

Introduction to the Extension of the Framework Types-To-Sets for Isabelle/HOL

Mihails Milehins

February 6, 2026

Abstract

In [19, 21], Ondřej Kunčar and Andrei Popescu propose an extension of the logic *Isabelle/HOL* and an associated algorithm for the relativization of *type-based theorems* to more flexible *set-based theorems*, collectively referred to as *Types-To-Sets*. One of the aims of their work was to open an opportunity for the development of a software tool for applied relativization in the implementation of the logic *Isabelle/HOL* in the proof assistant *Isabelle* [27]. In this document, we provide a prototype of a software framework for the interactive automated relativization of definitions and theorems in *Isabelle/HOL*, developed as an extension of the proof language *Isabelle/Isar* [31, 32]. The software framework incorporates the implementation of the proposed extension of the logic and associated tools provided in [19] and improved further in [14] by Fabian Immler and Bohua Zhan, and builds upon some of the ideas for further work expressed in [14] and [21].

Acknowledgements

The author would like to acknowledge the assistance that he received from the users of the mailing list of Isabelle in the form of answers given to his general queries and the persons responsible for the development of Types-To-Sets [19, 21] and its official extensions [14, 12] for providing explanations of the existing functionality of the framework and some of the ideas that laid the foundation of this work. Special thanks go to Fabian Immler for the conceptual design of the commands **tts_context**, **tts_lemmas** and **tts_lemma**, to Andrei Popescu for trying the software and providing feedback, and to Kevin Kappelmann for explaining certain aspects of [16]. Furthermore, the author would like to acknowledge the positive impact of [29] and [33] on his ability to code in Isabelle/ML [25, 33]. The author would also like to acknowledge the positive role that the numerous Q&A posted on the Stack Exchange network (especially Stack Overflow and TeX Stack Exchange) played in the development of this work. Finally, the author would like to express gratitude to all members of his family and friends for their continuous support.

Contents

1	ETTS: Reference Manual	6
1.1	Introduction	6
1.1.1	Background	6
1.1.2	Prerequisites and conventions	6
1.1.3	Previous work	7
1.1.4	Purpose and scope	8
1.2	ETTS and ERA	10
1.2.1	Background	10
1.2.2	Preliminaries	10
1.2.3	Set-based terms and their registration	10
1.2.4	Parameterization of the ERA	11
1.2.5	Definition of the ERA	12
1.3	Syntax	14
1.3.1	Background	14
1.3.2	Registration of the set-based terms	14
1.3.3	Relativization of theorems	14
1.4	ETTS by example	18
1.4.1	Background	18
1.4.2	First steps	18
2	ETTS Case Studies: Introduction	21
2.1	Background	21
2.1.1	Purpose	21
2.1.2	Related work	21
2.2	Examples: overview	21
2.2.1	Background	21
2.2.2	SML Relativization	21
2.2.3	TTS Vector Spaces	22
2.2.4	TTS Foundations	22
3	SML Relativization	23
3.1	Extension of the theory <i>Set</i>	23
3.2	Extension of the theory <i>Lifting-Set</i>	24
3.3	Extension of the theory <i>Product-Type-Ext</i>	26
3.4	Extension of the theory <i>Transfer</i>	27
3.5	Relativization of the results about relations	29
3.5.1	Definitions and common properties	29

3.5.2	Transfer rules I: <i>lfp</i> transfer	30
3.5.3	Transfer rules II: application-specific rules	31
3.6	Relativization of the results about orders	32
3.6.1	Class <i>ord</i>	32
3.6.2	Preorders	34
3.6.3	Partial orders	43
3.6.4	Dense orders	53
3.6.5	Partial orders with the greatest element and partial orders with the least elements	54
3.7	Relativization of the results about semigroups	60
3.7.1	Simple semigroups	60
3.7.2	Cancellative semigroups	61
3.7.3	Commutative semigroups	63
3.7.4	Cancellative commutative semigroups	65
3.8	Relativization of the results about monoids	67
3.8.1	Simple monoids	67
3.8.2	Commutative monoids	71
3.8.3	Cancellative commutative monoids	76
3.9	Relativization of the results about groups	78
3.9.1	Simple groups	78
3.9.2	Abelian groups	83
3.10	Relativization of the results about semirings	86
3.10.1	Semirings	86
3.10.2	Commutative semirings	86
3.10.3	Semirings with zero	87
3.10.4	Commutative semirings with zero	88
3.10.5	Cancellative semirings with zero	88
3.10.6	Commutative cancellative semirings with zero	89
3.10.7	Class <i>zero-neq-one</i>	90
3.10.8	Semirings with zero and one (rigs)	91
3.10.9	Commutative rigs	94
3.10.10	Cancellative rigs	96
3.10.11	Commutative cancellative rigs	97
3.11	Relativization of the results about rings	99
3.11.1	Rings	99
3.11.2	Commutative rings	101
3.11.3	Rings with identity	102
3.11.4	Commutative rings with identity	107
3.12	Relativization of the results about semilattices	110
3.12.1	Commutative bands	110
3.12.2	Simple upper and lower semilattices	111
3.12.3	Bounded semilattices	118
3.13	Relativization of the results about lattices	121
3.13.1	Simple lattices	121

3.13.2	Bounded lattices	124
3.13.3	Distributive lattices	127
3.14	Relativization of the results about complete lattices	129
3.14.1	Simple complete lattices	129
3.15	Relativization of the results about linear orders	144
3.15.1	Linear orders	144
3.15.2	Dense linear orders	149
3.16	Relativization of the results about simple topological spaces	152
3.16.1	Definitions and common properties	152
3.16.2	Transfer rules	154
3.16.3	Relativization	155
3.16.4	Further results	172
3.17	Relativization of the results related to the countability properties of topological spaces	173
3.17.1	First countable topological space	173
3.17.2	Topological space with a countable basis	175
3.17.3	Second countable topological space	176
3.18	Relativization of the results about ordered topological spaces	178
3.18.1	Ordered topological space	178
3.18.2	Linearly ordered topological space	179
3.19	Relativization of the results about product topologies	183
3.19.1	Definitions and common properties	183
3.19.2	Transfer rules	183
3.19.3	Relativization	183
4	TTS Vector Spaces	186
4.1	Introduction	186
4.1.1	Background	186
4.1.2	Prerequisites	186
4.2	Groups	187
4.2.1	Definitions and elementary properties	187
4.2.2	Instances (by type class constraints)	188
4.2.3	Transfer rules	189
4.2.4	Relativization.	190
4.3	Modules	192
4.3.1	<i>module-with</i>	192
4.3.2	<i>module-ow</i>	194
4.3.3	<i>module-on</i>	197
4.3.4	Relativization.	199
4.4	Vector spaces	210
4.4.1	<i>vector-space-with</i>	210
4.4.2	<i>vector-space-ow</i>	212
4.4.3	<i>vector-space-on</i>	219
4.4.4	Relativization : part I	221

4.4.5	Transfer: <i>dim</i>	225
4.4.6	Relativization: part II	225
5	TTS Foundations	241
5.1	Extension of the theory <i>Set</i>	241
5.2	Definite description operator	242
5.2.1	Definition and common properties	242
5.2.2	Transfer rules	243
5.3	Auxiliary	245
5.3.1	Methods	245
5.4	Abstract orders on types	246
5.4.1	Background	246
5.4.2	Order operations	246
5.4.3	Preorders	247
5.4.4	Partial orders	250
5.4.5	Dense orders	252
5.4.6	(Unique) top and bottom elements	254
5.4.7	(Unique) top and bottom elements for partial orders	254
5.4.8	Partial orders without top or bottom elements	255
5.4.9	Least and greatest operators	255
5.4.10	min and max	256
5.4.11	Monotonicity	257
5.4.12	Set intervals	259
5.4.13	Bounded sets	264
5.5	Abstract orders on explicit sets	268
5.5.1	Background	268
5.5.2	Order operations	268
5.5.3	Preorders	273
5.5.4	Partial orders	280
5.5.5	Dense orders	287
5.5.6	(Unique) top and bottom elements	290
5.5.7	Absence of top or bottom elements	292
5.6	Abstract semigroups on types	295
5.6.1	Background	295
5.6.2	Preliminaries	295
5.6.3	Binary operations	295
5.6.4	Simple semigroups	296
5.6.5	Commutative semigroups	296
5.6.6	Cancellative semigroups	297
5.6.7	Cancellative commutative semigroups	298
5.7	Extension of the theory <i>Lifting-Set</i>	301
5.8	Abstract semigroups on sets	303
5.8.1	Background	303
5.8.2	Binary operations	303

5.8.3	Simple semigroups	304
5.8.4	Commutative semigroups	305
5.8.5	Cancellative semigroups	307
5.8.6	Cancellative commutative semigroups	309
Bibliography		313

ETTS: Reference Manual

1.1 Introduction

1.1.1 Background

The *standard library* that is associated with the object logic Isabelle/HOL and provided as a part of the standard distribution of Isabelle [1] contains a significant number of formalized results from a variety of fields of mathematics (e.g., order theory, algebra, topology). Nevertheless, using the argot that was promoted in the original publication of Types-To-Sets [19], the formalization is performed using a type-based approach. Thus, for example, the carrier sets associated with the algebraic structures and the underlying sets of the topological spaces, effectively, consist of all terms of an arbitrary type. This restriction can create an inconvenience when working with mathematical objects induced on a subset of the carrier set/underlying set associated with the original object (e.g., see [14]).

To address this limitation, several additional libraries were developed upon the foundations of the standard library (e.g., *HOL-Algebra* [5] and *HOL-Analysis* [2]). In terms of the argot associated with Types-To-Sets, these libraries provide the set-based counterparts of many type-based theorems in the standard library, along with a plethora of additional results. Nonetheless, the proofs of the majority of the theorems that are available in the standard library are restated explicitly in the libraries that contain the set-based results. This unnecessary duplication of efforts is one of the primary problems that the framework Types-To-Sets is meant to address.

The framework Types-To-Sets offers a unified approach to structuring mathematical knowledge formalized in Isabelle/HOL: potentially, every type-based theorem can be converted to a set-based theorem in a semi-automated manner and the relationship between such type-based and set-based theorems can be articulated clearly and explicitly [19]. In this document, we describe a particular implementation of the framework Types-To-Sets in Isabelle/HOL that takes the form of a further extension of the language Isabelle/Isar with several new commands and attributes (e.g., see [34]).

1.1.2 Prerequisites and conventions

A reader of this document is assumed to be familiar with the proof assistant Isabelle, the proof language Isabelle/Isar, the meta-logic Isabelle/Pure and the object logic Isabelle/HOL, as described in, [27, 34], [31, 32, 34], [28, 34] and [20], respectively. Familiarity with the content of the original articles about Types-To-Sets [19, 21] and the first large-scale application of Types-To-Sets (as described in [14]) is not essential but can be useful.

The notational conventions that are used in this document are approximately equivalent to the conventions that were suggested in [19], [20] and [21]. However, a disparity comes from our use of explicit notation for the *schematic variables*. In Isabelle/HOL, free variables that occur in the theorems at the top-level in the theory context are generalized implicitly, which may be expressed by replacing fixed variables by schematic variables [35]. In this article, the schematic variables will be prefixed with the question mark “?”, like so: ?*a*. Nonetheless, explicit quantification over the type variables at the top-level is also allowed: $\forall\alpha.\phi[\alpha]$. Lastly, the square brackets may be

used either for the denotation of substitution or to indicate that certain types/terms are a part of a term: $t[?\alpha]$.

1.1.3 Previous work

Relativization Algorithm

Let ${}_{\alpha}(\beta \approx U)_{\text{Rep}}^{\text{Abs}}$ denote

$$\begin{aligned} & (\forall x_{\beta}.\text{Rep } x \in U) \wedge \\ & (\forall x_{\beta}.\text{Abs } (\text{Rep } x) = x) \wedge \quad , \\ & (\forall y_{\alpha}.y \in U \longrightarrow \text{Rep } (\text{Abs } y) = y) \end{aligned}$$

let \sim denote the constant/type *dependency relation* (see subsection 2.3 in [19]), let \sim^{\downarrow} be a *substitutive closure* of the constant/type dependency relation, let R^+ denote the transitive closure of the binary relation R , let Δ_c denote the set of all types for which c is *overloaded* and let $\sigma \notin S$ mean that σ is not an instance of any type in S (see [19] and [20]).

The framework Types-To-Sets extends Isabelle/HOL with two axioms: *Local Typedef Rule* (LT) and the *Unoverloading Rule* (UO). The consistency of Isabelle/HOL augmented with the LT and the UO is proved in Theorem 11 in [20].

The LT is given by

$$\frac{\Gamma \vdash U \neq \emptyset \quad \Gamma \vdash (\exists \text{Abs Rep}.\sigma(\beta \approx U)_{\text{Rep}}^{\text{Abs}} \longrightarrow \phi)}{\Gamma \vdash \phi} \quad \beta \notin U, \phi, \Gamma$$

Thus, if β is fresh for the non-empty set $U_{\sigma \text{ set}}$, the formula ϕ and the context Γ , then ϕ can be proved in Γ by assuming the existence of a type β isomorphic to U .

The UO is given by

$$\frac{\phi}{\forall x_{\sigma}.\phi[x_{\sigma}/c_{\sigma}]} \quad \forall u \text{ in } \phi.\neg(u \sim^{\downarrow+} c_{\sigma}); \quad \sigma \notin \Delta_c$$

Thus, a *constant-instance* c_{σ} may be replaced by a universally quantified variable x_{σ} in ϕ , if all types and constant-instances in ϕ do not semantically depend on c_{σ} through a chain of constant and type definitions and there is no matching definition for c_{σ} .

The statement of the *original relativization algorithm* (ORA) can be found in subsection 5.4 in [19]. Here, we present a variant of the algorithm that includes some of the amendments that were introduced in [14], which will be referred to as the *relativization algorithm* (RA). The differences between the ORA and the RA are implementation-specific and have no effect on the output of the algorithm, if applied to a conventional input. Let \bar{a} denote a finite sequence a_1, \dots, a_n for some positive integer n ; let Υ be a *type class* [26, 30, 10] that depends on the overloaded constants \bar{x} and let $U \downarrow \bar{f}$ be used to state that U is closed under the operations \bar{f} ; then the RA is given by

$$\begin{aligned} & \frac{}{\vdash \phi[?\alpha_{\Upsilon}]} \quad (1) \\ & \frac{}{\vdash \phi_{\text{with}}[?\alpha_{\Upsilon}, \bar{x}]} \quad (2) \\ & \frac{\vdash \Upsilon_{\text{with}} \bar{f}[?\alpha] \longrightarrow \phi_{\text{with}}[?\alpha, \bar{f}]}{\vdash \Upsilon_{\text{with}} \bar{f}[?\alpha] \longrightarrow \phi_{\text{with}}[?\alpha, \bar{f}]} \quad (3) \\ & \frac{U \neq \emptyset, \alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}} \vdash \Upsilon_{\text{with}} \bar{f}[?\alpha] \longrightarrow \phi_{\text{with}}[?\alpha, \bar{f}]}{U \neq \emptyset, \alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}} \vdash \Upsilon_{\text{with}} \bar{f}[\beta] \longrightarrow \phi_{\text{with}}[\beta, \bar{f}]} \quad (4) \\ & \frac{U \neq \emptyset, \alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}} \vdash U \downarrow \bar{f}[\alpha] \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{f}]}{U \neq \emptyset, \alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}} \vdash U \downarrow \bar{f}[\alpha] \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{f}]} \quad (5) \\ & \frac{U_{\alpha \text{ set}} \neq \emptyset \vdash U \downarrow \bar{f}[\alpha] \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{f}]}{\vdash ?U_{\alpha \text{ set}} \neq \emptyset \longrightarrow ?U \downarrow \bar{f}[\alpha] \longrightarrow \Upsilon_{\text{with}}^{\text{on}} ?U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[?\alpha, ?U, \bar{f}]} \quad (6) \\ & \frac{}{\vdash ?U_{\alpha \text{ set}} \neq \emptyset \longrightarrow ?U \downarrow \bar{f}[\alpha] \longrightarrow \Upsilon_{\text{with}}^{\text{on}} ?U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[?\alpha, ?U, \bar{f}]} \quad (7) \end{aligned}$$

The input to the RA is assumed to be a theorem $\vdash \phi[?\alpha\Upsilon]$. Step 1 will be referred to as the first step of the dictionary construction (subsection 5.2 in [19]); step 2 as unoverloading of the type $?\alpha\Upsilon$: it includes class internalization (subsection 5.1 in [19]) and the application of the UO (it corresponds to the application of the attribute *unoverload-type* [14]); step 3 provides the assumptions that are the prerequisites for the application of the LT; step 4 is reserved for the concrete type instantiation; step 5 is the application of *Transfer* (section 6 in [19]); step 6 refers to the application of the LT; step 7 is the export of the theorem from the local context [33].

Implementation of Types-To-Sets

In [19], the authors extended the implementation of Isabelle/HOL with the LT and UO. Also, they introduced the attributes *internalize-sort*, *unoverload* and *cancel-type-definition* that allowed for the execution of steps 1, 3 and 7 (respectively) of the ORA. Other steps could be performed using the technology that already existed. In [14], the implementation was augmented with the attribute *unoverload-type*, which largely subsumed the functionality of the attributes *internalize-sort* and *unoverload*.

The examples of the application of the ORA to theorems in Isabelle/HOL that were developed in [19] already contained an implicit suggestion that the constants and theorems needed for the first step of the dictionary construction in step 2 of the ORA and the *transfer rules* [18] needed for step 6 of the ORA can and should be obtained prior to the application of the algorithm. Thus, using the notation from subsection 1.1.3, for each constant-instance c_σ that occurs in the type-based theorem $\vdash \phi[?\alpha\Upsilon]$ prior to the application of the ORA with respect to $\alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}}$, the users were expected to provide an *unoverloaded* constant c_{with} such that $c_\sigma = c_{\text{with}} \bar{x}$, and a *relativized* constant $c_{\text{with}}^{\text{on}}$ such that $R[T_{\alpha \rightarrow \beta \rightarrow \mathbb{B}}](c_{\text{with}}^{\text{on}} U_{\alpha \text{ set}}) c_{\text{with}}$ (\mathbb{B} denotes the built-in Isabelle/HOL type *bool* [18]) is a conditional transfer rule (e.g., see [11]), with T being a binary relation that is at least right-total and bi-unique, assuming the default order on predicates in Isabelle/HOL (see [18]).

The unoverloading [17] and relativization of constants for the application of the RA was performed manually in [19, 21, 14]. Nonetheless, unoverloading could be performed using the *classical overloading elimination algorithm* proposed in [17], but it is likely that an implementation of this algorithm was not publicly available at the time of writing of this document. In [12], an alternative algorithm was implemented and made available via the command **unoverload-definition**, although it suffers from several limitations in comparison to the algorithm in [17]. The transfer rules for the constants that are conditionally parametric can be synthesized automatically using the command **parametric-constant** [9] from the standard distribution of Isabelle; the framework *autoref* [22] allows for the synthesis of transfer rules $R t t'$, including both the *parametricity relation* [18] R and the term t , based on t' , under favorable conditions; lastly, in [22] and [14], the authors suggest an outline of another feasible algorithm for the synthesis of the transfer rules based on the functionality of the framework *Transfer* [11], but do not provide an implementation.

Finally, the assumption $\alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}}$ can be stated using the constant *type-definition* from the standard library of Isabelle/HOL as *type-definition Rep Abs U*; the instantiation of types required in step 4 of the RA can be performed using the standard attributes of Isabelle; step 6 can be performed using the attribute *cancel-type-definition* developed in [19]; step 7 is expected to be performed manually by the user.

1.1.4 Purpose and scope

The extension of the framework Types-To-Sets that is described in this manual adds a further layer of automation to the existing implementation of the framework Types-To-Sets. The primary functionality of the extension is available via the following Isar commands: **tts-context**,

tts-lemmas and **tts-lemma** (and the synonymous commands **tts-corollary**, **tts-proposition** and **tts-theorem**¹). The commands **tts-lemmas** and **tts-lemma**, when invoked inside an appropriately defined **tts-context** provide the functionality that is approximately equivalent to the application of all steps of the RA and several additional steps of pre-processing of the input and post-processing of the result (collectively referred to as the *extended relativization algorithm* or ERA).

As part of our work on the ETTS, we have also designed the auxiliary framework *Conditional Transfer Rule* (CTR). The framework CTR provides the commands **ud** and **ctr** for the automation of unoverloading of definitions and synthesis of conditional transfer rules from definitions, respectively. Further information about this framework can be found in its reference manual [24]. In this context, we also mention that both the CTR and the ETTS were tested using the framework SpecCheck [16].²

The extension was designed under a policy of non-intervention with the existing implementation of the framework Types-To-Sets. Therefore, it does not reduce the scope of the applicability of the framework. However, the functionality that is provided by the commands associated with the extension is a proper subset of the functionality provided by the existing implementation. Nevertheless, the author of the extension believes that there exist very few practical applications of the relativization algorithm that can be solved using the original interface but cannot be solved using the commands that are introduced within the scope of the extension.

¹In what follows, any reference to the command **tts-lemma** should be viewed as a reference to the entire family of the commands with the identical functionality.

²Some of the elements of the content of the tests are based on the elements of the content of [6].

1.2 ETTS and ERA

1.2.1 Background

In this section, we describe our implementation of the prototype software framework ETTS that offers the integration of Types-To-Sets with the Isabelle/Isar infrastructure and full automation of the application of the ERA under favorable conditions. The design of the framework rests largely on our interpretation of several ideas expressed by the authors of [21] and [14].

It has already been mentioned that the primary functionality of the ETTS is available via the Isabelle/Isar [31, 32] commands **tts-context**, **tts-lemmas** and **tts-lemma**. There also exist secondary commands aimed at resolving certain specific problems that one may encounter during relativization: **tts-register-sbts** and **tts-find-sbts**. More specifically, these commands provide means for using transfer rules stated in a local context during the step of the ERA that is similar to step 5 of the RA. The functionality of these commands is explained in more detail in subsection 1.2.3 below.

It is important to note that the description of the ETTS presented in this subsection is only a simplified model of its programmatic implementation in *Isabelle/ML* [25, 33].

1.2.2 Preliminaries

The ERA is an extension of the RA that provides means for the automation of a design pattern similar to the one that was proposed in [14], as well as several additional steps for pre-processing of the input and post-processing of the result of the relativization. In a certain restricted sense the ERA can be seen as a localized form of the RA, as it provides additional infrastructure aimed specifically at making the relativization of theorems stated in the context of Isabelle's *locales* [15, 3, 4] more convenient.

In what follows, assume the existence of an underlying well-formed theory D that contains all definitional axioms that appear in the standard library of Isabelle/HOL. If $\Gamma \vdash_{\alpha} (\beta \approx U)_{\text{Rep}}^{\text{Abs}}$ and $\beta, U_{\alpha \text{ set}}, \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta} \in \Gamma$, then the 4-tuple $(U_{\alpha \text{ set}}, \beta, \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta})$, will be referred to as a *relativization isomorphism* (RI) *with respect to* Γ (or RI, if Γ can be inferred). Given the RI $(U_{\alpha \text{ set}}, \beta, \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta})$, the term $U_{\alpha \text{ set}}$ will be referred to as the *set associated with the RI*, β will be referred to as the *type variable associated with the RI*, $\text{Rep}_{\beta \rightarrow \alpha}$ will be referred to as the *representation associated with the RI* and $\text{Abs}_{\alpha \rightarrow \beta}$ will be referred to as the *abstraction associated with the RI*. Moreover, any typed term variable $T_{\alpha \rightarrow \beta \rightarrow \mathbb{B}}$ such that $\Gamma \vdash T = (\lambda x y. \text{Rep } y = x)$ will be referred to as the *transfer relation associated with the RI*. $\Gamma \vdash \text{Domainp } T = (\lambda x. x \in U)$ that holds for this transfer relation will be referred to as the *transfer domain rule associated with the RI*, $\Gamma \vdash \text{bi_unique } T$ and $\Gamma \vdash \text{right_total } T$ will be referred to as the *side conditions associated with the RI*. For brevity, the abbreviation $\text{dbr}[T_{\alpha \rightarrow \beta \rightarrow \mathbb{B}}, U_{\alpha \text{ set}}]$ will be used to mean that $\text{Domainp } T = (\lambda x. x \in U)$, $\text{bi_unique } T$ and $\text{right_total } T$ for any $\alpha, \beta, T_{\alpha \rightarrow \beta \rightarrow \mathbb{B}}$ and $U_{\alpha \text{ set}}$.

1.2.3 Set-based terms and their registration

Perhaps, one of the most challenging aspects of the automation of the relativization process is related to the application of Transfer during step 5 of the RA: a suitable transfer rule for a given constant-instance may exist only under non-conventional side conditions: an important example that showcases this issue is the built-in constant ε (see [21] and [14] for further information). Unfortunately, the ETTS does not offer a fundamental solution to this problem: the responsibility for providing suitable transfer rules for the application of the ERA remains at the discretion of the user. Nonetheless, the ETTS does provide additional infrastructure that may improve the user experience when dealing with the transfer rules that can only be conveniently stated in an explicitly relativized local context (usually a relativized locale): a common problem

that was already explored in [14].

The authors of [14] choose to perform the relativization of theorems that stem from their specifications in a locale context from within another dedicated relativized locale context. The relativized operations that are represented either by the locale parameters of the relativized locale or remain overloaded constants associated with a given class constraint are lifted to the type variables associated with the RIs that are used for the application of a variant of the relativization algorithm. This variant includes a step during which the variables introduced during unoverloading are substituted (albeit implicitly) for the terms that represent the lifted locale parameters and constants. The additional infrastructure and the additional step are needed, primarily, for the relativization of the constants whose transfer rules can only be stated conveniently in the context of the relativized locale.

A similar approach is used in the ETTS. However, instead of explicitly declaring the lifted constants in advance of the application of the RA, the user is expected to perform the registration of the so-called *set-based term* (sbterm) for each term of interest that is a relativization of a given concept.

The inputs to the algorithm that is associated with the registration of the sbterms are a context Γ , a term $t : \bar{\alpha} K$ (K , applied using a postfix notation, contains all information about the type constructors of the type $\bar{\alpha} K$) and a sequence of n distinct typed variables \bar{U} with distinct types of the form $\alpha \text{ set}$, such that $\bar{\alpha}$ is also of length n , all free variables and free type variables that occur in $t : \bar{\alpha} K$ also appear free in Γ and $\bar{U}_i : \bar{\alpha}_i \text{ set}$ for all i , $1 \leq i \leq n$.

Firstly, a term $\exists b. R[\bar{A}]_{\bar{\alpha} K \rightarrow \bar{\beta} K \rightarrow \mathbb{B}} t b$ is formed, such that $R[\bar{A}]$ is a parametricity relation associated with some type $\bar{\gamma} K$ for $\bar{\gamma}$ of length n , such that the sets of the elements of $\bar{\alpha}$, $\bar{\beta}$ and $\bar{\gamma}$ are pairwise disjoint, \bar{A} and $\bar{\beta}$ are both of length n , the elements of \bar{A} , $\bar{\beta}$ and $\bar{\gamma}$ are fresh for Γ and $\bar{A}_i : \bar{\alpha}_i \rightarrow \bar{\beta}_i \rightarrow \mathbb{B}$ for all i such that $1 \leq i \leq n$. Secondly, the context Γ' is built incrementally starting from Γ by adding the formulae $\text{dbr}[\bar{A}_i, \bar{U}_i]$ for each i such that $1 \leq i \leq n$. The term presented above serves as a goal that is meant to be discharged by the user in Γ' , resulting in the deduction

$$\Gamma \vdash \text{dbr}[\bar{A}_i, \bar{U}_i] \longrightarrow \exists b. R[\bar{A}]_{\bar{\alpha} K \rightarrow \bar{\beta} K \rightarrow \mathbb{B}} t b$$

(the index i is distributed over n) after the export to the context Γ . Once the proof is completed, the result is registered in the so-called *sbt-database* allowing a lookup of such results by the sbterm t (the terms and results are allowed to morph when the lookup is performed from within a context different from Γ [7]).

1.2.4 Parameterization of the ERA

Assuming the existence of some context Γ , the ERA is parameterized by the *RI specification*, the *sbterm specification*, the *rewrite rules for the set-based theorem*, the *known premises for the set-based theorem*, the *specification of the elimination of premises in the set-based theorem* and the *attributes for the set-based theorem*. A sequence of the entities in the list above will be referred to as the *ERA-parameterization for Γ* .

The RI Specification is a finite non-empty sequence of pairs of variables $(? \gamma, U_{\alpha \text{ set}})$, such that $U_{\alpha \text{ set}} \in \Gamma$. The individual elements of the RI specification will be referred to as the *RI specification elements*. The first element of the RI specification element will be referred to as the *schematic type variable associated with the RI specification element*, the second element will be referred to as the *set associated with the RI specification element*.

The sbterm specification is a finite sequence of pairs $(t : ? \bar{\alpha} K, u : \bar{\beta} K)$, where t is either a constant-instance or a schematic typed term variable and u is an sbterm with respect to Γ . The notation for the elements of the sbterm specification follows a convention similar to the one introduced for the RI specification elements.

The rewrite rules for the set-based theorem can be any set of valid rules for the Isabelle simplifier [34]; the known premises for the set-based theorem can be any finite sequence of deductions in Γ ; the specification of the elimination of premises in the set-based theorem is a pair (\bar{t}, m) , where \bar{t} is a sequence of formulae and m is a proof method; the attributes for the set-based theorem is a sequence of attributes of Isabelle (e.g., see [34]).

1.2.5 Definition of the ERA

The relativization is performed inside a local context Γ with an associated ERA-parameterization (such a context-parameterization pair will be called a *tts context*). The ERA provides explicit support for handling the transfer rules local to the context through the infrastructure for the registration of sbterms, as explained in subsection 1.2.3. Apart from this, the main part of the ERA largely follows the outline of the RA. However, the ERA also provides several tools for post-processing of the raw result of the relativization. The ERA also has an initialization stage, but this stage is largely hidden from the end-user. Thus, the ERA can be divided in three distinct parts: *initialization of the relativization context*, *kernel of the ERA* (KERA) and *post-processing*.

Assume that the context Γ contains the variable $U_{\alpha \text{ set}}$ and the finite sequences of variables \bar{g} and \bar{f} indexed by I and J , respectively, such that $\bar{g}_i : \alpha \bar{K}_i$ and $\bar{f}_j : \alpha \bar{L}_j$ for all $i \in I$ and $j \in J$ for some sequences of functions \bar{K} and \bar{L} also indexed by I and J , respectively, representing the type constructors. Also, assume that the input to the relativization algorithm is the type-based theorem $\vdash \phi[? \alpha_{\Upsilon}, ? \bar{h}[? \alpha_{\Upsilon}]]$ such that $? \bar{h}$ is indexed by I and Υ depends on the overloaded constants \bar{x} indexed by J . Finally, assume that the ERA is parameterized by the RI specification $(? \alpha_{\Upsilon}, U_{\alpha \text{ set}})$ and the sbterm specification elements $(? \bar{h}_i, \bar{g}_i)$ and (\bar{x}_j, \bar{f}_j) for all $i \in I$ and $j \in J$.

Initialization of the relativization context. Prior to the application of the relativization algorithm, the formula $\exists \text{Rep Abs. } \alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}}$ is added to the context Γ , with the type variable β being fresh for Γ , resulting in a new context Γ' such that $\Gamma \subseteq \Gamma'$ and $\exists \text{Rep Abs. } \alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}} \in \Gamma'$. Then, the properties of the Hilbert choice ε are used for the definition of **Rep** and **Abs** such that $\Gamma' \vdash \alpha(\beta \approx U)_{\text{Rep}}^{\text{Abs}}$ (e.g., see [18]). In this case, $(U_{\alpha \text{ set}}, \beta, \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta})$ is an RI with respect to Γ' . Furthermore, a fresh $T_{\alpha \rightarrow \beta \rightarrow \mathbb{B}}$ (for Γ) is defined as a transfer relation associated with the RI. Finally, the transfer domain rule associated with the RI and the side conditions associated with the RI are proved for T with respect to Γ' . For each \bar{g}_i such that $i \in I$, the sbt-database contains a deduction $\Gamma \vdash \text{dbr}[? A, U] \longrightarrow \exists a. R[? A]_{\alpha \bar{K}_i \rightarrow ? \delta \bar{K}_i \rightarrow \mathbb{B}} \bar{g}_i a$. Thence, for each $i \in I$, $? \delta$ is instantiated as β and $? A_{\alpha \rightarrow ? \delta \rightarrow \mathbb{B}}$ is instantiated as $T_{\alpha \rightarrow \beta \rightarrow \mathbb{B}}$. The resulting theorems are used for the definition of a fresh (for Γ) sequence of variables \bar{a} such that $\Gamma' \vdash R[T_{\alpha \rightarrow \beta \rightarrow \mathbb{B}}]_{\alpha \bar{K}_i \rightarrow \beta \bar{K}_i \rightarrow \mathbb{B}} \bar{g}_i \bar{a}_i$. Similar deductions are also established for the sequence \bar{f} , with the sequence of the variables appearing on the right-hand side of the transfer rules denoted by \bar{b} . These deductions are meant to be used by the transfer infrastructure during the step of the ERA that is equivalent to step 5 of the RA, as shown below. Thus, at the end of the initialization of the relativization context, the theorem is transformed into a deduction of the form $\Gamma' \vdash \phi[? \alpha_{\Upsilon}, ? \bar{h}[? \alpha_{\Upsilon}]]$, where the context Γ' (called the *relativization context*) is such that $\Gamma \subseteq \Gamma'$ and it has an associated relativization isomorphism $(U_{\alpha \text{ set}}, \beta, \text{Rep}_{\beta \rightarrow \alpha}, \text{Abs}_{\alpha \rightarrow \beta})$ for some fresh β , the associated fresh transfer relation T and fresh variables \bar{a} and \bar{b} indexed by I and J (with freshness being assessed with respect to Γ). Also, the following transfer rules are provided for all $i \in I$ and $j \in J$: $\Gamma' \vdash \bar{R}_i[T] \bar{g}_i \bar{a}_i$ and $\Gamma' \vdash \bar{R}'_j[T] \bar{f}_j \bar{b}_j$ (\bar{R} and \bar{R}' are sequences of parametricity relations indexed by I and J , respectively).

Kernel of ERA. The KERA is similar to the RA:

$$\frac{\Gamma' \vdash \phi[? \alpha_{\Upsilon}, ? \bar{h}[? \alpha_{\Upsilon}]]}{\Gamma' \vdash \phi_{\text{with}}[? \alpha_{\Upsilon}, ? \bar{h}[? \alpha_{\Upsilon}], \bar{x}]} \quad (1)$$

$$\frac{\Gamma' \vdash \Upsilon_{\text{with}} ? \bar{f}[? \alpha] \longrightarrow \phi_{\text{with}}[? \alpha, ? \bar{h}[? \alpha], ? \bar{f}]}{\Gamma' \vdash \Upsilon_{\text{with}} ? \bar{f}[\beta] \longrightarrow \phi_{\text{with}}[\beta, ? \bar{h}[\beta], ? \bar{f}]} \quad (2)$$

$$\frac{\Gamma' \vdash \Upsilon_{\text{with}} ? \bar{f}[\beta] \longrightarrow \phi_{\text{with}}[\beta, ? \bar{h}[\beta], ? \bar{f}]}{\Gamma' \vdash \Upsilon_{\text{with}} \bar{b} \longrightarrow \phi_{\text{with}}[\beta, \bar{a}, \bar{b}]} \quad (3)$$

$$\frac{\Gamma' \vdash \Upsilon_{\text{with}} \bar{b} \longrightarrow \phi_{\text{with}}[\beta, \bar{a}, \bar{b}]}{\Gamma' \vdash U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{g}, \bar{f}]} \quad (4)$$

$$\frac{\Gamma' \vdash U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{g}, \bar{f}]}{\Gamma \vdash \exists \text{Rep Abs.}_{\alpha}(\beta \approx U)_{\text{Rep}}^{\text{Abs}} \longrightarrow U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{g}, \bar{f}]} \quad (5)$$

$$\frac{\Gamma \vdash \exists \text{Rep Abs.}_{\alpha}(\beta \approx U)_{\text{Rep}}^{\text{Abs}} \longrightarrow U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{g}, \bar{f}]}{\Gamma \vdash U \neq \emptyset \longrightarrow U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{g}, \bar{f}]} \quad (6)$$

$$\frac{\Gamma \vdash U \neq \emptyset \longrightarrow U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{g}, \bar{f}]}{\Gamma \vdash U \neq \emptyset \longrightarrow U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon_{\text{with}}^{\text{on}} U \bar{f} \longrightarrow \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{g}, \bar{f}]} \quad (7)$$

Thus, step 1 will be referred to as the first step of the dictionary construction (similar to step 1 of the RA); step 2 will be referred to as unoverloading of the type $? \alpha_{\Upsilon}$: it includes class internalization and the application of the UO (similar to step 2 of the RA); in step 3, $? \alpha$ is instantiated as β using the RI specification (similar to step 4 in the RA); in step 4, the sbterm specification is used for the instantiation of $? \bar{h}$ as \bar{a} and $? \bar{f}$ as \bar{b} ; step 5 refers to the application of transfer, including the transfer rules associated with the sbterms (similar to step 5 in the RA); in step 6, the result is exported from Γ' to Γ , providing the additional premise $\exists \text{Rep Abs.}_{\alpha}(\beta \approx U)_{\text{Rep}}^{\text{Abs}}$; step 7 is the application of the attribute *cancel-type-definition* (similar to step 6 in the RA).

The RI specification and the sbterm specification provide the information that is necessary to perform the type and term substitutions in steps 3 and 4 of the KERA. If the specifications are viewed as finite maps, their domains morph along the transformations that the theorem undergoes until step 4.

Post-processing. The deduction that is obtained in the final step of the KERA can often be simplified further. The following post-processing steps were created to allow for the presentation of the set-based theorem in a format that is both desirable and convenient for the usual applications:

1. *Simplification.* The rewriting is performed using the rewrite rules for the set-based theorem: the implementation relies on the functionality of the Isabelle's simplifier.
2. *Substitution of known premises.* The known premises for the set-based theorem are matched with the premises of the set-based theorem, allowing for their elimination.
3. *Elimination of premises.* Each premise is matched against each term in the specification of the elimination of premises in the set-based theorem; the associated method is applied in an attempt to eliminate the matching premises (this can be useful for the elimination of the premises of the form $U \neq \emptyset$).
4. *Application of the attributes for the set-based theorem.* The attributes for the set-based theorem are applied as the final step during post-processing.

Generally, the desired form of the result after a successful application of post-processing is similar to $\Gamma \vdash \phi_{\text{with}}^{\text{on}}[\alpha, U, \bar{g}, \bar{f}]$ with the premises $U \neq \emptyset$, $U \downarrow \bar{g}, \bar{f}$ and $\Upsilon_{\text{with}}^{\text{on}} U \bar{f}$ eliminated completely (these premises can often be inferred from the context Γ).

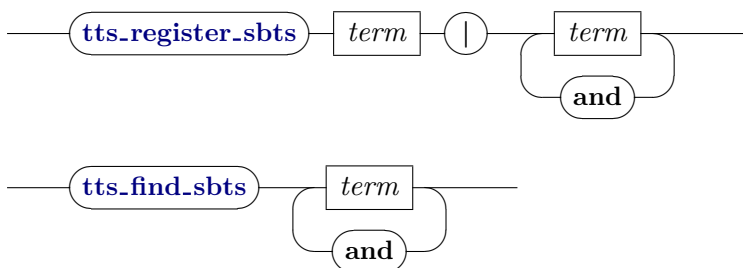
1.3 Syntax

1.3.1 Background

This section presents the syntactic categories that are associated with the commands **tts-context**, **tts-lemmas**, **tts-lemma**, and several other closely related auxiliary commands. It is important to note that the presentation of the syntax is approximate.

1.3.2 Registration of the set-based terms

tts-register-sbts : *local-theory* → *proof(prove)*
tts-find-sbts : *context* →



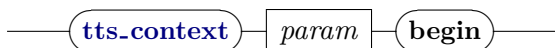
tts-register-sbts $t \mid U_1$ **and** ... **and** U_n allows for the registration of the set-based terms in the sbt-database. Generally, U_i ($1 \leq i \leq n$) must be distinct fixed variables with distinct types of the form '*a set*', with the set of the type variables that occur in the types of U_i equivalent to the set of the type variables that occur in the type of t .

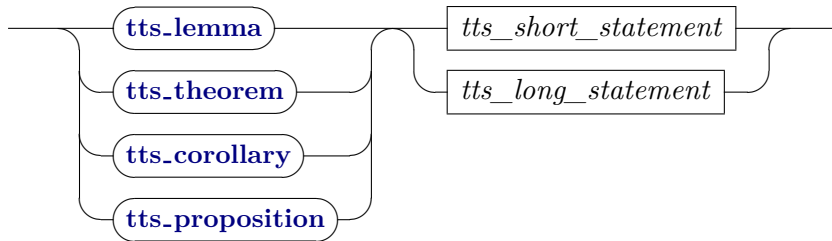
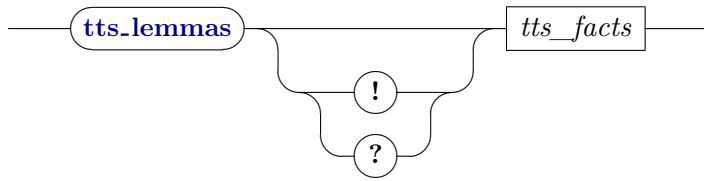
tts-find-sbts t_1 **and** ... **and** t_n prints the templates for the transfer rules for the set-based terms $t_1 \dots t_n$ for some positive integer n . If no arguments are provided, then the templates for all sbterms in the sbt-database are printed.

1.3.3 Relativization of theorems

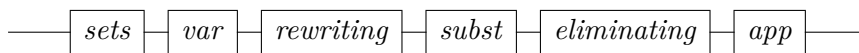
tts-context : *theory* → *local-theory*
tts-lemmas : *local-theory* → *local-theory*
tts-lemma : *local-theory* → *proof(prove)*
tts-theorem : *local-theory* → *proof(prove)*
tts-corollary : *local-theory* → *proof(prove)*
tts-proposition : *local-theory* → *proof(prove)*

The relativization of theorems should always be performed inside an appropriately parameterized tts context. The tts context can be set up using the command **tts-context**. The framework introduces two types of interfaces for the application of the extended relativization algorithm: **tts-lemmas** and the family of the commands with the identical functionality: **tts-lemma**, **tts-theorem**, **tts-corollary**, **tts-proposition**. Nonetheless, the primary purpose of the command **tts-lemmas** is the experimentation and the automated generation of the relativized results stated using the command **tts-lemma**.

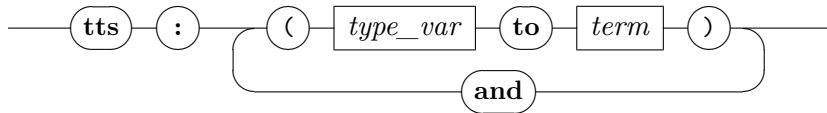




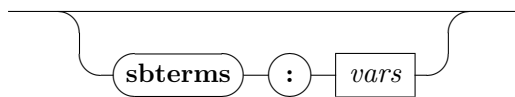
param



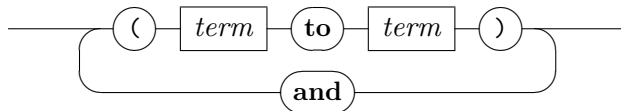
sets



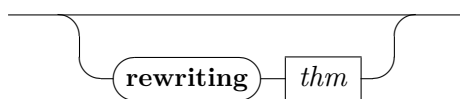
var



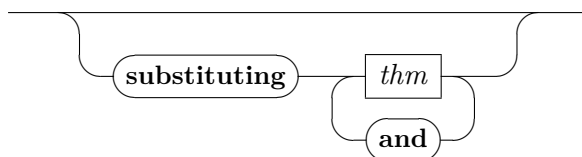
vars



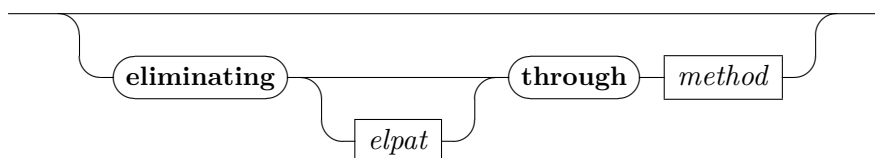
rewriting

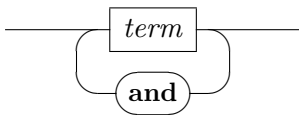
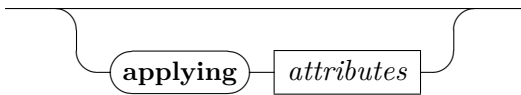
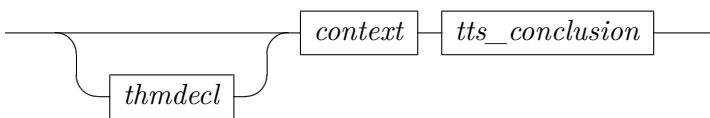
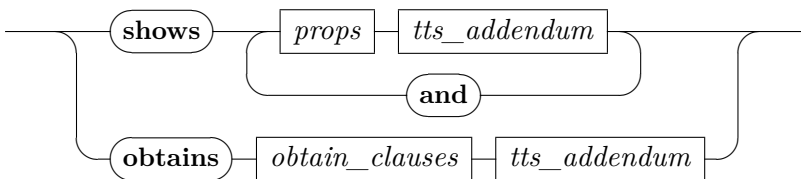
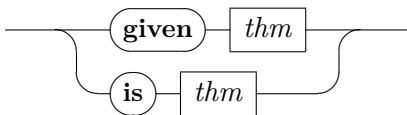
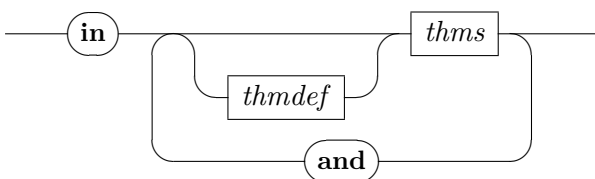


subst



eliminating



elpat*app**tts_short_statement**tts_long_statement**tts_conclusion**tts_addendum**tts_facts*

tts-context *param begin* provides means for the specification of a new (unnamed) tts context.

tts : ($?a_1$ to U_1) **and** ... **and** ($?a_n$ to U_n) provides means for the declaration of the RI specification. For each i ($1 \leq i \leq n$, n — positive integer), $?a_i$ must be a schematic type variable that occurs in each theorem provided as an input to the commands **tts-lemmas** and **tts-lemma** invoked inside the tts context and U_i can be any term of the type '*a set*', where '*a*' is a fixed type variable.

sbtterms : ($tbcv_1$ to sbt_1) **and** ... **and** ($tbcv_n$ to sbt_n) can be used for the declaration of the sbterm specification. For each individual entry i , such that $1 \leq i \leq n$ with n being a non-negative integer, $tbcv_i$ has to be either an overloaded operation that occurs in every theorem that is provided as an input to the extended relativization algorithm or a schematic variable that occurs in every theorem that is provided as an input to the command, and sbt_i has to be a term registered in the sbt-database.

rewriting *thm* provides means for the declaration of the rewrite rules for the set-based theorem.

substituting *thm*₁ **and** ... **and** *thm*_{*n*} (*n* — non-negative integer) provides means for the declaration of the known premises for the set-based theorem.

eliminating *term*₁ **and** ... **and** *term*_{*n*} **through** *method* (*n* — non-negative integer) provides means for the declaration of the specification of the elimination of premises in the set-based theorem.

applying [*attr*₁, ..., *attr*_{*n*}] (*n* — non-negative integer) provides means for the declaration of the attributes for the set-based theorem.

tts-lemmas applies the ERA to a list of facts and saves the resulting set-based facts in the context. The command **tts-lemmas** should always be invoked from within a **tts** context. If the statement of the command is followed immediately by the optional keyword **!**, then it operates in the verbose mode, printing the output of the application of the individual steps of the ERA. If the statement of the command is followed immediately by the optional keyword **?**, then the command operates in the active mode, outputting the set-based facts in the form of the “active areas” that can be embedded in the Isabelle theory file inside the **tts** context from which the command **tts-lemmas** was invoked. There is a further minor difference between the active mode and the other two modes of operation that is elaborated upon within the description of the keyword **in** below.

in *sb*₁ = *tb*₁ **and** ... **and** *sb*_{*n*} = *tb*_{*n*} is used for the specification of the type-based theorems and the output of the command. For each individual entry *i*, such that $1 \leq i \leq n$ with *n* being a positive integer, *tb*_{*i*} is used for the specification of the input of the extended relativization algorithm and *sb*_{*i*} is used for the specification of the name binding for the output of the extended relativization algorithm. The specification of the output is optional: if *sb*_{*i*} is omitted, then a default specification of the output is inferred automatically. *tb*_{*i*} must be a schematic fact available in the context, whereas *sb*_{*i*} can be any fresh name binding. Optionally, it is possible to provide attributes for each individual input and output, e.g., *sb*_{*i*}[*sb-attrb*] = *tb*_{*i*}[*tb-attrb*]. In this case, the list of the attributes *tb-attrb* is applied to *tb*_{*i*} during the first part (initialization of the relativization context) of the ERA. If the command operates in the active mode, then the attributes *sb-attrb* are included in the active area output, but not added to the list of the set-based attributes. For other modes of operation, the attributes *sb-attrb* are added to the list of the set-based attributes and applied during the third part (post-processing) of the ERA.

tts-lemma *a*: φ *tts-addendum*, enters proof mode with the main goal formed by an application of a tactic that depends on the settings specified in *tts-addendum* to φ . Eventually, this results in some fact $\vdash \varphi$ to be put back into the target context. The command should always be invoked from within a **tts** context.

A *tts-long-statement* is similar to the standard *long-statement* in that it allows to build up an initial proof context for the subsequent claim incrementally. Similarly, *tts-short-statement* can be viewed as a natural extension of the standard *short-statement*.

tts-addendum is used for the specification of the pre-processing strategy of the goal φ . φ **is** *thm* applies the extended relativization algorithm to *thm*. If the term that is associated with the resulting set-based theorem is α -equivalent to the term associated with the goal φ , then a specialized tactic solves the main goal, leaving only a trivial goal in its place (the trivial goal can be solved using the terminal proof step **.**). φ **given** *thm* also applies the extended relativization algorithm to *thm*, but the resulting set-based theorem is merely added as a premise to the goal φ .

1.4 ETTS by example

1.4.1 Background

In this section, some of the capabilities of the extension of the framework Types-To-Sets are demonstrated by example. The examples that are presented in this section are expected to be sufficient to begin an independent exploration of the extension, but do not cover the entire spectrum of its functionality.

1.4.2 First steps

Problem statement

Consider the task of the relativization of the type-based theorem *topological-space-class.closed-Un* from the standard library of Isabelle/HOL:

$$[[\text{closed } S; \text{closed } T]] \implies \text{closed } (S \cup T),$$

where $S::'a::\text{topological-space set}$ and $T::'a::\text{topological-space set}$.

Unoverloading

The constant *closed* that occurs in the theorem is an overloaded constant defined as $\text{closed } S = \text{open } (- S)$ for any $S::'a::\text{topological-space set}$. The constant may be unoverloaded with the help of the command **ud** that is provided as part of the framework CTR:

```
ud <topological-space.closed>
ud closed' <closed>
```

This invocation declares the constant *closed.with* that is defined as

$$\text{closed.with} \equiv \lambda \text{open } S. \text{open } (- S)$$

and provides the theorems *closed.with* given by

$$\text{topological-space.closed} \equiv \text{closed.with}$$

and *closed'.with* given by

$$\text{closed} \equiv \text{closed.with } \text{open}$$

These theorems are automatically added to the dynamic fact *ud.with*.

Conditional transfer rules

Before the relativization can be performed, the transfer rules need to be available for each constant that occurs in the type-based theorem immediately after step 4 of the KERA. All binary relations that are used in the transfer rules must be at least right total and bi-unique (assuming the default order on predicates in Isabelle/HOL). For the theorem *topological-space-class.closed-Un*, there are two such constants: *class.topological-space* and *closed.with*. The transfer rules can be obtained with the help of the command **ctr** from the framework CTR. The process may involve the synthesis of further relativized constants, as described in the reference manual for the framework CTR.

```
ctr
relativization
```

```

synthesis ctr-simps
assumes [transfer-domain-rule]: Domainp A = (λx. x ∈ U)
  and [transfer-rule]: right-total A bi-unique A
trp (?'a A)
in topological-space-ow: class.topological-space-def
  and closed-ow: closed.with-def

```

Relativization

As mentioned previously, the relativization of theorems can only be performed from within a suitable `tts` context. In setting up the `tts` context, the users always need to provide the RI specification elements that are compatible with the theorems that are meant to be relativized in the `tts` context. The set of the schematic type variables that occur in the theorem `topological-space-class.closed-Un` is $\{?'a\}$. Thus, there needs to be exactly one RI specification element of the form $(?'a, U::'a \text{ set})$:

```

tts-context
  tts: (?'a to <U::'a set>)
begin

```

The relativization can be performed by invoking the command `tts-lemmas` in the following manner:

```

tts-lemmas? in closed-Un' = topological-space-class.closed-Un

```

In this case, the command was invoked in the active mode, providing an active area that can be used to insert the following theorem directly into the theory file:

```

tts-lemma closed-Un':
  assumes  $U \neq \{\}$ 
    and  $\forall x \in S. x \in U$ 
    and  $\forall x \in T. x \in U$ 
    and topological-space-ow U opena
    and closed-ow U opena S
    and closed-ow U opena T
  shows closed-ow U opena (S ∪ T)
  is topological-space-class.closed-Un<proof>

```

The invocation of the command `tts-lemmas` in the active mode can be removed with no effect on the theorems that were generated using the command.

end

While our goal was achieved, that is, the theorem `closed-Un'` is, indeed, a relativization of the theorem `topological-space-class.closed-Un`, something does not appear right. Is the assumption $U \neq \{\}$ necessary? Is it possible to simplify $\forall x \in S. x \in U$? Is it necessary to use such a contrived name for the denotation of the open set predicate? Of course, all of these issues can be resolved by restating the theorem in the form that we would like to see and using `closed-Un'` in the proof of this theorem, e.g.

```

lemma closed-Un'':
  assumes  $S \subseteq U$ 
    and  $T \subseteq U$ 
    and topological-space-ow U τ
    and closed-ow U τ S
    and closed-ow U τ T
  shows closed-ow U τ (S ∪ T)
  <proof>

```

However, having to restate the theorem presents a grave inconvenience. This can be avoided by using a different format of the `tts-addendum`:

```

tts-context
  tts: (?'a to <U::'a set>)
begin

tts-lemma closed-Un''':
  assumes  $S \subseteq U$ 
    and  $T \subseteq U$ 
    and topological-space-ow  $U \tau$ 
    and closed-ow  $U \tau S$ 
    and closed-ow  $U \tau T$ 
  shows closed-ow  $U \tau (S \cup T)$ 
  given topological-space-class.closed-Un
  <proof>

```

end

Nevertheless, could there still be some space for improvement? It turns out that instead of having to state the theorem in the desired form manually, often enough, it suffices to provide additional parameters for post-processing of the raw set-based theorem, as demonstrated in the code below:

```

tts-context
  tts: (?'a to <U::'a set>)
  rewriting ctr-simps
  eliminating <?U#{}> through (auto simp: topological-space-ow-def)
  applying[of - - -  $\tau$ ]
begin

```

```

tts-lemma closed-Un'''':
  assumes  $S \subseteq U$ 
    and  $T \subseteq U$ 
    and topological-space-ow  $U \tau$ 
    and closed-ow  $U \tau S$ 
    and closed-ow  $U \tau T$ 
  shows closed-ow  $U \tau (S \cup T)$ 
  is topological-space-class.closed-Un(proof)

```

end

Finding the most suitable set of parameters for post-processing of the result of the relativization is an iterative process and requires practice before fluency can be achieved.

ETTS Case Studies: Introduction

2.1 Background

2.1.1 Purpose

The remainder of this document presents several examples of the application of the extension of the framework Types-To-Sets and provides the potential users of the extension with a plethora of design patterns to choose from for their own applied relativization needs.

2.1.2 Related work

Since the publication of the framework Types-To-Sets in [19], there has been a growing interest in its use in applied formalization. Some of the examples of the application of the framework include [8], [23] and [14]. However, this list is not exhaustive. Arguably, the most significant application example was developed in [14], where Fabian Immler and Bohua Zhan performed the relativization of over 200 theorems from the standard mathematics library of Isabelle/HOL. Nonetheless, it is likely that the work presented in this document is the first significant application of the ETTS: unsurprisingly, the content of this document was developed in parallel with the extension of the framework itself. Also, perhaps, it is the largest application of the framework Types-To-Sets at the time of writing: only one of the three libraries (SML Relativization) presented in the context of this work contains the relativization of over 800 theorems from the standard library of Isabelle/HOL.

2.2 Examples: overview

2.2.1 Background

The examples that are presented in this document were developed for the demonstration of the impact of various aspects of the relativization process on the outcome of the relativization. Three libraries of relativized results were developed in the context of this work:

- *SML Relativization*: a relativization of elements of the standard mathematics library of Isabelle/HOL
- *TTS Vector Spaces*: a renovation of the set-based library that was developed in [14] using the ETTS instead of the existing interface for Types-To-Sets
- *TTS Foundations*: a relativization of a miniature type-based library with every constant being parametric under the side conditions compatible with Types-To-Sets

2.2.2 SML Relativization

The standard library that is associated with the object logic Isabelle/HOL and provided as a part of the standard distribution of Isabelle [1] contains a significant number of formalized results from a variety of fields of mathematics. However, the formalization is performed using a type-based approach: for example, the carrier sets associated with the algebraic structures

and the underlying sets of the topological spaces consist of all terms of an arbitrary type. The ETTS was applied to the relativization of a certain number of results from the standard library. The results that are formalized in the library SML Relativization are taken from an array of topics that include order theory, group theory, ring theory and topology. However, only the results whose relativization could be nearly fully automated using the frameworks UD, CTR and ETTS with almost no additional proof effort are included.

2.2.3 TTS Vector Spaces

The TTS Vector Spaces is a remake of the library of relativized results that was developed in [14] using the ETTS. The theorems that are provided in the library TTS Vector Spaces are nearly identical to the results that are provided in [14].

A detailed description of the original library has already been given in [14] and will not be restated. The definitional frameworks that are used in [14] and the TTS Vector Spaces are similar. While the unoverloading of most of the constants could be performed by using the command `ud`, the command `ctr` could not be used to establish that the unoverloaded constants are parametric under a suitable set of side conditions. Therefore, like in [14], the proofs of the transfer rules were performed manually. However, the advantages of using the ETTS become apparent during the relativization of theorems: the complex infrastructure that was needed for compiling out dependencies on overloaded constants, the manual invocation of the attributes related to the individual steps of the relativization algorithm, the repeated explicit references to the theorem as it undergoes the transformations associated with the individual steps of the relativization algorithm, the explicitly stated names of the set-based theorems were no longer needed. Furthermore, the theorems synthesized by the ETTS in TTS Vector Spaces appear in the formal proof documents in a format that is similar to the canonical format of the Isabelle/Isar declarations associated with the standard commands such as `lemma`.

2.2.4 TTS Foundations

The most challenging aspect of the relativization process, perhaps, is related to the availability of the transfer rules for the constants in the type-based theorems. Nonetheless, even if the transfer rules are available, having to use the relativized constants in the set-based theorems that are different from the original constants that are used in the type-based theorems can be seen as unnatural and inconvenient. Unfortunately, the library SML Relativization suffers from both of the aforementioned problems. The library that was developed in [14] (hence, also the library TTS Vector Spaces) suffers, primarily, from the former problem, but, arguably, due to the methodology that was chosen for the relativization, the library has a more restricted scope of applicability.

The library TTS Foundations provides an example of a miniature type-based library such that all constants associated with the operations on mathematical structures (effectively, this excludes the constants associated with the locale predicates) in the library are parametric under the side conditions compatible with Types-To-Sets. The relativization is performed with respect to all possible type variables; in this case, the type classes are not used in the type-based library. Currently, the library includes the results from the areas of order theory and semigroups. However, it is hoped that it can be seen that the library can be extended to include most of the content that is available in the main library of Isabelle/HOL.

The library TTS Foundations demonstrates that the development of a set-based library can be nearly fully automated using the existing infrastructure associated with the UD, CTR and ETTS, and requires almost no explicit proofs on behalf of the users of these frameworks.

SML Relativization

3.1 Extension of the theory *Set*

lemma *set-comp-pair*: $\{f\ t\ r \mid t\ r.\ P\ t\ r\} = \{x.\ \exists\ t\ r.\ P\ t\ r \wedge x = (f\ t\ r)\}$
<proof>

lemma *image-iff'*: $(\forall\ x \in A.\ f\ x \in B) = (f\ ' A \subseteq B)$ *<proof>*

3.2 Extension of the theory *Lifting-Set*

context

includes *lifting-syntax*

begin

lemma *set-pred-eq-transfer*[*transfer-rule*]:

assumes *right-total A*

shows

$((rel\text{-}set\ A\ ==>\ (=))\ ==>\ (rel\text{-}set\ A\ ==>\ (=))\ ==>\ (=))$
 $(\lambda X\ Y.\ \forall s \subseteq Collect\ (Domainp\ A).\ X\ s = Y\ s)$
 $((=)::['b\ set\ \Rightarrow\ bool,\ 'b\ set\ \Rightarrow\ bool]\ \Rightarrow\ bool)$

<proof> **lemma** *vimage-fst-transfer-h*:

$pred\text{-}prod\ (Domainp\ A)\ (Domainp\ B)\ x =$
 $(x \in Collect\ (Domainp\ A) \times Collect\ (Domainp\ B))$

<proof>

lemma *vimage-fst-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A right-total B*

shows

$((rel\text{-}prod\ A\ B\ ==>\ A)\ ==>\ rel\text{-}set\ A\ ==>\ rel\text{-}set\ (rel\text{-}prod\ A\ B))$
 $(\lambda f\ S.\ (f\ -'\ S) \cap ((Collect\ (Domainp\ A)) \times (Collect\ (Domainp\ B))))$
vimage

<proof>

lemma *vimage-snd-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total A bi-unique B right-total B*

shows

$((rel\text{-}prod\ A\ B\ ==>\ B)\ ==>\ rel\text{-}set\ B\ ==>\ rel\text{-}set\ (rel\text{-}prod\ A\ B))$
 $(\lambda f\ S.\ (f\ -'\ S) \cap ((Collect\ (Domainp\ A)) \times (Collect\ (Domainp\ B))))$
vimage

<proof>

lemma *vimage-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique B right-total A*

shows

$((A\ ==>\ B)\ ==>\ (rel\text{-}set\ B)\ ==>\ rel\text{-}set\ A)$
 $(\lambda f\ s.\ (vimage\ f\ s) \cap (Collect\ (Domainp\ A)))\ (-')$

<proof>

lemma *pairwise-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A*

shows $((A\ ==>\ A\ ==>\ (=))\ ==>\ rel\text{-}set\ A\ ==>\ (=))$ *pairwise pairwise*

<proof>

lemma *disjnt-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A*

shows $(rel\text{-}set\ A\ ==>\ rel\text{-}set\ A\ ==>\ (=))$ *disjnt disjnt*

<proof>

lemma *bij-betw-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A bi-unique B*

shows $((A\ ==>\ B)\ ==>\ rel\text{-}set\ A\ ==>\ rel\text{-}set\ B\ ==>\ (=))$ *bij-betw bij-betw*

<proof>

end

3.3 Extension of the theory *Product-Type-Ext*

context

includes *lifting-syntax*

begin

lemma *Sigma-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total A*

shows

$(\text{rel-set } A \implies (A \implies \text{rel-set } B) \implies \text{rel-set } (\text{rel-prod } A \ B))$

Sigma Sigma

<proof>

end

3.4 Extension of the theory *Transfer*

lemma *bi-unique-intersect-r*:

assumes *bi-unique* T
and *rel-set* T a a'
and *rel-set* T b b'
and *rel-set* T $(a \cap b)$ xr
shows $a' \cap b' = xr$

<proof>

lemma *bi-unique-intersect-l*:

assumes *bi-unique* T
and *rel-set* T a a'
and *rel-set* T b b'
and *rel-set* T xl $(a' \cap b')$
shows $a \cap b = xl$

<proof>

lemma *bi-unique-intersect*:

assumes *bi-unique* T **and** *rel-set* T a a' **and** *rel-set* T b b'
shows *rel-set* T $(a \cap b)$ $(a' \cap b')$

<proof>

lemma *bi-unique-union-r*:

assumes *bi-unique* T
and *rel-set* T a a'
and *rel-set* T b b'
and *rel-set* T $(a \cup b)$ xr
shows $a' \cup b' = xr$

<proof>

lemma *bi-unique-union-l*:

assumes *bi-unique* T
and *rel-set* T a a'
and *rel-set* T b b'
and *rel-set* T xl $(a' \cup b')$
shows $a \cup b = xl$

<proof>

lemma *bi-unique-union*:

assumes *bi-unique* T **and** *rel-set* T a a' **and** *rel-set* T b b'
shows *rel-set* T $(a \cup b)$ $(a' \cup b')$

<proof>

lemma *bi-unique-Union-r*:

fixes $T :: ['a, 'b] \Rightarrow \text{bool}$ **and** K
defines K' : $K' \equiv \{(x, y). \text{rel-set } T \ x \ y\}$ “ K
assumes *bi-unique* T
and $\bigcup K \subseteq \text{Collect } (\text{Domainp } T)$
and *rel-set* T $(\bigcup K)$ xr
shows $\bigcup K' = xr$

<proof>

lemma *bi-unique-Union-l*:

fixes $T :: ['a, 'b] \Rightarrow \text{bool}$ **and** K'
defines K : $K \equiv \{(x, y). \text{rel-set } (\lambda y \ x. T \ x \ y) \ x \ y\}$ “ K'
assumes *bi-unique* T

```

and  $\cup K' \subseteq \text{Collect } (\text{Rangep } T)$ 
and  $\text{rel-set } T \text{ xl } (\cup K')$ 
shows  $\cup K = \text{xl}$ 
<proof>

```

context

```

includes lifting-syntax
begin

```

The lemma *Domainp-applyI* was adopted from the lemma with the identical name in the theory *Types-To-Sets/Group-on-With.thy*.

lemma *Domainp-applyI*:

```

includes lifting-syntax
shows  $(A \implies B) \text{ f g } \implies A \text{ x y } \implies \text{Domainp } B \text{ (f x)}$ 
<proof>

```

lemma *Domainp-fun*:

```

assumes left-unique A
shows
   $\text{Domainp } (\text{rel-fun } A \text{ B}) =$ 
   $(\lambda f. f ' (\text{Collect } (\text{Domainp } A)) \subseteq (\text{Collect } (\text{Domainp } B)))$ 
<proof>

```

lemma *Bex-fun-transfer[transfer-rule]*:

```

assumes bi-unique A right-total B
shows
   $((A \implies B) \implies (=)) \implies (=)$ 
   $(\text{Bex } (\text{Collect } (\lambda f. f ' (\text{Collect } (\text{Domainp } A)) \subseteq (\text{Collect } (\text{Domainp } B))))))$ 
  Ex
<proof>

```

end

3.5 Relativization of the results about relations

3.5.1 Definitions and common properties

context

notes $[[\text{inductive-internals}]]$

begin

inductive-set $\text{trancl-on} :: ['a \text{ set}, ('a \times 'a) \text{ set}] \Rightarrow ('a \times 'a) \text{ set}$
 $(\langle \text{on } - / (-^+) \rangle [1000, 1000] 999)$

for $U :: 'a \text{ set}$ **and** $r :: ('a \times 'a) \text{ set}$

where

$r\text{-into-trancl}[\text{intro}, \text{Pure.intro}]$:

$[[a \in U; b \in U; (a, b) \in r]] \Longrightarrow (a, b) \in \text{on } U r^+$

$| \text{trancl-into-trancl}[\text{Pure.intro}]$:

$[[a \in U; b \in U; c \in U; (a, b) \in \text{on } U r^+; (b, c) \in r]] \Longrightarrow$
 $(a, c) \in \text{on } U r^+$

abbreviation $\text{tranclp-on} (\langle \text{on } - / (-^{++}) \rangle [1000, 1000] 1000)$ **where**

$\text{tranclp-on} \equiv \text{trancl-onp}$

declare $\text{trancl-on-def}[\text{nitpick-unfold del}]$

lemmas $\text{tranclp-on-def} = \text{trancl-onp-def}$

end

definition $\text{transp-on} :: ['a \text{ set}, ['a, 'a] \Rightarrow \text{bool}] \Rightarrow \text{bool}$

where $\text{transp-on } U = (\lambda r. (\forall x \in U. \forall y \in U. \forall z \in U. r x y \longrightarrow r y z \longrightarrow r x z))$

definition $\text{acyclic-on} :: ['a \text{ set}, ('a \times 'a) \text{ set}] \Rightarrow \text{bool}$

where $\text{acyclic-on } U = (\lambda r. (\forall x \in U. (x, x) \notin \text{on } U r^+))$

lemma $\text{trancl-on-eq-tranclp-on}$:

$\text{on } P (\lambda x y. (x, y) \in r)^{++} x y = ((x, y) \in \text{on } (\text{Collect } P) r^+)$
 $\langle \text{proof} \rangle$

lemma trancl-on-imp-U : $(x, y) \in \text{on } U r^+ \Longrightarrow (x, y) \in U \times U$

$\langle \text{proof} \rangle$

lemmas $\text{tranclp-on-imp-P} = \text{trancl-on-imp-U}[\text{to-pred}, \text{simplified}]$

lemma $\text{trancl-on-imp-trancl}$: $(x, y) \in \text{on } U r^+ \Longrightarrow (x, y) \in r^+$

$\langle \text{proof} \rangle$

lemmas $\text{tranclp-on-imp-tranclp} = \text{trancl-on-imp-trancl}[\text{to-pred}]$

lemma $\text{tranclp-eq-tranclp-on}$: $r^{++} = \text{on } (\lambda x. \text{True}) r^{++}$

$\langle \text{proof} \rangle$

lemma $\text{trancl-eq-trancl-on}$: $r^+ = \text{on } \text{UNIV } r^+$

$\langle \text{proof} \rangle$

lemma $\text{transp-on-empty}[\text{simp}]$: $\text{transp-on } \{ \} r \langle \text{proof} \rangle$

lemma $\text{transp-eq-transp-on}$: $\text{transp} = \text{transp-on } \text{UNIV}$

<proof>

lemma *acyclic-on-empty*[*simp*]: *acyclic-on* {} *r* *<proof>*

lemma *acyclic-eq-acyclic-on*: *acyclic* = *acyclic-on UNIV*
<proof>

3.5.2 Transfer rules I: *lfp* transfer

The following context contains code from [13].

context

includes *lifting-syntax*

begin

lemma *Inf-transfer*[*transfer-rule*]:

(*rel-set* (*A* \implies (=)) \implies *A* \implies (=)) *Inf Inf*
<proof>

lemma *less-eq-pred-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total A*

shows

((*A* \implies (=)) \implies (*A* \implies (=)) \implies (=))
($\lambda f g. \forall x \in \text{Collect}(\text{Domainp } A). f x \leq g x$) (\leq)
<proof>

lemma *lfp-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

defines *R* \equiv (((*A* \implies (=)) \implies (*A* \implies (=))) \implies (*A* \implies (=)))

shows *R* ($\lambda f. \text{lfp } (\lambda u x. \text{if } \text{Domainp } A \ x \ \text{then } f \ u \ x \ \text{else } \text{bot})$) *lfp*

<proof>

lemma *Inf2-transfer*[*transfer-rule*]:

(*rel-set* (*T* \implies *T* \implies (=)) \implies *T* \implies *T* \implies (=)) *Inf Inf*
<proof>

lemma *less-eq2-pred-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total T*

shows

((*T* \implies *T* \implies (=)) \implies (*T* \implies *T* \implies (=)) \implies (=))
($\lambda f g. \forall x \in \text{Collect}(\text{Domainp } T). \forall y \in \text{Collect}(\text{Domainp } T). f x y \leq g x y$) (\leq)
<proof>

lemma *lfp2-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

defines

R \equiv

(((*A* \implies *A* \implies (=)) \implies (*A* \implies *A* \implies (=))) \implies (*A* \implies *A* \implies (=)))

shows

R

(

$\lambda f. \text{lfp}$

(

$\lambda u x y.$

if *Domainp A x*

then if *Domainp A y* *then* (*f u*) *x y* *else bot*

else bot

)

```

    )
    lfp
  <proof>

```

```
end
```

3.5.3 Transfer rules II: application-specific rules

```
context
```

```
  includes lifting-syntax
```

```
begin
```

```
lemma transp-rt-transfer[transfer-rule]:
```

```
  assumes[transfer-rule]: right-total A
```

```
  shows
```

```
    ((A ==> A ==> (=)) ==> (=)) (transp-on (Collect (Domainp A))) transp
```

```
  <proof>
```

```
lemma tranclp-rt-bu-transfer[transfer-rule]:
```

```
  assumes[transfer-rule]: bi-unique A right-total A
```

```
  shows
```

```
    ((A ==> A ==> (=)) ==> (A ==> A ==> (=)))
```

```
    (tranclp-on (Domainp A)) tranclp
```

```
  <proof>
```

```
lemma trancl-rt-bu-transfer[transfer-rule]:
```

```
  assumes[transfer-rule]: bi-unique A right-total A
```

```
  shows
```

```
    (rel-set (rel-prod A A) ==> rel-set (rel-prod A A))
```

```
    (trancl-on (Collect (Domainp A))) trancl
```

```
  <proof>
```

```
lemma acyclic-rt-bu-transfer[transfer-rule]:
```

```
  assumes[transfer-rule]: bi-unique A right-total A
```

```
  shows
```

```
    ((rel-set (rel-prod A A)) ==> (=))
```

```
    (acyclic-on (Collect (Domainp A))) acyclic
```

```
  <proof>
```

```
end
```

3.6 Relativization of the results about orders

3.6.1 Class *ord*

Definitions and common properties

locale *ord-ow* =

fixes $U :: 'ao\ set$

and $le :: ['ao, 'ao] \Rightarrow bool$ (infix $\langle \leq_{ow} \rangle$ 50)

and $ls :: ['ao, 'ao] \Rightarrow bool$ (infix $\langle <_{ow} \rangle$ 50)

begin

notation le ($\langle'(\leq_{ow}')\rangle$)

and le (infix $\langle \leq_{ow} \rangle$ 50)

and ls ($\langle'(<_{ow}')\rangle$)

and ls (infix $\langle <_{ow} \rangle$ 50)

abbreviation (input) ge (infix $\langle \geq_{ow} \rangle$ 50) where $x \geq_{ow} y \equiv y \leq_{ow} x$

abbreviation (input) gt (infix $\langle >_{ow} \rangle$ 50) where $x >_{ow} y \equiv y <_{ow} x$

notation ge ($\langle'(\geq_{ow}')\rangle$)

and ge (infix $\langle \geq_{ow} \rangle$ 50)

and gt ($\langle'(>_{ow}')\rangle$)

and gt (infix $\langle >_{ow} \rangle$ 50)

tts-register-sbts $\langle (\leq_{ow}) \rangle \mid U$

$\langle proof \rangle$

tts-register-sbts $\langle (<_{ow}) \rangle \mid U$

$\langle proof \rangle$

end

locale *ord-pair-ow* = *ord*₁: *ord-ow* U_1 le_1 ls_1 + *ord*₂: *ord-ow* U_2 le_2 ls_2

for $U_1 :: 'ao\ set$ and le_1 ls_1 and $U_2 :: 'bo\ set$ and le_2 ls_2

begin

notation le_1 ($\langle'(\leq_{ow.1}')\rangle$)

and le_1 (infix $\langle \leq_{ow.1} \rangle$ 50)

and ls_1 ($\langle'(<_{ow.1}')\rangle$)

and ls_1 (infix $\langle <_{ow.1} \rangle$ 50)

and le_2 ($\langle'(\leq_{ow.2}')\rangle$)

and le_2 (infix $\langle \leq_{ow.2} \rangle$ 50)

and ls_2 ($\langle'(<_{ow.2}')\rangle$)

and ls_2 (infix $\langle <_{ow.2} \rangle$ 50)

notation $ord_1.ge$ ($\langle'(\geq_{ow.1}')\rangle$)

and $ord_1.ge$ (infix $\langle \geq_{ow.1} \rangle$ 50)

and $ord_1.gt$ ($\langle'(>_{ow.1}')\rangle$)

and $ord_1.gt$ (infix $\langle >_{ow.1} \rangle$ 50)

and $ord_2.ge$ ($\langle'(\geq_{ow.2}')\rangle$)

and $ord_2.ge$ (infix $\langle \geq_{ow.2} \rangle$ 50)

and $ord_2.gt$ ($\langle'(>_{ow.2}')\rangle$)

and $ord_2.gt$ (infix $\langle >_{ow.2} \rangle$ 50)

end

ud $\langle ord.lessThan \rangle$ ($\langle (with - : \{..<- \}) \rangle$ [1000] 10)

ud $lessThan'$ $\langle lessThan \rangle$

```

ud <ord.atMost> (<(with - : ({..-}))> [1000] 10)
ud atMost' <atMost>
ud <ord.greaterThan> (<(with - : ({-<..})}> [1000] 10)
ud greaterThan' <greaterThan>
ud <ord.atLeast> (<(with - : ({..-}))> [1000] 10)
ud atLeast' <atLeast>
ud <ord.greaterThanLessThan> (<(with - : ({-<..<-})}> [1000, 999, 1000] 10)
ud greaterThanLessThan' <greaterThanLessThan>
ud <ord.atLeastLessThan> (<(with - - : ({-..<-})}> [1000, 999, 1000, 1000] 10)
ud atLeastLessThan' <atLeastLessThan>
ud <ord.greaterThanAtMost> (<(with - - : ({-<..-})}> [1000, 999, 1000, 999] 10)
ud greaterThanAtMost' <greaterThanAtMost>
ud <ord.atLeastAtMost> (<(with - - : ({-..-})}> [1000, 1000, 1000] 10)
ud atLeastAtMost' <atLeastAtMost>
ud <ord.min> (<(with - : «min» - -)> [1000, 1000, 999] 10)
ud min' <min>
ud <ord.max> (<(with - : «max» - -)> [1000, 1000, 999] 10)
ud max' <max>

```

ctr relativization**synthesis** *ctr-simps***assumes** [*transfer-domain-rule*, *transfer-rule*]: $\text{Domain}p\ A = (\lambda x. x \in U)$ **and** [*transfer-rule*]: *right-total* A **trp** ($?a\ A$)**in** *lessThan-ow*: *lessThan.with-def*

(<(on - with - : ({..<-})}> [1000, 1000, 1000] 10)

and *atMost-ow*: *atMost.with-def*

(<(on - with - : ({..-})}> [1000, 1000, 1000] 10)

and *greaterThan-ow*: *greaterThan.with-def*

(<(on - with - : ({-<..})}> [1000, 1000, 1000] 10)

and *atLeast-ow*: *atLeast.with-def*

(<(on - with - : ({-..})}> [1000, 1000, 1000] 10)

ctr relativization**synthesis** *ctr-simps***assumes** [*transfer-domain-rule*, *transfer-rule*]: $\text{Domain}p\ A = (\lambda x. x \in U)$ **and** [*transfer-rule*]: *bi-unique* A *right-total* A **trp** ($?a\ A$)**in** *greaterThanLessThan-ow*: *greaterThanLessThan.with-def*

(<(on - with - : ({-<..<-})}> [1000, 1000, 1000, 1000] 10)

and *atLeastLessThan-ow*: *atLeastLessThan.with-def*

(<(on - with - - : ({-..<-})}> [1000, 1000, 999, 1000, 1000] 10)

and *greaterThanAtMost-ow*: *greaterThanAtMost.with-def*

(<(on - with - - : ({-<..-})}> [1000, 1000, 999, 1000, 1000] 10)

and *atLeastAtMost-ow*: *atLeastAtMost.with-def*

(<(on - with - : ({-..-})}> [1000, 1000, 1000, 1000] 10)

ctr parametricity**in** *min-ow*: *min.with-def***and** *max-ow*: *max.with-def***context** *ord-ow***begin****abbreviation** *lessThan* :: 'ao \Rightarrow 'ao set (<(1{..<_{ow}-})>)**where** $\{..<_{ow}\ u\} \equiv \text{on } U \text{ with } (<_{ow}) : \{..<u\}$ **abbreviation** *atMost* :: 'ao \Rightarrow 'ao set (<(1{..<_{ow}-})>)**where** $\{..<_{ow}\ u\} \equiv \text{on } U \text{ with } (\leq_{ow}) : \{..u\}$

abbreviation *greaterThan* :: 'ao ⇒ 'ao set (⟨(1{<_{ow}..})⟩)

where {<_{ow}..} ≡ on U with (<_{ow}) : {l<..}

abbreviation *atLeast* :: 'ao ⇒ 'ao set (⟨(1{<.._{ow}})⟩)

where *atLeast* l ≡ on U with (≤_{ow}) : {l..}

abbreviation *greaterThanLessThan* :: 'ao ⇒ 'ao ⇒ 'ao set (⟨(1{<_{ow}..<_{ow}-})⟩)

where {<_{ow}..<_{ow}u} ≡ on U with (<_{ow}) : {l<..<u}

abbreviation *atLeastLessThan* :: 'ao ⇒ 'ao ⇒ 'ao set (⟨(1{<..<_{ow}-})⟩)

where {l.<_{ow} u} ≡ on U with (≤_{ow}) (<_{ow}) : {l<..u}

abbreviation *greaterThanAtMost* :: 'ao ⇒ 'ao ⇒ 'ao set (⟨(1{<_{ow}..-})⟩)

where {<_{ow}..u} ≡ on U with (≤_{ow}) (<_{ow}) : {l<..u}

abbreviation *atLeastAtMost* :: 'ao ⇒ 'ao ⇒ 'ao set (⟨(1{<.._{ow}-})⟩)

where {l.<_{ow}u} ≡ on U with (≤_{ow}) : {l..u}

abbreviation *min* :: 'ao ⇒ 'ao ⇒ 'ao **where** *min* ≡ *min.with* (≤_{ow})

abbreviation *max* :: 'ao ⇒ 'ao ⇒ 'ao **where** *max* ≡ *max.with* (≤_{ow})

end

context *ord-pair-ow*

begin

notation *ord₁.lessThan* (⟨(1{<..<_{ow}.1-})⟩)

notation *ord₁.atMost* (⟨(1{<.._{ow}.1-})⟩)

notation *ord₁.greaterThan* (⟨(1{<-<_{ow}.1..})⟩)

notation *ord₁.atLeast* (⟨(1{<-.._{ow}.1})⟩)

notation *ord₁.greaterThanLessThan* (⟨(1{<-<_{ow}.1..<_{ow}.1-})⟩)

notation *ord₁.atLeastLessThan* (⟨(1{<-..<_{ow}.1-})⟩)

notation *ord₁.greaterThanAtMost* (⟨(1{<-<_{ow}.1..-})⟩)

notation *ord₁.atLeastAtMost* (⟨(1{<-.._{ow}.1-})⟩)

notation *ord₂.lessThan* (⟨(1{<..<_{ow}.2-})⟩)

notation *ord₂.atMost* (⟨(1{<.._{ow}.2-})⟩)

notation *ord₂.greaterThan* (⟨(1{<-<_{ow}.2..})⟩)

notation *ord₂.atLeast* (⟨(1{<-.._{ow}.2})⟩)

notation *ord₂.greaterThanLessThan* (⟨(1{<-<_{ow}.2..<_{ow}.2-})⟩)

notation *ord₂.atLeastLessThan* (⟨(1{<-..<_{ow}.2-})⟩)

notation *ord₂.greaterThanAtMost* (⟨(1{<-<_{ow}.2..-})⟩)

notation *ord₂.atLeastAtMost* (⟨(1{<-.._{ow}.2-})⟩)

end

3.6.2 Preorders

Definitions and common properties

locale *partial-preordering-ow* =

fixes U :: 'ao set

and *le* :: 'ao ⇒ 'ao ⇒ bool (**infix** <≤_{ow}> 50)

assumes *refl*: a ∈ U ⇒ a ≤_{ow} a

and *trans*: [[a ∈ U; b ∈ U; c ∈ U; a ≤_{ow} b; b ≤_{ow} c]] ⇒ a ≤_{ow} c

begin

notation *le* (**infix** <≤_{ow}> 50)

end

locale *preordering-ow* = *partial-preordering-ow* U *le*

for U :: 'ao set

and *le* :: 'ao ⇒ 'ao ⇒ bool (**infix** <≤_{ow}> 50) +

```

fixes ls :: ⟨'ao ⇒ 'ao ⇒ bool⟩ (infix <<ow> 50)
assumes strict-iff-not:
  [[ a ∈ U; b ∈ U ]] ⇒ a <<ow b ↔ a ≤ow b ∧ ¬ b ≤ow a

locale preorder-ow = ord-ow U le ls
for U :: 'ao set and le ls +
assumes less-le-not-le:
  [[ x ∈ U; y ∈ U ]] ⇒ x <<ow y ↔ x ≤ow y ∧ ¬ (y ≤ow x)
and order-refl[iff]: x ∈ U ⇒ x ≤ow x
and order-trans: [[ x ∈ U; y ∈ U; z ∈ U; x ≤ow y; y ≤ow z ]] ⇒ x ≤ow z
begin

sublocale
  order: preordering-ow U ⟨(≤ow)⟩ ⟨(<sub>ow)>⟩ +
  dual-order: preordering-ow U ⟨(≥ow)⟩ ⟨(>sub>ow)>⟩
  ⟨proof⟩

end

locale ord-preorder-ow =
  ord1: ord-ow U1 le1 ls1 + ord2: preorder-ow U2 le2 ls2
for U1 :: 'ao set and le1 ls1 and U2 :: 'bo set and le2 ls2
begin

sublocale ord-pair-ow ⟨proof⟩

end

locale preorder-pair-ow =
  ord1: preorder-ow U1 le1 ls1 + ord2: preorder-ow U2 le2 ls2
for U1 :: 'ao set and le1 and ls1 and U2 :: 'bo set and le2 and ls2
begin

sublocale ord-preorder-ow ⟨proof⟩

end

ud ⟨preordering-bdd.bdd⟩
ud ⟨preorder.bdd-above⟩
ud bdd-above' ⟨bdd-above⟩
ud ⟨preorder.bdd-below⟩
ud bdd-below' ⟨bdd-below⟩

ctr relativization
synthesis ctr-simps
assumes [transfer-domain-rule, transfer-rule]: Domainp A = (λx. x ∈ U)
and [transfer-rule]: right-total A
trp (?a A)
in bdd-ow: bdd.with-def
  (⟨(on - with - : «bdd» -)⟩ [1000, 1000, 1000] 10)
and bdd-above-ow: bdd-above.with-def
  (⟨(on - with - : «bdd'-above» -)⟩ [1000, 1000, 1000] 10)
and bdd-below-ow: bdd-below.with-def
  (⟨(on - with - : «bdd'-below» -)⟩ [1000, 1000, 1000] 10)

declare bdd.with[ud-with del]

context preorder-ow

```

begin

abbreviation *bdd-above* :: 'a o set ⇒ bool
where *bdd-above* ≡ *bdd-above-ow* U (\leq_{ow})
abbreviation *bdd-below* :: 'a o set ⇒ bool
where *bdd-below* ≡ *bdd-below-ow* U (\leq_{ow})

end

Transfer rules

context

includes *lifting-syntax*

begin

lemma *partial-preordering-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total* A

shows

((A ===> A ===> (=)) ===> (=))
(*partial-preordering-ow* (Collect (Domainp A))) *partial-preordering*
⟨*proof*⟩

lemma *preordering-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total* A

shows

((A ===> A ===> (=)) ===> (A ===> A ===> (=)) ===> (=))
(*preordering-ow* (Collect (Domainp A))) *preordering*
⟨*proof*⟩

lemma *preorder-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total* A

shows

((A ===> A ===> (=)) ===> (A ===> A ===> (=)) ===> (=))
(*preorder-ow* (Collect (Domainp A))) *class.preorder*
⟨*proof*⟩

end

Relativization

context *preordering-ow*

begin

tts-context

tts: (?'a to U)

rewriting *ctr-simps*

substituting *preordering-ow-axioms*

eliminating through *auto*

begin

tts-lemma *strict-implies-order*:

assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$

shows $a \leq_{ow} b$

is *preordering.strict-implies-order*⟨*proof*⟩

tts-lemma *irrefl*:

assumes $a \in U$

shows $\neg a <_{ow} a$

```

    is preordering.irrefl{proof}

tts-lemma asym:
  assumes  $a \in U$  and  $b \in U$  and  $a <_{ow} b$  and  $b <_{ow} a$ 
  shows False
  is preordering.asym{proof}

tts-lemma strict-trans1:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$  and  $a \leq_{ow} b$  and  $b <_{ow} c$ 
  shows  $a <_{ow} c$ 
  is preordering.strict-trans1{proof}

tts-lemma strict-trans2:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$  and  $a <_{ow} b$  and  $b \leq_{ow} c$ 
  shows  $a <_{ow} c$ 
  is preordering.strict-trans2{proof}

tts-lemma strict-trans:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$  and  $a <_{ow} b$  and  $b <_{ow} c$ 
  shows  $a <_{ow} c$ 
  is preordering.strict-trans{proof}

end

end

context preorder-ow
begin

tts-context
  tts: (?a to U)
  rewriting ctr-simps
  substituting preorder-ow-axioms
  eliminating through auto
begin

tts-lemma less-irrefl:
  assumes  $x \in U$ 
  shows  $\neg x <_{ow} x$ 
  is preorder-class.less-irrefl{proof}

tts-lemma bdd-below-Ioc:
  assumes  $a \in U$  and  $b \in U$ 
  shows bdd-below  $\{a <_{ow} ..b\}$ 
  is preorder-class.bdd-below-Ioc{proof}

tts-lemma bdd-above-Ioc:
  assumes  $a \in U$  and  $b \in U$ 
  shows bdd-above  $\{a <_{ow} ..b\}$ 
  is preorder-class.bdd-above-Ioc{proof}

tts-lemma bdd-above-Iic:
  assumes  $b \in U$ 
  shows bdd-above  $\{..owb\}$ 
  is preorder-class.bdd-above-Iic{proof}

tts-lemma bdd-above-Iio:
  assumes  $b \in U$ 

```

shows *bdd-above* $\{..<_{ow}b\}$
is *preorder-class.bdd-above-Ioi*(*proof*)

tts-lemma *bdd-below-Ici*:
assumes $a \in U$
shows *bdd-below* $\{a.._{ow}\}$
is *preorder-class.bdd-below-Ici*(*proof*)

tts-lemma *bdd-below-Ioi*:
assumes $a \in U$
shows *bdd-below* $\{a<_{ow}..\}$
is *preorder-class.bdd-below-Ioi*(*proof*)

tts-lemma *bdd-above-Icc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a.._{ow}b\}$
is *preorder-class.bdd-above-Icc*(*proof*)

tts-lemma *bdd-above-Ioo*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a<_{ow}..<_{ow}b\}$
is *preorder-class.bdd-above-Ioo*(*proof*)

tts-lemma *bdd-below-Icc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a.._{ow}b\}$
is *preorder-class.bdd-below-Icc*(*proof*)

tts-lemma *bdd-below-Ioo*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a<_{ow}..<_{ow}b\}$
is *preorder-class.bdd-below-Ioo*(*proof*)

tts-lemma *bdd-above-Ico*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* (on U with (\leq_{ow}) $(<_{ow})$) : $\{a..<b\}$
is *preorder-class.bdd-above-Ico*(*proof*)

tts-lemma *bdd-below-Ico*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* (on U with (\leq_{ow}) $(<_{ow})$) : $\{a..<b\}$
is *preorder-class.bdd-below-Ico*(*proof*)

tts-lemma *Ioi-le-Ico*:
assumes $a \in U$
shows $\{a<_{ow}..\} \subseteq \{a.._{ow}\}$
is *preorder-class.Ioi-le-Ico*(*proof*)

tts-lemma *eq-refl*:
assumes $y \in U$ **and** $x = y$
shows $x \leq_{ow} y$
is *preorder-class.eq-refl*(*proof*)

tts-lemma *less-imp-le*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $x \leq_{ow} y$
is *preorder-class.less-imp-le*(*proof*)

tts-lemma *less-not-sym*:

assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $\neg y <_{ow} x$
is *preorder-class.less-not-sym*{proof}

tts-lemma *less-imp-not-less*:

assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $(\neg y <_{ow} x) = True$
is *preorder-class.less-imp-not-less*{proof}

tts-lemma *less-asym'*:

assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$ **and** $b <_{ow} a$
shows P
is *preorder-class.less-asym'*{proof}

tts-lemma *less-imp-triv*:

assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $(y <_{ow} x \longrightarrow P) = True$
is *preorder-class.less-imp-triv*{proof}

tts-lemma *less-trans*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x <_{ow} y$ **and** $y <_{ow} z$
shows $x <_{ow} z$
is *preorder-class.less-trans*{proof}

tts-lemma *less-le-trans*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x <_{ow} y$ **and** $y \leq_{ow} z$
shows $x <_{ow} z$
is *preorder-class.less-le-trans*{proof}

tts-lemma *le-less-trans*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x \leq_{ow} y$ **and** $y <_{ow} z$
shows $x <_{ow} z$
is *preorder-class.le-less-trans*{proof}

tts-lemma *bdd-aboveI*:

assumes $A \subseteq U$ **and** $M \in U$ **and** $\bigwedge x. [[x \in U; x \in A]] \Longrightarrow x \leq_{ow} M$
shows *bdd-above* A
is *preorder-class.bdd-aboveI*{proof}

tts-lemma *bdd-belowI*:

assumes $A \subseteq U$ **and** $m \in U$ **and** $\bigwedge x. [[x \in U; x \in A]] \Longrightarrow m \leq_{ow} x$
shows *bdd-below* A
is *preorder-class.bdd-belowI*{proof}

tts-lemma *less-asym*:

assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$ **and** $\neg P \Longrightarrow y <_{ow} x$
shows P
is *preorder-class.less-asym*{proof}

tts-lemma *bdd-above-Int1*:

assumes $A \subseteq U$ **and** $B \subseteq U$ **and** *bdd-above* A
shows *bdd-above* $(A \cap B)$
is *preorder-class.bdd-above-Int1*{proof}

tts-lemma *bdd-above-Int2*:

assumes $B \subseteq U$ **and** $A \subseteq U$ **and** *bdd-above* B
shows *bdd-above* $(A \cap B)$

is *preorder-class.bdd-above-Int2*(proof)

tts-lemma *bdd-below-Int1*:

assumes $A \subseteq U$ and $B \subseteq U$ and *bdd-below* A

shows *bdd-below* $(A \cap B)$

is *preorder-class.bdd-below-Int1*(proof)

tts-lemma *bdd-below-Int2*:

assumes $B \subseteq U$ and $A \subseteq U$ and *bdd-below* B

shows *bdd-below* $(A \cap B)$

is *preorder-class.bdd-below-Int2*(proof)

tts-lemma *bdd-above-mono*:

assumes $B \subseteq U$ and *bdd-above* B and $A \subseteq B$

shows *bdd-above* A

is *preorder-class.bdd-above-mono*(proof)

tts-lemma *bdd-below-mono*:

assumes $B \subseteq U$ and *bdd-below* B and $A \subseteq B$

shows *bdd-below* A

is *preorder-class.bdd-below-mono*(proof)

tts-lemma *atLeastAtMost-subseteq-atLeastLessThan-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$

shows $(\{a.._{ow}b\} \subseteq (\text{on } U \text{ with } (\leq_{ow}) (\lt_{ow}) : \{c..<d\})) =$

$(a \leq_{ow} b \longrightarrow b \lt_{ow} d \wedge c \leq_{ow} a)$

is *preorder-class.atLeastAtMost-subseteq-atLeastLessThan-iff*(proof)

tts-lemma *atMost-subset-iff*:

assumes $x \in U$ and $y \in U$

shows $(\{.._{ow}x\} \subseteq \{.._{ow}y\}) = (x \leq_{ow} y)$

is *Set-Interval.atMost-subset-iff*(proof)

tts-lemma *single-Diff-lessThan*:

assumes $k \in U$

shows $\{k\} - \{..<_{ow}k\} = \{k\}$

is *Set-Interval.single-Diff-lessThan*(proof)

tts-lemma *atLeast-subset-iff*:

assumes $x \in U$ and $y \in U$

shows $(\{x.._{ow}\} \subseteq \{y.._{ow}\}) = (y \leq_{ow} x)$

is *Set-Interval.atLeast-subset-iff*(proof)

tts-lemma *atLeastatMost-psubset-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$

shows

$(\{a.._{ow}b\} \subseteq \{c.._{ow}d\}) =$

$(c \leq_{ow} d \wedge (\neg a \leq_{ow} b \vee c \leq_{ow} a \wedge b \leq_{ow} d \wedge (c \lt_{ow} a \vee b \lt_{ow} d)))$

is *preorder-class.atLeastatMost-psubset-iff*(proof)

tts-lemma *not-empty-eq-Iic-eq-empty*:

assumes $h \in U$

shows $\{\} \neq \{.._{ow}h\}$

is *preorder-class.not-empty-eq-Iic-eq-empty*(proof)

tts-lemma *atLeastatMost-subset-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$

shows $(\{a.._{ow}b\} \subseteq \{c.._{ow}d\}) = (\neg a \leq_{ow} b \vee b \leq_{ow} d \wedge c \leq_{ow} a)$

is *preorder-class.atLeastatMost-subset-iff* \langle proof \rangle

tts-lemma *Icc-subset-Ici-iff*:

assumes $l \in U$ and $h \in U$ and $l' \in U$
 shows $(\{l.._{ow}h\} \subseteq \{l'.._{ow}\}) = (\neg l \leq_{ow} h \vee l' \leq_{ow} l)$
 is *preorder-class.Icc-subset-Ici-iff* \langle proof \rangle

tts-lemma *Icc-subset-Iic-iff*:

assumes $l \in U$ and $h \in U$ and $h' \in U$
 shows $(\{l.._{ow}h\} \subseteq \{.._{ow}h'\}) = (\neg l \leq_{ow} h \vee h \leq_{ow} h')$
 is *preorder-class.Icc-subset-Iic-iff* \langle proof \rangle

tts-lemma *not-empty-eq-Ici-eq-empty*:

assumes $l \in U$
 shows $\{\} \neq \{l.._{ow}\}$
 is *preorder-class.not-empty-eq-Ici-eq-empty* \langle proof \rangle

tts-lemma *not-Ici-eq-empty*:

assumes $l \in U$
 shows $\{l.._{ow}\} \neq \{\}$
 is *preorder-class.not-Ici-eq-empty* \langle proof \rangle

tts-lemma *not-Iic-eq-empty*:

assumes $h \in U$
 shows $\{.._{ow}h\} \neq \{\}$
 is *preorder-class.not-Iic-eq-empty* \langle proof \rangle

tts-lemma *atLeastatMost-empty-iff2*:

assumes $a \in U$ and $b \in U$
 shows $(\{\} = \{a.._{ow}b\}) = (\neg a \leq_{ow} b)$
 is *preorder-class.atLeastatMost-empty-iff2* \langle proof \rangle

tts-lemma *atLeastLessThan-empty-iff2*:

assumes $a \in U$ and $b \in U$
 shows $(\{\} = (\text{on } U \text{ with } (\leq_{ow}) (\lt_{ow}) : \{a..<b\})) = (\neg a <_{ow} b)$
 is *preorder-class.atLeastLessThan-empty-iff2* \langle proof \rangle

tts-lemma *greaterThanAtMost-empty-iff2*:

assumes $k \in U$ and $l \in U$
 shows $(\{\} = \{k<_{ow}..l\}) = (\neg k <_{ow} l)$
 is *preorder-class.greaterThanAtMost-empty-iff2* \langle proof \rangle

tts-lemma *atLeastatMost-empty-iff*:

assumes $a \in U$ and $b \in U$
 shows $(\{a.._{ow}b\} = \{\}) = (\neg a \leq_{ow} b)$
 is *preorder-class.atLeastatMost-empty-iff* \langle proof \rangle

tts-lemma *atLeastLessThan-empty-iff*:

assumes $a \in U$ and $b \in U$
 shows $((\text{on } U \text{ with } (\leq_{ow}) (\lt_{ow}) : \{a..<b\}) = \{\}) = (\neg a <_{ow} b)$
 is *preorder-class.atLeastLessThan-empty-iff* \langle proof \rangle

tts-lemma *greaterThanAtMost-empty-iff*:

assumes $k \in U$ and $l \in U$
 shows $(\{k<_{ow}..l\} = \{\}) = (\neg k <_{ow} l)$
 is *preorder-class.greaterThanAtMost-empty-iff* \langle proof \rangle

tts-lemma *lift-Suc-mono-less-iff*:

assumes $\text{range } f \subseteq U$ **and** $\bigwedge n. f\ n <_{ow} f\ (Suc\ n)$
shows $(f\ n <_{ow} f\ m) = (n < m)$
is *preorder.lift-Suc-mono-less-iff*{proof}

tts-lemma *lift-Suc-mono-less*:

assumes $\text{range } f \subseteq U$ **and** $\bigwedge n. f\ n <_{ow} f\ (Suc\ n)$ **and** $n < n'$
shows $f\ n <_{ow} f\ n'$
is *preorder-class.lift-Suc-mono-less*{proof}

tts-lemma *lift-Suc-mono-le*:

assumes $\text{range } f \subseteq U$ **and** $\bigwedge n. f\ n \leq_{ow} f\ (Suc\ n)$ **and** $n \leq n'$
shows $f\ n \leq_{ow} f\ n'$
is *preorder.lift-Suc-mono-le*{proof}

tts-lemma *lift-Suc-antimono-le*:

assumes $\text{range } f \subseteq U$ **and** $\bigwedge n. f\ (Suc\ n) \leq_{ow} f\ n$ **and** $n \leq n'$
shows $f\ n' \leq_{ow} f\ n$
is *preorder-class.lift-Suc-antimono-le*{proof}

end

tts-context

tts: (*?a* **to** U)
substituting *preorder-ow-axioms*
begin

tts-lemma *bdd-above-empty*:

assumes $U \neq \{\}$
shows *bdd-above* $\{\}$
is *preorder-class.bdd-above-empty*{proof}

tts-lemma *bdd-below-empty*:

assumes $U \neq \{\}$
shows *bdd-below* $\{\}$
is *preorder-class.bdd-below-empty*{proof}

end

tts-context

tts: (*?a* **to** U) **and** (*?b* **to** $\langle U_2::'a\ \text{set} \rangle$)
rewriting *ctr-simps*
substituting *preorder-ow-axioms*
eliminating through (*auto intro: bdd-above-empty bdd-below-empty*)
begin

tts-lemma *bdd-belowI2*:

assumes $A \subseteq U_2$
and $m \in U$
and $\forall x \in U_2. f\ x \in U$
and $\bigwedge x. x \in A \implies m \leq_{ow} f\ x$
shows *bdd-below* $(f\ ` A)$
given *preorder-class.bdd-belowI2*
{proof}

tts-lemma *bdd-aboveI2*:

assumes $A \subseteq U_2$
and $\forall x \in U_2. f\ x \in U$
and $M \in U$

```

and  $\wedge x. x \in A \implies f x \leq_{ow} M$ 
shows bdd-above ( $f \text{ ' } A$ )
given preorder-class.bdd-aboveI2
<proof>

```

end

end

3.6.3 Partial orders

Definitions and common properties

```

locale ordering-ow = partial-preordering-ow  $U$   $le$ 
for  $U :: 'ao$  set and  $le :: 'ao \Rightarrow 'ao \Rightarrow bool$  (infix  $\langle \leq_{ow} \rangle$  50) +
fixes  $ls :: 'ao \Rightarrow 'ao \Rightarrow bool$  (infix  $\langle <_{ow} \rangle$  50)
assumes strict-iff-order:  $\llbracket a \in U; b \in U \rrbracket \implies a \langle <_{ow} \rangle b \longleftrightarrow a \leq_{ow} b \wedge a \neq b$ 
and antisym:  $\llbracket a \in U; b \in U; a \leq_{ow} b; b \leq_{ow} a \rrbracket \implies a = b$ 
begin

```

```

notation  $le$  (infix  $\langle \leq_{ow} \rangle$  50)
and  $ls$  (infix  $\langle <_{ow} \rangle$  50)

```

```

sublocale preordering-ow  $U$   $\langle (\leq_{ow}) \rangle$   $\langle (\langle_{ow} \rangle)$ 
<proof>

```

end

```

locale order-ow = preorder-ow  $U$   $le$   $ls$  for  $U :: 'ao$  set and  $le$   $ls$  +
assumes antisym:  $\llbracket x \in U; y \in U; x \leq_{ow} y; y \leq_{ow} x \rrbracket \implies x = y$ 
begin

```

```

sublocale
order: ordering-ow  $U$   $\langle (\leq_{ow}) \rangle$   $\langle (\langle_{ow} \rangle)$  +
dual-order: ordering-ow  $U$   $\langle (\geq_{ow}) \rangle$   $\langle (\rangle_{ow}) \rangle$ 
<proof>

```

```

no-notation  $le$  (infix  $\langle \leq_{ow} \rangle$  50)
and  $ls$  (infix  $\langle <_{ow} \rangle$  50)

```

end

```

locale ord-order-ow = ord1: ord-ow  $U_1$   $le_1$   $ls_1$  + ord2: order-ow  $U_2$   $le_2$   $ls_2$ 
for  $U_1 :: 'ao$  set and  $le_1$   $ls_1$  and  $U_2 :: 'bo$  set and  $le_2$   $ls_2$ 

```

```

sublocale ord-order-ow  $\subseteq$  ord-preorder-ow <proof>

```

```

locale preorder-order-ow =
ord1: preorder-ow  $U_1$   $le_1$   $ls_1$  + ord2: order-ow  $U_2$   $le_2$   $ls_2$ 
for  $U_1 :: 'ao$  set and  $le_1$   $ls_1$  and  $U_2 :: 'bo$  set and  $le_2$   $ls_2$ 

```

```

sublocale preorder-order-ow  $\subseteq$  preorder-pair-ow <proof>

```

```

locale order-pair-ow = ord1: order-ow  $U_1$   $le_1$   $ls_1$  + ord2: order-ow  $U_2$   $le_2$   $ls_2$ 
for  $U_1 :: 'ao$  set and  $le_1$   $ls_1$  and  $U_2 :: 'bo$  set and  $le_2$   $ls_2$ 

```

```

sublocale order-pair-ow  $\subseteq$  preorder-order-ow <proof>

```

ud $\langle monoseq \rangle (\langle with - : \langle monoseq \rangle - \rangle [1000, 1000] 10)$

ctr relativization

synthesis *ctr-simps*
assumes [*transfer-domain-rule, transfer-rule*]:
 $Domainp (B::'c \Rightarrow 'd \Rightarrow bool) = (\lambda x. x \in U_2)$
and [*transfer-rule*]: *right-total B*
trp ($?'b \langle A::'a \Rightarrow 'b \Rightarrow bool \rangle$) **and** ($?'a B$)
in *monoseq-ow: monoseq.with-def*

Transfer rules

context

includes *lifting-syntax*

begin

lemma *ordering-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 $((A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } (A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } (=)$
 $(\text{ordering-ow } (Collect (Domainp A))) \text{ ordering}$
 $\langle proof \rangle$

lemma *order-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 $((A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } (A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } (=)$
 $(\text{order-ow } (Collect (Domainp A))) \text{ class.order}$
 $\langle proof \rangle$

end

Relativization

context *ordering-ow*

begin

tts-context

tts: ($?'a \text{ to } U$)
rewriting *ctr-simps*
substituting *ordering-ow-axioms*
eliminating through *simp*

begin

tts-lemma *strict-implies-not-eq*:

assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$
shows $a \neq b$
is *ordering.strict-implies-not-eq* $\langle proof \rangle$

tts-lemma *order-iff-strict*:

assumes $a \in U$ **and** $b \in U$
shows $(a \leq_{ow} b) = (a <_{ow} b \vee a = b)$
is *ordering.order-iff-strict* $\langle proof \rangle$

tts-lemma *not-eq-order-implies-strict*:

assumes $a \in U$ **and** $b \in U$ **and** $a \neq b$ **and** $a \leq_{ow} b$
shows $a <_{ow} b$
is *ordering.not-eq-order-implies-strict* $\langle proof \rangle$

```

tts-lemma eq-iff:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $(a = b) = (a \leq_{ow} b \wedge b \leq_{ow} a)$ 
  is ordering.eq-iff{proof}

end

end

context order-ow
begin

tts-context
  tts: (?a to U)
  rewriting ctr-simps
  substituting order-ow-axioms
  eliminating through clarsimp
begin

tts-lemma atLeastAtMost-singleton:
  assumes  $a \in U$ 
  shows  $\{a.._{ow}a\} = \{a\}$ 
  is order-class.atLeastAtMost-singleton{proof}

tts-lemma less-imp-neq:
  assumes  $y \in U$  and  $x <_{ow} y$ 
  shows  $x \neq y$ 
  is order-class.less-imp-neq{proof}

tts-lemma atLeastatMost-empty:
  assumes  $b \in U$  and  $a \in U$  and  $b <_{ow} a$ 
  shows  $\{a.._{ow}b\} = \{\}$ 
  is order-class.atLeastatMost-empty{proof}

tts-lemma less-imp-not-eq:
  assumes  $y \in U$  and  $x <_{ow} y$ 
  shows  $(x = y) = False$ 
  is order-class.less-imp-not-eq{proof}

tts-lemma less-imp-not-eq2:
  assumes  $y \in U$  and  $x <_{ow} y$ 
  shows  $(y = x) = False$ 
  is order-class.less-imp-not-eq2{proof}

tts-lemma atLeastLessThan-empty:
  assumes  $b \in U$  and  $a \in U$  and  $b \leq_{ow} a$ 
  shows (on U with  $(\leq_{ow})$   $(<_{ow})$  :  $\{a..<b\} = \{\}$ )
  is order-class.atLeastLessThan-empty{proof}

tts-lemma greaterThanAtMost-empty:
  assumes  $l \in U$  and  $k \in U$  and  $l \leq_{ow} k$ 
  shows  $\{k<_{ow}..l\} = \{\}$ 
  is order-class.greaterThanAtMost-empty{proof}

tts-lemma antisym-conv1:
  assumes  $x \in U$  and  $y \in U$  and  $\neg x <_{ow} y$ 
  shows  $(x \leq_{ow} y) = (x = y)$ 

```

is *order-class.antisym-conv1*(proof)

tts-lemma *antisym-conv2*:

assumes $x \in U$ and $y \in U$ and $x \leq_{ow} y$

shows $(\neg x <_{ow} y) = (x = y)$

is *order-class.antisym-conv2*(proof)

tts-lemma *greaterThanLessThan-empty*:

assumes $l \in U$ and $k \in U$ and $l \leq_{ow} k$

shows $\{k <_{ow} .. <_{ow} l\} = \{\}$

is *order-class.greaterThanLessThan-empty*(proof)

tts-lemma *atLeastLessThan-eq-atLeastAtMost-diff*:

assumes $a \in U$ and $b \in U$

shows $(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{a..<b\}) = \{a.._{ow}b\} - \{b\}$

is *order-class.atLeastLessThan-eq-atLeastAtMost-diff*(proof)

tts-lemma *greaterThanAtMost-eq-atLeastAtMost-diff*:

assumes $a \in U$ and $b \in U$

shows $\{a <_{ow} .. b\} = \{a.._{ow}b\} - \{a\}$

is *order-class.greaterThanAtMost-eq-atLeastAtMost-diff*(proof)

tts-lemma *less-separate*:

assumes $x \in U$ and $y \in U$ and $x <_{ow} y$

shows

$\exists x' \in U. \exists y' \in U. x \in \{.. <_{ow} x'\} \wedge y \in \{y' <_{ow} ..\} \wedge \{.. <_{ow} x'\} \cap \{y' <_{ow} ..\} = \{\}$

is *order-class.less-separate*(proof)

tts-lemma *order-iff-strict*:

assumes $a \in U$ and $b \in U$

shows $(a \leq_{ow} b) = (a <_{ow} b \vee a = b)$

is *order-class.order.order-iff-strict*(proof)

tts-lemma *le-less*:

assumes $x \in U$ and $y \in U$

shows $(x \leq_{ow} y) = (x <_{ow} y \vee x = y)$

is *order-class.le-less*(proof)

tts-lemma *strict-iff-order*:

assumes $a \in U$ and $b \in U$

shows $(a <_{ow} b) = (a \leq_{ow} b \wedge a \neq b)$

is *order-class.order.strict-iff-order*(proof)

tts-lemma *less-le*:

assumes $x \in U$ and $y \in U$

shows $(x <_{ow} y) = (x \leq_{ow} y \wedge x \neq y)$

is *order-class.less-le*(proof)

tts-lemma *atLeastAtMost-singleton'*:

assumes $b \in U$ and $a = b$

shows $\{a.._{ow}b\} = \{a\}$

is *order-class.atLeastAtMost-singleton'*(proof)

tts-lemma *le-imp-less-or-eq*:

assumes $x \in U$ and $y \in U$ and $x \leq_{ow} y$

shows $x <_{ow} y \vee x = y$

is *order-class.le-imp-less-or-eq*(proof)

tts-lemma *antisym-conv*:

assumes $y \in U$ **and** $x \in U$ **and** $y \leq_{ow} x$
shows $(x \leq_{ow} y) = (x = y)$
is *order-class.antisym-conv* \langle *proof* \rangle

tts-lemma *le-neq-trans*:

assumes $a \in U$ **and** $b \in U$ **and** $a \leq_{ow} b$ **and** $a \neq b$
shows $a <_{ow} b$
is *order-class.le-neq-trans* \langle *proof* \rangle

tts-lemma *neq-le-trans*:

assumes $a \in U$ **and** $b \in U$ **and** $a \neq b$ **and** $a \leq_{ow} b$
shows $a <_{ow} b$
is *order-class.neq-le-trans* \langle *proof* \rangle

tts-lemma *Iio-Int-singleton*:

assumes $k \in U$ **and** $x \in U$
shows $\{..<_{ow}k\} \cap \{x\} = (\text{if } x <_{ow} k \text{ then } \{x\} \text{ else } \{\})$
is *order-class.Iio-Int-singleton* \langle *proof* \rangle

tts-lemma *atLeastAtMost-singleton-iff*:

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(\{a.._{ow}b\} = \{c\}) = (a = b \wedge b = c)$
is *order-class.atLeastAtMost-singleton-iff* \langle *proof* \rangle

tts-lemma *Icc-eq-Icc*:

assumes $l \in U$ **and** $h \in U$ **and** $l' \in U$ **and** $h' \in U$
shows $(\{l.._{ow}h\} = \{l'.._{ow}h'\}) = (h = h' \wedge l = l' \vee \neg l' \leq_{ow} h' \wedge \neg l \leq_{ow} h)$
is *order-class.Icc-eq-Icc* \langle *proof* \rangle

tts-lemma *ivl-disj-int-two*:

assumes $l \in U$ **and** $m \in U$ **and** $u \in U$
shows
 $\{l <_{ow} .. <_{ow} m\} \cap (\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{m..<u\}) = \{\}$
 $\{l <_{ow} .. m\} \cap \{m <_{ow} .. <_{ow} u\} = \{\}$
 $(\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{l..<m\}) \cap (\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{m..<u\}) = \{\}$
 $\{l.._{ow}m\} \cap \{m <_{ow} .. <_{ow} u\} = \{\}$
 $\{l <_{ow} .. <_{ow} m\} \cap \{m.._{ow}u\} = \{\}$
 $\{l <_{ow} .. m\} \cap \{m <_{ow} .. u\} = \{\}$
 $(\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{l..<m\}) \cap \{m.._{ow}u\} = \{\}$
 $\{l.._{ow}m\} \cap \{m <_{ow} .. u\} = \{\}$
is *Set-Interval.ivl-disj-int-two* \langle *proof* \rangle

tts-lemma *ivl-disj-int-one*:

assumes $l \in U$ **and** $u \in U$
shows
 $\{.._{ow}l\} \cap \{l <_{ow} .. <_{ow} u\} = \{\}$
 $\{..<_{ow}l\} \cap (\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{l..<u\}) = \{\}$
 $\{.._{ow}l\} \cap \{l <_{ow} .. u\} = \{\}$
 $\{..<_{ow}l\} \cap \{l.._{ow}u\} = \{\}$
 $\{l <_{ow} .. u\} \cap \{u <_{ow} ..\} = \{\}$
 $\{l <_{ow} .. <_{ow} u\} \cap \{u.._{ow}\} = \{\}$
 $\{l.._{ow}u\} \cap \{u <_{ow} ..\} = \{\}$
 $(\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{l..<u\}) \cap \{u.._{ow}\} = \{\}$
is *Set-Interval.ivl-disj-int-one* \langle *proof* \rangle

tts-lemma *min-absorb2*:

assumes $y \in U$ **and** $x \in U$ **and** $y \leq_{ow} x$

shows *local.min* $x y = y$
is *Orderings.min-absorb2*{proof}

tts-lemma *max-absorb1*:
assumes $y \in U$ **and** $x \in U$ **and** $y \leq_{ow} x$
shows *local.max* $x y = x$
is *Orderings.max-absorb1*{proof}

tts-lemma *atMost-Int-atLeast*:
assumes $n \in U$
shows $\{\cdot_{ow}n\} \cap \{n\cdot_{ow}\} = \{n\}$
is *Set-Interval.atMost-Int-atLeast*{proof}

tts-lemma *monoseq-Suc*:
assumes $range X \subseteq U$
shows
 (*with* $(\leq_{ow}) : \ll monoseq \gg X$) =
 $((\forall x. X x \leq_{ow} X (Suc x)) \vee (\forall x. X (Suc x) \leq_{ow} X x))$
is *Topological-Spaces.monoseq-Suc*{proof}

tts-lemma *mono-SucI2*:
assumes $range X \subseteq U$ **and** $\forall x. X (Suc x) \leq_{ow} X x$
shows *with* $(\leq_{ow}) : \ll monoseq \gg X$
is *Topological-Spaces.mono-SucI2*{proof}

tts-lemma *mono-SucI1*:
assumes $range X \subseteq U$ **and** $\forall x. X x \leq_{ow} X (Suc x)$
shows *with* $(\leq_{ow}) : \ll monoseq \gg X$
is *Topological-Spaces.mono-SucI1*{proof}

tts-lemma *monoI2*:
assumes $range X \subseteq U$ **and** $\forall x y. x \leq y \longrightarrow X y \leq_{ow} X x$
shows *with* $(\leq_{ow}) : \ll monoseq \gg X$
is *Topological-Spaces.monoI2*{proof}

tts-lemma *monoI1*:
assumes $range X \subseteq U$ **and** $\forall x y. x \leq y \longrightarrow X x \leq_{ow} X y$
shows *with* $(\leq_{ow}) : \ll monoseq \gg X$
is *Topological-Spaces.monoI1*{proof}

end

tts-context
tts: (*?a to U*)
rewriting *ctr-simps*
substituting *order-ow-axioms*
eliminating through *clarsimp*
begin

tts-lemma *ex-min-if-finite*:
assumes $S \subseteq U$
and *finite S*
and $S \neq \{\}$
shows $\exists x \in S. \neg (\exists y \in S. y <_{ow} x)$
is *Lattices-Big.ex-min-if-finite*{proof}

end

tts-context

tts: ($?a$ to U)

sbterms: ($\langle(\leq)::['a::order, 'a::order] \Rightarrow bool\rangle$ to $\langle(\leq_{ow})\rangle$)

and ($\langle(\leq)::['a::order, 'a::order] \Rightarrow bool\rangle$ to $\langle(\leq_{ow})\rangle$)

substituting *order-ow-axioms*

eliminating through *clarsimp*

begin

tts-lemma *xt1*:

shows

$\llbracket a = b; c <_{ow} b \rrbracket \Longrightarrow c <_{ow} a$

$\llbracket b <_{ow} a; b = c \rrbracket \Longrightarrow c <_{ow} a$

$\llbracket a = b; c \leq_{ow} b \rrbracket \Longrightarrow c \leq_{ow} a$

$\llbracket b \leq_{ow} a; b = c \rrbracket \Longrightarrow c \leq_{ow} a$

$\llbracket y \in U; x \in U; y \leq_{ow} x; x \leq_{ow} y \rrbracket \Longrightarrow x = y$

$\llbracket y \in U; x \in U; z \in U; y \leq_{ow} x; z \leq_{ow} y \rrbracket \Longrightarrow z \leq_{ow} x$

$\llbracket y \in U; x \in U; z \in U; y <_{ow} x; z \leq_{ow} y \rrbracket \Longrightarrow z <_{ow} x$

$\llbracket y \in U; x \in U; z \in U; y \leq_{ow} x; z <_{ow} y \rrbracket \Longrightarrow z <_{ow} x$

$\llbracket b \in U; a \in U; b <_{ow} a; a <_{ow} b \rrbracket \Longrightarrow P$

$\llbracket y \in U; x \in U; z \in U; y <_{ow} x; z <_{ow} y \rrbracket \Longrightarrow z <_{ow} x$

$\llbracket b \in U; a \in U; b \leq_{ow} a; a \neq b \rrbracket \Longrightarrow b <_{ow} a$

$\llbracket a \in U; b \in U; a \neq b; b \leq_{ow} a \rrbracket \Longrightarrow b <_{ow} a$

\llbracket

$b \in U;$

$c \in U;$

$a = f b;$

$c <_{ow} b;$

$\wedge x y. \llbracket x \in U; y \in U; y <_{ow} x \rrbracket \Longrightarrow f y <_{ow} f x$

$\rrbracket \Longrightarrow f c <_{ow} a$

\llbracket

$b \in U;$

$a \in U;$

$b <_{ow} a;$

$f b = c;$

$\wedge x y. \llbracket x \in U; y \in U; y <_{ow} x \rrbracket \Longrightarrow f y <_{ow} f x$

$\rrbracket \Longrightarrow c <_{ow} f a$

\llbracket

$b \in U;$

$c \in U;$

$a = f b;$

$c \leq_{ow} b;$

$\wedge x y. \llbracket x \in U; y \in U; y \leq_{ow} x \rrbracket \Longrightarrow f y \leq_{ow} f x$

$\rrbracket \Longrightarrow f c \leq_{ow} a$

\llbracket

$b \in U;$

$a \in U;$

$b \leq_{ow} a;$

$f b = c;$

$\wedge x y. \llbracket x \in U; y \in U; y \leq_{ow} x \rrbracket \Longrightarrow f y \leq_{ow} f x$

$\rrbracket \Longrightarrow c \leq_{ow} f a$

is *Orderings.xt1*{proof}

end

end

context *ord-order-ow*

begin

tts-context

tts: ($?a$ to U_2) and ($?b$ to U_1)
sbterms: ($\langle(\leq)::[?a::order, ?a::order] \Rightarrow bool\rangle$ to $\langle(\leq_{ow.2})\rangle$)
 and ($\langle(\leq)::[?a::order, ?a::order] \Rightarrow bool\rangle$ to $\langle(\leq_{ow.2})\rangle$)
 and ($\langle(\leq)::[?b::ord, ?b::ord] \Rightarrow bool\rangle$ to $\langle(\leq_{ow.1})\rangle$)
 and ($\langle(\leq)::[?b::ord, ?b::ord] \Rightarrow bool\rangle$ to $\langle(\leq_{ow.1})\rangle$)
rewriting *ctr-simps*
substituting $ord_2.order-ow-axioms$
eliminating through *clarsimp*

begin

tts-lemma *xt2:*

assumes $\forall x \in U_1. f x \in U_2$
 and $b \in U_1$
 and $a \in U_2$
 and $c \in U_1$
 and $f b \leq_{ow.2} a$
 and $c \leq_{ow.1} b$
 and $\bigwedge x y. [x \in U_1; y \in U_1; y \leq_{ow.1} x] \Longrightarrow f y \leq_{ow.2} f x$
shows $f c \leq_{ow.2} a$
is *Orderings.xt2<proof>*

tts-lemma *xt6:*

assumes $\forall x \in U_1. f x \in U_2$
 and $b \in U_1$
 and $a \in U_2$
 and $c \in U_1$
 and $f b \leq_{ow.2} a$
 and $c <_{ow.1} b$
 and $\bigwedge x y. [x \in U_1; y \in U_1; y <_{ow.1} x] \Longrightarrow f y <_{ow.2} f x$
shows $f c <_{ow.2} a$
is *Orderings.xt6<proof>*

end

end

context *order-pair-ow*

begin

tts-context

tts: ($?a$ to U_1) and ($?b$ to U_2)
sbterms: ($\langle(\leq)::[?a::order, ?a::order] \Rightarrow bool\rangle$ to $\langle(\leq_{ow.1})\rangle$)
 and ($\langle(\leq)::[?a::order, ?a::order] \Rightarrow bool\rangle$ to $\langle(\leq_{ow.1})\rangle$)
 and ($\langle(\leq)::[?b::order, ?b::order] \Rightarrow bool\rangle$ to $\langle(\leq_{ow.2})\rangle$)
 and ($\langle(\leq)::[?b::order, ?b::order] \Rightarrow bool\rangle$ to $\langle(\leq_{ow.2})\rangle$)
rewriting *ctr-simps*
substituting $ord_1.order-ow-axioms$ and $ord_2.order-ow-axioms$
eliminating through *clarsimp*

begin

tts-lemma *xt3:*

assumes $b \in U_1$
 and $a \in U_1$
 and $c \in U_2$
 and $\forall x \in U_1. f x \in U_2$
 and $b \leq_{ow.1} a$

and $c \leq_{ow.2} f b$
and $\wedge x y. [[x \in U_1; y \in U_1; y \leq_{ow.1} x]] \implies f y \leq_{ow.2} f x$
shows $c \leq_{ow.2} f a$
is *Orderings.xt3* \langle *proof* \rangle

tts-lemma *xt4*:

assumes $\forall x \in U_2. f x \in U_1$
and $b \in U_2$
and $a \in U_1$
and $c \in U_2$
and $f b <_{ow.1} a$
and $c \leq_{ow.2} b$
and $\wedge x y. [[x \in U_2; y \in U_2; y \leq_{ow.2} x]] \implies f y \leq_{ow.1} f x$
shows $f c <_{ow.1} a$
is *Orderings.xt4* \langle *proof* \rangle

tts-lemma *xt5*:

assumes $b \in U_1$
and $a \in U_1$
and $c \in U_2$
and $\forall x \in U_1. f x \in U_2$
and $b <_{ow.1} a$
and $c \leq_{ow.2} f b$
and $\wedge x y. [[x \in U_1; y \in U_1; y <_{ow.1} x]] \implies f y <_{ow.2} f x$
shows $c <_{ow.2} f a$
is *Orderings.xt5* \langle *proof* \rangle

tts-lemma *xt7*:

assumes $b \in U_1$
and $a \in U_1$
and $c \in U_2$
and $\forall x \in U_1. f x \in U_2$
and $b \leq_{ow.1} a$
and $c <_{ow.2} f b$
and $\wedge x y. [[x \in U_1; y \in U_1; y \leq_{ow.1} x]] \implies f y \leq_{ow.2} f x$
shows $c <_{ow.2} f a$
is *Orderings.xt7* \langle *proof* \rangle

tts-lemma *xt8*:

assumes $\forall x \in U_2. f x \in U_1$
and $b \in U_2$
and $a \in U_1$
and $c \in U_2$
and $f b <_{ow.1} a$
and $c <_{ow.2} b$
and $\wedge x y. [[x \in U_2; y \in U_2; y <_{ow.2} x]] \implies f y <_{ow.1} f x$
shows $f c <_{ow.1} a$
is *Orderings.xt8* \langle *proof* \rangle

tts-lemma *xt9*:

assumes $b \in U_1$
and $a \in U_1$
and $c \in U_2$
and $\forall x \in U_1. f x \in U_2$
and $b <_{ow.1} a$
and $c <_{ow.2} f b$
and $\wedge x y. [[x \in U_1; y \in U_1; y <_{ow.1} x]] \implies f y <_{ow.2} f x$
shows $c <_{ow.2} f a$

```

is Orderings.xt9<proof>

end

tts-context
  tts: (?'a to U1) and (?'b to U2)
  sbterms: (⟨(≤)::[?'a::order, ?'a::order] ⇒ bool⟩ to ⟨(≤ow.1)⟩)
    and (⟨(<)::[?'a::order, ?'a::order] ⇒ bool⟩ to ⟨(<ow.1)⟩)
    and (⟨(≤)::[?'b::order, ?'b::order] ⇒ bool⟩ to ⟨(≤ow.2)⟩)
    and (⟨(<)::[?'b::order, ?'b::order] ⇒ bool⟩ to ⟨(<ow.2)⟩)
  rewriting ctr-simps
  substituting ord1.order-ow-axioms and ord2.order-ow-axioms
  eliminating through simp
begin

tts-lemma order-less-subst1:
  assumes a ∈ U1
    and ∀ x ∈ U2. f x ∈ U1
    and b ∈ U2
    and c ∈ U2
    and a <ow.1 f b
    and b <ow.2 c
    and ∧ x y. [[x ∈ U2; y ∈ U2; x <ow.2 y]] ⇒ f x <ow.1 f y
  shows a <ow.1 f c
  is Orderings.order-less-subst1<proof>

tts-lemma order-subst1:
  assumes a ∈ U1
    and ∀ x ∈ U2. f x ∈ U1
    and b ∈ U2
    and c ∈ U2
    and a ≤ow.1 f b
    and b ≤ow.2 c
    and ∧ x y. [[x ∈ U2; y ∈ U2; x ≤ow.2 y]] ⇒ f x ≤ow.1 f y
  shows a ≤ow.1 f c
  is Orderings.order-subst1<proof>

end

tts-context
  tts: (?'a to U1) and (?'c to U2)
  sbterms: (⟨(≤)::[?'a::order, ?'a::order] ⇒ bool⟩ to ⟨(≤ow.1)⟩)
    and (⟨(<)::[?'a::order, ?'a::order] ⇒ bool⟩ to ⟨(<ow.1)⟩)
    and (⟨(≤)::[?'c::order, ?'c::order] ⇒ bool⟩ to ⟨(≤ow.2)⟩)
    and (⟨(<)::[?'c::order, ?'c::order] ⇒ bool⟩ to ⟨(<ow.2)⟩)
  rewriting ctr-simps
  substituting ord1.order-ow-axioms and ord2.order-ow-axioms
  eliminating through simp
begin

tts-lemma order-less-le-subst2:
  assumes a ∈ U1
    and b ∈ U1
    and ∀ x ∈ U1. f x ∈ U2
    and c ∈ U2
    and a <ow.1 b
    and f b ≤ow.2 c
    and ∧ x y. [[x ∈ U1; y ∈ U1; x <ow.1 y]] ⇒ f x <ow.2 f y

```

shows $f a <_{ow.2} c$
is *Orderings.order-less-le-subst2*{proof}

tts-lemma *order-le-less-subst2*:

assumes $a \in U_1$
and $b \in U_1$
and $\forall x \in U_1. f x \in U_2$
and $c \in U_2$
and $a \leq_{ow.1} b$
and $f b <_{ow.2} c$
and $\bigwedge x y. [[x \in U_1; y \in U_1; x \leq_{ow.1} y]] \implies f x \leq_{ow.2} f y$
shows $f a <_{ow.2} c$
is *Orderings.order-le-less-subst2*{proof}

tts-lemma *order-less-subst2*:

assumes $a \in U_1$
and $b \in U_1$
and $\forall x \in U_1. f x \in U_2$
and $c \in U_2$
and $a <_{ow.1} b$
and $f b <_{ow.2} c$
and $\bigwedge x y. [[x \in U_1; y \in U_1; x <_{ow.1} y]] \implies f x <_{ow.2} f y$
shows $f a <_{ow.2} c$
is *Orderings.order-less-subst2*{proof}

tts-lemma *order-subst2*:

assumes $a \in U_1$
and $b \in U_1$
and $\forall x \in U_1. f x \in U_2$
and $c \in U_2$
and $a \leq_{ow.1} b$
and $f b \leq_{ow.2} c$
and $\bigwedge x y. [[x \in U_1; y \in U_1; x \leq_{ow.1} y]] \implies f x \leq_{ow.2} f y$
shows $f a \leq_{ow.2} c$
is *Orderings.order-subst2*{proof}

end

end

3.6.4 Dense orders

Definitions and common properties

locale *dense-order-ow = order-ow U le ls*

for $U :: 'ao \text{ set}$ **and** $le \text{ ls} +$

assumes *dense*: $[[x \in U; y \in U; x <_{ow} y]] \implies (\exists z \in U. x <_{ow} z \wedge z <_{ow} y)$

Transfer rules

context

includes *lifting-syntax*

begin

lemma *dense-order-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$((A \implies A \implies (=)) \implies (A \implies A \implies (=)) \implies (=))$
(dense-order-ow (Collect (Domainp A))) class.dense-order

$\langle proof \rangle$

end

3.6.5 Partial orders with the greatest element and partial orders with the least elements

Definitions and common properties

locale *ordering-top-ow* = *ordering-ow* *U* *le* *ls*
 for *U* :: 'ao set and *le* *ls* +
 fixes *top* :: 'ao ($\langle \top_{ow} \rangle$)
 assumes *top-closed*[*simp*]: $\top_{ow} \in U$
 assumes *extremum*[*simp*]: $a \in U \implies a \leq_{ow} \top_{ow}$
 begin

notation *top* ($\langle \top_{ow} \rangle$)

end

locale *bot-ow* =
 fixes *U* :: 'ao set and *bot* ($\langle \perp_{ow} \rangle$)
 assumes *bot-closed*[*simp*]: $\perp_{ow} \in U$
 begin

notation *bot* ($\langle \perp_{ow} \rangle$)

end

locale *bot-pair-ow* = *ord*₁: *bot-ow* *U*₁ *bot*₁ + *ord*₂: *bot-ow* *U*₂ *bot*₂
 for *U*₁ :: 'ao set and *bot*₁ and *U*₂ :: 'bo set and *bot*₂
 begin

notation *bot*₁ ($\langle \perp_{ow.1} \rangle$)

notation *bot*₂ ($\langle \perp_{ow.2} \rangle$)

end

locale *order-bot-ow* = *order-ow* *U* *le* *ls* + *bot-ow* *U* *bot*
 for *U* :: 'ao set and *bot* *le* *ls* +
 assumes *bot-least*: $a \in U \implies \perp_{ow} \leq_{ow} a$
 begin

sublocale *bot*: *ordering-top-ow* *U* ($\langle \geq_{ow} \rangle$) ($\langle >_{ow} \rangle$) ($\langle \perp_{ow} \rangle$)
 $\langle proof \rangle$

no-notation *le* (**infix** $\langle \leq_{ow} \rangle$ 50)
 and *ls* (**infix** $\langle <_{ow} \rangle$ 50)
 and *top* ($\langle \top_{ow} \rangle$)

end

locale *order-bot-pair-ow* =
*ord*₁: *order-bot-ow* *U*₁ *bot*₁ *le*₁ *ls*₁ + *ord*₂: *order-bot-ow* *U*₂ *bot*₂ *le*₂ *ls*₂
 for *U*₁ :: 'ao set and *bot*₁ *le*₁ *ls*₁ and *U*₂ :: 'bo set and *bot*₂ *le*₂ *ls*₂
 begin

sublocale *bot-pair-ow* $\langle proof \rangle$

```

sublocale order-pair-ow ⟨proof⟩

end

locale top-ow =
  fixes  $U :: 'ao\ set$  and  $top \langle \top_{ow} \rangle$ 
  assumes top-closed[simp]:  $\top_{ow} \in U$ 
begin

  notation  $top \langle \top_{ow} \rangle$ 

end

locale top-pair-ow =  $ord_1: top-ow\ U_1\ top_1 + ord_2: top-ow\ U_2\ top_2$ 
  for  $U_1 :: 'ao\ set$  and  $top_1$  and  $U_2 :: 'bo\ set$  and  $top_2$ 
begin

  notation  $top_1 \langle \top_{ow.1} \rangle$ 
  notation  $top_2 \langle \top_{ow.2} \rangle$ 

end

locale order-top-ow =  $order-ow\ U\ le\ ls + top-ow\ U\ top$ 
  for  $U :: 'ao\ set$  and  $le\ ls\ top +$ 
  assumes top-greatest:  $a \in U \implies a \leq_{ow} \top_{ow}$ 
begin

  sublocale top: ordering-top-ow  $U \langle \leq_{ow} \rangle \langle \langle \leq_{ow} \rangle \top_{ow} \rangle$ 
  ⟨proof⟩

  no-notation  $le$  (infix  $\langle \leq_{ow} \rangle$  50)
  and  $ls$  (infix  $\langle \leq_{ow} \rangle$  50)
  and  $top$  ( $\langle \top_{ow} \rangle$ )

end

locale order-top-pair-ow =
   $ord_1: order-top-ow\ U_1\ le_1\ ls_1\ top_1 + ord_2: order-top-ow\ U_2\ le_2\ ls_2\ top_2$ 
  for  $U_1 :: 'ao\ set$  and  $le_1\ ls_1\ top_1$  and  $U_2 :: 'bo\ set$  and  $le_2\ ls_2\ top_2$ 
begin

  sublocale top-pair-ow ⟨proof⟩
  sublocale order-pair-ow ⟨proof⟩

end

Transfer rules

context
  includes lifting-syntax
begin

lemma ordering-top-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique  $A$  right-total  $A$ 
  shows
   $((A \implies A \implies (=)) \implies (A \implies A \implies (=)) \implies A \implies (=))$ 
  (ordering-top-ow (Collect (Domainp  $A$ ))) ordering-top

```

<proof>

lemma *order-bot-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$(A \text{ ===> } (A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } (A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } (=))$
(order-bot-ow (Collect (Domainp A))) class.order-bot

<proof>

lemma *order-top-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$((A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } (A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } A \text{ ===> } (=))$
(order-top-ow (Collect (Domainp A))) class.order-top

<proof>

end

Relativization

context *ordering-top-ow*

begin

tts-context

tts: (*?a to U*)

rewriting *ctr-simps*

substituting *ordering-top-ow-axioms*

eliminating through *simp*

applying [*OF top-closed*]

begin

tts-lemma *extremum-uniqueI*:

assumes $\top_{ow} \leq_{ow} \top_{ow}$

shows $\top_{ow} = \top_{ow}$

is *ordering-top.extremum-uniqueI**<proof>*

tts-lemma *extremum-unique*:

shows $(\top_{ow} \leq_{ow} \top_{ow}) = (\top_{ow} = \top_{ow})$

is *ordering-top.extremum-unique**<proof>*

tts-lemma *extremum-strict*:

shows $\neg \top_{ow} <_{ow} \top_{ow}$

is *ordering-top.extremum-strict**<proof>*

tts-lemma *not-eq-extremum*:

shows $(\top_{ow} \neq \top_{ow}) = (\top_{ow} <_{ow} \top_{ow})$

is *ordering-top.not-eq-extremum**<proof>*

end

end

context *order-bot-ow*

begin

tts-context

tts: (*?a to U*)

rewriting *ctr-simps*

```

substituting order-bot-ow-axioms
eliminating through simp
begin

tts-lemma bdd-above-bot:
  assumes  $A \subseteq U$ 
  shows bdd-below  $A$ 
  is order-bot-class.bdd-below-bot{proof}

tts-lemma not-less-bot:
  assumes  $a \in U$ 
  shows  $\neg a <_{ow} \perp_{ow}$ 
  is order-bot-class.not-less-bot{proof}

tts-lemma max-bot:
  assumes  $x \in U$ 
  shows  $max \perp_{ow} x = x$ 
  is order-bot-class.max-bot{proof}

tts-lemma max-bot2:
  assumes  $x \in U$ 
  shows  $max x \perp_{ow} = x$ 
  is order-bot-class.max-bot2{proof}

tts-lemma min-bot:
  assumes  $x \in U$ 
  shows  $min \perp_{ow} x = \perp_{ow}$ 
  is order-bot-class.min-bot{proof}

tts-lemma min-bot2:
  assumes  $x \in U$ 
  shows  $min x \perp_{ow} = \perp_{ow}$ 
  is order-bot-class.min-bot2{proof}

tts-lemma bot-unique:
  assumes  $a \in U$ 
  shows  $(a \leq_{ow} \perp_{ow}) = (a = \perp_{ow})$ 
  is order-bot-class.bot-unique{proof}

tts-lemma bot-less:
  assumes  $a \in U$ 
  shows  $(a \neq \perp_{ow}) = (\perp_{ow} <_{ow} a)$ 
  is order-bot-class.bot-less{proof}

tts-lemma atLeast-eq-UNIV-iff:
  assumes  $x \in U$ 
  shows  $(\{x.._{ow}\} = U) = (x = \perp_{ow})$ 
  is order-bot-class.atLeast-eq-UNIV-iff{proof}

tts-lemma le-bot:
  assumes  $a \in U$  and  $a \leq_{ow} \perp_{ow}$ 
  shows  $a = \perp_{ow}$ 
  is order-bot-class.le-bot{proof}

end

end

```

```

context order-top-ow
begin

tts-context
  tts: (?a to U)
  rewriting ctr-simps
  substituting order-top-ow-axioms
  eliminating through simp
begin

tts-lemma bdd-above-top:
  assumes  $A \subseteq U$ 
  shows bdd-above  $A$ 
  is order-top-class.bdd-above-top{proof}

tts-lemma not-top-less:
  assumes  $a \in U$ 
  shows  $\neg \top_{ow} <_{ow} a$ 
  is order-top-class.not-top-less{proof}

tts-lemma max-top:
  assumes  $x \in U$ 
  shows  $\max \top_{ow} x = \top_{ow}$ 
  is order-top-class.max-top{proof}

tts-lemma max-top2:
  assumes  $x \in U$ 
  shows  $\max x \top_{ow} = \top_{ow}$ 
  is order-top-class.max-top2{proof}

tts-lemma min-top:
  assumes  $x \in U$ 
  shows  $\min \top_{ow} x = x$ 
  is order-top-class.min-top{proof}

tts-lemma min-top2:
  assumes  $x \in U$ 
  shows  $\min x \top_{ow} = x$ 
  is order-top-class.min-top2{proof}

tts-lemma top-unique:
  assumes  $a \in U$ 
  shows  $(\top_{ow} \leq_{ow} a) = (a = \top_{ow})$ 
  is order-top-class.top-unique{proof}

tts-lemma less-top:
  assumes  $a \in U$ 
  shows  $(a \neq \top_{ow}) = (a <_{ow} \top_{ow})$ 
  is order-top-class.less-top{proof}

tts-lemma atMost-eq-UNIV-iff:
  assumes  $x \in U$ 
  shows  $(\{\cdot_{ow}x\} = U) = (x = \top_{ow})$ 
  is order-top-class.atMost-eq-UNIV-iff{proof}

tts-lemma top-le:
  assumes  $a \in U$  and  $\top_{ow} \leq_{ow} a$ 
  shows  $a = \top_{ow}$ 

```

```
    is order-top-class.top-le(proof)
end
end
```

3.7 Relativization of the results about semigroups

3.7.1 Simple semigroups

Definitions and common properties

locale *semigroup-ow* =

fixes $U :: 'ag\ set$ and $f :: ['ag, 'ag] \Rightarrow 'ag$ (**infixl** $\langle *_{ow} \rangle$ 70)

assumes *f-closed*: $[[a \in U; b \in U]] \Longrightarrow a *_{ow} b \in U$

assumes *assoc*: $[[a \in U; b \in U; c \in U]] \Longrightarrow a *_{ow} b *_{ow} c = a *_{ow} (b *_{ow} c)$

begin

notation f (**infixl** $\langle *_{ow} \rangle$ 70)

lemma *f-closed*'[*simp*]: $\forall x \in U. \forall y \in U. x *_{ow} y \in U$ *<proof>*

tts-register-sbts $\langle (*_{ow}) \rangle \mid U$ *<proof>*

end

lemma *semigroup-ow*: *semigroup* = *semigroup-ow UNIV*

<proof>

locale *plus-ow* =

fixes $U :: 'ag\ set$ and *plus* :: $['ag, 'ag] \Rightarrow 'ag$ (**infixl** $\langle +_{ow} \rangle$ 65)

assumes *plus-closed*[*simp, intro*]: $[[a \in U; b \in U]] \Longrightarrow a +_{ow} b \in U$

begin

notation *plus* (**infixl** $\langle +_{ow} \rangle$ 65)

lemma *plus-closed*'[*simp*]: $\forall x \in U. \forall y \in U. x +_{ow} y \in U$ *<proof>*

tts-register-sbts $\langle (+_{ow}) \rangle \mid U$ *<proof>*

end

locale *times-ow* =

fixes $U :: 'ag\ set$ and *times* :: $['ag, 'ag] \Rightarrow 'ag$ (**infixl** $\langle *_{ow} \rangle$ 70)

assumes *times-closed*[*simp, intro*]: $[[a \in U; b \in U]] \Longrightarrow a *_{ow} b \in U$

begin

notation *times* (**infixl** $\langle *_{ow} \rangle$ 70)

lemma *times-closed*'[*simp*]: $\forall x \in U. \forall y \in U. x *_{ow} y \in U$ *<proof>*

tts-register-sbts $\langle (*_{ow}) \rangle \mid U$ *<proof>*

end

locale *semigroup-add-ow* = *plus-ow U plus*

for $U :: 'ag\ set$ and *plus* +

assumes *plus-assoc*:

$[[a \in U; b \in U; c \in U]] \Longrightarrow a +_{ow} b +_{ow} c = a +_{ow} (b +_{ow} c)$

begin

sublocale *add*: *semigroup-ow U* $\langle (+_{ow}) \rangle$

<proof>

end

lemma *semigroup-add-ow*: *class.semigroup-add = semigroup-add-ow UNIV*
 ⟨*proof*⟩

locale *semigroup-mult-ow = times-ow U times*
for *U* :: 'ag set **and** *times* +
assumes *mult-assoc*:
 [[*a* ∈ *U*; *b* ∈ *U*; *c* ∈ *U*]] $\implies a *_{ow} b *_{ow} c = a *_{ow} (b *_{ow} c)$
begin

sublocale *mult*: *semigroup-ow U* ⟨ $(*_ow)$ ⟩
 ⟨*proof*⟩

end

lemma *semigroup-mult-ow*: *class.semigroup-mult = semigroup-mult-ow UNIV*
 ⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semigroup-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 ((*A* \implies *A* \implies *A*) \implies (=))
 (*semigroup-ow* (*Collect* (*Domainp A*))) *semigroup*
 ⟨*proof*⟩

lemma *semigroup-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 ((*A* \implies *A* \implies *A*) \implies (=))
 (*semigroup-add-ow* (*Collect* (*Domainp A*))) *class.semigroup-add*
 ⟨*proof*⟩

lemma *semigroup-mult-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 ((*A* \implies *A* \implies *A*) \implies (=))
 (*semigroup-mult-ow* (*Collect* (*Domainp A*))) *class.semigroup-mult*
 ⟨*proof*⟩

end

3.7.2 Cancellative semigroups

Definitions and common properties

locale *cancel-semigroup-add-ow = semigroup-add-ow U plus*
for *U* :: 'ag set **and** *plus* +
assumes *add-left-imp-eq*:
 [[*a* ∈ *U*; *b* ∈ *U*; *c* ∈ *U*; *a* +_{ow} *b* = *a* +_{ow} *c*]] $\implies b = c$
assumes *add-right-imp-eq*:
 [[*b* ∈ *U*; *a* ∈ *U*; *c* ∈ *U*; *b* +_{ow} *a* = *c* +_{ow} *a*]] $\implies b = c$

lemma *cancel-semigroup-add-ow*:

```
class.cancel-semigroup-add = cancel-semigroup-add-ow UNIV
⟨proof⟩
```

Transfer rules

context

includes *lifting-syntax*

begin

lemma *cancel-semigroup-add-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

```
((A ==> A ==> A) ==> (=))
(cancel-semigroup-add-ow (Collect (Domainp A)))
class.cancel-semigroup-add
⟨proof⟩
```

end

Relativization

context *cancel-semigroup-add-ow*

begin

tts-context

tts: (*?'a to U*)

rewriting *ctr-simps*

substituting *cancel-semigroup-add-ow-axioms*

eliminating through *simp*

begin

tts-lemma *add-right-cancel*:

assumes *b ∈ U and a ∈ U and c ∈ U*

shows $(b +_{ow} a = c +_{ow} a) = (b = c)$

is *cancel-semigroup-add-class.add-right-cancel*⟨*proof*⟩

tts-lemma *add-left-cancel*:

assumes *a ∈ U and b ∈ U and c ∈ U*

shows $(a +_{ow} b = a +_{ow} c) = (b = c)$

is *cancel-semigroup-add-class.add-left-cancel*⟨*proof*⟩

tts-lemma *bij-betw-add*:

assumes *a ∈ U and A ⊆ U and B ⊆ U*

shows *bij-betw* $((+_{ow}) a) A B = ((+_{ow}) a) A = B$

is *cancel-semigroup-add-class.bij-betw-add*⟨*proof*⟩

tts-lemma *inj-on-add*:

assumes *a ∈ U and A ⊆ U*

shows *inj-on* $((+_{ow}) a) A$

is *cancel-semigroup-add-class.inj-on-add*⟨*proof*⟩

tts-lemma *inj-on-add'*:

assumes *a ∈ U and A ⊆ U*

shows *inj-on* $(\lambda b. b +_{ow} a) A$

is *cancel-semigroup-add-class.inj-on-add'*⟨*proof*⟩

end

end

3.7.3 Commutative semigroups

Definitions and common properties

locale *abel-semigroup-ow* =
semigroup-ow *U* *f* **for** *U* :: 'ag set **and** *f* +
assumes *commute*: $[[a \in U; b \in U]] \implies a *_{ow} b = b *_{ow} a$
begin

lemma *fun-left-comm*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $y *_{ow} (x *_{ow} z) = x *_{ow} (y *_{ow} z)$
 ⟨*proof*⟩

end

lemma *abel-semigroup-ow*: *abel-semigroup* = *abel-semigroup-ow* *UNIV*
 ⟨*proof*⟩

locale *ab-semigroup-add-ow* =
semigroup-add-ow *U* *plus* **for** *U* :: 'ag set **and** *plus* +
assumes *add-commute*: $[[a \in U; b \in U]] \implies a +_{ow} b = b +_{ow} a$
begin

sublocale *add*: *abel-semigroup-ow* *U* ⟨*(+_{ow})*⟩
 ⟨*proof*⟩

end

lemma *ab-semigroup-add-ow*: *class.ab-semigroup-add* = *ab-semigroup-add-ow* *UNIV*
 ⟨*proof*⟩

locale *ab-semigroup-mult-ow* =
semigroup-mult-ow *U* *times* **for** *U* :: 'ag set **and** *times* +
assumes *mult-commute*: $[[a \in U; b \in U]] \implies a *_{ow} b = b *_{ow} a$
begin

sublocale *mult*: *abel-semigroup-ow* *U* ⟨*(*_{ow})*⟩
 ⟨*proof*⟩

end

lemma *ab-semigroup-mult-ow*:
class.ab-semigroup-mult = *ab-semigroup-mult-ow* *UNIV*
 ⟨*proof*⟩

Transfer rules

context

includes *lifting-syntax*

begin

lemma *abel-semigroup-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
 $((A \implies A \implies A) \implies (=))$
 (*abel-semigroup-ow* (*Collect* (*Domainp* *A*))) *abel-semigroup*

<proof>

lemma *ab-semigroup-add-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
 ((*A* \implies *A* \implies *A*) \implies (=))
 (*ab-semigroup-add-ow* (*Collect* (*Domainp A*))) *class.ab-semigroup-add*
<proof>

lemma *ab-semigroup-mult-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
 ((*A* \implies *A* \implies *A*) \implies (=))
 (*ab-semigroup-mult-ow* (*Collect* (*Domainp A*))) *class.ab-semigroup-mult*
<proof>

end

Relativization

context *abel-semigroup-ow*

begin

tts-context

tts: (*?a to U*)

rewriting *ctr-simps*

substituting *abel-semigroup-ow-axioms*

eliminating through *simp*

begin

tts-lemma *left-commute*:

assumes *b* \in *U* **and** *a* \in *U* **and** *c* \in *U*

shows *b* \ast_{ow} (*a* \ast_{ow} *c*) = *a* \ast_{ow} (*b* \ast_{ow} *c*)

is *abel-semigroup.left-commute**<proof>*

end

end

context *ab-semigroup-add-ow*

begin

tts-context

tts: (*?a to U*)

rewriting *ctr-simps*

substituting *ab-semigroup-add-ow-axioms*

eliminating through *simp*

begin

tts-lemma *add-ac*:

shows $[[a \in U; b \in U; c \in U]] \implies a +_{ow} b +_{ow} c = a +_{ow} (b +_{ow} c)$

is *ab-semigroup-add-class.add-ac(1)*

and $[[a \in U; b \in U]] \implies a +_{ow} b = b +_{ow} a$

is *ab-semigroup-add-class.add-ac(2)*

and $[[b \in U; a \in U; c \in U]] \implies b +_{ow} (a +_{ow} c) = a +_{ow} (b +_{ow} c)$

is *ab-semigroup-add-class.add-ac(3)**<proof>*

end

end

context *ab-semigroup-mult-ow*
begin

tts-context

tts: (?*a* to *U*)

rewriting *ctr-simps*

substituting *ab-semigroup-mult-ow-axioms*

eliminating through *simp*

begin

tts-lemma *mult-ac*:

shows $[[a \in U; b \in U; c \in U]] \implies a *_{ow} b *_{ow} c = a *_{ow} (b *_{ow} c)$

is *ab-semigroup-mult-class.mult-ac(1)*

and $[[a \in U; b \in U]] \implies a *_{ow} b = b *_{ow} a$

is *ab-semigroup-mult-class.mult-ac(2)*

and $[[b \in U; a \in U; c \in U]] \implies b *_{ow} (a *_{ow} c) = a *_{ow} (b *_{ow} c)$

is *ab-semigroup-mult-class.mult-ac(3)**<proof>*

end

end

3.7.4 Cancellative commutative semigroups

Definitions and common properties

locale *minus-ow* =

fixes *U* :: '*ag set* and *minus* :: [*'ag*, '*ag*] \Rightarrow '*ag* (infixl $\langle -_{ow} \rangle$ 65)

assumes *minus-closed*[*simp,intro*]: $[[a \in U; b \in U]] \implies a -_{ow} b \in U$

begin

notation *minus* (infixl $\langle -_{ow} \rangle$ 65)

lemma *minus-closed'*[*simp*]: $\forall x \in U. \forall y \in U. x -_{ow} y \in U$ *<proof>*

tts-register-sbts $\langle (-_{ow}) \rangle$ | *U* *<proof>*

end

locale *cancel-ab-semigroup-add-ow* =

ab-semigroup-add-ow U plus + minus-ow U minus

for *U* :: '*ag set* and *plus* *minus* +

assumes *add-diff-cancel-left'*[*simp*]:

$[[a \in U; b \in U]] \implies (a +_{ow} b) -_{ow} a = b$

assumes *diff-diff-add*:

$[[a \in U; b \in U; c \in U]] \implies a -_{ow} b -_{ow} c = a -_{ow} (b +_{ow} c)$

begin

sublocale *cancel-semigroup-add-ow U* $\langle (+_{ow}) \rangle$
<proof>

end

lemma *cancel-ab-semigroup-add-ow*:

class.cancel-ab-semigroup-add = cancel-ab-semigroup-add-ow UNIV

<proof>

Transfer rules

context

includes *lifting-syntax*

begin

lemma *cancel-ab-semigroup-add-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$((A \text{ ===> } A \text{ ===> } A) \text{ ===> } (A \text{ ===> } A \text{ ===> } A) \text{ ===> } (=))$
(cancel-ab-semigroup-add-ow (Collect (Domainp A)))
class.cancel-ab-semigroup-add

<proof>

end

Relativization

context *cancel-ab-semigroup-add-ow*

begin

tts-context

tts: (*?'a to U*)

rewriting *ctr-simps*

substituting *cancel-ab-semigroup-add-ow-axioms*

eliminating through *simp*

begin

tts-lemma *add-diff-cancel-right'*:

assumes $a \in U$ **and** $b \in U$

shows $a +_{ow} b -_{ow} b = a$

is *cancel-ab-semigroup-add-class.add-diff-cancel-right'**<proof>*

tts-lemma *add-diff-cancel-right*:

assumes $a \in U$ **and** $c \in U$ **and** $b \in U$

shows $a +_{ow} c -_{ow} (b +_{ow} c) = a -_{ow} b$

is *cancel-ab-semigroup-add-class.add-diff-cancel-right**<proof>*

tts-lemma *add-diff-cancel-left*:

assumes $c \in U$ **and** $a \in U$ **and** $b \in U$

shows $c +_{ow} a -_{ow} (c +_{ow} b) = a -_{ow} b$

is *cancel-ab-semigroup-add-class.add-diff-cancel-left**<proof>*

tts-lemma *diff-right-commute*:

assumes $a \in U$ **and** $c \in U$ **and** $b \in U$

shows $a -_{ow} c -_{ow} b = a -_{ow} b -_{ow} c$

is *cancel-ab-semigroup-add-class.diff-right-commute**<proof>*

tts-lemma *diff-diff-eq*:

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$

shows $a -_{ow} b -_{ow} c = a -_{ow} (b +_{ow} c)$

is *diff-diff-eq**<proof>*

end

end

3.8 Relativization of the results about monoids

3.8.1 Simple monoids

Definitions and common properties

locale *neutral-ow* =

fixes $U :: 'ag\ set$ and $z :: 'ag\ (\langle 1_{ow} \rangle)$

assumes $z\text{-closed}[simp]: 1_{ow} \in U$

begin

notation $z\ (\langle 1_{ow} \rangle)$

tts-register-sbts $\langle 1_{ow} \rangle \mid U\ \langle proof \rangle$

lemma *not-empty*[simp]: $U \neq \{\}$ $\langle proof \rangle$

lemma *neutral-map*: $(\lambda y. 1_{ow}) \cdot A \subseteq U\ \langle proof \rangle$

end

locale *monoid-ow* = *semigroup-ow* $U\ f$ + *neutral-ow* $U\ z$

for $U :: 'ag\ set$ and $f\ z$ +

assumes *left-neutral-mow*[simp]: $a \in U \implies (1_{ow} *_{ow} a) = a$

and *right-neutral-mow*[simp]: $a \in U \implies (a *_{ow} 1_{ow}) = a$

locale *zero-ow* = *zero*: *neutral-ow* $U\ zero$

for $U :: 'ag\ set$ and $zero :: 'ag\ (\langle 0_{ow} \rangle)$

begin

notation $zero\ (\langle 0_{ow} \rangle)$

lemma *zero-closed*: $0_{ow} \in U\ \langle proof \rangle$

end

lemma *monoid-ow*: *monoid* = *monoid-ow* *UNIV*

$\langle proof \rangle$

locale *one-ow* = *one*: *neutral-ow* $U\ one$

for $U :: 'ag\ set$ and $one :: 'ag\ (\langle 1_{ow} \rangle)$

begin

notation $one\ (\langle 1_{ow} \rangle)$

lemma *one-closed*: $1_{ow} \in U\ \langle proof \rangle$

end

locale *power-ow* = *one-ow* $U\ one$ + *times-ow* $U\ times$

for $U :: 'ag\ set$ and $one :: 'ag\ (\langle 1_{ow} \rangle)$ and *times* (**infixl** $\langle *_{ow} \rangle\ 70$)

primrec *power-with* :: $['a, ['a, 'a] \Rightarrow 'a, 'a, nat] \Rightarrow 'a$

$(\langle (/with\ -\ -\ -\ \hat{\ }_{ow}\ -/)\rangle [1000, 999, 1000, 1000] 10)$

where

power-0: *power-with* *one* *times* $a\ 0 = one$ **for** *one* *times*

| *power-Suc*: *power-with* *one* *times* $a\ (Suc\ n) =$

times $a\ (power-with\ one\ times\ a\ n)$ **for** *one* *times*

lemma *power-with*[*ud-with*]: $power = power-with\ 1\ (*)$
 ⟨*proof*⟩

context *power-ow*
begin

abbreviation *power* ⟨ $(-\ \widehat{\ }_{ow}\ -)$ ⟩ [81, 80] 80 **where**
power $\equiv power-with\ 1_{ow}\ (*_{ow})$

end

locale *monoid-add-ow* =
semigroup-add-ow *U plus* + *zero-ow* *U zero* **for** *U* :: '*ag set* **and** *plus zero* +
assumes *add-0-left*: $a \in U \implies (0_{ow} +_{ow} a) = a$
assumes *add-0-right*: $a \in U \implies (a +_{ow} 0_{ow}) = a$
begin

sublocale *add*: *monoid-ow* *U* ⟨ $(+_{ow})$ ⟩ ⟨ 0_{ow} ⟩
 ⟨*proof*⟩

end

lemma *monoid-add-ow*: *class.monoid-add* = *monoid-add-ow* *UNIV*
 ⟨*proof*⟩

locale *monoid-mult-ow* = *semigroup-mult-ow* *U times* + *one-ow* *U one*
for *U* :: '*ag set* **and** *one times* +
assumes *mult-1-left*: $a \in U \implies (1_{ow} *_{ow} a) = a$
assumes *mult-1-right*: $a \in U \implies (a *_{ow} 1_{ow}) = a$
begin

sublocale *mult*: *monoid-ow* *U* ⟨ $(*_{ow})$ ⟩ ⟨ 1_{ow} ⟩
 ⟨*proof*⟩

sublocale *power-ow* ⟨*proof*⟩

end

lemma *monoid-mult-ow*: *class.monoid-mult* = *monoid-mult-ow* *UNIV*
 ⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *monoid-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
 $((A \implies A \implies A) \implies A \implies (=))$
 (*monoid-ow* (*Collect* (*Domainp* *A*))) *monoid*
 ⟨*proof*⟩

lemma *monoid-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
 $((A \implies A \implies A) \implies A \implies (=))$

(*monoid-add-ow* (*Collect* (*Domainp* *A*))) *class.monoid-add*
 {proof}

lemma *monoid-mult-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
 (*A* \implies (*A* \implies *A* \implies *A*) \implies (=))
 (*monoid-mult-ow* (*Collect* (*Domainp* *A*))) *class.monoid-mult*
 {proof}

lemma *power-with-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
 (*A* \implies (*A* \implies *A* \implies *A*) \implies *A* \implies (=) \implies *A*) *power-with* *power-with*
 {proof}

end

Relativization

context *power-ow*
begin

tts-context
tts: (*?a* **to** *U*)
sbterms: ($\langle (*):[?a::power, ?a::power] \Rightarrow ?a::power \rangle$ **to** $\langle (*_{ow}) \rangle$)
and ($\langle 1::?a::power \rangle$ **to** $\langle 1_{ow} \rangle$)
rewriting *ctr-simps*
substituting *power-ow-axioms* **and** *one.not-empty*
begin

tts-lemma *power-Suc*:
assumes $a \in U$
shows $a \widehat{_{ow}} Suc\ n = a *_{ow} a \widehat{_{ow}} n$
is *power-class.power.power-Suc*{proof}

tts-lemma *power-0*:
assumes $a \in U$
shows $a \widehat{_{ow}} 0 = 1_{ow}$
is *power-class.power.power-0*{proof}

tts-lemma *power-eq-if*:
assumes $p \in U$
shows $p \widehat{_{ow}} m = (\text{if } m = 0 \text{ then } 1_{ow} \text{ else } p *_{ow} p \widehat{_{ow}} (m - 1))$
is *power-class.power-eq-if*{proof}

tts-lemma *simps*:
assumes $a \in U$
shows $a \widehat{_{ow}} 0 = 1_{ow}$
is *power-class.power.simps(1)*
and $a \widehat{_{ow}} Suc\ n = a *_{ow} a \widehat{_{ow}} n$
is *power-class.power.simps(2)*{proof}

end

end

context *monoid-mult-ow*

begin

tts-context

tts: (*?a to U*)

rewriting *ctr-simps*

substituting *monoid-mult-ow-axioms and one.not-empty*

applying [*OF one-closed mult.f-closed'*]

begin

tts-lemma *power-commuting-commutes:*

assumes $x \in U$ **and** $y \in U$ **and** $x *_{ow} y = y *_{ow} x$

shows $x \hat{\ }_{ow} n *_{ow} y = y *_{ow} x \hat{\ }_{ow} n$

is *monoid-mult-class.power-commuting-commutes*(*proof*)

tts-lemma *left-right-inverse-power:*

assumes $x \in U$ **and** $y \in U$ **and** $x *_{ow} y = 1_{ow}$

shows $x \hat{\ }_{ow} n *_{ow} y \hat{\ }_{ow} n = 1_{ow}$

is *monoid-mult-class.left-right-inverse-power*(*proof*)

tts-lemma *power-numeral-even:*

assumes $z \in U$

shows $z \hat{\ }_{ow} \text{numeral } (\text{num.Bit0 } w) = (\text{let } w = z \hat{\ }_{ow} \text{numeral } w \text{ in } w *_{ow} w)$

is *monoid-mult-class.power-numeral-even*(*proof*)

tts-lemma *power-numeral-odd:*

assumes $z \in U$

shows $z \hat{\ }_{ow} \text{numeral } (\text{num.Bit1 } w) = (\text{let } w = z \hat{\ }_{ow} \text{numeral } w \text{ in } z *_{ow} w *_{ow} w)$

is *monoid-mult-class.power-numeral-odd*(*proof*)

tts-lemma *power-minus-mult:*

assumes $a \in U$ **and** $0 < n$

shows $a \hat{\ }_{ow} (n - 1) *_{ow} a = a \hat{\ }_{ow} n$

is *monoid-mult-class.power-minus-mult*(*proof*)

tts-lemma *power-Suc0-right:*

assumes $a \in U$

shows $a \hat{\ }_{ow} \text{Suc } 0 = a$

is *monoid-mult-class.power-Suc0-right*(*proof*)

tts-lemma *power2-eq-square:*

assumes $a \in U$

shows $a \hat{\ }_{ow} 2 = a *_{ow} a$

is *monoid-mult-class.power2-eq-square*(*proof*)

tts-lemma *power-one-right:*

assumes $a \in U$

shows $a \hat{\ }_{ow} 1 = a$

is *monoid-mult-class.power-one-right*(*proof*)

tts-lemma *power-commutes:*

assumes $a \in U$

shows $a \hat{\ }_{ow} n *_{ow} a = a *_{ow} a \hat{\ }_{ow} n$

is *monoid-mult-class.power-commutes*(*proof*)

tts-lemma *power3-eq-cube:*

assumes $a \in U$

shows $a \hat{\ }_{ow} 3 = a *_{ow} a *_{ow} a$

```

    is monoid-mult-class.power3-eq-cube{proof}

tts-lemma power-even-eq:
  assumes a ∈ U
  shows a  $\hat{\ }_{ow}$  (2 * n) = (a  $\hat{\ }_{ow}$  n)  $\hat{\ }_{ow}$  2
  is monoid-mult-class.power-even-eq{proof}

tts-lemma power-odd-eq:
  assumes a ∈ U
  shows a  $\hat{\ }_{ow}$  Suc (2 * n) = a *ow (a  $\hat{\ }_{ow}$  n)  $\hat{\ }_{ow}$  2
  is monoid-mult-class.power-odd-eq{proof}

tts-lemma power-mult:
  assumes a ∈ U
  shows a  $\hat{\ }_{ow}$  (m * n) = (a  $\hat{\ }_{ow}$  m)  $\hat{\ }_{ow}$  n
  is monoid-mult-class.power-mult{proof}

tts-lemma power-Suc2:
  assumes a ∈ U
  shows a  $\hat{\ }_{ow}$  Suc n = a  $\hat{\ }_{ow}$  n *ow a
  is monoid-mult-class.power-Suc2{proof}

tts-lemma power-one: 1ow  $\hat{\ }_{ow}$  n = 1ow
  is monoid-mult-class.power-one{proof}

tts-lemma power-add:
  assumes a ∈ U
  shows a  $\hat{\ }_{ow}$  (m + n) = a  $\hat{\ }_{ow}$  m *ow a  $\hat{\ }_{ow}$  n
  is monoid-mult-class.power-add{proof}

tts-lemma power-mult-numeral:
  assumes a ∈ U
  shows (a  $\hat{\ }_{ow}$  numeral m)  $\hat{\ }_{ow}$  numeral n = a  $\hat{\ }_{ow}$  numeral (m * n)
  is Power.power-mult-numeral{proof}

tts-lemma power-add-numeral2:
  assumes a ∈ U and b ∈ U
  shows
    a  $\hat{\ }_{ow}$  numeral m *ow (a  $\hat{\ }_{ow}$  numeral n *ow b) = a  $\hat{\ }_{ow}$  numeral (m + n) *ow b
  is Power.power-add-numeral2{proof}

tts-lemma power-add-numeral:
  assumes a ∈ U
  shows a  $\hat{\ }_{ow}$  numeral m *ow a  $\hat{\ }_{ow}$  numeral n = a  $\hat{\ }_{ow}$  numeral (m + n)
  is Power.power-add-numeral{proof}

end

end

```

3.8.2 Commutative monoids

Definitions and common properties

```

locale comm-monoid-ow =
  abel-semigroup-ow U f + neutral-ow U z for U :: 'ag set and f z +
  assumes comm-neutral: a ∈ U  $\implies$  (a *ow 1ow) = a
begin

```

sublocale *monoid-ow* $U \langle (*_{ow}) \rangle \langle \mathbf{1}_{ow} \rangle$
 ⟨*proof*⟩

end

lemma *comm-monoid-ow*: *comm-monoid* = *comm-monoid-ow* UNIV
 ⟨*proof*⟩

locale *comm-monoid-set-ow* = *comm-monoid-ow* U *fz* **for** $U :: 'ag\ set$ **and** fz
begin

tts-register-sbts $\langle (*_{ow}) \rangle \mid U$ ⟨*proof*⟩

end

lemma *comm-monoid-set-ow*: *comm-monoid-set* = *comm-monoid-set-ow* UNIV
 ⟨*proof*⟩

locale *comm-monoid-add-ow* =
ab-semigroup-add-ow U *plus* + *zero-ow* U *zero*
for $U :: 'ag\ set$ **and** *plus* *zero* +
assumes *add-0[simp]*: $a \in U \implies 0_{ow} +_{ow} a = a$
begin

sublocale *add*: *comm-monoid-ow* $U \langle (+_{ow}) \rangle \langle 0_{ow} \rangle$
 ⟨*proof*⟩

sublocale *monoid-add-ow* $U \langle (+_{ow}) \rangle \langle 0_{ow} \rangle$ ⟨*proof*⟩

sublocale *sum*: *comm-monoid-set-ow* $U \langle (+_{ow}) \rangle \langle 0_{ow} \rangle$ ⟨*proof*⟩

notation *sum.F* (⟨⟨*sum*⟩⟩)

abbreviation *Sum* ($\langle \sum_{ow} / - \rangle [1000] 1000$)
where $\sum_{ow} A \equiv (\langle \langle \textit{sum} \rangle \rangle (\lambda x. x) A)$

notation *Sum* ($\langle \sum_{ow} / - \rangle [1000] 1000$)

end

lemma *comm-monoid-add-ow*: *class.comm-monoid-add* = *comm-monoid-add-ow* UNIV
 ⟨*proof*⟩

locale *dvd-ow* = *times-ow* U *times*
for $U :: 'ag\ set$ **and** *times*

ud $\langle dvd.dvd \rangle$

ud *dvd'* $\langle dvd-class.dvd \rangle$

ctr relativization

synthesis *ctr-simps*

assumes [*transfer-domain-rule*, *transfer-rule*]: *Domainp* $A = (\lambda x. x \in U)$
and [*transfer-rule*]: *bi-unique* A *right-total* A

trp ($?a$ A)

in *dvd-ow'*: *dvd.with-def*

($\langle (on - with \:- \langle \langle \textit{dvd} \rangle \rangle -) \rangle [1000, 1000, 1000, 1000] 50$)

ctr parametricity

in $dvd-ow''$: $dvd-ow'-def$

context $dvd-ow$

begin

abbreviation dvd (**infixr** $\langle\langle dvd \rangle\rangle$ 50) **where** $a \langle\langle dvd \rangle\rangle b \equiv dvd-ow' U (*_{ow}) a b$

notation dvd (**infixr** $\langle\langle dvd \rangle\rangle$ 50)

end

locale $comm-monoid-mult-ow =$

$ab-semigroup-mult-ow U times + one-ow U one$

for $U :: 'ag set$ **and** $times one +$

assumes $mult-1[simp]$: $a \in U \implies 1_{ow} *_{ow} a = a$

begin

sublocale $dvd-ow \langle proof \rangle$

sublocale $mult$: $comm-monoid-ow U \langle (*_{ow}) \rangle \langle 1_{ow} \rangle$
 $\langle proof \rangle$

sublocale $monoid-mult-ow U \langle 1_{ow} \rangle \langle (*_{ow}) \rangle \langle proof \rangle$

sublocale $prod$: $comm-monoid-set-ow U \langle (*_{ow}) \rangle \langle 1_{ow} \rangle \langle proof \rangle$

notation $prod.F$ ($\langle\langle prod \rangle\rangle$)

abbreviation $Prod$ ($\langle\prod_{ow} \rightarrow [1000] 1000$)

where $\prod_{ow} A \equiv (\langle\langle prod \rangle\rangle (\lambda x. x) A)$

notation $Prod$ ($\langle\prod_{ow} \rightarrow [1000] 1000$)

end

lemma $comm-monoid-mult-ow$: $class.comm-monoid-mult = comm-monoid-mult-ow UNIV$
 $\langle proof \rangle$

Transfer rules

context

includes $lifting-syntax$

begin

lemma $bij-betw-transfer[transfer-rule]$:

assumes $[transfer-rule]$:

$bi-unique A right-total A bi-unique B right-total B$

shows

$((A \implies B) \implies rel-set A \implies rel-set B \implies (=))$ $bij-betw$ $bij-betw$

$\langle proof \rangle$

lemma $comm-monoid-transfer[transfer-rule]$:

assumes $[transfer-rule]$: $bi-unique A right-total A$

shows

$((A \implies A \implies A) \implies A \implies (=))$

$(comm-monoid-ow (Collect (Domainp A)))$ $comm-monoid$

$\langle proof \rangle$

lemma *comm-monoid-set-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 $((A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } (=))$
(*comm-monoid-set-ow (Collect (Domainp A))*) *comm-monoid-set*
 $\langle \text{proof} \rangle$

lemma *comm-monoid-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 $((A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } (=))$
(*comm-monoid-add-ow (Collect (Domainp A))*) *class.comm-monoid-add*
 $\langle \text{proof} \rangle$

lemma *comm-monoid-mult-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 $((A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } (=))$
(*comm-monoid-mult-ow (Collect (Domainp A))*) *class.comm-monoid-mult*
 $\langle \text{proof} \rangle$

lemma *dvd-with-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 $((A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } A \text{ ===> } (=))$
(*dvd-ow' (Collect (Domainp A))*) *dvd.with*
 $\langle \text{proof} \rangle$

end

Relativization

context *dvd-ow*

begin

tts-context

tts: (*?a to U*)

sbterms: $\langle (*):[?a::times, ?a::times] \Rightarrow ?a::times \rangle$ **to** $\langle (*_{ow}) \rangle$

rewriting *ctr-simps*

substituting *dvd-ow-axioms*

eliminating through *simp*

begin

tts-lemma *dvdI*:

assumes $b \in U$ **and** $k \in U$ **and** $a = b *_{ow} k$

shows $b \ll \text{dvd} \gg a$

is *dvd-class.dvdI* $\langle \text{proof} \rangle$

tts-lemma *dvdE*:

assumes $b \in U$

and $a \in U$

and $b \ll \text{dvd} \gg a$

and $\bigwedge k. [[k \in U; a = b *_{ow} k]] \implies P$

shows P

is *dvd-class.dvdE* $\langle \text{proof} \rangle$

end

end

context *comm-monoid-mult-ow*
begin

tts-context

tts: (?*a* to *U*)

rewriting *ctr-simps*

substituting *comm-monoid-mult-ow-axioms* and *one.not-empty*

applying [*OF mult.f-closed' one-closed*]

begin

tts-lemma *strict-subset-divisors-dvd*:

assumes $a \in U$ and $b \in U$

shows

$(\{x \in U. x \ll \text{dvd} \gg a\} \subseteq \{x \in U. x \ll \text{dvd} \gg b\}) = (a \ll \text{dvd} \gg b \wedge \neg b \ll \text{dvd} \gg a)$

is *comm-monoid-mult-class.strict-subset-divisors-dvd*{proof}

tts-lemma *subset-divisors-dvd*:

assumes $a \in U$ and $b \in U$

shows $(\{x \in U. x \ll \text{dvd} \gg a\} \subseteq \{x \in U. x \ll \text{dvd} \gg b\}) = (a \ll \text{dvd} \gg b)$

is *comm-monoid-mult-class.subset-divisors-dvd*{proof}

tts-lemma *power-mult-distrib*:

assumes $a \in U$ and $b \in U$

shows $(a *_{ow} b) \hat{\ }_{ow} n = a \hat{\ }_{ow} n *_{ow} b \hat{\ }_{ow} n$

is *Power.comm-monoid-mult-class.power-mult-distrib*{proof}

tts-lemma *dvd-triv-right*:

assumes $a \in U$ and $b \in U$

shows $a \ll \text{dvd} \gg b *_{ow} a$

is *comm-monoid-mult-class.dvd-triv-right*{proof}

tts-lemma *dvd-mult-right*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $a *_{ow} b \ll \text{dvd} \gg c$

shows $b \ll \text{dvd} \gg c$

is *comm-monoid-mult-class.dvd-mult-right*{proof}

tts-lemma *mult-dvd-mono*:

assumes $a \in U$

and $b \in U$

and $c \in U$

and $d \in U$

and $a \ll \text{dvd} \gg b$

and $c \ll \text{dvd} \gg d$

shows $a *_{ow} c \ll \text{dvd} \gg b *_{ow} d$

is *comm-monoid-mult-class.mult-dvd-mono*{proof}

tts-lemma *dvd-triv-left*:

assumes $a \in U$ and $b \in U$

shows $a \ll \text{dvd} \gg a *_{ow} b$

is *comm-monoid-mult-class.dvd-triv-left*{proof}

tts-lemma *dvd-mult-left*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $a *_{ow} b \ll \text{dvd} \gg c$

shows $a \ll \text{dvd} \gg c$

is *comm-monoid-mult-class.dvd-mult-left*{proof}

```

tts-lemma dvd-trans:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$  and  $a \ll \text{dvd} \gg b$  and  $b \ll \text{dvd} \gg c$ 
  shows  $a \ll \text{dvd} \gg c$ 
  is comm-monoid-mult-class.dvd-trans{proof}

tts-lemma dvd-mult2:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$  and  $a \ll \text{dvd} \gg b$ 
  shows  $a \ll \text{dvd} \gg b *_{ow} c$ 
  is comm-monoid-mult-class.dvd-mult2{proof}

tts-lemma dvd-refl:
  assumes  $a \in U$ 
  shows  $a \ll \text{dvd} \gg a$ 
  is comm-monoid-mult-class.dvd-refl{proof}

tts-lemma dvd-mult:
  assumes  $a \in U$  and  $c \in U$  and  $b \in U$  and  $a \ll \text{dvd} \gg c$ 
  shows  $a \ll \text{dvd} \gg b *_{ow} c$ 
  is comm-monoid-mult-class.dvd-mult{proof}

tts-lemma one-dvd:
  assumes  $a \in U$ 
  shows  $1_{ow} \ll \text{dvd} \gg a$ 
  is comm-monoid-mult-class.one-dvd{proof}

end

end

```

3.8.3 Cancellative commutative monoids

Definitions and common properties

```

locale cancel-comm-monoid-add-ow =
  cancel-ab-semigroup-add-ow  $U$  plus minus +
  comm-monoid-add-ow  $U$  plus zero
  for  $U :: 'a$  g set and plus minus zero

```

```

lemma cancel-comm-monoid-add-ow:
  class.cancel-comm-monoid-add = cancel-comm-monoid-add-ow UNIV
  {proof}

```

Transfer rules

```

context
  includes lifting-syntax
begin

lemma cancel-comm-monoid-add-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique  $A$  right-total  $A$ 
  shows
    ( $(A \text{ ===> } A \text{ ===> } A) \text{ ===> } (A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } (=)$ )
    (cancel-comm-monoid-add-ow (Collect (Domainp  $A$ )))
    class.cancel-comm-monoid-add
  {proof}

end

```

Relativization

```

context cancel-comm-monoid-add-ow
begin

tts-context
  tts: (?a to U)
  rewriting ctr-simps
  substituting cancel-comm-monoid-add-ow-axioms and zero.not-empty
  applying [OF add.f-closed' minus-closed' zero-closed]
begin

tts-lemma add-cancel-right-right:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $(a = a +_{ow} b) = (b = 0_{ow})$ 
  is cancel-comm-monoid-add-class.add-cancel-right-right(proof)

tts-lemma add-cancel-right-left:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $(a = b +_{ow} a) = (b = 0_{ow})$ 
  is cancel-comm-monoid-add-class.add-cancel-right-left(proof)

tts-lemma add-cancel-left-right:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $(a +_{ow} b = a) = (b = 0_{ow})$ 
  is cancel-comm-monoid-add-class.add-cancel-left-right(proof)

tts-lemma add-cancel-left-left:
  assumes  $b \in U$  and  $a \in U$ 
  shows  $(b +_{ow} a = a) = (b = 0_{ow})$ 
  is cancel-comm-monoid-add-class.add-cancel-left-left(proof)

tts-lemma add-implies-diff:
  assumes  $c \in U$  and  $b \in U$  and  $a \in U$  and  $c +_{ow} b = a$ 
  shows  $c = a -_{ow} b$ 
  is cancel-comm-monoid-add-class.add-implies-diff(proof)

tts-lemma diff-cancel:
  assumes  $a \in U$ 
  shows  $a -_{ow} a = 0_{ow}$ 
  is cancel-comm-monoid-add-class.diff-cancel(proof)

tts-lemma diff-zero:
  assumes  $a \in U$ 
  shows  $a -_{ow} 0_{ow} = a$ 
  is cancel-comm-monoid-add-class.diff-zero(proof)

end

end

```

3.9 Relativization of the results about groups

3.9.1 Simple groups

Definitions and common properties

locale *group-ow* = *semigroup-ow* *U f* for *U* :: 'ag set and *f* +
fixes *z* ($\langle \mathbf{1}_{ow} \rangle$)
and *inverse* :: 'ag \Rightarrow 'ag
assumes *z-closed*[*simp*]: $\mathbf{1}_{ow} \in U$
and *inverse-closed*[*simp*]: $a \in U \Longrightarrow \text{inverse } a \in U$
and *group-left-neutral*: $a \in U \Longrightarrow \mathbf{1}_{ow} *_{ow} a = a$
and *left-inverse*[*simp*]: $a \in U \Longrightarrow \text{inverse } a *_{ow} a = \mathbf{1}_{ow}$
begin

notation *z* ($\langle \mathbf{1}_{ow} \rangle$)

lemma *inverse-closed'*: *inverse* ' $U \subseteq U$ \langle proof \rangle

lemma *inverse-closed''*: $\forall x \in U. \text{inverse } x \in U$ \langle proof \rangle

lemma *left-cancel*:

assumes $a \in U$ and $b \in U$ and $c \in U$

shows $a *_{ow} b = a *_{ow} c \longleftrightarrow b = c$

\langle proof \rangle

sublocale *monoid-ow* U \langle $\langle *_{ow} \rangle$ \rangle $\langle \mathbf{1}_{ow} \rangle$

\langle proof \rangle

lemma *inverse-image*[*simp*]: *inverse* ' $U \subseteq U$ \langle proof \rangle

end

lemma *group-ow*: *group* = *group-ow* *UNIV*

\langle proof \rangle

locale *uminus-ow* =

fixes *U* :: 'ag set and *uminus* :: 'ag \Rightarrow 'ag ($\langle -_{ow} \rightarrow$ [81] 80)

assumes *uminus-closed*: $a \in U \Longrightarrow -_{ow} a \in U$

begin

notation *uminus* ($\langle -_{ow} \rightarrow$ [81] 80)

lemma *uminus-closed'*: *uminus* ' $U \subseteq U$ \langle proof \rangle

lemma *uminus-closed''*: $\forall a \in U. -_{ow} a \in U$ \langle proof \rangle

tts-register-sbts *uminus* | U \langle proof \rangle

end

locale *group-add-ow* =

minus-ow U *minus* + *uminus-ow* U *uminus* + *monoid-add-ow* U *plus zero*

for *U* :: 'ag set and *minus plus zero* *uminus* +

assumes *left-inverse*: $a \in U \Longrightarrow (-_{ow} a) +_{ow} a = \mathbf{0}_{ow}$

and *add-inv-conv-diff*: $[[a \in U; b \in U]] \Longrightarrow a +_{ow} (-_{ow} b) = a -_{ow} b$

begin

sublocale *add*: *group-ow* U \langle $\langle +_{ow} \rangle$ \rangle $\langle \mathbf{0}_{ow} \rangle$ *uminus*

\langle proof \rangle

lemma *inverse-unique*:

assumes $a \in U$ **and** $b \in U$ **and** $a +_{ow} b = 0_{ow}$
shows $-_{ow} a = b$

<proof>

lemma *inverse-neutral[simp]*: $-_{ow} 0_{ow} = 0_{ow}$

<proof>

lemma *inverse-inverse*:

assumes $a \in U$

shows $-_{ow} (-_{ow} a) = a$

<proof>

lemma *right-inverse*:

assumes $a \in U$

shows $a +_{ow} (-_{ow} a) = 0_{ow}$

<proof>

sublocale *cancel-semigroup-add-ow* $U \langle (+_{ow}) \rangle$

<proof>

end

lemma *group-add-ow*: $class.group-add = group-add-ow UNIV$

<proof>

Transfer rules

context

includes *lifting-syntax*

begin

lemma *group-transfer[transfer-rule]*:

assumes [*transfer-rule*]: *bi-unique* A *right-total* A

shows $((A \implies A \implies A) \implies A \implies (A \implies A) \implies (=))$

$(group-ow (Collect (Domainp A))) group$

<proof>

lemma *group-add-transfer[transfer-rule]*:

assumes [*transfer-rule*]: *bi-unique* A *right-total* A

shows

$((A \implies A \implies A) \implies (A \implies A \implies A) \implies A \implies (A \implies A) \implies (=))$

$(group-add-ow (Collect (Domainp A))) class.group-add$

<proof>

end

Relativization

context *group-ow*

begin

tts-context

tts: (*?a to* U)

rewriting *ctr-simps*

substituting *group-ow-axioms* **and** *not-empty*

applying [*OF f-closed' z-closed inverse-closed''*]

begin

```

tts-lemma inverse-neutral: inverse  $\mathbf{1}_{ow} = \mathbf{1}_{ow}$ 
  is group.inverse-neutral{proof}

tts-lemma inverse-inverse:
  assumes  $a \in U$ 
  shows inverse (inverse  $a$ ) =  $a$ 
  is group.inverse-inverse{proof}

tts-lemma right-inverse:
  assumes  $a \in U$ 
  shows  $a *_{ow} \textit{inverse } a = \mathbf{1}_{ow}$ 
  is group.right-inverse{proof}

tts-lemma inverse-distrib-swap:
  assumes  $a \in U$  and  $b \in U$ 
  shows inverse ( $a *_{ow} b$ ) = inverse  $b *_{ow} \textit{inverse } a$ 
  is group.inverse-distrib-swap{proof}

tts-lemma right-cancel:
  assumes  $b \in U$  and  $a \in U$  and  $c \in U$ 
  shows ( $b *_{ow} a = c *_{ow} a$ ) = ( $b = c$ )
  is group.right-cancel{proof}

tts-lemma inverse-unique:
  assumes  $a \in U$  and  $b \in U$  and  $a *_{ow} b = \mathbf{1}_{ow}$ 
  shows inverse  $a = b$ 
  is group.inverse-unique{proof}
end

end

context group-add-ow
begin

tts-context
  tts: (?a to  $U$ )
  rewriting ctr-simps
  substituting group-add-ow-axioms and zero.not-empty
  applying [OF minus-closed' plus-closed' zero-closed add.inverse-closed']
begin

tts-lemma diff-0:
  assumes  $a \in U$ 
  shows  $0_{ow} -_{ow} a = -_{ow} a$ 
  is group-add-class.diff-0{proof}

tts-lemma diff-0-right:
  assumes  $a \in U$ 
  shows  $a -_{ow} 0_{ow} = a$ 
  is group-add-class.diff-0-right{proof}

tts-lemma diff-self:
  assumes  $a \in U$ 
  shows  $a -_{ow} a = 0_{ow}$ 
  is group-add-class.diff-self{proof}

tts-lemma group-left-neutral:

```

assumes $a \in U$
shows $0_{ow} +_{ow} a = a$
is *group-add-class.add.group-left-neutral*(proof)

tts-lemma *minus-minus*:
assumes $a \in U$
shows $-_{ow} (-_{ow} a) = a$
is *group-add-class.minus-minus*(proof)

tts-lemma *right-minus*:
assumes $a \in U$
shows $a +_{ow} -_{ow} a = 0_{ow}$
is *group-add-class.right-minus*(proof)

tts-lemma *left-minus*:
assumes $a \in U$
shows $-_{ow} a +_{ow} a = 0_{ow}$
is *group-add-class.left-minus*(proof)

tts-lemma *add-diff-cancel*:
assumes $a \in U$ **and** $b \in U$
shows $a +_{ow} b -_{ow} b = a$
is *group-add-class.add-diff-cancel*(proof)

tts-lemma *diff-add-cancel*:
assumes $a \in U$ **and** $b \in U$
shows $a -_{ow} b +_{ow} b = a$
is *group-add-class.diff-add-cancel*(proof)

tts-lemma *diff-conv-add-uminus*:
assumes $a \in U$ **and** $b \in U$
shows $a -_{ow} b = a +_{ow} -_{ow} b$
is *group-add-class.diff-conv-add-uminus*(proof)

tts-lemma *diff-minus-eq-add*:
assumes $a \in U$ **and** $b \in U$
shows $a -_{ow} -_{ow} b = a +_{ow} b$
is *group-add-class.diff-minus-eq-add*(proof)

tts-lemma *add-uminus-conv-diff*:
assumes $a \in U$ **and** $b \in U$
shows $a +_{ow} -_{ow} b = a -_{ow} b$
is *group-add-class.add-uminus-conv-diff*(proof)

tts-lemma *minus-diff-eq*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} (a -_{ow} b) = b -_{ow} a$
is *group-add-class.minus-diff-eq*(proof)

tts-lemma *add-minus-cancel*:
assumes $a \in U$ **and** $b \in U$
shows $a +_{ow} (-_{ow} a +_{ow} b) = b$
is *group-add-class.add-minus-cancel*(proof)

tts-lemma *minus-add-cancel*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} a +_{ow} (a +_{ow} b) = b$
is *group-add-class.minus-add-cancel*(proof)

tts-lemma *neg-0-equal-iff-equal*:
assumes $a \in U$
shows $(0_{ow} = -_{ow} a) = (0_{ow} = a)$
is *group-add-class.neg-0-equal-iff-equal*{proof}

tts-lemma *neg-equal-0-iff-equal*:
assumes $a \in U$
shows $(-_{ow} a = 0_{ow}) = (a = 0_{ow})$
is *group-add-class.neg-equal-0-iff-equal*{proof}

tts-lemma *eq-iff-diff-eq-0*:
assumes $a \in U$ **and** $b \in U$
shows $(a = b) = (a -_{ow} b = 0_{ow})$
is *group-add-class.eq-iff-diff-eq-0*{proof}

tts-lemma *equation-minus-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(a = -_{ow} b) = (b = -_{ow} a)$
is *group-add-class.equation-minus-iff*{proof}

tts-lemma *minus-equation-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(-_{ow} a = b) = (-_{ow} b = a)$
is *group-add-class.minus-equation-iff*{proof}

tts-lemma *neg-equal-iff-equal*:
assumes $a \in U$ **and** $b \in U$
shows $(-_{ow} a = -_{ow} b) = (a = b)$
is *group-add-class.neg-equal-iff-equal*{proof}

tts-lemma *right-minus-eq*:
assumes $a \in U$ **and** $b \in U$
shows $(a -_{ow} b = 0_{ow}) = (a = b)$
is *group-add-class.right-minus-eq*{proof}

tts-lemma *minus-add*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} (a +_{ow} b) = -_{ow} b +_{ow} -_{ow} a$
is *group-add-class.minus-add*{proof}

tts-lemma *eq-neg-iff-add-eq-0*:
assumes $a \in U$ **and** $b \in U$
shows $(a = -_{ow} b) = (a +_{ow} b = 0_{ow})$
is *group-add-class.eq-neg-iff-add-eq-0*{proof}

tts-lemma *neg-eq-iff-add-eq-0*:
assumes $a \in U$ **and** $b \in U$
shows $(-_{ow} a = b) = (a +_{ow} b = 0_{ow})$
is *group-add-class.neg-eq-iff-add-eq-0*{proof}

tts-lemma *add-eq-0-iff2*:
assumes $a \in U$ **and** $b \in U$
shows $(a +_{ow} b = 0_{ow}) = (a = -_{ow} b)$
is *group-add-class.add-eq-0-iff2*{proof}

tts-lemma *add-eq-0-iff*:
assumes $a \in U$ **and** $b \in U$

shows $(a +_{ow} b = 0_{ow}) = (b = -_{ow} a)$
is *group-add-class.add-eq-0-iff*{proof}

tts-lemma *diff-diff-eq2*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a -_{ow} (b -_{ow} c) = a +_{ow} c -_{ow} b$
is *group-add-class.diff-diff-eq2*{proof}

tts-lemma *diff-add-eq-diff-diff-swap*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a -_{ow} (b +_{ow} c) = a -_{ow} c -_{ow} b$
is *group-add-class.diff-add-eq-diff-diff-swap*{proof}

tts-lemma *add-diff-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a +_{ow} (b -_{ow} c) = a +_{ow} b -_{ow} c$
is *group-add-class.add-diff-eq*{proof}

tts-lemma *eq-diff-eq*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$
shows $(a = c -_{ow} b) = (a +_{ow} b = c)$
is *group-add-class.eq-diff-eq*{proof}

tts-lemma *diff-eq-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a -_{ow} b = c) = (a = c +_{ow} b)$
is *group-add-class.diff-eq-eq*{proof}

tts-lemma *left-cancel*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a +_{ow} b = a +_{ow} c) = (b = c)$
is *group-add-class.add.left-cancel*{proof}

tts-lemma *right-cancel*:
assumes $b \in U$ **and** $a \in U$ **and** $c \in U$
shows $(b +_{ow} a = c +_{ow} a) = (b = c)$
is *group-add-class.add.right-cancel*{proof}

tts-lemma *minus-unique*:
assumes $a \in U$ **and** $b \in U$ **and** $a +_{ow} b = 0_{ow}$
shows $-_{ow} a = b$
is *group-add-class.minus-unique*{proof}

tts-lemma *diff-eq-diff-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$ **and** $a -_{ow} b = c -_{ow} d$
shows $(a = b) = (c = d)$
is *group-add-class.diff-eq-diff-eq*{proof}

end

end

3.9.2 Abelian groups

Definitions and common properties

locale *ab-group-add-ow* =

minus-ow U minus + uminus-ow U uminus + comm-monoid-add-ow U plus zero

```

for  $U :: 'ag\ set$  and  $plus\ zero\ minus\ uminus\ +$ 
assumes  $ab\text{-left}\text{-minus}: a \in U \implies -_{ow}\ a +_{ow}\ a = 0_{ow}$ 
assumes  $ab\text{-diff}\text{-conv}\text{-add}\text{-uminus}:$ 
   $[[ a \in U; b \in U ]] \implies a -_{ow}\ b = a +_{ow}\ (-_{ow}\ b)$ 
begin

sublocale  $group\text{-add}\text{-ow}$ 
   $\langle proof \rangle$ 

sublocale  $cancel\text{-comm}\text{-monoid}\text{-add}\text{-ow}$ 
   $\langle proof \rangle$ 

end

lemma  $ab\text{-group}\text{-add}\text{-ow}: class.ab\text{-group}\text{-add} = ab\text{-group}\text{-add}\text{-ow}\ UNIV$ 
   $\langle proof \rangle$ 

lemma  $ab\text{-group}\text{-add}\text{-ow}\text{-UNIV}\text{-axioms}:$ 
   $ab\text{-group}\text{-add}\text{-ow}\ (UNIV::'a::ab\text{-group}\text{-add}\ set)\ (+)\ 0\ (-)\ uminus$ 
   $\langle proof \rangle$ 

```

Transfer rules

```

context
  includes  $lifting\text{-syntax}$ 
begin

lemma  $ab\text{-group}\text{-add}\text{-transfer}[transfer\text{-rule}]:$ 
  assumes  $[transfer\text{-rule}]: bi\text{-unique}\ A\ right\text{-total}\ A$ 
  shows
     $((A \implies A \implies A) \implies A \implies (A \implies A \implies A) \implies (A \implies A) \implies (=))$ 
     $(ab\text{-group}\text{-add}\text{-ow}\ (Collect\ (Domainp\ A)))\ class.ab\text{-group}\text{-add}$ 
   $\langle proof \rangle$ 

end

```

Relativization

```

context  $ab\text{-group}\text{-add}\text{-ow}$ 
begin

tts-context
  tts:  $(?a\ to\ U)$ 
  rewriting  $ctr\text{-simps}$ 
  substituting  $ab\text{-group}\text{-add}\text{-ow}\text{-axioms}$  and  $zero.\text{not}\text{-empty}$ 
  applying  $[OF\ plus\text{-closed}'\ zero\text{-closed}\ minus\text{-closed}'\ add.\text{inverse}\text{-closed}']$ 
begin

tts-lemma  $uminus\text{-add}\text{-conv}\text{-diff}:$ 
  assumes  $a \in U$  and  $b \in U$ 
  shows  $-_{ow}\ a +_{ow}\ b = b -_{ow}\ a$ 
  is  $ab\text{-group}\text{-add}\text{-class.uminus}\text{-add}\text{-conv}\text{-diff}\langle proof \rangle$ 

tts-lemma  $diff\text{-add}\text{-eq}:$ 
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$ 
  shows  $a -_{ow}\ b +_{ow}\ c = a +_{ow}\ c -_{ow}\ b$ 
  is  $ab\text{-group}\text{-add}\text{-class.diff}\text{-add}\text{-eq}\langle proof \rangle$ 

```

end

end

3.10 Relativization of the results about semirings

3.10.1 Semirings

Definitions and common properties

locale *semiring-ow* =

ab-semigroup-add-ow *U plus* + *semigroup-mult-ow* *U times*

for *U* :: 'ag set and plus times +

assumes *distrib-right[simp]*:

$$\llbracket a \in U; b \in U; c \in U \rrbracket \Longrightarrow (a +_{ow} b) *_{ow} c = a *_{ow} c +_{ow} b *_{ow} c$$

assumes *distrib-left[simp]*:

$$\llbracket a \in U; b \in U; c \in U \rrbracket \Longrightarrow a *_{ow} (b +_{ow} c) = a *_{ow} b +_{ow} a *_{ow} c$$

lemma *semiring-ow*: *class.semiring* = *semiring-ow UNIV*

<proof>

Transfer rules

context

includes *lifting-syntax*

begin

lemma *semiring-transfer[transfer-rule]*:

assumes[*transfer-rule*]: *bi-unique A right-total A*

shows

$$\begin{aligned} ((A \text{ ===> } A \text{ ===> } A) \text{ ===> } (A \text{ ===> } A \text{ ===> } A) \text{ ===> } (=)) \\ (\text{semiring-ow } (\text{Collect } (\text{Domainp } A))) \text{ class.semiring} \end{aligned}$$

<proof>

end

Relativization

context *semiring-ow*

begin

tts-context

tts: (?*a* to *U*)

substituting *semiring-ow-axioms*

eliminating through *simp*

begin

tts-lemma *combine-common-factor*:

assumes *a* ∈ *U* and *e* ∈ *U* and *b* ∈ *U* and *c* ∈ *U*

shows $a *_{ow} e +_{ow} (b *_{ow} e +_{ow} c) = (a +_{ow} b) *_{ow} e +_{ow} c$

is *semiring-class.combine-common-factor**<proof>*

end

end

3.10.2 Commutative semirings

Definitions and common properties

locale *comm-semiring-ow* =

ab-semigroup-add-ow *U plus* + *ab-semigroup-mult-ow* *U times*

for *U* :: 'ag set and plus times +

assumes *distrib*:

$\llbracket a \in U; b \in U; c \in U \rrbracket \implies (a +_{ow} b) *_{ow} c = a *_{ow} c +_{ow} b *_{ow} c$
begin

sublocale *semiring-ow*
 ⟨*proof*⟩

end

lemma *comm-semiring-ow*: *class.comm-semiring = comm-semiring-ow UNIV*
 ⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *comm-semiring-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
 ((*A* \implies *A* \implies *A*) \implies (*A* \implies *A* \implies *A*) \implies (=))
 (*comm-semiring-ow* (*Collect* (*Domainp A*))) *class.comm-semiring*
 (**is** ?*PR* (*comm-semiring-ow* (*Collect* (*Domainp A*))) *class.comm-semiring*)
 ⟨*proof*⟩

end

3.10.3 Semirings with zero

Definitions and further results

locale *mult-zero-ow* = *times-ow U times + zero-ow U zero*
for *U* :: 'ag set **and** *times zero +*
assumes *mult-zero-left*[*simp*]: $a \in U \implies 0_{ow} *_{ow} a = 0_{ow}$
assumes *mult-zero-right*[*simp*]: $a \in U \implies a *_{ow} 0_{ow} = 0_{ow}$

lemma *mult-zero-ow*: *class.mult-zero = mult-zero-ow UNIV*
 ⟨*proof*⟩

locale *semiring-0-ow* =
semiring-ow U plus times +
comm-monoid-add-ow U plus zero +
mult-zero-ow U times zero
for *U* :: 'ag set **and** *plus zero times*

lemma *semiring-0-ow*: *class.semiring-0 = semiring-0-ow UNIV*
 ⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semiring-0-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
 ((*A* \implies *A* \implies *A*) \implies *A* \implies (*A* \implies *A* \implies *A*) \implies (=))
 (*semiring-0-ow* (*Collect* (*Domainp A*))) *class.semiring-0*

```

  (is ?PR (semiring-0-ow (Collect (Domainp A))) class.semiring-0)
  <proof>
end

```

3.10.4 Commutative semirings with zero

Definitions and common properties

```

locale comm-semiring-0-ow =
  comm-semiring-ow U plus times +
  comm-monoid-add-ow U plus zero +
  mult-zero-ow U times zero
  for U :: 'ag set and plus zero times
begin

```

```

sublocale semiring-0-ow <proof>

```

```

end

```

```

lemma comm-semiring-0-ow: class.comm-semiring-0 = comm-semiring-0-ow UNIV
  <proof>

```

Transfer rules

```

context
  includes lifting-syntax
begin

```

```

lemma comm-semiring-0-transfer[transfer-rule]:
  assumes[transfer-rule]: bi-unique A right-total A
  shows
    ((A ==> A ==> A) ==> A ==> (A ==> A ==> A) ==> (=))
    (comm-semiring-0-ow (Collect (Domainp A))) class.comm-semiring-0
    (is ?PR (comm-semiring-0-ow (Collect (Domainp A))) class.comm-semiring-0)
  <proof>

```

```

end

```

3.10.5 Cancellative semirings with zero

Definitions and common properties

```

locale semiring-0-cancel-ow =
  semiring-ow U plus times + cancel-comm-monoid-add-ow U plus minus zero
  for U :: 'ag set and plus minus zero times
begin

```

```

sublocale semiring-0-ow
  <proof>

```

```

end

```

```

lemma semiring-0-cancel-ow:
  class.semiring-0-cancel = semiring-0-cancel-ow UNIV
  <proof>

```

Transfer rules**context****includes** *lifting-syntax***begin****lemma** *semiring-0-cancel-transfer*[*transfer-rule*]:**assumes**[*transfer-rule*]: *bi-unique A right-total A***shows**

```

(
  (A ==> A ==> A) ==>
  (A ==> A ==> A) ==>
  A ==>
  (A ==> A ==> A) ==>
  (=)
) (semiring-0-cancel-ow (Collect (Domainp A))) class.semiring-0-cancel
⟨proof⟩

```

end**3.10.6 Commutative cancellative semirings with zero****Definitions and common properties****locale** *comm-semiring-0-cancel-ow* =*comm-semiring-ow U plus times +**cancel-comm-monoid-add-ow U plus minus zero***for** *U :: 'ag set and plus minus zero times***begin****sublocale** *semiring-0-cancel-ow* ⟨proof⟩**sublocale** *comm-semiring-0-ow* ⟨proof⟩**end****lemma** *comm-semiring-0-cancel-ow*:*class.comm-semiring-0-cancel = comm-semiring-0-cancel-ow UNIV*

⟨proof⟩

Transfer rules**context****includes** *lifting-syntax***begin****lemma** *comm-semiring-0-cancel-transfer*[*transfer-rule*]:**assumes**[*transfer-rule*]: *bi-unique A right-total A***shows**

```

(
  (A ==> A ==> A) ==>
  (A ==> A ==> A) ==>
  A ==>
  (A ==> A ==> A) ==>
  (=)
)
(comm-semiring-0-cancel-ow (Collect (Domainp A)))
class.comm-semiring-0-cancel
⟨proof⟩

```

end

3.10.7 Class *zero-neq-one*

Definitions and common properties

locale *zero-neq-one-ow* =
zero-ow *U* *zero* + *one-ow* *U* *one*
for *U* :: 'ag set **and** *one* ($\langle 1_{ow} \rangle$) **and** *zero* ($\langle 0_{ow} \rangle$) +
assumes *zero-neq-one[simp]*: $0_{ow} \neq 1_{ow}$

lemma *zero-neq-one-ow*: *class.zero-neq-one* = *zero-neq-one-ow* *UNIV*
 $\langle proof \rangle$

ud $\langle zero-neq-one.of\text{-}bool \rangle$ ($\langle with\ -\ - : \langle of'\text{-}bool \rangle \ - \rangle$) [1000, 999, 1000] 10)
ud *of\text{-}bool'* $\langle of\text{-}bool \rangle$

ctr *parametricity*
in *of\text{-}bool.with\text{-}def*

context *zero-neq-one-ow*
begin

abbreviation *of\text{-}bool* **where** *of\text{-}bool* \equiv *of\text{-}bool.with* 1_{ow} 0_{ow}

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *zero-neq-one-transfer[transfer-rule]*:
assumes [*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
 $(A \text{ ===> } A \text{ ===> } (=))$
 $(zero-neq-one-ow (Collect (Domainp\ A))) \text{ class.zero-neq-one}$
 $(is\ ?PR (zero-neq-one-ow (Collect (Domainp\ A))) \text{ class.zero-neq-one})$
 $\langle proof \rangle$

end

Relativization

context *zero-neq-one-ow*
begin

tts-context
tts: ($?a$ to *U*)
rewriting *ctr-simps*
substituting *zero-neq-one-ow-axioms*
eliminating through *simp*
begin

tts-lemma *split-of-bool-asm*:
shows $P (of\text{-}bool\ p) = (\neg (p \wedge \neg P\ 1_{ow} \vee \neg p \wedge \neg P\ 0_{ow}))$
is *zero-neq-one-class.split-of-bool-asm* $\langle proof \rangle$

tts-lemma *of-bool-eq-iff*:

shows $(\text{of-bool } p = \text{local.of-bool } q) = (p = q)$
is *zero-neq-one-class.of-bool-eq-iff* $\langle \text{proof} \rangle$

tts-lemma *split-of-bool*:

shows $P (\text{of-bool } p) = ((p \longrightarrow P \ 1_{ow}) \wedge (\neg p \longrightarrow P \ 0_{ow}))$
is *zero-neq-one-class.split-of-bool* $\langle \text{proof} \rangle$

tts-lemma *one-neq-zero*: $1_{ow} \neq 0_{ow}$

is *zero-neq-one-class.one-neq-zero* $\langle \text{proof} \rangle$

tts-lemma *of-bool-eq*:

shows $\text{of-bool } \text{False} = 0_{ow}$
is *zero-neq-one-class.of-bool-eq*(1)
and $\text{of-bool } \text{True} = 1_{ow}$
is *zero-neq-one-class.of-bool-eq*(2) $\langle \text{proof} \rangle$

end

end

3.10.8 Semirings with zero and one (rigs)

Definitions and common properties

locale *semiring-1-ow* =

zero-neq-one-ow U *one zero* +
semiring-0-ow U *plus zero times* +
monoid-mult-ow U *one times*
for $U :: 'a$ g set **and** *one times plus zero*

lemma *semiring-1-ow*: *class.semiring-1* = *semiring-1-ow* *UNIV*
 $\langle \text{proof} \rangle$

ud $\langle \text{semiring-1.of-nat} \rangle$ ($\langle (\text{with } - - - : \langle \text{of}'\text{-nat} \rangle -) \rangle$ [1000, 999, 998, 1000] 10)
ud $\langle \text{of-nat}' \rangle \langle \text{of-nat} \rangle$

ud $\langle \text{semiring-1.Nats} \rangle$ ($\langle (\text{with } - - - : \mathbf{N}) \rangle$ [1000, 999, 998] 10)
ud $\langle \text{Nats}' \rangle \langle \text{Nats} \rangle$

ctr *parametricity*

in *of-nat-ow*: *of-nat.with-def*
and *Nats-ow*: *Nats.with-def*

context *semiring-1-ow*

begin

abbreviation *of-nat* **where** $\text{of-nat} \equiv \text{of-nat.with } 1_{ow} (+_{ow}) 0_{ow}$

abbreviation *Nats* ($\langle \langle \mathbf{N} \rangle \rangle$) **where** $\langle \langle \mathbf{N} \rangle \rangle \equiv \text{Nats.with } 1_{ow} (+_{ow}) 0_{ow}$

notation *Nats* ($\langle \langle \mathbf{N} \rangle \rangle$)

end

context *semiring-1*

begin

lemma *Nat-ss-UNIV*: $\mathbf{N} \subseteq \text{UNIV}$ $\langle \text{proof} \rangle$

end

Transfer rules

context

includes *lifting-syntax*

begin

lemma *semiring-1-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$(A \text{ ===> } (A \text{ ===> } A \text{ ===> } A) \text{ ===> } (A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } (=))$

(semiring-1-ow (Collect (Domainp A))) class.semiring-1

<proof>

end

Relativization

context *semiring-1-ow*

begin

tts-context

tts: (*?'a to U*)

rewriting *ctr-simps*

substituting *semiring-1-ow-axioms* and *zero.not-empty*

eliminating through *simp*

begin

tts-lemma *Nat-ss-UNIV*[*simp*]:

shows $\langle \mathbf{N} \rangle \subseteq U$

is *Nat-ss-UNIV**<proof>*

end

lemma *Nat-closed*[*simp, intro*]: $a \in \langle \mathbf{N} \rangle \implies a \in U$ *<proof>*

tts-context

tts: (*?'a to U*)

rewriting *ctr-simps*

substituting *semiring-1-ow-axioms* and *zero.not-empty*

eliminating through *auto*

begin

tts-lemma *mult-of-nat-commute*:

assumes $y \in U$

shows $\text{of-nat } x *_{ow} y = y *_{ow} \text{of-nat } x$

is *semiring-1-class.mult-of-nat-commute**<proof>*

tts-lemma *of-bool-conj*: $\text{of-bool } (P \wedge Q) = \text{of-bool } P *_{ow} \text{of-bool } Q$

is *semiring-1-class.of-bool-conj**<proof>*

tts-lemma *power-0-left*: $0_{ow} \hat{\ }_{ow} n = (\text{if } n = 0 \text{ then } 1_{ow} \text{ else } 0_{ow})$

is *semiring-1-class.power-0-left**<proof>*

tts-lemma *of-nat-power*: $\text{of-nat } ((\text{with } 1 (*): m \hat{\ }_{ow} n)) = \text{of-nat } m \hat{\ }_{ow} n$

is *semiring-1-class.of-nat-power**<proof>*

tts-lemma *of-nat-of-bool*: $of\text{-nat}$ (with $1\ 0 : \langle\langle of\text{-bool} \rangle\rangle P$) = $of\text{-bool}$ P
is *semiring-1-class.of-nat-of-bool*{*proof*}

tts-lemma *of-nat-in-Nats*: $of\text{-nat}$ $n \in \langle\langle \mathbf{N} \rangle\rangle$
is *semiring-1-class.of-nat-in-Nats*{*proof*}

tts-lemma *zero-power2*: $0_{ow} \hat{\ }_{ow} 2 = 0_{ow}$
is *semiring-1-class.zero-power2*{*proof*}

tts-lemma *power-0-Suc*: $0_{ow} \hat{\ }_{ow} Suc\ n = 0_{ow}$
is *semiring-1-class.power-0-Suc*{*proof*}

tts-lemma *zero-power*:
assumes $0 < n$
shows $0_{ow} \hat{\ }_{ow} n = 0_{ow}$
is *semiring-1-class.zero-power*{*proof*}

tts-lemma *one-power2*: $1_{ow} \hat{\ }_{ow} 2 = 1_{ow}$
is *semiring-1-class.one-power2*{*proof*}

tts-lemma *of-nat-simps*:
shows $of\text{-nat}\ 0 = 0_{ow}$
is *semiring-1-class.of-nat-simps*(1)
and $of\text{-nat}\ (Suc\ m) = 1_{ow} +_{ow} of\text{-nat}\ m$
is *semiring-1-class.of-nat-simps*(2){*proof*}

tts-lemma *of-nat-mult*: $of\text{-nat}\ (m * n) = of\text{-nat}\ m *_{ow} of\text{-nat}\ n$
is *semiring-1-class.of-nat-mult*{*proof*}

tts-lemma *Nats-induct*:
assumes $x \in \langle\langle \mathbf{N} \rangle\rangle$ **and** $\bigwedge n. P\ (of\text{-nat}\ n)$
shows $P\ x$
is *semiring-1-class.Nats-induct*{*proof*}

tts-lemma *of-nat-add*: $of\text{-nat}\ (m + n) = of\text{-nat}\ m +_{ow} of\text{-nat}\ n$
is *semiring-1-class.of-nat-add*{*proof*}

tts-lemma *of-nat-Suc*: $of\text{-nat}\ (Suc\ m) = 1_{ow} +_{ow} of\text{-nat}\ m$
is *semiring-1-class.of-nat-Suc*{*proof*}

tts-lemma *Nats-cases*:
assumes $x \in \langle\langle \mathbf{N} \rangle\rangle$
obtains $(of\text{-nat})\ n$ **where** $x = of\text{-nat}\ n$
given *semiring-1-class.Nats-cases* {*proof*}

tts-lemma *Nats-mult*:
assumes $a \in \langle\langle \mathbf{N} \rangle\rangle$ **and** $b \in \langle\langle \mathbf{N} \rangle\rangle$
shows $a *_{ow} b \in \langle\langle \mathbf{N} \rangle\rangle$
is *semiring-1-class.Nats-mult*{*proof*}

tts-lemma *of-nat-1*: $of\text{-nat}\ 1 = 1_{ow}$
is *semiring-1-class.of-nat-1*{*proof*}

tts-lemma *of-nat-0*: $of\text{-nat}\ 0 = 0_{ow}$
is *semiring-1-class.of-nat-0*{*proof*}

tts-lemma *Nats-add*:

```

assumes  $a \in \langle\langle \mathbb{N} \rangle\rangle$  and  $b \in \langle\langle \mathbb{N} \rangle\rangle$ 
shows  $a +_{ow} b \in \langle\langle \mathbb{N} \rangle\rangle$ 
is semiring-1-class.Nats-add $\langle$ proof $\rangle$ 

```

```

tts-lemma Nats-1:  $1_{ow} \in \langle\langle \mathbb{N} \rangle\rangle$ 
is semiring-1-class.Nats-1 $\langle$ proof $\rangle$ 

```

```

tts-lemma Nats-0:  $0_{ow} \in \langle\langle \mathbb{N} \rangle\rangle$ 
is semiring-1-class.Nats-0 $\langle$ proof $\rangle$ 

```

end

end

3.10.9 Commutative rigs

Definitions and common properties

```

locale comm-semiring-1-ow =
  zero-neg-one-ow U one zero +
  comm-semiring-0-ow U plus zero times +
  comm-monoid-mult-ow U times one
for  $U :: 'a$  g set and times one plus zero
begin

```

```

sublocale semiring-1-ow  $\langle$ proof $\rangle$ 

```

end

```

lemma comm-semiring-1-ow: class.comm-semiring-1 = comm-semiring-1-ow UNIV
 $\langle$ proof $\rangle$ 

```

Transfer rules

```

context
includes lifting-syntax
begin

```

```

lemma comm-semiring-1-transfer $[$ transfer-rule $]$ :
assumes $[$ transfer-rule $]$ : bi-unique  $A$  right-total  $A$ 
shows
   $((A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } (A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } (=))$ 
   $(\text{comm-semiring-1-ow } (\text{Collect } (\text{Domainp } A))) \text{ class.comm-semiring-1}$ 
 $\langle$ proof $\rangle$ 

```

end

Relativization

```

context comm-semiring-1-ow
begin

```

```

tts-context
tts:  $(?'a \text{ to } U)$ 
rewriting ctr-simps
substituting comm-semiring-1-ow-axioms and zero.not-empty
applying  $[$ OF times-closed' one-closed plus-closed' zero-closed $]$ 
begin

```

tts-lemma *semiring-normalization-rules*:

shows

$$\begin{aligned}
& \llbracket a \in U; m \in U; b \in U \rrbracket \Longrightarrow a *_{ow} m +_{ow} b *_{ow} m = (a +_{ow} b) *_{ow} m \\
& \llbracket a \in U; m \in U \rrbracket \Longrightarrow a *_{ow} m +_{ow} m = (a +_{ow} 1_{ow}) *_{ow} m \\
& \llbracket m \in U; a \in U \rrbracket \Longrightarrow m +_{ow} a *_{ow} m = (a +_{ow} 1_{ow}) *_{ow} m \\
& m \in U \Longrightarrow m +_{ow} m = (1_{ow} +_{ow} 1_{ow}) *_{ow} m \\
& a \in U \Longrightarrow 0_{ow} +_{ow} a = a \\
& a \in U \Longrightarrow a +_{ow} 0_{ow} = a \\
& \llbracket a \in U; b \in U \rrbracket \Longrightarrow a *_{ow} b = b *_{ow} a \\
& \llbracket a \in U; b \in U; c \in U \rrbracket \Longrightarrow (a +_{ow} b) *_{ow} c = a *_{ow} c +_{ow} b *_{ow} c \\
& a \in U \Longrightarrow 0_{ow} *_{ow} a = 0_{ow} \\
& a \in U \Longrightarrow a *_{ow} 0_{ow} = 0_{ow} \\
& a \in U \Longrightarrow 1_{ow} *_{ow} a = a \\
& a \in U \Longrightarrow a *_{ow} 1_{ow} = a \\
& \llbracket lx \in U; ly \in U; rx \in U; ry \in U \rrbracket \Longrightarrow \\
& \quad lx *_{ow} ly *_{ow} (rx *_{ow} ry) = lx *_{ow} rx *_{ow} (ly *_{ow} ry) \\
& \llbracket lx \in U; ly \in U; rx \in U; ry \in U \rrbracket \Longrightarrow \\
& \quad lx *_{ow} ly *_{ow} (rx *_{ow} ry) = lx *_{ow} (ly *_{ow} (rx *_{ow} ry)) \\
& \llbracket lx \in U; ly \in U; rx \in U; ry \in U \rrbracket \Longrightarrow \\
& \quad lx *_{ow} ly *_{ow} (rx *_{ow} ry) = rx *_{ow} (lx *_{ow} ly *_{ow} ry) \\
& \llbracket lx \in U; ly \in U; rx \in U \rrbracket \Longrightarrow lx *_{ow} ly *_{ow} rx = lx *_{ow} rx *_{ow} ly \\
& \llbracket lx \in U; ly \in U; rx \in U \rrbracket \Longrightarrow lx *_{ow} ly *_{ow} rx = lx *_{ow} (ly *_{ow} rx) \\
& \llbracket lx \in U; rx \in U; ry \in U \rrbracket \Longrightarrow lx *_{ow} (rx *_{ow} ry) = lx *_{ow} rx *_{ow} ry \\
& \llbracket lx \in U; rx \in U; ry \in U \rrbracket \Longrightarrow lx *_{ow} (rx *_{ow} ry) = rx *_{ow} (lx *_{ow} ry) \\
& \llbracket a \in U; b \in U; c \in U; d \in U \rrbracket \Longrightarrow \\
& \quad a +_{ow} b +_{ow} (c +_{ow} d) = a +_{ow} c +_{ow} (b +_{ow} d) \\
& \llbracket a \in U; b \in U; c \in U \rrbracket \Longrightarrow a +_{ow} b +_{ow} c = a +_{ow} (b +_{ow} c) \\
& \llbracket a \in U; c \in U; d \in U \rrbracket \Longrightarrow a +_{ow} (c +_{ow} d) = c +_{ow} (a +_{ow} d) \\
& \llbracket a \in U; b \in U; c \in U \rrbracket \Longrightarrow a +_{ow} b +_{ow} c = a +_{ow} c +_{ow} b \\
& \llbracket a \in U; c \in U \rrbracket \Longrightarrow a +_{ow} c = c +_{ow} a \\
& \llbracket a \in U; c \in U; d \in U \rrbracket \Longrightarrow a +_{ow} (c +_{ow} d) = a +_{ow} c +_{ow} d \\
& x \in U \Longrightarrow x \widehat{ow} p *_{ow} x \widehat{ow} q = x \widehat{ow} (p + q) \\
& x \in U \Longrightarrow x *_{ow} x \widehat{ow} q = x \widehat{ow} Suc\ q \\
& x \in U \Longrightarrow x \widehat{ow} q *_{ow} x = x \widehat{ow} Suc\ q \\
& x \in U \Longrightarrow x *_{ow} x = x \widehat{ow} 2 \\
& \llbracket x \in U; y \in U \rrbracket \Longrightarrow (x *_{ow} y) \widehat{ow} q = x \widehat{ow} q *_{ow} y \widehat{ow} q \\
& x \in U \Longrightarrow (x \widehat{ow} p) \widehat{ow} q = x \widehat{ow} (p * q) \\
& x \in U \Longrightarrow x \widehat{ow} 0 = 1_{ow} \\
& x \in U \Longrightarrow x \widehat{ow} 1 = x \\
& \llbracket x \in U; y \in U; z \in U \rrbracket \Longrightarrow x *_{ow} (y +_{ow} z) = x *_{ow} y +_{ow} x *_{ow} z \\
& x \in U \Longrightarrow x \widehat{ow} Suc\ q = x *_{ow} x \widehat{ow} q \\
& x \in U \Longrightarrow x \widehat{ow} (2 * n) = x \widehat{ow} n *_{ow} x \widehat{ow} n \\
& \text{is comm-semiring-1-class.semiring-normalization-rules}\langle\text{proof}\rangle
\end{aligned}$$

tts-lemma *le-imp-power-dvd*:

assumes $a \in U$ and $m \leq n$

shows $a \widehat{ow} m \ll\text{dvd}\gg a \widehat{ow} n$

is *comm-semiring-1-class.le-imp-power-dvd*\langle proof \rangle

tts-lemma *dvd-0-left-iff*:

assumes $a \in U$

shows $(0_{ow} \ll\text{dvd}\gg a) = (a = 0_{ow})$

is *comm-semiring-1-class.dvd-0-left-iff*\langle proof \rangle

tts-lemma *dvd-power-same*:

assumes $x \in U$ and $y \in U$ and $x \ll\text{dvd}\gg y$

shows $x \widehat{ow} n \ll\text{dvd}\gg y \widehat{ow} n$

is *comm-semiring-1-class.dvd-power-same*\langle proof \rangle

```

tts-lemma power-le-dvd:
  assumes  $a \in U$  and  $b \in U$  and  $a \hat{\ }_{ow} n \ll dvd \gg b$  and  $m \leq n$ 
  shows  $a \hat{\ }_{ow} m \ll dvd \gg b$ 
  is comm-semiring-1-class.power-le-dvd{proof}

tts-lemma dvd-0-right:
  assumes  $a \in U$ 
  shows  $a \ll dvd \gg 0_{ow}$ 
  is comm-semiring-1-class.dvd-0-right{proof}

tts-lemma dvd-0-left:
  assumes  $a \in U$  and  $0_{ow} \ll dvd \gg a$ 
  shows  $a = 0_{ow}$ 
  is comm-semiring-1-class.dvd-0-left{proof}

tts-lemma dvd-power:
  assumes  $x \in U$  and  $0 < n \vee x = 1_{ow}$ 
  shows  $x \ll dvd \gg x \hat{\ }_{ow} n$ 
  is comm-semiring-1-class.dvd-power{proof}

tts-lemma dvd-add:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$  and  $a \ll dvd \gg b$  and  $a \ll dvd \gg c$ 
  shows  $a \ll dvd \gg b +_{ow} c$ 
  is comm-semiring-1-class.dvd-add{proof}

```

end

end

3.10.10 Cancellative rigs

Definitions and common properties

```

locale semiring-1-cancel-ow =
  semiring-ow U plus times +
  cancel-comm-monoid-add-ow U plus minus zero +
  zero-neq-one-ow U one zero +
  monoid-mult-ow U one times
  for  $U :: 'ag\ set$  and plus minus zero one times
begin

```

```

sublocale semiring-0-cancel-ow {proof}
sublocale semiring-1-ow {proof}

```

end

```

lemma semiring-1-cancel-ow:
  class.semiring-1-cancel = semiring-1-cancel-ow UNIV
  {proof}

```

Transfer rules

```

context
  includes lifting-syntax
begin

```

```

lemma semiring-1-cancel-transfer[transfer-rule]:
  includes lifting-syntax

```

```

assumes[transfer-rule]: bi-unique A right-total A
shows
  (
    (A ==> A ==> A) ==>
    (A ==> A ==> A) ==>
    A ==>
    A ==>
    (A ==> A ==> A) ==>
    (=)
  ) (semiring-1-cancel-ow (Collect (Domainp A))) class.semiring-1-cancel
  <proof>

```

end

3.10.11 Commutative cancellative rigs

Definitions and common properties

```

locale comm-semiring-1-cancel-ow =
  comm-semiring-ow U plus times +
  cancel-comm-monoid-add-ow U plus minus zero +
  zero-neg-one-ow U one zero +
  comm-monoid-mult-ow U times one
for U :: 'ag set and plus minus zero times one +
assumes right-diff-distrib'[algebra-simps]:
  
$$\llbracket a \in U; b \in U; c \in U \rrbracket \implies a *_{ow} (b -_{ow} c) = a *_{ow} b -_{ow} a *_{ow} c$$

begin

```

```

sublocale semiring-1-cancel-ow <proof>
sublocale comm-semiring-0-cancel-ow <proof>
sublocale comm-semiring-1-ow <proof>

```

end

Transfer rules

```

context
  includes lifting-syntax
begin

```

```

lemma comm-semiring-1-cancel-transfer[transfer-rule]:
assumes[transfer-rule]: bi-unique A right-total A
shows
  (
    (A ==> A ==> A) ==>
    (A ==> A ==> A) ==>
    A ==>
    (A ==> A ==> A) ==>
    A ==>
    (=)
  )
  (comm-semiring-1-cancel-ow (Collect (Domainp A)))
  class.comm-semiring-1-cancel
  <proof>

```

end

Relativization

context *comm-semiring-1-cancel-ow*

begin

tts-context

tts: (*?'a to U*)

rewriting *ctr-simps*

substituting *comm-semiring-1-cancel-ow-axioms* **and** *zero.not-empty*

applying [*OF plus-closed' minus-closed' zero-closed times-closed' one-closed*]

begin

tts-lemma *dvd-add-times-triv-right-iff:*

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$

shows $(a \ll \text{dvd} \gg b +_{ow} c *_{ow} a) = (a \ll \text{dvd} \gg b)$

is *comm-semiring-1-cancel-class.dvd-add-times-triv-right-iff*{*proof*}

tts-lemma *dvd-add-times-triv-left-iff:*

assumes $a \in U$ **and** $c \in U$ **and** $b \in U$

shows $(a \ll \text{dvd} \gg c *_{ow} a +_{ow} b) = (a \ll \text{dvd} \gg b)$

is *comm-semiring-1-cancel-class.dvd-add-times-triv-left-iff*{*proof*}

tts-lemma *dvd-add-triv-right-iff:*

assumes $a \in U$ **and** $b \in U$

shows $(a \ll \text{dvd} \gg b +_{ow} a) = (a \ll \text{dvd} \gg b)$

is *comm-semiring-1-cancel-class.dvd-add-triv-right-iff*{*proof*}

tts-lemma *dvd-add-triv-left-iff:*

assumes $a \in U$ **and** $b \in U$

shows $(a \ll \text{dvd} \gg a +_{ow} b) = (a \ll \text{dvd} \gg b)$

is *comm-semiring-1-cancel-class.dvd-add-triv-left-iff*{*proof*}

tts-lemma *left-diff-distrib':*

assumes $b \in U$ **and** $c \in U$ **and** $a \in U$

shows $(b -_{ow} c) *_{ow} a = b *_{ow} a -_{ow} c *_{ow} a$

is *comm-semiring-1-cancel-class.left-diff-distrib'*{*proof*}

tts-lemma *dvd-add-right-iff:*

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $a \ll \text{dvd} \gg b$

shows $(a \ll \text{dvd} \gg b +_{ow} c) = (a \ll \text{dvd} \gg c)$

is *comm-semiring-1-cancel-class.dvd-add-right-iff*{*proof*}

tts-lemma *dvd-add-left-iff:*

assumes $a \in U$ **and** $c \in U$ **and** $b \in U$ **and** $a \ll \text{dvd} \gg c$

shows $(a \ll \text{dvd} \gg b +_{ow} c) = (a \ll \text{dvd} \gg b)$

is *comm-semiring-1-cancel-class.dvd-add-left-iff*{*proof*}

end

end

3.11 Relativization of the results about rings

3.11.1 Rings

Definitions and common properties

```

locale ring-ow =
  semiring-ow U plus times + ab-group-add-ow U plus zero minus uminus
  for U :: 'ag set and plus zero minus uminus times
begin

  sublocale semiring-0-cancel-ow ⟨proof⟩

end

lemma ring-ow: class.ring = ring-ow UNIV
  ⟨proof⟩

lemma ring-ow-UNIV-axioms: ring-ow (UNIV::'a::ring set) (+) 0 (-) uminus (*)
  ⟨proof⟩

```

Transfer rules

```

context
  includes lifting-syntax
begin

lemma ring-transfer[transfer-rule]:
  assumes[transfer-rule]: bi-unique A right-total A
  shows
  (
    (A =====> A =====> A) =====>
    A =====>
    (A =====> A =====> A) =====>
    (A =====> A) =====>
    (A =====> A =====> A) =====>
    (=)
  )
  (ring-ow (Collect (Domainp A))) class.ring
  ⟨proof⟩

```

end

Relativization

```

context ring-ow
begin

tts-context
  tts: (?'a to U)
  rewriting ctr-simps
  substituting ring-ow-axioms and zero.not-empty
  applying
  [
    OF
    plus-closed'
    zero-closed
    minus-closed'
    add.inverse-closed''
  ]

```

```

    times-closed'
  ]
begin

tts-lemma right-diff-distrib:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$ 
  shows  $a *_{ow} (b -_{ow} c) = a *_{ow} b -_{ow} a *_{ow} c$ 
  is Rings.ring-class.right-diff-distrib{proof}

tts-lemma minus-mult-commute:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $-_{ow} a *_{ow} b = a *_{ow} -_{ow} b$ 
  is Rings.ring-class.minus-mult-commute{proof}

tts-lemma left-diff-distrib:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$ 
  shows  $(a -_{ow} b) *_{ow} c = a *_{ow} c -_{ow} b *_{ow} c$ 
  is Rings.ring-class.left-diff-distrib{proof}

tts-lemma mult-minus-right:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $a *_{ow} -_{ow} b = -_{ow} (a *_{ow} b)$ 
  is Rings.ring-class.mult-minus-right{proof}

tts-lemma minus-mult-right:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $-_{ow} (a *_{ow} b) = a *_{ow} -_{ow} b$ 
  is Rings.ring-class.minus-mult-right{proof}

tts-lemma minus-mult-minus:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $-_{ow} a *_{ow} -_{ow} b = a *_{ow} b$ 
  is Rings.ring-class.minus-mult-minus{proof}

tts-lemma mult-minus-left:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $-_{ow} a *_{ow} b = -_{ow} (a *_{ow} b)$ 
  is Rings.ring-class.mult-minus-left{proof}

tts-lemma minus-mult-left:
  assumes  $a \in U$  and  $b \in U$ 
  shows  $-_{ow} (a *_{ow} b) = -_{ow} a *_{ow} b$ 
  is Rings.ring-class.minus-mult-left{proof}

tts-lemma ring-distrib:
  assumes  $a \in U$  and  $b \in U$  and  $c \in U$ 
  shows
     $a *_{ow} (b +_{ow} c) = a *_{ow} b +_{ow} a *_{ow} c$ 
     $(a +_{ow} b) *_{ow} c = a *_{ow} c +_{ow} b *_{ow} c$ 
     $(a -_{ow} b) *_{ow} c = a *_{ow} c -_{ow} b *_{ow} c$ 
     $a *_{ow} (b -_{ow} c) = a *_{ow} b -_{ow} a *_{ow} c$ 
  is Rings.ring-class.ring-distrib{proof}

tts-lemma eq-add-iff2:
  assumes  $a \in U$  and  $e \in U$  and  $c \in U$  and  $b \in U$  and  $d \in U$ 
  shows  $(a *_{ow} e +_{ow} c = b *_{ow} e +_{ow} d) = (c = (b -_{ow} a) *_{ow} e +_{ow} d)$ 
  is Rings.ring-class.eq-add-iff2{proof}

```

```

tts-lemma eq-add-iff1:
  assumes  $a \in U$  and  $e \in U$  and  $c \in U$  and  $b \in U$  and  $d \in U$ 
  shows  $(a *_{ow} e +_{ow} c = b *_{ow} e +_{ow} d) = ((a -_{ow} b) *_{ow} e +_{ow} c = d)$ 
  is Rings.ring-class.eq-add-iff1 $\langle$ proof $\rangle$ 

tts-lemma mult-diff-mult:
  assumes  $x \in U$  and  $y \in U$  and  $a \in U$  and  $b \in U$ 
  shows  $x *_{ow} y -_{ow} a *_{ow} b = x *_{ow} (y -_{ow} b) +_{ow} (x -_{ow} a) *_{ow} b$ 
  is Real.mult-diff-mult $\langle$ proof $\rangle$ 

end

end

```

3.11.2 Commutative rings

Definitions and common properties

```

locale comm-ring-ow =
  comm-semiring-ow  $U$  plus times + ab-group-add-ow  $U$  plus zero minus uminus
  for  $U :: 'ag$  set and plus zero minus uminus times
begin

  sublocale ring-ow  $\langle$ proof $\rangle$ 
  sublocale comm-semiring-0-cancel-ow  $\langle$ proof $\rangle$ 

end

```

```

lemma comm-ring-ow: class.comm-ring = comm-ring-ow UNIV
   $\langle$ proof $\rangle$ 

```

Transfer rules

```

context
  includes lifting-syntax
begin

lemma comm-ring-transfer $[$ transfer-rule $]$ :
  assumes $[$ transfer-rule $]$ : bi-unique  $A$  right-total  $A$ 
  shows
    (
       $(A \implies A \implies A) \implies$ 
       $A \implies$ 
       $(A \implies A \implies A) \implies$ 
       $(A \implies A) \implies$ 
       $(A \implies A \implies A) \implies$ 
       $(=)$ 
    )
    (comm-ring-ow (Collect (Domainp  $A$ ))) class.comm-ring
   $\langle$ proof $\rangle$ 

end

```

Relativization

```

context comm-ring-ow
begin

tts-context

```

```

tts: (?a to U)
rewriting ctr-simps
substituting comm-ring-ow-axioms and zero.not-empty
applying
  [
    OF
    plus-closed'
    zero-closed
    minus-closed'
    add.inverse-closed''
    times-closed'
  ]
begin

```

```

tts-lemma square-diff-square-factored:
  assumes  $x \in U$  and  $y \in U$ 
  shows  $x *_{ow} x -_{ow} y *_{ow} y = (x +_{ow} y) *_{ow} (x -_{ow} y)$ 
  is comm-ring-class.square-diff-square-factored(proof)

```

end

end

3.11.3 Rings with identity

Definitions and common properties

```

locale ring-1-ow =
  ring-ow U plus zero minus uminus times +
  zero-neq-one-ow U one zero +
  monoid-mult-ow U one times
  for  $U :: 'a$  set and one times plus zero minus uminus
begin

```

```

sublocale semiring-1-cancel-ow (proof)

```

end

```

lemma ring-1-ow: class.ring-1 = ring-1-ow UNIV
  (proof)

```

```

lemma ring-1-ow-UNIV-axioms:
  ring-1-ow (UNIV::'a::ring-1 set) 1 (*) (+) 0 (-) uminus
  (proof)

```

```

ud ring-1.iszero (⟨(with - : «iszero» -)⟩ [1000, 1000] 10)
ud iszero' ⟨iszero⟩
ud ring-1.of-int
  (⟨(with - - - : «of-int» -)⟩ [1000, 999, 998, 997, 1000] 10)
ud of-int' ⟨of-int⟩
ud ring-1.Ints (⟨(with - - - :  $\mathbf{Z}$ )⟩ [1000, 999, 998, 997] 10)
ud Ints' ⟨Ints⟩
ud diffs (⟨(with - - - : «diffs» -)⟩ [1000, 999, 998, 997, 1000] 10)

```

ctr parametricity

```

in iszero-ow: iszero.with-def
and of-int-ow: of-int.with-def
and Ints-ow: Ints.with-def

```

and *diffs-ow*: *diffs.with-def*

context *ring-1-ow*
begin

abbreviation *iszero* where *iszero* \equiv *iszero.with* 0_{ow}

abbreviation *of-int* where *of-int* \equiv *of-int.with* 1_{ow} $(+_{ow})$ 0_{ow} $(-_{ow})$

abbreviation *Ints* ($\langle\langle\mathbf{Z}\rangle\rangle$) where $\langle\langle\mathbf{Z}\rangle\rangle \equiv$ *Ints.with* 1_{ow} $(+_{ow})$ 0_{ow} $(-_{ow})$

notation *Ints* ($\langle\langle\mathbf{Z}\rangle\rangle$)

end

context *ring-1*
begin

lemma *Int-ss-UNIV*: $\mathbf{Z} \subseteq UNIV$ *<proof>*

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *ring-1-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
(
 A \equiv \equiv \equiv \equiv
 (*A* \equiv \equiv \equiv *A*) \equiv \equiv \equiv
 (*A* \equiv \equiv \equiv *A*) \equiv \equiv \equiv
 A \equiv \equiv \equiv
 (*A* \equiv \equiv \equiv *A*) \equiv \equiv \equiv
 (*A* \equiv \equiv \equiv *A*) \equiv \equiv \equiv
 (=)
)
 (*ring-1-ow* (*Collect* (*Domainp* *A*))) *class.ring-1*
 <proof>

end

Relativization

declare *dvd.with*[*ud-with del*]
declare *dvd'.with*[*ud-with del*]

context *ring-1-ow*
begin

tts-context
tts: (*?a to U*)
substituting *ring-1-ow-axioms* and *zero.not-empty*
eliminating through *simp*
begin

tts-lemma *Int-ss-UNIV*[*simp*]: $\langle\langle\mathbf{Z}\rangle\rangle \subseteq U$
is *Int-ss-UNIV* *<proof>*

end

lemma *Int-closed*[*simp,intro*]: $a \in \langle\langle \mathbb{Z} \rangle\rangle \implies a \in U$ *{proof}*

tts-context

tts: (*?'a to U*)

rewriting *ctr-simps*

substituting *ring-1-ow-axioms* **and** *zero.not-empty*

eliminating through *auto*

begin

tts-lemma *iszero-0*: *iszero* 0_{ow}

is *ring-1-class.iszero-0**{proof}*

tts-lemma *not-iszero-1*: \neg *iszero* 1_{ow}

is *ring-1-class.not-iszero-1**{proof}*

tts-lemma *Nats-subset-Ints*: $\langle\langle \mathbb{N} \rangle\rangle \subseteq \langle\langle \mathbb{Z} \rangle\rangle$

is *Int.ring-1-class.Nats-subset-Ints**{proof}*

tts-lemma *Ints-1*: $1_{ow} \in \langle\langle \mathbb{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-1**{proof}*

tts-lemma *Ints-0*: $0_{ow} \in \langle\langle \mathbb{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-0**{proof}*

tts-lemma *not-iszero-neg-1*: \neg *iszero* $(-_{ow} 1_{ow})$

is *Num.ring-1-class.not-iszero-neg-1**{proof}*

tts-lemma *of-int-1*: *of-int* $1 = 1_{ow}$

is *Int.ring-1-class.of-int-1**{proof}*

tts-lemma *of-int-0*: *of-int* $0 = 0_{ow}$

is *Int.ring-1-class.of-int-0**{proof}*

tts-lemma *Ints-of-int*: *of-int* $z \in \langle\langle \mathbb{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-of-int**{proof}*

tts-lemma *Ints-of-nat*: *of-nat* $n \in \langle\langle \mathbb{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-of-nat**{proof}*

tts-lemma *of-int-of-nat-eq*:

shows *local.of-int* (with $1 (+) 0 : \langle\langle \text{of-nat} \rangle\rangle n$) = *local.of-nat* n

is *Int.ring-1-class.of-int-of-nat-eq**{proof}*

tts-lemma *of-int-of-bool*:

of-int (with $1 0 : \langle\langle \text{of-bool} \rangle\rangle P$) = *of-bool* P

is *Int.ring-1-class.of-int-of-bool**{proof}*

tts-lemma *of-int-minus*: *of-int* $(- z) = -_{ow}$ *of-int* z

is *Int.ring-1-class.of-int-minus**{proof}*

tts-lemma *mult-minus1-right*:

assumes $z \in U$

shows $z *_{ow} -_{ow} 1_{ow} = -_{ow} z$

is *Num.ring-1-class.mult-minus1-right**{proof}*

tts-lemma *mult-minus1*:

assumes $z \in U$

shows $-_{ow} 1_{ow} *_{ow} z = -_{ow} z$

is *Num.ring-1-class.mult-minus1*{proof}

tts-lemma *eq-iff-iszero-diff*:

assumes $x \in U$ **and** $y \in U$

shows $(x = y) = iszero (x -_{ow} y)$

is *Num.ring-1-class.eq-iff-iszero-diff*{proof}

tts-lemma *minus-in-Ints-iff*:

assumes $x \in U$

shows $(-_{ow} x \in \langle\langle \mathbb{Z} \rangle\rangle) = (x \in \langle\langle \mathbb{Z} \rangle\rangle)$

is *Int.ring-1-class.minus-in-Ints-iff*{proof}

tts-lemma *mult-of-int-commute*:

assumes $y \in U$

shows $of-int x *_{ow} y = y *_{ow} of-int x$

is *Int.ring-1-class.mult-of-int-commute*{proof}

tts-lemma *of-int-power*:

$of-int ((with\ 1\ (*)) : z \hat{\ }_{ow} n) = of-int z \hat{\ }_{ow} n$

is *Int.ring-1-class.of-int-power*{proof}

tts-lemma *Ints-minus*:

assumes $a \in \langle\langle \mathbb{Z} \rangle\rangle$

shows $-_{ow} a \in \langle\langle \mathbb{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-minus*{proof}

tts-lemma *of-int-diff*: $of-int (w - z) = of-int w -_{ow} of-int z$

is *Int.ring-1-class.of-int-diff*{proof}

tts-lemma *of-int-add*: $of-int (w + z) = of-int w +_{ow} of-int z$

is *Int.ring-1-class.of-int-add*{proof}

tts-lemma *of-int-mult*: $of-int (w * z) = of-int w *_{ow} of-int z$

is *Int.ring-1-class.of-int-mult*{proof}

tts-lemma *power-minus1-even*: $(-_{ow} 1_{ow}) \hat{\ }_{ow} (2 * n) = 1_{ow}$

is *Power.ring-1-class.power-minus1-even*{proof}

tts-lemma *Ints-power*:

assumes $a \in \langle\langle \mathbb{Z} \rangle\rangle$

shows $a \hat{\ }_{ow} n \in \langle\langle \mathbb{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-power*{proof}

tts-lemma *of-nat-nat*:

assumes $0 \leq z$

shows $of-nat (nat z) = of-int z$

is *Int.ring-1-class.of-nat-nat*{proof}

tts-lemma *power2-minus*:

assumes $a \in U$

shows $(-_{ow} a) \hat{\ }_{ow} 2 = a \hat{\ }_{ow} 2$

is *Power.ring-1-class.power2-minus*{proof}

tts-lemma *power-minus1-odd*:

shows $(-_{ow} 1_{ow}) \hat{\ }_{ow} Suc (2 * n) = -_{ow} 1_{ow}$

is *Power.ring-1-class.power-minus1-odd*{proof}

tts-lemma *power-minus*:

assumes $a \in U$

shows $(-_{ow} a) \hat{_{ow}} n = (-_{ow} 1_{ow}) \hat{_{ow}} n *_{ow} a \hat{_{ow}} n$

is *Power.ring-1-class.power-minus*{proof}

tts-lemma *square-diff-one-factored*:

assumes $x \in U$

shows $x *_{ow} x -_{ow} 1_{ow} = (x +_{ow} 1_{ow}) *_{ow} (x -_{ow} 1_{ow})$

is *Rings.ring-1-class.square-diff-one-factored*{proof}

tts-lemma *neg-one-even-power*:

assumes *even* n

shows $(-_{ow} 1_{ow}) \hat{_{ow}} n = 1_{ow}$

is *Parity.ring-1-class.neg-one-even-power*{proof}

tts-lemma *minus-one-power-iff*:

$(-_{ow} 1_{ow}) \hat{_{ow}} n = (\text{if } \text{even } n \text{ then } 1_{ow} \text{ else } -_{ow} 1_{ow})$

is *Parity.ring-1-class.minus-one-power-iff*{proof}

tts-lemma *Nats-altdef1*: $\langle\langle \mathbf{N} \rangle\rangle = \{x \in U. \exists y \geq 0. x = \text{of-int } y\}$

is *Int.ring-1-class.Nats-altdef1*{proof}

tts-lemma *Ints-induct*:

assumes $q \in \langle\langle \mathbf{Z} \rangle\rangle$ and $\bigwedge z. P (\text{of-int } z)$

shows $P q$

is *Int.ring-1-class.Ints-induct*{proof}

tts-lemma *of-int-of-nat*:

shows

$\text{of-int } k = (\text{if } k < 0 \text{ then } -_{ow} \text{of-nat } (\text{nat } (-k)) \text{ else } \text{of-nat } (\text{nat } k))$

is *Int.ring-1-class.of-int-of-nat*{proof}

tts-lemma *Ints-diff*:

assumes $a \in \langle\langle \mathbf{Z} \rangle\rangle$ and $b \in \langle\langle \mathbf{Z} \rangle\rangle$

shows $a -_{ow} b \in \langle\langle \mathbf{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-diff*{proof}

tts-lemma *Ints-add*:

assumes $a \in \langle\langle \mathbf{Z} \rangle\rangle$ and $b \in \langle\langle \mathbf{Z} \rangle\rangle$

shows $a +_{ow} b \in \langle\langle \mathbf{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-add*{proof}

tts-lemma *Ints-mult*:

assumes $a \in \langle\langle \mathbf{Z} \rangle\rangle$ and $b \in \langle\langle \mathbf{Z} \rangle\rangle$

shows $a *_{ow} b \in \langle\langle \mathbf{Z} \rangle\rangle$

is *Int.ring-1-class.Ints-mult*{proof}

tts-lemma *power-minus-even'*:

assumes $a \in U$ and *even* n

shows $(-_{ow} a) \hat{_{ow}} n = a \hat{_{ow}} n$

is *Parity.ring-1-class.power-minus-even*{proof}

tts-lemma *power-minus-even*:

assumes $a \in U$

shows $(-_{ow} a) \hat{_{ow}} (2 * n) = a \hat{_{ow}} (2 * n)$

is *Power.ring-1-class.power-minus-even*{proof}

tts-lemma *power-minus-odd*:
assumes $a \in U$ **and** *odd* n
shows $(-_{ow} a) \hat{_{ow}} n = -_{ow} (a \hat{_{ow}} n)$
is *Parity.ring-1-class.power-minus-odd*{*proof*}

tts-lemma *uminus-power-if*:
assumes $a \in U$
shows $(-_{ow} a) \hat{_{ow}} n = (\text{if even } n \text{ then } a \hat{_{ow}} n \text{ else } -_{ow} (a \hat{_{ow}} n))$
is *Parity.ring-1-class.uminus-power-if*{*proof*}

tts-lemma *neg-one-power-add-eq-neg-one-power-diff*:
assumes $k \leq n$
shows $(-_{ow} 1_{ow}) \hat{_{ow}} (n + k) = (-_{ow} 1_{ow}) \hat{_{ow}} (n - k)$
is *Parity.ring-1-class.neg-one-power-add-eq-neg-one-power-diff*{*proof*}

tts-lemma *neg-one-odd-power*:
assumes *odd* n
shows $(-_{ow} 1_{ow}) \hat{_{ow}} n = -_{ow} 1_{ow}$
is *Parity.ring-1-class.neg-one-odd-power*{*proof*}

tts-lemma *Ints-cases*:
assumes $q \in \langle\mathbb{Z}\rangle$ **and** $\bigwedge z. q = \text{of-int } z \implies \text{thesis}$
shows *thesis*
is *Int.ring-1-class.Ints-cases*{*proof*}

end

end

lemmas [*ud-with*] = *dvd.with dvd'.with*

3.11.4 Commutative rings with identity

Definitions and common properties

locale *comm-ring-1-ow* =
comm-ring-ow U plus zero minus uminus times +
zero-neg-one-ow U one zero +
comm-monoid-mult-ow U times one
for $U :: 'a$ *g set* **and** *times one plus zero minus uminus*
begin

sublocale *ring-1-ow* {*proof*}

sublocale *comm-semiring-1-cancel-ow*
{*proof*}

end

lemma *comm-ring-1-ow*: *class.comm-ring-1 = comm-ring-1-ow UNIV*
{*proof*}

lemma *comm-ring-1-ow-UNIV-axioms*:
comm-ring-1-ow (UNIV::'a::comm-ring-1 set) () 1 (+) 0 (-) uminus*
{*proof*}

Transfer rules

context

```

includes lifting-syntax
begin

lemma comm-ring-1-transfer[transfer-rule]:
  assumes[transfer-rule]: bi-unique A right-total A
  shows
    (
      (A  $\implies$  A  $\implies$  A)  $\implies$ 
      A  $\implies$ 
      (A  $\implies$  A  $\implies$  A)  $\implies$ 
      A  $\implies$ 
      (A  $\implies$  A  $\implies$  A)  $\implies$ 
      (A  $\implies$  A)  $\implies$ 
      (=)
    ) (comm-ring-1-ow (Collect (Domainp A))) class.comm-ring-1
  <proof>

end

```

Relativization

```

context comm-ring-1-ow
begin

tts-context
  tts: (?a to U)
  rewriting ctr-simps
  substituting comm-ring-1-ow-axioms and zero.not-empty
  applying
    [
      OF
      times-closed'
      one-closed
      plus-closed'
      zero-closed
      minus-closed'
      add.inverse-closed''
    ]
begin

tts-lemma ring-normalization-rules:
  assumes x  $\in$  U
  shows  $-_{ow} x = -_{ow} 1_{ow} *_{ow} x$   $y \in U \implies x -_{ow} y = x +_{ow} -_{ow} y$ 
  is comm-ring-1-class.ring-normalization-rules{proof}

tts-lemma left-minus-one-mult-self:
  assumes a  $\in$  U
  shows  $(-_{ow} 1_{ow}) \hat{_{ow}} n *_{ow} ((-_{ow} 1_{ow}) \hat{_{ow}} n *_{ow} a) = a$ 
  is Power.comm-ring-1-class.left-minus-one-mult-self{proof}

tts-lemma minus-power-mult-self:
  assumes a  $\in$  U
  shows  $(-_{ow} a) \hat{_{ow}} n *_{ow} (-_{ow} a) \hat{_{ow}} n = a \hat{_{ow}} (2 * n)$ 
  is Power.comm-ring-1-class.minus-power-mult-self{proof}

tts-lemma minus-one-mult-self:  $(-_{ow} 1_{ow}) \hat{_{ow}} n *_{ow} (-_{ow} 1_{ow}) \hat{_{ow}} n = 1_{ow}$ 
  is comm-ring-1-class.minus-one-mult-self{proof}

```

tts-lemma *power2-commute*:

assumes $x \in U$ **and** $y \in U$

shows $(x \text{ }_{-ow} \text{ } y) \widehat{\text{ }_{ow} 2} = (y \text{ }_{-ow} \text{ } x) \widehat{\text{ }_{ow} 2}$

is *comm-ring-1-class.power2-commute*{proof}

tts-lemma *minus-dvd-iff*:

assumes $x \in U$ **and** $y \in U$

shows $(\text{ }_{-ow} x \ll \text{dvd} \gg y) = (x \ll \text{dvd} \gg y)$

is *comm-ring-1-class.minus-dvd-iff*{proof}

tts-lemma *dvd-minus-iff*:

assumes $x \in U$ **and** $y \in U$

shows $(x \ll \text{dvd} \gg \text{ }_{-ow} y) = (x \ll \text{dvd} \gg y)$

is *comm-ring-1-class.dvd-minus-iff*{proof}

tts-lemma *dvd-diff*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x \ll \text{dvd} \gg y$ **and** $x \ll \text{dvd} \gg z$

shows $x \ll \text{dvd} \gg y \text{ }_{-ow} z$

is *comm-ring-1-class.dvd-diff*{proof}

end

end

3.12 Relativization of the results about semilattices

3.12.1 Commutative bands

Definitions and common properties

```
locale semilattice-ow = abel-semigroup-ow U f
  for U :: 'al set and f +
  assumes idem[simp]:  $x \in U \implies x *_{ow} x = x$ 
```

```
locale semilattice-set-ow =
  semilattice-ow U f for U :: 'al set and f (infixl <*_ow> 70)
```

Transfer rules

context

```
includes lifting-syntax
```

begin

```
lemma semilattice-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique A right-total A
  shows
    ((A ==> A ==> A) ==> (=))
    ( $\lambda f. \text{semilattice-ow } (\text{Collect } (\text{Domainp } A)) f \text{ semilattice}$ 
    <proof>
```

```
lemma semilattice-set-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique A right-total A
  shows
    ((A ==> A ==> A) ==> (=))
    ( $\lambda f. \text{semilattice-set-ow } (\text{Collect } (\text{Domainp } A)) f \text{ semilattice-set}$ 
    <proof>
```

end

Relativization

context semilattice-ow

begin

tts-context

```
tts: (?'a to U)
```

```
substituting semilattice-ow-axioms
```

```
eliminating through simp
```

begin

tts-lemma left-idem:

```
assumes  $a \in U$  and  $b \in U$ 
```

```
shows  $a *_{ow} (a *_{ow} b) = a *_{ow} b$ 
```

```
is semilattice.left-idem<proof>
```

tts-lemma right-idem:

```
assumes  $a \in U$  and  $b \in U$ 
```

```
shows  $a *_{ow} b *_{ow} b = a *_{ow} b$ 
```

```
is semilattice.right-idem<proof>
```

end

end

3.12.2 Simple upper and lower semilattices

Definitions and common properties

locale *semilattice-order-ow* = *semilattice-ow* *U f*

for *U* :: 'al set **and** *f* +

fixes *le* :: ['al, 'al] ⇒ bool (**infix** <≤_{ow}> 50)

and *ls* :: ['al, 'al] ⇒ bool (**infix** <<_{ow}> 50)

assumes *order-iff*: [[*a* ∈ *U*; *b* ∈ *U*]] ⇒ $a \leq_{ow} b \leftrightarrow a = a *_{ow} b$

and *strict-order-iff*: [[*a* ∈ *U*; *b* ∈ *U*]] ⇒ $a <_{ow} b \leftrightarrow a = a *_{ow} b \wedge a \neq b$

begin

sublocale *ordering-ow* *U* <(≤_{ow})> <<_{ow}>

<proof>

notation *le* (**infix** <≤_{ow}> 50)

and *ls* (**infix** <<_{ow}> 50)

end

locale *semilattice-order-set-ow* =

semilattice-order-ow *U f le ls* + *semilattice-set-ow* *U f*

for *U* :: 'al set **and** *f le ls*

locale *inf-ow* =

fixes *U* :: 'al set **and** *inf* (**infixl** <⊔_{ow}> 70)

assumes *inf-closed[simp]*: [[*x* ∈ *U*; *y* ∈ *U*]] ⇒ $x \sqcap_{ow} y \in U$

begin

notation *inf* (**infixl** <⊔_{ow}> 70)

lemma *inf-closed'[simp]*: $\forall x \in U. \forall y \in U. x \sqcap_{ow} y \in U$ <proof>

end

locale *inf-pair-ow* = *inf*₁: *inf-ow* *U*₁ *inf*₁ + *inf*₂: *inf-ow* *U*₂ *inf*₂

for *U*₁ :: 'al set **and** *inf*₁

and *U*₂ :: 'bl set **and** *inf*₂

begin

notation *inf*₁ (**infixl** <⊔_{ow.1}> 70)

notation *inf*₂ (**infixl** <⊔_{ow.2}> 70)

end

locale *semilattice-inf-ow* = *inf-ow* *U inf* + *order-ow* *U le ls*

for *U* :: 'al set **and** *inf le ls* +

assumes *inf-le1[simp]*: [[*x* ∈ *U*; *y* ∈ *U*]] ⇒ $x \sqcap_{ow} y \leq_{ow} x$

and *inf-le2[simp]*: [[*x* ∈ *U*; *y* ∈ *U*]] ⇒ $x \sqcap_{ow} y \leq_{ow} y$

and *inf-greatest*:

[[*x* ∈ *U*; *y* ∈ *U*; *z* ∈ *U*; $x \leq_{ow} y$; $x \leq_{ow} z$]] ⇒ $x \leq_{ow} y \sqcap_{ow} z$

begin

sublocale *inf*: *semilattice-order-ow* *U* <(⊔_{ow})> <(≤_{ow})> <<_{ow}>

<proof>

sublocale *Inf-fin*: *semilattice-order-set-ow* *U* <(⊔_{ow})> <(≤_{ow})> <<_{ow}> <proof>

end

```

locale semilattice-inf-pair-ow =
  sl-inf1: semilattice-inf-ow U1 inf1 le1 ls1 +
  sl-inf2: semilattice-inf-ow U2 inf2 le2 ls2
  for U1 :: 'al set and inf1 le1 ls1
    and U2 :: 'bl set and inf2 le2 ls2
begin

sublocale inf-pair-ow ⟨proof⟩
sublocale order-pair-ow ⟨proof⟩

end

locale sup-ow =
  fixes U :: 'ao set and sup :: ['ao, 'ao] ⇒ 'ao (infixl ⟨⊔ow⟩ 70)
  assumes sup-closed[simp]: [[ x ∈ U; y ∈ U ]] ⇒ sup x y ∈ U
begin

notation sup (infixl ⟨⊔ow⟩ 70)

lemma sup-closed'[simp]: ∀ x ∈ U. ∀ y ∈ U. x ⊔ow y ∈ U ⟨proof⟩

end

locale sup-pair-ow = sup1: sup-ow U1 sup1 + sup2: sup-ow U2 sup2
  for U1 :: 'al set and sup1
    and U2 :: 'bl set and sup2
begin

notation sup1 (infixl ⟨⊔ow.1⟩ 70)
notation sup2 (infixl ⟨⊔ow.2⟩ 70)

end

locale semilattice-sup-ow = sup-ow U sup + order-ow U le ls
  for U :: 'al set and sup le ls +
  assumes sup-ge1[simp]: [[ x ∈ U; y ∈ U ]] ⇒ x ≤ow x ⊔ow y
    and sup-ge2[simp]: [[ y ∈ U; x ∈ U ]] ⇒ y ≤ow x ⊔ow y
    and sup-least:
      [[ y ∈ U; x ∈ U; z ∈ U; y ≤ow x; z ≤ow x ]] ⇒ y ⊔ow z ≤ow x
begin

sublocale sup: semilattice-order-ow U ⟨(⊔ow)⟩ ⟨(≥ow)⟩ ⟨(>ow)⟩
  ⟨proof⟩

sublocale Sup-fin: semilattice-order-set-ow U sup (≥ow) (>ow) ⟨proof⟩

end

locale semilattice-sup-pair-ow =
  sl-sup1: semilattice-sup-ow U1 sup1 le1 ls1 +
  sl-sup2: semilattice-sup-ow U2 sup2 le2 ls2
  for U1 :: 'al set and sup1 le1 ls1
    and U2 :: 'bl set and sup2 le2 ls2
begin

sublocale sup-pair-ow ⟨proof⟩

```

sublocale *order-pair-ow* *<proof>*

end

Transfer rules

context

includes *lifting-syntax*

begin

lemma *semilattice-order-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(
 $(A \text{ ===> } A \text{ ===> } A) \text{ ===>}$
 $(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$
 $(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$
 $(=)$
) (*semilattice-order-ow (Collect (Domainp A)) semilattice-order*
<proof>)

lemma *semilattice-order-set-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(
 $(A \text{ ===> } A \text{ ===> } A) \text{ ===>}$
 $(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$
 $(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$
 $(=)$
) (*semilattice-order-set-ow (Collect (Domainp A)) semilattice-order-set*
<proof>)

lemma *semilattice-inf-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(
 $(A \text{ ===> } A \text{ ===> } A) \text{ ===>}$
 $(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$
 $(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$
 $(=)$
) (*semilattice-inf-ow (Collect (Domainp A)) class.semilattice-inf*
<proof>)

lemma *semilattice-sup-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(
 $(A \text{ ===> } A \text{ ===> } A) \text{ ===>}$
 $(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$
 $(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$
 $(=)$
) (*semilattice-sup-ow (Collect (Domainp A)) class.semilattice-sup*
<proof>)

end

Relativization

context *semilattice-order-ow*

begin

tts-context

tts: (*?'a to U*)

rewriting *ctr-simps*

substituting *semilattice-order-ow-axioms*

eliminating through *simp*

begin

tts-lemma *cobounded1:*

assumes $a \in U$ **and** $b \in U$

shows $a *_{ow} b \leq_{ow} a$

is *semilattice-order.cobounded1* \langle *proof* \rangle

tts-lemma *cobounded2:*

assumes $a \in U$ **and** $b \in U$

shows $a *_{ow} b \leq_{ow} b$

is *semilattice-order.cobounded2* \langle *proof* \rangle

tts-lemma *absorb-iff1:*

assumes $a \in U$ **and** $b \in U$

shows $(a \leq_{ow} b) = (a *_{ow} b = a)$

is *semilattice-order.absorb-iff1* \langle *proof* \rangle

tts-lemma *absorb-iff2:*

assumes $b \in U$ **and** $a \in U$

shows $(b \leq_{ow} a) = (a *_{ow} b = b)$

is *semilattice-order.absorb-iff2* \langle *proof* \rangle

tts-lemma *strict-coboundedI1:*

assumes $a \in U$ **and** $c \in U$ **and** $b \in U$ **and** $a <_{ow} c$

shows $a *_{ow} b <_{ow} c$

is *semilattice-order.strict-coboundedI1* \langle *proof* \rangle

tts-lemma *strict-coboundedI2:*

assumes $b \in U$ **and** $c \in U$ **and** $a \in U$ **and** $b <_{ow} c$

shows $a *_{ow} b <_{ow} c$

is *semilattice-order.strict-coboundedI2* \langle *proof* \rangle

tts-lemma *absorb1:*

assumes $a \in U$ **and** $b \in U$ **and** $a \leq_{ow} b$

shows $a *_{ow} b = a$

is *semilattice-order.absorb1* \langle *proof* \rangle

tts-lemma *coboundedI1:*

assumes $a \in U$ **and** $c \in U$ **and** $b \in U$ **and** $a \leq_{ow} c$

shows $a *_{ow} b \leq_{ow} c$

is *semilattice-order.coboundedI1* \langle *proof* \rangle

tts-lemma *absorb2:*

assumes $b \in U$ **and** $a \in U$ **and** $b \leq_{ow} a$

shows $a *_{ow} b = b$

is *semilattice-order.absorb2* \langle *proof* \rangle

tts-lemma *coboundedI2:*

assumes $b \in U$ and $c \in U$ and $a \in U$ and $b \leq_{ow} c$
 shows $a *_{ow} b \leq_{ow} c$
 is *semilattice-order.coboundedI2*{proof}

tts-lemma *orderI*:

assumes $a \in U$ and $b \in U$ and $a = a *_{ow} b$
 shows $a \leq_{ow} b$
 is *semilattice-order.orderI*{proof}

tts-lemma *bounded-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$
 shows $(a \leq_{ow} b *_{ow} c) = (a \leq_{ow} b \wedge a \leq_{ow} c)$
 is *semilattice-order.bounded-iff*{proof}

tts-lemma *boundedI*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $a \leq_{ow} b$ and $a \leq_{ow} c$
 shows $a \leq_{ow} b *_{ow} c$
 is *semilattice-order.boundedI*{proof}

tts-lemma *orderE*:

assumes $a \in U$ and $b \in U$ and $a \leq_{ow} b$ and $a = a *_{ow} b \implies thesis$
 shows *thesis*
 is *semilattice-order.orderE*{proof}

tts-lemma *mono*:

assumes $a \in U$
 and $c \in U$
 and $b \in U$
 and $d \in U$
 and $a \leq_{ow} c$
 and $b \leq_{ow} d$
 shows $a *_{ow} b \leq_{ow} c *_{ow} d$
 is *semilattice-order.mono*{proof}

tts-lemma *strict-boundedE*:

assumes $a \in U$
 and $b \in U$
 and $c \in U$
 and $a <_{ow} b *_{ow} c$
 obtains $a <_{ow} b$ and $a <_{ow} c$
 given *semilattice-order.strict-boundedE* {proof}

tts-lemma *boundedE*:

assumes $a \in U$
 and $b \in U$
 and $c \in U$
 and $a \leq_{ow} b *_{ow} c$
 obtains $a \leq_{ow} b$ and $a \leq_{ow} c$
 given *semilattice-order.boundedE* {proof}

end

end

context *semilattice-inf-ow*
 begin

tts-context

tts: (*?a to U*)
rewriting *ctr-simps*
substituting *semilattice-inf-ow-axioms*
eliminating through *simp*
begin

tts-lemma *le-iff-inf*:
assumes $x \in U$ **and** $y \in U$
shows $(x \leq_{ow} y) = (x \sqcap_{ow} y = x)$
is *semilattice-inf-class.le-iff-inf*{*proof*}

tts-lemma *less-infI1*:
assumes $a \in U$ **and** $x \in U$ **and** $b \in U$ **and** $a <_{ow} x$
shows $a \sqcap_{ow} b <_{ow} x$
is *semilattice-inf-class.less-infI1*{*proof*}

tts-lemma *less-infI2*:
assumes $b \in U$ **and** $x \in U$ **and** $a \in U$ **and** $b <_{ow} x$
shows $a \sqcap_{ow} b <_{ow} x$
is *semilattice-inf-class.less-infI2*{*proof*}

tts-lemma *le-infI1*:
assumes $a \in U$ **and** $x \in U$ **and** $b \in U$ **and** $a \leq_{ow} x$
shows $a \sqcap_{ow} b \leq_{ow} x$
is *semilattice-inf-class.le-infI1*{*proof*}

tts-lemma *le-infI2*:
assumes $b \in U$ **and** $x \in U$ **and** $a \in U$ **and** $b \leq_{ow} x$
shows $a \sqcap_{ow} b \leq_{ow} x$
is *semilattice-inf-class.le-infI2*{*proof*}

tts-lemma *le-inf-iff*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(x \leq_{ow} y \sqcap_{ow} z) = (x \leq_{ow} y \wedge x \leq_{ow} z)$
is *semilattice-inf-class.le-inf-iff*{*proof*}

tts-lemma *le-infI*:
assumes $x \in U$ **and** $a \in U$ **and** $b \in U$ **and** $x \leq_{ow} a$ **and** $x \leq_{ow} b$
shows $x \leq_{ow} a \sqcap_{ow} b$
is *semilattice-inf-class.le-infI*{*proof*}

tts-lemma *le-infE*:
assumes $x \in U$
and $a \in U$
and $b \in U$
and $x \leq_{ow} a \sqcap_{ow} b$
and $[[x \leq_{ow} a; x \leq_{ow} b]] \implies P$
shows P
is *semilattice-inf-class.le-infE*{*proof*}

tts-lemma *inf-mono*:
assumes $a \in U$
and $c \in U$
and $b \in U$
and $d \in U$
and $a \leq_{ow} c$
and $b \leq_{ow} d$
shows $a \sqcap_{ow} b \leq_{ow} c \sqcap_{ow} d$

```

    is semilattice-inf-class.inf-mono{proof}

end

end

context semilattice-sup-ow
begin

tts-context
  tts: (?a to U)
  rewriting ctr-simps
  substituting semilattice-sup-ow-axioms
  eliminating through simp
begin

tts-lemma le-iff-sup:
  assumes  $x \in U$  and  $y \in U$ 
  shows  $(x \leq_{ow} y) = (x \sqcup_{ow} y = y)$ 
  is semilattice-sup-class.le-iff-sup{proof}

tts-lemma less-supI1:
  assumes  $x \in U$  and  $a \in U$  and  $b \in U$  and  $x <_{ow} a$ 
  shows  $x <_{ow} a \sqcup_{ow} b$ 
  is semilattice-sup-class.less-supI1{proof}

tts-lemma less-supI2:
  assumes  $x \in U$  and  $b \in U$  and  $a \in U$  and  $x <_{ow} b$ 
  shows  $x <_{ow} a \sqcup_{ow} b$ 
  is semilattice-sup-class.less-supI2{proof}

tts-lemma le-supI1:
  assumes  $x \in U$  and  $a \in U$  and  $b \in U$  and  $x \leq_{ow} a$ 
  shows  $x \leq_{ow} a \sqcup_{ow} b$ 
  is semilattice-sup-class.le-supI1{proof}

tts-lemma le-supI2:
  assumes  $x \in U$  and  $b \in U$  and  $a \in U$  and  $x \leq_{ow} b$ 
  shows  $x \leq_{ow} a \sqcup_{ow} b$ 
  is semilattice-sup-class.le-supI2{proof}

tts-lemma le-sup-iff:
  assumes  $x \in U$  and  $y \in U$  and  $z \in U$ 
  shows  $(x \sqcup_{ow} y \leq_{ow} z) = (x \leq_{ow} z \wedge y \leq_{ow} z)$ 
  is semilattice-sup-class.le-sup-iff{proof}

tts-lemma le-supI:
  assumes  $a \in U$  and  $x \in U$  and  $b \in U$  and  $a \leq_{ow} x$  and  $b \leq_{ow} x$ 
  shows  $a \sqcup_{ow} b \leq_{ow} x$ 
  is semilattice-sup-class.le-supI{proof}

tts-lemma le-supE:
  assumes  $a \in U$ 
  and  $b \in U$ 
  and  $x \in U$ 
  and  $a \sqcup_{ow} b \leq_{ow} x$ 
  and  $[[a \leq_{ow} x; b \leq_{ow} x]] \implies P$ 
  shows  $P$ 

```

```

    is semilattice-sup-class.le-supE⟨proof⟩

tts-lemma sup-unique:
  assumes  $\forall x \in U. \forall y \in U. f x y \in U$ 
    and  $x \in U$ 
    and  $y \in U$ 
    and  $\wedge x y. [[x \in U; y \in U]] \implies x \leq_{ow} f x y$ 
    and  $\wedge x y. [[x \in U; y \in U]] \implies y \leq_{ow} f x y$ 
    and  $\wedge x y z. [[x \in U; y \in U; z \in U; y \leq_{ow} x; z \leq_{ow} x]] \implies f y z \leq_{ow} x$ 
  shows  $x \sqcup_{ow} y = f x y$ 
  is semilattice-sup-class.sup-unique⟨proof⟩

```

```

tts-lemma sup-mono:
  assumes  $a \in U$ 
    and  $c \in U$ 
    and  $b \in U$ 
    and  $d \in U$ 
    and  $a \leq_{ow} c$ 
    and  $b \leq_{ow} d$ 
  shows  $a \sqcup_{ow} b \leq_{ow} c \sqcup_{ow} d$ 
  is semilattice-sup-class.sup-mono⟨proof⟩

```

end

end

3.12.3 Bounded semilattices

Definitions and common properties

```

locale semilattice-neutral-ow = semilattice-ow U f + comm-monoid-ow U f z
  for U :: 'al set and f z

```

```

locale semilattice-neutral-order-ow =
  sl-neut: semilattice-neutral-ow U f z +
  sl-ord: semilattice-order-ow U f le ls
  for U :: 'al set and f z le ls

```

begin

```

sublocale order-top-ow U <( $\leq_{ow}$ )> <( $<_{ow}$ )> < $\mathbf{1}_{ow}$ >
  ⟨proof⟩

```

end

```

locale bounded-semilattice-inf-top-ow =
  semilattice-inf-ow U inf le ls + order-top-ow U le ls top
  for U :: 'al set and inf le ls top

```

begin

```

sublocale inf-top: semilattice-neutral-order-ow U <( $\sqcap_{ow}$ )> < $\top_{ow}$ > <( $\leq_{ow}$ )> <( $<_{ow}$ )>
  ⟨proof⟩

```

end

```

locale bounded-semilattice-sup-bot-ow =
  semilattice-sup-ow U sup le ls + order-bot-ow U bot le ls
  for U :: 'al set and sup le ls bot

```

begin

sublocale *sup-bot: semilattice-neutral-order-ow* $U \langle (\sqcup_{ow}) \rangle \langle \perp_{ow} \rangle \langle (\geq_{ow}) \rangle \langle (>_{ow}) \rangle$
 $\langle \text{proof} \rangle$

end

Transfer rules

context

includes *lifting-syntax*

begin

lemma *semilattice-neutral-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$((A \text{ ===> } A \text{ ===> } A) \text{ ===> } A \text{ ===> } (=))$

$(\text{semilattice-neutral-ow } (\text{Collect } (\text{Domainp } A))) \text{ semilattice-neutr}$
 $\langle \text{proof} \rangle$

lemma *semilattice-neutr-order-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(

$(A \text{ ===> } A \text{ ===> } A) \text{ ===>}$

$A \text{ ===>}$

$(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$

$(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$

$(=)$

)

$(\text{semilattice-neutral-order-ow } (\text{Collect } (\text{Domainp } A)))$

semilattice-neutr-order

$\langle \text{proof} \rangle$

lemma *bounded-semilattice-inf-top-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(

$(A \text{ ===> } A \text{ ===> } A) \text{ ===>}$

$(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$

$(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$

$A \text{ ===>}$

$(=)$

)

$(\text{bounded-semilattice-inf-top-ow } (\text{Collect } (\text{Domainp } A)))$

class.bounded-semilattice-inf-top

$\langle \text{proof} \rangle$

lemma *bounded-semilattice-sup-bot-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(

$(A \text{ ===> } A \text{ ===> } A) \text{ ===>}$

$(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$

$(A \text{ ===> } A \text{ ===> } (=)) \text{ ===>}$

$A \text{ ===>}$

$(=)$

)

$(\text{bounded-semilattice-sup-bot-ow } (\text{Collect } (\text{Domainp } A)))$

```
class.bounded-semilattice-sup-bot  
<proof>
```

```
end
```

3.13 Relativization of the results about lattices

3.13.1 Simple lattices

Definitions and common properties

```

locale lattice-ow =
  semilattice-inf-ow U inf le ls + semilattice-sup-ow U sup le ls
  for U :: 'a list and inf le ls sup

```

```

locale lattice-pair-ow =
  lattice1: lattice-ow U1 inf1 le1 ls1 sup1 +
  lattice2: lattice-ow U2 inf2 le2 ls2 sup2
  for U1 :: 'a list and inf1 le1 ls1 sup1
     and U2 :: 'b list and inf2 le2 ls2 sup2
begin

```

```

sublocale semilattice-inf-pair-ow <proof>
sublocale semilattice-sup-pair-ow <proof>

```

end

Transfer rules

```

context
  includes lifting-syntax
begin

```

```

lemma lattice-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique A right-total A
  shows
    (
      (A =====> A =====> A) =====>
      (A =====> A =====> (=)) =====>
      (A =====> A =====> (=)) =====>
      (A =====> A =====> A) =====>
      (=)
    )
    (lattice-ow (Collect (Domainp A))) class.lattice
  <proof>

```

end

Relativization

```

context lattice-ow
begin

```

```

tts-context
  tts: (?'a to U)
  rewriting ctr-simps
  substituting lattice-ow-axioms
  eliminating through simp
begin

```

```

tts-lemma inf-sup-aci:
  assumes x ∈ U and y ∈ U
  shows
    x ⊔ow y = y ⊔ow x

```

$z \in U \implies x \sqcap_{ow} y \sqcap_{ow} z = x \sqcap_{ow} (y \sqcap_{ow} z)$
 $z \in U \implies x \sqcap_{ow} (y \sqcap_{ow} z) = y \sqcap_{ow} (x \sqcap_{ow} z)$
 $x \sqcap_{ow} (x \sqcap_{ow} y) = x \sqcap_{ow} y$
 $x \sqcup_{ow} y = y \sqcup_{ow} x$
 $z \in U \implies x \sqcup_{ow} y \sqcup_{ow} z = x \sqcup_{ow} (y \sqcup_{ow} z)$
 $z \in U \implies x \sqcup_{ow} (y \sqcup_{ow} z) = y \sqcup_{ow} (x \sqcup_{ow} z)$
 $x \sqcup_{ow} (x \sqcup_{ow} y) = x \sqcup_{ow} y$
is *lattice-class.inf-sup-aci*{proof}

tts-lemma *inf-sup-absorb*:

assumes $x \in U$ **and** $y \in U$
shows $x \sqcap_{ow} (x \sqcup_{ow} y) = x$
is *lattice-class.inf-sup-absorb*{proof}

tts-lemma *sup-inf-absorb*:

assumes $x \in U$ **and** $y \in U$
shows $x \sqcup_{ow} (x \sqcap_{ow} y) = x$
is *lattice-class.sup-inf-absorb*{proof}

tts-lemma *bdd-above-insert*:

assumes $a \in U$ **and** $A \subseteq U$
shows *local.bdd-above* (*insert a A*) = *local.bdd-above A*
is *lattice-class.bdd-above-insert*{proof}

tts-lemma *bdd-below-insert*:

assumes $a \in U$ **and** $A \subseteq U$
shows *local.bdd-below* (*insert a A*) = *local.bdd-below A*
is *lattice-class.bdd-below-insert*{proof}

tts-lemma *distrib-sup-le*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $x \sqcup_{ow} (y \sqcap_{ow} z) \leq_{ow} x \sqcup_{ow} y \sqcap_{ow} (x \sqcup_{ow} z)$
is *lattice-class.distrib-sup-le*{proof}

tts-lemma *distrib-inf-le*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $x \sqcap_{ow} y \sqcup_{ow} (x \sqcap_{ow} z) \leq_{ow} x \sqcap_{ow} (y \sqcup_{ow} z)$
is *lattice-class.distrib-inf-le*{proof}

tts-lemma *distrib-imp1*:

assumes $x \in U$
and $y \in U$
and $z \in U$
and
 $\bigwedge x y z. \llbracket x \in U; y \in U; z \in U \rrbracket \implies$
 $x \sqcap_{ow} (y \sqcup_{ow} z) = x \sqcap_{ow} y \sqcup_{ow} (x \sqcap_{ow} z)$
shows $x \sqcup_{ow} (y \sqcap_{ow} z) = x \sqcup_{ow} y \sqcap_{ow} (x \sqcup_{ow} z)$
is *lattice-class.distrib-imp1*{proof}

tts-lemma *distrib-imp2*:

assumes $x \in U$
and $y \in U$
and $z \in U$
and
 $\bigwedge x y z. \llbracket x \in U; y \in U; z \in U \rrbracket \implies$
 $x \sqcup_{ow} (y \sqcap_{ow} z) = x \sqcup_{ow} y \sqcap_{ow} (x \sqcup_{ow} z)$
shows $x \sqcap_{ow} (y \sqcup_{ow} z) = x \sqcap_{ow} y \sqcup_{ow} (x \sqcap_{ow} z)$
is *lattice-class.distrib-imp2*{proof}

tts-lemma *bdd-above-Un*:

assumes $A \subseteq U$ **and** $B \subseteq U$

shows $local.bdd-above (A \cup B) = (local.bdd-above A \wedge local.bdd-above B)$

is *lattice-class.bdd-above-Un*{proof}

tts-lemma *bdd-below-Un*:

assumes $A \subseteq U$ **and** $B \subseteq U$

shows $local.bdd-below (A \cup B) = (local.bdd-below A \wedge local.bdd-below B)$

is *lattice-class.bdd-below-Un*{proof}

tts-lemma *bdd-above-image-sup*:

assumes $range f \subseteq U$ **and** $range g \subseteq U$

shows $local.bdd-above ((\lambda x. f x \sqcup_{ow} g x) ' A) =$
 $(local.bdd-above (f ' A) \wedge local.bdd-above (g ' A))$

is *lattice-class.bdd-above-image-sup*{proof}

tts-lemma *bdd-below-image-inf*:

assumes $range f \subseteq U$ **and** $range g \subseteq U$

shows $local.bdd-below ((\lambda x. f x \sqcap_{ow} g x) ' A) =$
 $(local.bdd-below (f ' A) \wedge local.bdd-below (g ' A))$

is *lattice-class.bdd-below-image-inf*{proof}

end

tts-context

tts: (*?a* to U)

rewriting *ctr-simps*

substituting *lattice-ow-axioms*

eliminating through *simp*

begin

tts-lemma *bdd-above-UN*:

assumes $U \neq \{\}$ **and** $range A \subseteq Pow U$ **and** *finite I*

shows $bdd-above (\bigcup (A ' I)) = (\forall x \in I. bdd-above (A x))$

is *lattice-class.bdd-above-UN* {proof}

tts-lemma *bdd-below-UN*:

assumes $U \neq \{\}$ **and** $range A \subseteq Pow U$ **and** *finite I*

shows $local.bdd-below (\bigcup (A ' I)) = (\forall x \in I. local.bdd-below (A x))$

is *lattice-class.bdd-below-UN*{proof}

tts-lemma *bdd-above-finite*:

assumes $U \neq \{\}$ **and** $A \subseteq U$ **and** *finite A*

shows $local.bdd-above A$

is *lattice-class.bdd-above-finite*{proof}

tts-lemma *bdd-below-finite*:

assumes $U \neq \{\}$ **and** $A \subseteq U$ **and** *finite A*

shows $local.bdd-below A$

is *lattice-class.bdd-below-finite*{proof}

end

end

3.13.2 Bounded lattices

Definitions and common properties

locale *bounded-lattice-bot-ow* =

lattice-ow U inf le ls sup + order-bot-ow U bot le ls

for *U* :: 'al set **and** *inf le ls sup bot*

begin

sublocale *bounded-semilattice-sup-bot-ow U* $\langle(\sqcup_{ow})\rangle$ $\langle(\leq_{ow})\rangle$ $\langle(<_{ow})\rangle$ $\langle\perp_{ow}\rangle$ $\langle proof \rangle$

end

locale *bounded-lattice-bot-pair-ow* =

blb₁: bounded-lattice-bot-ow U₁ inf₁ le₁ ls₁ sup₁ bot₁ +

blb₂: bounded-lattice-bot-ow U₂ inf₂ le₂ ls₂ sup₂ bot₂

for *U₁* :: 'al set **and** *inf₁ le₁ ls₁ sup₁ bot₁*

and *U₂* :: 'bl set **and** *inf₂ le₂ ls₂ sup₂ bot₂*

begin

sublocale *lattice-pair-ow* $\langle proof \rangle$

sublocale *order-bot-pair-ow U₁ bot₁ le₁ ls₁ U₂ bot₂ le₂ ls₂* $\langle proof \rangle$

end

locale *bounded-lattice-top-ow* =

lattice-ow U inf le ls sup + order-top-ow U le ls top

for *U* :: 'al set **and** *inf le ls sup top*

begin

sublocale *bounded-semilattice-inf-top-ow U* $\langle(\sqcap_{ow})\rangle$ $\langle(\leq_{ow})\rangle$ $\langle(<_{ow})\rangle$ $\langle\top_{ow}\rangle$ $\langle proof \rangle$

end

locale *bounded-lattice-top-pair-ow* =

blb₁: bounded-lattice-top-ow U₁ inf₁ le₁ ls₁ sup₁ top₁ +

blb₂: bounded-lattice-top-ow U₂ inf₂ le₂ ls₂ sup₂ top₂

for *U₁* :: 'al set **and** *inf₁ le₁ ls₁ sup₁ top₁*

and *U₂* :: 'bl set **and** *inf₂ le₂ ls₂ sup₂ top₂*

begin

sublocale *lattice-pair-ow* $\langle proof \rangle$

sublocale *order-top-pair-ow U₁ le₁ ls₁ top₁ U₂ le₂ ls₂ top₂* $\langle proof \rangle$

end

locale *bounded-lattice-ow* =

lattice-ow U inf le ls sup +

order-bot-ow U bot le ls +

order-top-ow U le ls top

for *U* :: 'al set **and** *inf le ls sup bot top*

begin

sublocale *bounded-lattice-bot-ow U* $\langle(\sqcap_{ow})\rangle$ $\langle(\leq_{ow})\rangle$ $\langle(<_{ow})\rangle$ $\langle(\sqcup_{ow})\rangle$ $\langle\perp_{ow}\rangle$ $\langle proof \rangle$

sublocale *bounded-lattice-top-ow U* $\langle(\sqcap_{ow})\rangle$ $\langle(\leq_{ow})\rangle$ $\langle(<_{ow})\rangle$ $\langle(\sqcup_{ow})\rangle$ $\langle\top_{ow}\rangle$ $\langle proof \rangle$

end

locale *bounded-lattice-pair-ow* =

```

bl1: bounded-lattice-ow U1 inf1 le1 ls1 sup1 bot1 top1 +
bl2: bounded-lattice-ow U2 inf2 le2 ls2 sup2 bot2 top2
for U1 :: 'al set and inf1 le1 ls1 sup1 bot1 top1
and U2 :: 'bl set and inf2 le2 ls2 sup2 bot2 top2
begin

sublocale bounded-lattice-bot-pair-ow ⟨proof⟩
sublocale bounded-lattice-top-pair-ow ⟨proof⟩

end

```

Transfer rules

context

includes *lifting-syntax*

begin

lemma *bounded-lattice-bot-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 (
 (*A* ==> *A* ==> *A*) ==>
 (*A* ==> *A* ==> (=)) ==>
 (*A* ==> *A* ==> (=)) ==>
 (*A* ==> *A* ==> *A*) ==>
A ==>
 (=)
)
 (*bounded-lattice-bot-ow* (*Collect* (*Domainp A*)))
class.bounded-lattice-bot
 ⟨*proof*⟩

lemma *bounded-lattice-top-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 (
 (*A* ==> *A* ==> *A*) ==>
 (*A* ==> *A* ==> (=)) ==>
 (*A* ==> *A* ==> (=)) ==>
 (*A* ==> *A* ==> *A*) ==>
A ==>
 (=)
)
 (*bounded-lattice-top-ow* (*Collect* (*Domainp A*)))
class.bounded-lattice-top
 ⟨*proof*⟩

lemma *bounded-lattice-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 (
 (*A* ==> *A* ==> *A*) ==>
 (*A* ==> *A* ==> (=)) ==>
 (*A* ==> *A* ==> (=)) ==>
 (*A* ==> *A* ==> *A*) ==>
A ==>
A ==>

```

    (=)
  )
  (bounded-lattice-ow (Collect (Domainp A))) class.bounded-lattice
  <proof>

```

end

Relativization

```

context bounded-lattice-bot-ow
begin

```

```

tts-context

```

```

  tts: (?a to U)

```

```

  rewriting ctr-simps

```

```

  substituting bounded-lattice-bot-ow-axioms and sup-bot.sl-neut.not-empty

```

```

  applying [OF inf-closed' sup-closed' bot-closed]

```

```

begin

```

```

tts-lemma inf-bot-left:

```

```

  assumes  $x \in U$ 

```

```

  shows  $\perp_{ow} \sqcap_{ow} x = \perp_{ow}$ 

```

```

  is bounded-lattice-bot-class.inf-bot-left<proof>

```

```

tts-lemma inf-bot-right:

```

```

  assumes  $x \in U$ 

```

```

  shows  $x \sqcap_{ow} \perp_{ow} = \perp_{ow}$ 

```

```

  is bounded-lattice-bot-class.inf-bot-right<proof>

```

end

end

```

context bounded-lattice-top-ow
begin

```

```

tts-context

```

```

  tts: (?a to U)

```

```

  rewriting ctr-simps

```

```

  substituting bounded-lattice-top-ow-axioms and inf-top.sl-neut.not-empty

```

```

  applying [OF inf-closed' sup-closed' top-closed]

```

```

begin

```

```

tts-lemma sup-top-left:

```

```

  assumes  $x \in U$ 

```

```

  shows  $\top_{ow} \sqcup_{ow} x = \top_{ow}$ 

```

```

  is bounded-lattice-top-class.sup-top-left<proof>

```

```

tts-lemma sup-top-right:

```

```

  assumes  $x \in U$ 

```

```

  shows  $x \sqcup_{ow} \top_{ow} = \top_{ow}$ 

```

```

  is bounded-lattice-top-class.sup-top-right<proof>

```

end

end

```

context bounded-lattice-ow

```

begin

tts-context

tts: (*'a* to *U*)

rewriting *ctr-simps*

substituting *bounded-lattice-ow-axioms*

applying [*OF sup-bot.sl-neut.not-empty, simplified*]

begin

tts-lemma *atLeastAtMost-eq-UNIV-iff*:

assumes $x \in U$ **and** $y \in U$

shows $(\{x.._{ow}y\} = U) = (x = \perp_{ow} \wedge y = \top_{ow})$

is *bounded-lattice-class.atLeastAtMost-eq-UNIV-iff* {*proof*}

end

end

3.13.3 Distributive lattices

Definitions and common properties

locale *distrib-lattice-ow* =

lattice-ow U inf le ls sup for U :: 'a set **and** *inf le ls sup +*

assumes *sup-inf-distrib1*:

$\llbracket x \in U; y \in U; z \in U \rrbracket \implies x \sqcup_{ow} (y \sqcap_{ow} z) = (x \sqcup_{ow} y) \sqcap_{ow} (x \sqcup_{ow} z)$

Transfer rules

context

includes *lifting-syntax*

begin

lemma *distrib-lattice-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(
 $(A \implies A \implies A) \implies$
 $(A \implies A \implies (=)) \implies$
 $(A \implies A \implies (=)) \implies$
 $(A \implies A \implies A) \implies$
 $(=)$
)

(*distrib-lattice-ow (Collect (Domainp A))*) *class.distrib-lattice*

{*proof*}

end

Relativization

context *distrib-lattice-ow*

begin

tts-context

tts: (*'a* to *U*)

rewriting *ctr-simps*

substituting *distrib-lattice-ow-axioms*

eliminating through *simp*

begin

tts-lemma *inf-sup-distrib1*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $x \sqcap_{ow} (y \sqcup_{ow} z) = x \sqcap_{ow} y \sqcup_{ow} (x \sqcap_{ow} z)$
is *distrib-lattice-class.inf-sup-distrib1*{*proof*}

tts-lemma *inf-sup-distrib2*:
assumes $y \in U$ **and** $z \in U$ **and** $x \in U$
shows $y \sqcup_{ow} z \sqcap_{ow} x = y \sqcap_{ow} x \sqcup_{ow} (z \sqcap_{ow} x)$
is *distrib-lattice-class.inf-sup-distrib2*{*proof*}

tts-lemma *sup-inf-distrib2*:
assumes $y \in U$ **and** $z \in U$ **and** $x \in U$
shows $y \sqcap_{ow} z \sqcup_{ow} x = y \sqcup_{ow} x \sqcap_{ow} (z \sqcup_{ow} x)$
is *distrib-lattice-class.sup-inf-distrib2*{*proof*}

end

end

3.14 Relativization of the results about complete lattices

3.14.1 Simple complete lattices

Definitions and common properties

locale *Inf-ow* =

fixes $U :: 'al\ set$ **and** Inf ($\langle \sqcap_{ow} \rangle$)
assumes $Inf-closed[simp]: \sqcap_{ow} ' Pow\ U \subseteq U$

begin

notation Inf ($\langle \sqcap_{ow} \rangle$)

lemma $Inf-closed'[simp]: \forall x \subseteq U. \sqcap_{ow} x \in U$ *<proof>*

end

locale *Inf-pair-ow* = $Inf_1: Inf-ow\ U_1\ Inf_1 + Inf_2: Inf-ow\ U_2\ Inf_2$

for $U_1 :: 'al\ set$ **and** Inf_1 **and** $U_2 :: 'bl\ set$ **and** Inf_2

begin

notation Inf_1 ($\langle \sqcap_{ow.1} \rangle$)

notation Inf_2 ($\langle \sqcap_{ow.2} \rangle$)

end

locale *Sup-ow* =

fixes $U :: 'al\ set$ **and** Sup ($\langle \sqcup_{ow} \rangle$)
assumes $Sup-closed[simp]: \sqcup_{ow} ' Pow\ U \subseteq U$

begin

notation Sup ($\langle \sqcup_{ow} \rangle$)

lemma $Sup-closed'[simp]: \forall x \subseteq U. \sqcup_{ow} x \in U$ *<proof>*

end

locale *Sup-pair-ow* = $Sup_1: Sup-ow\ U_1\ Sup_1 + Sup_2: Sup-ow\ U_2\ Sup_2$

for $U_1 :: 'al\ set$ **and** Sup_1 **and** $U_2 :: 'bl\ set$ **and** Sup_2

begin

notation Sup_1 ($\langle \sqcup_{ow.1} \rangle$)

notation Sup_2 ($\langle \sqcup_{ow.2} \rangle$)

end

locale *complete-lattice-ow* =

lattice-ow U *inf* *le* *ls* *sup* +

Inf-ow U *Inf* +

Sup-ow U *Sup* +

bot-ow U *bot* +

top-ow U *top*

for $U :: 'al\ set$ **and** Inf Sup *inf* *le* *ls* *sup* *bot* *top* +

assumes $Inf-lower: \llbracket A \subseteq U; x \in A \rrbracket \Longrightarrow \sqcap_{ow} A \leq_{ow} x$

and $Inf-greatest:$

$\llbracket A \subseteq U; z \in U; (\bigwedge x. x \in A \Longrightarrow z \leq_{ow} x) \rrbracket \Longrightarrow z \leq_{ow} \sqcap_{ow} A$

and $Sup-upper: \llbracket x \in A; A \subseteq U \rrbracket \Longrightarrow x \leq_{ow} \sqcup_{ow} A$

and $Sup-least:$

$\llbracket A \subseteq U; z \in U; (\bigwedge x. x \in A \Longrightarrow x \leq_{ow} z) \rrbracket \Longrightarrow \sqcup_{ow} A \leq_{ow} z$

```

    and Inf-empty[simp]:  $\prod_{ow} \{\} = \top_{ow}$ 
    and Sup-empty[simp]:  $\sqcup_{ow} \{\} = \perp_{ow}$ 
begin
  sublocale bounded-lattice-ow U <( $\prod_{ow}$ )> <( $\leq_{ow}$ )> <( $<_{ow}$ )> <( $\sqcup_{ow}$ )> < $\perp_{ow}$ > < $\top_{ow}$ >
    <proof>

  tts-register-sbts < $\perp_{ow}$ > | U
    <proof>

  tts-register-sbts < $\top_{ow}$ > | U
    <proof>

end

```

```

locale complete-lattice-pair-ow =
  cl1: complete-lattice-ow U1 Inf1 Sup1 inf1 le1 ls1 sup1 bot1 top1 +
  cl2: complete-lattice-ow U2 Inf2 Sup2 inf2 le2 ls2 sup2 bot2 top2
  for U1 :: 'al set and Inf1 Sup1 inf1 le1 ls1 sup1 bot1 top1
    and U2 :: 'bl set and Inf2 Sup2 inf2 le2 ls2 sup2 bot2 top2
begin

  sublocale bounded-lattice-pair-ow <proof>
  sublocale Inf-pair-ow <proof>
  sublocale Sup-pair-ow <proof>

end

```

Transfer rules

```

context
  includes lifting-syntax
begin

lemma complete-lattice-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique A right-total A
  shows
    (
      (rel-set A ==> A) ==>
      (rel-set A ==> A) ==>
      (A ==> A ==> A) ==>
      (A ==> A ==> (=)) ==>
      (A ==> A ==> (=)) ==>
      (A ==> A ==> A) ==>
      A ==> A ==>
      (=)
    )
    (complete-lattice-ow (Collect (Domainp A))) class.complete-lattice
  <proof>

end

```

Relativization

```

context complete-lattice-ow
begin

tts-context

```

tts: ($?a$ to U)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* and *sup-bot.sl-neut.not-empty*
eliminating $\langle ?a \in ?A \rangle$ and $\langle ?A \subseteq ?B \rangle$ **through** *auto*
applying [*OF Inf-closed' Sup-closed' inf-closed' sup-closed'*]
begin

tts-lemma *Inf-atLeast:*

assumes $x \in U$
shows $\prod_{ow} \{x..ow\} = x$
is *complete-lattice-class.Inf-atLeast*(*proof*)

tts-lemma *Inf-atMost:*

assumes $x \in U$
shows $\prod_{ow} \{..owx\} = \perp_{ow}$
is *complete-lattice-class.Inf-atMost*(*proof*)

tts-lemma *Sup-atLeast:*

assumes $x \in U$
shows $\sqcup_{ow} \{x..ow\} = \top_{ow}$
is *complete-lattice-class.Sup-atLeast*(*proof*)

tts-lemma *Sup-atMost:*

assumes $y \in U$
shows $\sqcup_{ow} \{..owy\} = y$
is *complete-lattice-class.Sup-atMost*(*proof*)

tts-lemma *Inf-insert:*

assumes $a \in U$ and $A \subseteq U$
shows $\prod_{ow} (\text{insert } a \ A) = a \prod_{ow} \prod_{ow} A$
is *complete-lattice-class.Inf-insert*(*proof*)

tts-lemma *Sup-insert:*

assumes $a \in U$ and $A \subseteq U$
shows $\sqcup_{ow} (\text{insert } a \ A) = a \sqcup_{ow} \sqcup_{ow} A$
is *complete-lattice-class.Sup-insert*(*proof*)

tts-lemma *Inf-atMostLessThan:*

assumes $x \in U$ and $\top_{ow} <_{ow} x$
shows $\prod_{ow} \{..<_{ow}x\} = \perp_{ow}$
is *complete-lattice-class.Inf-atMostLessThan*(*proof*)

tts-lemma *Sup-greaterThanAtLeast:*

assumes $x \in U$ and $x <_{ow} \top_{ow}$
shows $\sqcup_{ow} \{x<_{ow}..\} = \top_{ow}$
is *complete-lattice-class.Sup-greaterThanAtLeast*(*proof*)

tts-lemma *le-Inf-iff:*

assumes $b \in U$ and $A \subseteq U$
shows $(b \leq_{ow} \prod_{ow} A) = \text{Ball } A ((\leq_{ow}) b)$
is *complete-lattice-class.le-Inf-iff*(*proof*)

tts-lemma *Sup-le-iff:*

assumes $A \subseteq U$ and $b \in U$
shows $(\sqcup_{ow} A \leq_{ow} b) = (\forall x \in A. x \leq_{ow} b)$
is *complete-lattice-class.Sup-le-iff*(*proof*)

tts-lemma *Inf-atLeastLessThan:*

assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $\prod_{ow} (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{x..<y\}) = x$
is *complete-lattice-class.Inf-atLeastLessThan* $\langle proof \rangle$

tts-lemma *Sup-greaterThanAtMost*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $\sqcup_{ow} \{x <_{ow} .. y\} = y$
is *complete-lattice-class.Sup-greaterThanAtMost* $\langle proof \rangle$

tts-lemma *Inf-atLeastAtMost*:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_{ow} y$
shows $\prod_{ow} \{x.._{ow} y\} = x$
is *complete-lattice-class.Inf-atLeastAtMost* $\langle proof \rangle$

tts-lemma *Sup-atLeastAtMost*:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_{ow} y$
shows $\sqcup_{ow} \{x.._{ow} y\} = y$
is *complete-lattice-class.Sup-atLeastAtMost* $\langle proof \rangle$

tts-lemma *Inf-lower2*:
assumes $A \subseteq U$ **and** $v \in U$ **and** $u \in A$ **and** $u \leq_{ow} v$
shows $\prod_{ow} A \leq_{ow} v$
is *complete-lattice-class.Inf-lower2* $\langle proof \rangle$

tts-lemma *Sup-upper2*:
assumes $A \subseteq U$ **and** $v \in U$ **and** $u \in A$ **and** $v \leq_{ow} u$
shows $v \leq_{ow} \sqcup_{ow} A$
is *complete-lattice-class.Sup-upper2* $\langle proof \rangle$

tts-lemma *Inf-less-eq*:
assumes $A \subseteq U$ **and** $u \in U$ **and** $\wedge v. v \in A \implies v \leq_{ow} u$ **and** $A \neq \{\}$
shows $\prod_{ow} A \leq_{ow} u$
given *complete-lattice-class.Inf-less-eq* $\langle proof \rangle$

tts-lemma *less-eq-Sup*:
assumes $A \subseteq U$ **and** $u \in U$ **and** $\wedge v. v \in A \implies u \leq_{ow} v$ **and** $A \neq \{\}$
shows $u \leq_{ow} \sqcup_{ow} A$
given *complete-lattice-class.less-eq-Sup* $\langle proof \rangle$

tts-lemma *Sup-eqI*:
assumes $A \subseteq U$
and $x \in U$
and $\wedge y. y \in A \implies y \leq_{ow} x$
and $\wedge y. [\![y \in U; \wedge z. z \in A \implies z \leq_{ow} y]\!] \implies x \leq_{ow} y$
shows $\sqcup_{ow} A = x$
given *complete-lattice-class.Sup-eqI*
 $\langle proof \rangle$

tts-lemma *Inf-eqI*:
assumes $A \subseteq U$
and $x \in U$
and $\wedge i. i \in A \implies x \leq_{ow} i$
and $\wedge y. [\![y \in U; \wedge i. i \in A \implies y \leq_{ow} i]\!] \implies y \leq_{ow} x$
shows $\prod_{ow} A = x$
given *complete-lattice-class.Inf-eqI*
 $\langle proof \rangle$

tts-lemma *Inf-union-distrib*:

assumes $A \subseteq U$ **and** $B \subseteq U$
shows $\prod_{ow} (A \cup B) = \prod_{ow} A \prod_{ow} \prod_{ow} B$
is *complete-lattice-class.Inf-union-distrib*{proof}

tts-lemma *Sup-union-distrib*:
assumes $A \subseteq U$ **and** $B \subseteq U$
shows $\sqcup_{ow} (A \cup B) = \sqcup_{ow} A \sqcup_{ow} \sqcup_{ow} B$
is *complete-lattice-class.Sup-union-distrib*{proof}

tts-lemma *Sup-inter-less-eq*:
assumes $A \subseteq U$ **and** $B \subseteq U$
shows $\sqcup_{ow} (A \cap B) \leq_{ow} \sqcup_{ow} A \prod_{ow} \sqcup_{ow} B$
is *complete-lattice-class.Sup-inter-less-eq*{proof}

tts-lemma *less-eq-Inf-inter*:
assumes $A \subseteq U$ **and** $B \subseteq U$
shows $\prod_{ow} A \sqcup_{ow} \prod_{ow} B \leq_{ow} \prod_{ow} (A \cap B)$
is *complete-lattice-class.less-eq-Inf-inter*{proof}

tts-lemma *Sup-subset-mono*:
assumes $B \subseteq U$ **and** $A \subseteq B$
shows $\sqcup_{ow} A \leq_{ow} \sqcup_{ow} B$
is *complete-lattice-class.Sup-subset-mono*{proof}

tts-lemma *Inf-superset-mono*:
assumes $A \subseteq U$ **and** $B \subseteq A$
shows $\prod_{ow} A \leq_{ow} \prod_{ow} B$
is *complete-lattice-class.Inf-superset-mono*{proof}

tts-lemma *Sup-bot-conv*:
assumes $A \subseteq U$
shows
 $(\sqcup_{ow} A = \perp_{ow}) = (\forall x \in A. x = \perp_{ow})$
 $(\perp_{ow} = \sqcup_{ow} A) = (\forall x \in A. x = \perp_{ow})$
is *complete-lattice-class.Sup-bot-conv*{proof}

tts-lemma *Inf-top-conv*:
assumes $A \subseteq U$
shows
 $(\prod_{ow} A = \top_{ow}) = (\forall x \in A. x = \top_{ow})$
 $(\top_{ow} = \prod_{ow} A) = (\forall x \in A. x = \top_{ow})$
is *complete-lattice-class.Inf-top-conv*{proof}

tts-lemma *Inf-le-Sup*:
assumes $A \subseteq U$ **and** $A \neq \{\}$
shows $\prod_{ow} A \leq_{ow} \sqcup_{ow} A$
is *complete-lattice-class.Inf-le-Sup*{proof}

tts-lemma *INF-UNIV-bool-expand*:
assumes *range* $A \subseteq U$
shows $\prod_{ow} (\text{range } A) = A \text{ True } \prod_{ow} A \text{ False}$
is *complete-lattice-class.INF-UNIV-bool-expand*{proof}

tts-lemma *SUP-UNIV-bool-expand*:
assumes *range* $A \subseteq U$
shows $\sqcup_{ow} (\text{range } A) = A \text{ True } \sqcup_{ow} A \text{ False}$
is *complete-lattice-class.SUP-UNIV-bool-expand*{proof}

tts-lemma *Sup-mono*:

assumes $A \subseteq U$ **and** $B \subseteq U$ **and** $\bigwedge a. a \in A \implies B \in B ((\leq_{ow}) a)$
shows $\sqcup_{ow} A \leq_{ow} \sqcup_{ow} B$
given *complete-lattice-class.Sup-mono* $\langle proof \rangle$

tts-lemma *Inf-mono*:

assumes $B \subseteq U$
and $A \subseteq U$
and $\bigwedge b. b \in B \implies \exists x \in A. x \leq_{ow} b$
shows $\prod_{ow} A \leq_{ow} \prod_{ow} B$
given *complete-lattice-class.Inf-mono* $\langle proof \rangle$

tts-lemma *Sup-Inf-le*:

assumes $A \subseteq Pow U$
shows \sqcup_{ow}
 (
 $\prod_{ow} \{x. x \subseteq U \wedge (\exists f \in \{f. f' Pow U \subseteq U\}. x = f' A \wedge (\forall Y \in A. f Y \in Y))\}$
 $\leq_{ow} \prod_{ow} (\sqcup_{ow} 'A)$
is *complete-lattice-class.Sup-Inf-le* $\langle proof \rangle$

end

tts-context

tts: ($'a$ to U)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
applying [
 OF *Inf-closed' Sup-closed' inf-closed' sup-closed' bot-closed top-closed*
 $]$

begin

tts-lemma *Inf-UNIV*: $\prod_{ow} U = \perp_{ow}$

is *complete-lattice-class.Inf-UNIV* $\langle proof \rangle$

tts-lemma *Sup-UNIV*: $\sqcup_{ow} U = \top_{ow}$

is *complete-lattice-class.Sup-UNIV* $\langle proof \rangle$

end

context

fixes $U_2 :: 'b$ set

begin

lemmas [*transfer-rule*] =

image-transfer [**where** $'a = 'b$]

tts-context

tts: ($'a$ to U) **and** ($'b$ to $\langle U_2 :: 'b$ set \rangle)

rewriting *ctr-simps*

substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*

eliminating through (*insert Sup-least, auto*)

begin

tts-lemma *SUP-upper2*:

assumes $A \subseteq U_2$

and $u \in U$

and $\forall x \in U_2. f x \in U$

and $i \in A$

and $u \leq_{ow} f i$
shows $u \leq_{ow} \sqcup_{ow} (f ' A)$
is *complete-lattice-class.SUP-upper2*{proof}

tts-lemma *INF-lower2*:
assumes $A \subseteq U_2$
and $\forall x \in U_2. f x \in U$
and $u \in U$
and $i \in A$
and $f i \leq_{ow} u$
shows $\prod_{ow} (f ' A) \leq_{ow} u$
is *complete-lattice-class.INF-lower2*{proof}

tts-lemma *less-INF-D*:
assumes $y \in U$
and $\forall x \in U_2. f x \in U$
and $A \subseteq U_2$
and $y <_{ow} \prod_{ow} (f ' A)$
and $i \in A$
shows $y <_{ow} f i$
is *complete-lattice-class.less-INF-D*{proof}

tts-lemma *SUP-lessD*:
assumes $\forall x \in U_2. f x \in U$
and $A \subseteq U_2$
and $y \in U$
and $\sqcup_{ow} (f ' A) <_{ow} y$
and $i \in A$
shows $f i <_{ow} y$
is *complete-lattice-class.SUP-lessD*{proof}

tts-lemma *SUP-le-iff*:
assumes $\forall x \in U_2. f x \in U$ **and** $A \subseteq U_2$ **and** $u \in U$
shows $(\sqcup_{ow} (f ' A) \leq_{ow} u) = (\forall x \in A. f x \leq_{ow} u)$
is *complete-lattice-class.SUP-le-iff*{proof}

end

end

tts-context
tts: (*?a to U*) **and** (*?b to $\langle U_2::'b \text{ set} \rangle$*)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through (*auto dest: top-le*)
begin

tts-lemma *INF-eqI*:
assumes $A \subseteq U_2$
and $x \in U$
and $\forall x \in U_2. f x \in U$
and $\wedge i. \llbracket i \in U_2; i \in A \rrbracket \Longrightarrow x \leq_{ow} f i$
and $\wedge y. \llbracket y \in U; \wedge i. \llbracket i \in U_2; i \in A \rrbracket \Longrightarrow y \leq_{ow} f i \rrbracket \Longrightarrow y \leq_{ow} x$
shows $\prod_{ow} (f ' A) = x$
is *complete-lattice-class.INF-eqI*{proof}

end

tts-context

tts: (*?a* to U) and (*?b* to $\langle U_2::'b \text{ set} \rangle$)

rewriting *ctr-simps*

substituting *complete-lattice-ow-axioms* and *sup-bot.sl-neut.not-empty*

eliminating through (*auto simp: order.eq-iff*)

begin

tts-lemma *SUP-eqI*:

assumes $A \subseteq U_2$

and $\forall x \in U_2. f x \in U$

and $x \in U$

and $\bigwedge i. \llbracket i \in U_2; i \in A \rrbracket \implies f i \leq_{ow} x$

and $\bigwedge y. \llbracket y \in U; \bigwedge i. \llbracket i \in U_2; i \in A \rrbracket \implies f i \leq_{ow} y \rrbracket \implies x \leq_{ow} y$

shows $\sqcup_{ow} (f \text{ ` } A) = x$

is *complete-lattice-class.SUP-eqI* \langle *proof* \rangle

end

tts-context

tts: (*?a* to U) and (*?b* to $\langle U_2::'b \text{ set} \rangle$)

rewriting *ctr-simps*

substituting *complete-lattice-ow-axioms* and *sup-bot.sl-neut.not-empty*

eliminating through *simp*

begin

tts-lemma *INF-le-SUP*:

assumes $A \subseteq U_2$ and $\forall x \in U_2. f x \in U$ and $A \neq \{\}$

shows $\prod_{ow} (f \text{ ` } A) \leq_{ow} \sqcup_{ow} (f \text{ ` } A)$

is *complete-lattice-class.INF-le-SUP* \langle *proof* \rangle

tts-lemma *INF-inf-const1*:

assumes $I \subseteq U_2$ and $x \in U$ and $\forall x \in U_2. f x \in U$ and $I \neq \{\}$

shows $\prod_{ow} ((\lambda i. x \sqcap_{ow} f i) \text{ ` } I) = x \sqcap_{ow} \prod_{ow} (f \text{ ` } I)$

is *complete-lattice-class.INF-inf-const1* \langle *proof* \rangle

tts-lemma *INF-inf-const2*:

assumes $I \subseteq U_2$ and $\forall x \in U_2. f x \in U$ and $x \in U$ and $I \neq \{\}$

shows $\prod_{ow} ((\lambda i. f i \sqcap_{ow} x) \text{ ` } I) = \prod_{ow} (f \text{ ` } I) \sqcap_{ow} x$

is *complete-lattice-class.INF-inf-const2* \langle *proof* \rangle

end

tts-context

tts: (*?a* to U) and (*?b* to $\langle U_2::'b \text{ set} \rangle$)

rewriting *ctr-simps*

substituting *complete-lattice-ow-axioms* and *sup-bot.sl-neut.not-empty*

eliminating through *auto*

begin

tts-lemma *INF-eq-const*:

assumes $I \subseteq U_2$

and $\forall x \in U_2. f x \in U$

and $I \neq \{\}$

and $\bigwedge i. i \in I \implies f i = x$

shows $\prod_{ow} (f \text{ ` } I) = x$

given *complete-lattice-class.INF-eq-const* \langle *proof* \rangle

tts-lemma *SUP-eq-const*:

assumes $I \subseteq U_2$
and $\forall x \in U_2. f x \in U$
and $I \neq \{\}$
and $\bigwedge i. i \in I \implies f i = x$
shows $\sqcup_{ow} (f ' I) = x$
given *complete-lattice-class.SUP-eq-const*

<proof>

tts-lemma *SUP-eq-iff*:

assumes $I \subseteq U_2$
and $c \in U$
and $\forall x \in U_2. f x \in U$
and $I \neq \{\}$
and $\bigwedge i. i \in I \implies c \leq_{ow} f i$
shows $(\sqcup_{ow} (f ' I) = c) = (\forall x \in I. f x = c)$
given *complete-lattice-class.SUP-eq-iff*

<proof>

tts-lemma *INF-eq-iff*:

assumes $I \subseteq U_2$
and $\forall x \in U_2. f x \in U$
and $c \in U$
and $I \neq \{\}$
and $\bigwedge i. i \in I \implies f i \leq_{ow} c$
shows $(\sqcap_{ow} (f ' I) = c) = (\forall x \in I. f x = c)$
given *complete-lattice-class.INF-eq-iff*

<proof>

end

context

fixes $U_2 :: 'b \text{ set}$

begin

lemmas [*transfer-rule*] =
image-transfer[**where** $?'a = 'b$]

tts-context

tts: ($?'a$ to U) **and** ($?'b$ to $\langle U_2 :: 'b \text{ set} \rangle$)

rewriting *ctr-simps*

substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*

eliminating through (*auto simp: sup-bot.top-unique top-le intro: Sup-least*)

begin

tts-lemma *INF-insert*:

assumes $\forall x \in U_2. f x \in U$ **and** $a \in U_2$ **and** $A \subseteq U_2$
shows $\sqcap_{ow} (f ' \text{insert } a A) = f a \sqcap_{ow} \sqcap_{ow} (f ' A)$
is *complete-lattice-class.INF-insert**<proof>*

tts-lemma *SUP-insert*:

assumes $\forall x \in U_2. f x \in U$ **and** $a \in U_2$ **and** $A \subseteq U_2$
shows $\sqcup_{ow} (f ' \text{insert } a A) = f a \sqcup_{ow} \sqcup_{ow} (f ' A)$
is *complete-lattice-class.SUP-insert**<proof>*

tts-lemma *le-INF-iff*:

assumes $u \in U$ **and** $\forall x \in U_2. f x \in U$ **and** $A \subseteq U_2$

shows $(u \leq_{ow} \prod_{ow} (f \text{ ' } A)) = (\forall x \in A. u \leq_{ow} f x)$
is *complete-lattice-class.le-INF-iff*{proof}

tts-lemma *INF-union*:

assumes $\forall x \in U_2. M x \in U$ **and** $A \subseteq U_2$ **and** $B \subseteq U_2$
shows $\prod_{ow} (M \text{ ' } (A \cup B)) = \prod_{ow} (M \text{ ' } A) \prod_{ow} \prod_{ow} (M \text{ ' } B)$
is *complete-lattice-class.INF-union*{proof}

tts-lemma *SUP-union*:

assumes $\forall x \in U_2. M x \in U$ **and** $A \subseteq U_2$ **and** $B \subseteq U_2$
shows $\sqcup_{ow} (M \text{ ' } (A \cup B)) = \sqcup_{ow} (M \text{ ' } A) \sqcup_{ow} \sqcup_{ow} (M \text{ ' } B)$
is *complete-lattice-class.SUP-union*{proof}

tts-lemma *SUP-bot-conv*:

assumes $\forall x \in U_2. B x \in U$ **and** $A \subseteq U_2$
shows
 $(\sqcup_{ow} (B \text{ ' } A) = \perp_{ow}) = (\forall x \in A. B x = \perp_{ow})$
 $(\perp_{ow} = \sqcup_{ow} (B \text{ ' } A)) = (\forall x \in A. B x = \perp_{ow})$
is *complete-lattice-class.SUP-bot-conv*{proof}

tts-lemma *INF-top-conv*:

assumes $\forall x \in U_2. B x \in U$ **and** $A \subseteq U_2$
shows
 $(\prod_{ow} (B \text{ ' } A) = \top_{ow}) = (\forall x \in A. B x = \top_{ow})$
 $(\top_{ow} = \prod_{ow} (B \text{ ' } A)) = (\forall x \in A. B x = \top_{ow})$
is *complete-lattice-class.INF-top-conv*{proof}

tts-lemma *SUP-upper*:

assumes $A \subseteq U_2$ **and** $\forall x \in U_2. f x \in U$ **and** $i \in A$
shows $f i \leq_{ow} \sqcup_{ow} (f \text{ ' } A)$
is *complete-lattice-class.SUP-upper*{proof}

tts-lemma *INF-lower*:

assumes $A \subseteq U_2$ **and** $\forall x \in U_2. f x \in U$ **and** $i \in A$
shows $\prod_{ow} (f \text{ ' } A) \leq_{ow} f i$
is *complete-lattice-class.INF-lower*{proof}

tts-lemma *INF-inf-distrib*:

assumes $\forall x \in U_2. f x \in U$ **and** $A \subseteq U_2$ **and** $\forall x \in U_2. g x \in U$
shows $\prod_{ow} (f \text{ ' } A) \prod_{ow} \prod_{ow} (g \text{ ' } A) = \prod_{ow} ((\lambda a. f a \prod_{ow} g a) \text{ ' } A)$
is *complete-lattice-class.INF-inf-distrib*{proof}

tts-lemma *SUP-sup-distrib*:

assumes $\forall x \in U_2. f x \in U$ **and** $A \subseteq U_2$ **and** $\forall x \in U_2. g x \in U$
shows $\sqcup_{ow} (f \text{ ' } A) \sqcup_{ow} \sqcup_{ow} (g \text{ ' } A) = \sqcup_{ow} ((\lambda a. f a \sqcup_{ow} g a) \text{ ' } A)$
is *complete-lattice-class.SUP-sup-distrib*{proof}

tts-lemma *SUP-subset-mono*:

assumes $B \subseteq U_2$
and $\forall x \in U_2. f x \in U$
and $\forall x \in U_2. g x \in U$
and $A \subseteq B$
and $\bigwedge x. x \in A \implies f x \leq_{ow} g x$
shows $\sqcup_{ow} (f \text{ ' } A) \leq_{ow} \sqcup_{ow} (g \text{ ' } B)$
given *complete-lattice-class.SUP-subset-mono*
{proof}

tts-lemma *INF-superset-mono*:

assumes $A \subseteq U_2$
and $\forall x \in U_2. f x \in U$
and $\forall x \in U_2. g x \in U$
and $B \subseteq A$
and $\bigwedge x. \llbracket x \in U_2; x \in B \rrbracket \implies f x \leq_{ow} g x$
shows $\prod_{ow} (f ' A) \leq_{ow} \prod_{ow} (g ' B)$
given *complete-lattice-class.INF-superset-mono*
 {proof}

tts-lemma *INF-absorb*:
assumes $I \subseteq U_2$ **and** $\forall x \in U_2. A x \in U$ **and** $k \in I$
shows $A k \prod_{ow} \prod_{ow} (A ' I) = \prod_{ow} (A ' I)$
is *complete-lattice-class.INF-absorb*{proof}

tts-lemma *SUP-absorb*:
assumes $I \subseteq U_2$ **and** $\forall x \in U_2. A x \in U$ **and** $k \in I$
shows $A k \sqcup_{ow} \sqcup_{ow} (A ' I) = \sqcup_{ow} (A ' I)$
is *complete-lattice-class.SUP-absorb*{proof}

end

end

context
fixes $f :: 'b \Rightarrow 'al$ **and** $U_2 :: 'b \text{ set}$
assumes $f[\text{transfer-rule}]: \forall x \in U_2. f x = \perp_{ow}$
begin

tts-context
tts: ($?a$ to U) **and** ($?b$ to $\langle U_2 :: 'b \text{ set} \rangle$)
sbterms: ($\langle Orderings.bot :: ?a :: complete-lattice \rangle$ to $\langle \perp_{ow} \rangle$)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through *simp*
begin

tts-lemma *SUP-bot*:
assumes $A \subseteq U_2$
shows $\sqcup_{ow} (f ' A) = \perp_{ow}$
is *complete-lattice-class.SUP-bot[folded const-fun-def]*{proof}

end

end

context
fixes $f :: 'b \Rightarrow 'al$ **and** $U_2 :: 'b \text{ set}$
assumes $f[\text{transfer-rule}]: \forall x \in U_2. f x = \top_{ow}$
begin

tts-context
tts: ($?a$ to U) **and** ($?b$ to $\langle U_2 :: 'b \text{ set} \rangle$)
sbterms: ($\langle Orderings.top :: ?a :: complete-lattice \rangle$ to $\langle \top_{ow} \rangle$)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through *simp*
begin

```

tts-lemma INF-top:
  assumes  $A \subseteq U_2$ 
  shows  $\sqcap_{ow} (f \text{ ' } A) = \top_{ow}$ 
  is complete-lattice-class.INF-top[folded const-fun-def]{proof}

end

end

context
  fixes  $f :: 'b \Rightarrow 'a$  and  $c$  and  $F$  and  $U_2 :: 'b$  set
  assumes c-closed[transfer-rule]:  $c \in U$ 
  assumes f[transfer-rule]:  $\forall x \in U_2. f x = c$ 
begin

tts-register-sbts  $\langle c \rangle \mid U$ 
  {proof}

tts-context
  tts: (?a to U) and (?b to  $\langle U_2 :: 'b \text{ set} \rangle$ )
  sbterms: ( $\langle ?c :: ?'a :: \text{complete-lattice} \rangle$  to  $\langle c \rangle$ )
  rewriting ctr-simps
  substituting complete-lattice-ow-axioms and sup-bot.sl-neut.not-empty
  eliminating through simp
begin

tts-lemma INF-constant:
  assumes  $A \subseteq U_2$ 
  shows  $\sqcap_{ow} (f \text{ ' } A) = (\text{if } A = \{\} \text{ then } \top_{ow} \text{ else } c)$ 
  is complete-lattice-class.INF-constant[folded const-fun-def]{proof}

tts-lemma SUP-constant:
  assumes  $A \subseteq U_2$ 
  shows  $\sqcup_{ow} (f \text{ ' } A) = (\text{if } A = \{\} \text{ then } \perp_{ow} \text{ else } c)$ 
  is complete-lattice-class.SUP-constant[folded const-fun-def]{proof}

end

tts-context
  tts: (?a to U) and (?b to  $\langle U_2 :: 'b \text{ set} \rangle$ )
  sbterms: ( $\langle ?f :: ?'a :: \text{complete-lattice} \rangle$  to  $\langle c \rangle$ )
  rewriting ctr-simps
  substituting complete-lattice-ow-axioms and sup-bot.sl-neut.not-empty
  eliminating through simp
begin

tts-lemma INF-const:
  assumes  $A \subseteq U_2$  and  $A \neq \{\}$ 
  shows  $\sqcap_{ow} ((\lambda i. c) \text{ ' } A) = c$ 
  is complete-lattice-class.INF-const{proof}

tts-lemma SUP-const:
  assumes  $A \subseteq U_2$  and  $A \neq \{\}$ 
  shows  $\sqcup_{ow} ((\lambda i. c) \text{ ' } A) = c$ 
  is complete-lattice-class.SUP-const{proof}

end

```

end

end

context *complete-lattice-ow*
begin

tts-context

tts: (*?a to U*) **and** (*?b to $\langle U_2 :: 'b \text{ set} \rangle$*) **and** (*?c to $\langle U_3 :: 'c \text{ set} \rangle$*)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms and sup-bot.sl-neut.not-empty*
eliminating $\langle ?U \neq \{\} \rangle$ **through** (*force simp: subset-iff INF-top equals0D*)
applying [
OF Inf-closed' Sup-closed' inf-closed' sup-closed' bot-closed top-closed
]

begin

tts-lemma *SUP-eq:*

assumes $A \subseteq U_2$
and $B \subseteq U_3$
and $\forall x \in U_2. f(x :: 'a) \in U$
and $\forall x \in U_3. g x \in U$
and $\wedge i. i \in A \implies \exists x \in B. f i \leq_{ow} g x$
and $\wedge j. j \in B \implies \exists x \in A. g j \leq_{ow} f x$
shows $\sqcup_{ow} (f 'A) = \sqcup_{ow} (g 'B)$
given *complete-lattice-class.SUP-eq*

(proof)

tts-lemma *INF-eq:*

assumes $A \subseteq U_2$
and $B \subseteq U_3$
and $\forall x \in U_3. g x \in U$
and $\forall x \in U_2. f(x :: 'a) \in U$
and $\wedge i. i \in A \implies \exists x \in B. g x \leq_{ow} f i$
and $\wedge j. j \in B \implies \exists x \in A. f x \leq_{ow} g j$
shows $\sqcap_{ow} (f 'A) = \sqcap_{ow} (g 'B)$
given *complete-lattice-class.INF-eq*

(proof)

end

end

context *complete-lattice-ow*
begin

context

fixes $U_2 :: 'b \text{ set}$ **and** $U_3 :: 'c \text{ set}$

begin

lemmas [*transfer-rule*] =

image-transfer[**where** $?a = 'b$]
image-transfer[**where** $?a = 'c$]

tts-context

tts: (*?a to U*) **and** (*?b to $\langle U_2 :: 'b \text{ set} \rangle$*) **and** (*?c to $\langle U_3 :: 'c \text{ set} \rangle$*)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms and sup-bot.sl-neut.not-empty*

```

applying [
  OF - - Inf-closed' Sup-closed' inf-closed' sup-closed' bot-closed top-closed
]
begin

tts-lemma ne-INF-commute:
assumes  $U_2 \neq \{\}$ 
and  $U_3 \neq \{\}$ 
and  $\forall x \in U_2. \forall y \in U_3. f(x::'b) y \in U$ 
and  $B \subseteq U_3$ 
and  $A \subseteq U_2$ 
shows  $\prod_{ow} ((\lambda i. \prod_{ow} (f i ' B)) ' A) = \prod_{ow} ((\lambda j. \prod_{ow} ((\lambda i. f i j) ' A)) ' B)$ 
is complete-lattice-class.INF-commute{proof}

tts-lemma ne-SUP-commute:
assumes  $U_2 \neq \{\}$ 
and  $U_3 \neq \{\}$ 
and  $\forall x \in U_2. \forall y \in U_3. f(x::'b) y \in U$ 
and  $B \subseteq U_3$ 
and  $A \subseteq U_2$ 
shows  $\sqcup_{ow} ((\lambda i. \sqcup_{ow} (f i ' B)) ' A) = \sqcup_{ow} ((\lambda j. \sqcup_{ow} ((\lambda i. f i j) ' A)) ' B)$ 
is complete-lattice-class.SUP-commute{proof}

tts-lemma ne-SUP-mono:
assumes  $U_2 \neq \{\}$ 
and  $U_3 \neq \{\}$ 
and  $A \subseteq U_2$ 
and  $B \subseteq U_3$ 
and  $\forall x \in U_2. f(x::'b) \in U$ 
and  $\forall x \in U_3. g x \in U$ 
and  $\bigwedge n. [[n \in U_2; n \in A]] \implies \exists x \in B. f n \leq_{ow} g x$ 
shows  $\sqcup_{ow} (f ' A) \leq_{ow} \sqcup_{ow} (g ' B)$ 
is complete-lattice-class.SUP-mono{proof}

tts-lemma ne-INF-mono:
assumes  $U_2 \neq \{\}$ 
and  $U_3 \neq \{\}$ 
and  $B \subseteq U_2$ 
and  $A \subseteq U_3$ 
and  $\forall x \in U_3. f x \in U$ 
and  $\forall x \in U_2. g(x::'b) \in U$ 
and  $\bigwedge m. [[m \in U_2; m \in B]] \implies \exists x \in A. f x \leq_{ow} g m$ 
shows  $\prod_{ow} (f ' A) \leq_{ow} \prod_{ow} (g ' B)$ 
is complete-lattice-class.INF-mono{proof}

end

end

lemma INF-commute:
assumes  $\forall x \in U_2. \forall y \in U_3. f x y \in U$  and  $B \subseteq U_3$  and  $A \subseteq U_2$ 
shows
 $\prod_{ow} ((\lambda x. \prod_{ow} (f x ' B)) ' A) = \prod_{ow} ((\lambda j. \prod_{ow} ((\lambda i. f i j) ' A)) ' B)$ 
{proof}

lemma SUP-commute:
assumes  $\forall x \in U_2. \forall y \in U_3. f x y \in U$  and  $B \subseteq U_3$  and  $A \subseteq U_2$ 
shows

```

$\sqcup_{ow} ((\lambda x. \sqcup_{ow} (f x \text{ ' } B)) \text{ ' } A) = \sqcup_{ow} ((\lambda j. \sqcup_{ow} ((\lambda i. f i j) \text{ ' } A)) \text{ ' } B)$
 <proof>

lemma *SUP-mono*:

assumes $A \subseteq U_2$

and $B \subseteq U_3$

and $\forall x \in U_2. f x \in U$

and $\forall x \in U_3. g x \in U$

and $\wedge n. n \in A \implies \exists m \in B. f n \leq_{ow} g m$

shows $\sqcup_{ow} (f \text{ ' } A) \leq_{ow} \sqcup_{ow} (g \text{ ' } B)$

<proof>

lemma *INF-mono*:

assumes $B \subseteq U_2$

and $A \subseteq U_3$

and $\forall x \in U_3. f x \in U$

and $\forall x \in U_2. g x \in U$

and $\wedge m. m \in B \implies \exists n \in A. f n \leq_{ow} g m$

shows $\sqcap_{ow} (f \text{ ' } A) \leq_{ow} \sqcap_{ow} (g \text{ ' } B)$

<proof>

end

context *complete-lattice-pair-ow*

begin

context

fixes $U_3 :: 'c \text{ set}$

begin

lemmas [*transfer-rule*] =

image-transfer[**where** $?'a='c$]

end

end

3.15 Relativization of the results about linear orders

3.15.1 Linear orders

Definitions and further properties

locale *linorder-ow* = *order-ow* +
assumes *linear*: $[[x \in U; y \in U]] \implies x \leq_{ow} y \vee y \leq_{ow} x$
begin

sublocale *min*:

semilattice-order-ow *U* $\langle (\lambda x y. (\text{with } (\leq_{ow}) : \langle \text{min} \rangle x y)) \rangle \langle (\leq_{ow}) \rangle \langle (<_{ow}) \rangle$
 $\langle \text{proof} \rangle$

sublocale *max*:

semilattice-order-ow *U* $\langle (\lambda x y. (\text{with } (\leq_{ow}) : \langle \text{max} \rangle x y)) \rangle \langle (\geq_{ow}) \rangle \langle (>_{ow}) \rangle$
 $\langle \text{proof} \rangle$

end

locale *ord-linorder-ow* =

ord?: *ord-ow* *U*₁ *le*₁ *ls*₁ + *lo?*: *linorder-ow* *U*₂ *le*₂ *ls*₂
for *U*₁ :: '*ao set* **and** *le*₁ *ls*₁ **and** *U*₂ :: '*bo set* **and** *le*₂ *ls*₂

begin

sublocale *ord-order-ow* $\langle \text{proof} \rangle$

end

locale *preorder-linorder-ow* =

po?: *preorder-ow* *U*₁ *le*₁ *ls*₁ + *lo?*: *linorder-ow* *U*₂ *le*₂ *ls*₂
for *U*₁ :: '*ao set* **and** *le*₁ *ls*₁ **and** *U*₂ :: '*bo set* **and** *le*₂ *ls*₂

begin

sublocale *preorder-order-ow* $\langle \text{proof} \rangle$

end

locale *order-linorder-ow* =

order?: *order-ow* *U*₁ *le*₁ *ls*₁ + *lo?*: *linorder-ow* *U*₂ *le*₂ *ls*₂
for *U*₁ :: '*ao set* **and** *le*₁ *ls*₁ **and** *U*₂ :: '*bo set* **and** *le*₂ *ls*₂

begin

sublocale *order-pair-ow* $\langle \text{proof} \rangle$

end

locale *linorder-pair-ow* =

*lo*₁?: *linorder-ow* *U*₁ *le*₁ *ls*₁ + *lo*₂?: *linorder-ow* *U*₂ *le*₂ *ls*₂
for *U*₁ :: '*ao set* **and** *le*₁ *ls*₁ **and** *U*₂ :: '*bo set* **and** *le*₂ *ls*₂

begin

sublocale *order-linorder-ow* $\langle \text{proof} \rangle$

end

Transfer rules

context

```

includes lifting-syntax
begin

lemma linorder-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique A right-total A
  shows
    ((A ===> A ===> (=)) ===> (A ===> A ===> (=)) ===> (=))
    (linorder-ow (Collect (Domainp A))) class.linorder
  <proof>

end

```

Relativization

```

context linorder-ow
begin

```

```

tts-context
  tts: (?'a to U)
  rewriting ctr-simps
  substituting linorder-ow-axioms
  eliminating through simp
begin

```

```

tts-lemma le-less-linear:
  assumes x ∈ U and y ∈ U
  shows x ≤ow y ∨ y <ow x
  is linorder-class.le-less-linear<proof>

```

```

tts-lemma not-less:
  assumes x ∈ U and y ∈ U
  shows (¬ x <ow y) = (y ≤ow x)
  is linorder-class.not-less<proof>

```

```

tts-lemma not-le:
  assumes x ∈ U and y ∈ U
  shows (¬ x ≤ow y) = (y <ow x)
  is linorder-class.not-le<proof>

```

```

tts-lemma lessThan-minus-lessThan:
  assumes n ∈ U and m ∈ U
  shows {..own} − {..owm} = (on U with (≤ow) (<ow) : {m..own})
  is linorder-class.lessThan-minus-lessThan<proof>

```

```

tts-lemma Ici-subset-Ioi-iff:
  assumes a ∈ U and b ∈ U
  shows ({a..ow}) ⊆ {b<ow..}) = (b <ow a)
  is linorder-class.Ici-subset-Ioi-iff<proof>

```

```

tts-lemma Iic-subset-Iio-iff:
  assumes a ∈ U and b ∈ U
  shows ({..owa}) ⊆ {..owb}) = (a <ow b)
  is linorder-class.Iic-subset-Iio-iff<proof>

```

```

tts-lemma leI:
  assumes x ∈ U and y ∈ U and ¬ x <ow y
  shows y ≤ow x
  is linorder-class.leI<proof>

```

tts-lemma *not-le-imp-less*:

assumes $y \in U$ **and** $x \in U$ **and** $\neg y \leq_{ow} x$
shows $x <_{ow} y$
is *linorder-class.not-le-imp-less*(proof)

tts-lemma *Int-atMost*:

assumes $a \in U$ **and** $b \in U$
shows $\{.._{ow}a\} \cap \{.._{ow}b\} = \{.._{ow}min\ a\ b\}$
is *linorder-class.Int-atMost*(proof)

tts-lemma *lessThan-Int-lessThan*:

assumes $a \in U$ **and** $b \in U$
shows $\{a <_{ow}..\} \cap \{b <_{ow}..\} = \{max\ a\ b <_{ow}..\}$
is *linorder-class.lessThan-Int-lessThan*(proof)

tts-lemma *greaterThan-Int-greaterThan*:

assumes $a \in U$ **and** $b \in U$
shows $\{.. <_{ow}a\} \cap \{.. <_{ow}b\} = \{.. <_{ow}min\ a\ b\}$
is *linorder-class.greaterThan-Int-greaterThan*(proof)

tts-lemma *less-linear*:

assumes $x \in U$ **and** $y \in U$
shows $x <_{ow} y \vee x = y \vee y <_{ow} x$
is *linorder-class.less-linear*(proof)

tts-lemma *Int-atLeastAtMostR2*:

assumes $a \in U$ **and** $c \in U$ **and** $d \in U$
shows $\{a.._{ow}\} \cap \{c.._{ow}d\} = \{max\ a\ c.._{ow}d\}$
is *linorder-class.Int-atLeastAtMostR2*(proof)

tts-lemma *Int-atLeastAtMostR1*:

assumes $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $\{.._{ow}b\} \cap \{c.._{ow}d\} = \{c.._{ow}min\ b\ d\}$
is *linorder-class.Int-atLeastAtMostR1*(proof)

tts-lemma *Int-atLeastAtMostL2*:

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $\{a.._{ow}b\} \cap \{c.._{ow}\} = \{max\ a\ c.._{ow}b\}$
is *linorder-class.Int-atLeastAtMostL2*(proof)

tts-lemma *Int-atLeastAtMostL1*:

assumes $a \in U$ **and** $b \in U$ **and** $d \in U$
shows $\{a.._{ow}b\} \cap \{.._{ow}d\} = \{a.._{ow}min\ b\ d\}$
is *linorder-class.Int-atLeastAtMostL1*(proof)

tts-lemma *neq-iff*:

assumes $x \in U$ **and** $y \in U$
shows $(x \neq y) = (x <_{ow} y \vee y <_{ow} x)$
is *linorder-class.neq-iff*(proof)

tts-lemma *not-less-iff-gr-or-eq*:

assumes $x \in U$ **and** $y \in U$
shows $(\neg x <_{ow} y) = (y <_{ow} x \vee x = y)$
is *linorder-class.not-less-iff-gr-or-eq*(proof)

tts-lemma *max-min-distrib2*:

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$

shows $\max a (\min b c) = \min (\max a b) (\max a c)$
is *linorder-class.max-min-distrib2*{proof}

tts-lemma *max-min-distrib1*:

assumes $b \in U$ **and** $c \in U$ **and** $a \in U$
shows $\max (\min b c) a = \min (\max b a) (\max c a)$
is *linorder-class.max-min-distrib1*{proof}

tts-lemma *min-max-distrib2*:

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $\min a (\max b c) = \max (\min a b) (\min a c)$
is *linorder-class.min-max-distrib2*{proof}

tts-lemma *min-max-distrib1*:

assumes $b \in U$ **and** $c \in U$ **and** $a \in U$
shows $\min (\max b c) a = \max (\min b a) (\min c a)$
is *linorder-class.min-max-distrib1*{proof}

tts-lemma *atLeastAtMost-diff-ends*:

assumes $a \in U$ **and** $b \in U$
shows $\{a.._{ow} b\} - \{a, b\} = \{a <_{ow} .. <_{ow} b\}$
is *linorder-class.atLeastAtMost-diff-ends*{proof}

tts-lemma *less-max-iff-disj*:

assumes $z \in U$ **and** $x \in U$ **and** $y \in U$
shows $(z <_{ow} \max x y) = (z <_{ow} x \vee z <_{ow} y)$
is *linorder-class.less-max-iff-disj*{proof}

tts-lemma *min-less-iff-conj*:

assumes $z \in U$ **and** $x \in U$ **and** $y \in U$
shows $(z <_{ow} \min x y) = (z <_{ow} x \wedge z <_{ow} y)$
is *linorder-class.min-less-iff-conj*{proof}

tts-lemma *max-less-iff-conj*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(\max x y <_{ow} z) = (x <_{ow} z \wedge y <_{ow} z)$
is *linorder-class.max-less-iff-conj*{proof}

tts-lemma *min-less-iff-disj*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(\min x y <_{ow} z) = (x <_{ow} z \vee y <_{ow} z)$
is *linorder-class.min-less-iff-disj*{proof}

tts-lemma *le-max-iff-disj*:

assumes $z \in U$ **and** $x \in U$ **and** $y \in U$
shows $(z \leq_{ow} \max x y) = (z \leq_{ow} x \vee z \leq_{ow} y)$
is *linorder-class.le-max-iff-disj*{proof}

tts-lemma *min-le-iff-disj*:

assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(\min x y \leq_{ow} z) = (x \leq_{ow} z \vee y \leq_{ow} z)$
is *linorder-class.min-le-iff-disj*{proof}

tts-lemma *antisym-conv3*:

assumes $y \in U$ **and** $x \in U$ **and** $\neg y <_{ow} x$
shows $(\neg x <_{ow} y) = (x = y)$
is *linorder-class.antisym-conv3*{proof}

tts-lemma *Int-atLeastAtMost*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$
 shows $\{a..{}_{ow}b\} \cap \{c..{}_{ow}d\} = \{\max a c..{}_{ow}\min b d\}$
 is *linorder-class.Int-atLeastAtMost*(proof)

tts-lemma *Int-atLeastLessThan*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$
 shows
 $(\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{a..<b\}) \cap (\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{c..<d\}) =$
 $(\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{\max a c..<(\min b d)\})$
 is *linorder-class.Int-atLeastLessThan*(proof)

tts-lemma *Int-greaterThanAtMost*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$
 shows $\{a<{}_{ow}..b\} \cap \{c<{}_{ow}..d\} = \{\max a c<{}_{ow}..min b d\}$
 is *linorder-class.Int-greaterThanAtMost*(proof)

tts-lemma *Int-greaterThanLessThan*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$
 shows $\{a<{}_{ow}..<{}_{ow}b\} \cap \{c<{}_{ow}..<{}_{ow}d\} = \{\max a c<{}_{ow}..<{}_{ow}\min b d\}$
 is *linorder-class.Int-greaterThanLessThan*(proof)

tts-lemma *le-cases*:

assumes $x \in U$ and $y \in U$ and $x \leq_{ow} y \implies P$ and $y \leq_{ow} x \implies P$
 shows P
 is *linorder-class.le-cases*(proof)

tts-lemma *split-max*:

assumes $i \in U$ and $j \in U$
 shows $P (\max i j) = ((i \leq_{ow} j \implies P j) \wedge (\neg i \leq_{ow} j \implies P i))$
 is *linorder-class.split-max*(proof)

tts-lemma *split-min*:

assumes $i \in U$ and $j \in U$
 shows $P (\min i j) = ((i \leq_{ow} j \implies P i) \wedge (\neg i \leq_{ow} j \implies P j))$
 is *linorder-class.split-min*(proof)

tts-lemma *Ioc-subset-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$
 shows $(\{a<{}_{ow}..b\} \subseteq \{c<{}_{ow}..d\}) = (b \leq_{ow} a \vee b \leq_{ow} d \wedge c \leq_{ow} a)$
 is *linorder-class.Ioc-subset-iff*(proof)

tts-lemma *atLeastLessThan-subset-iff*:

assumes $a \in U$
 and $b \in U$
 and $c \in U$
 and $d \in U$
 and $(\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{a..<b\}) \subseteq (\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{c..<d\})$
 shows $b \leq_{ow} a \vee b \leq_{ow} d \wedge c \leq_{ow} a$
 is *linorder-class.atLeastLessThan-subset-iff*(proof)

tts-lemma *Ioc-inj*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$
 shows $(\{a<{}_{ow}..b\} = \{c<{}_{ow}..d\}) = (b \leq_{ow} a \wedge d \leq_{ow} c \vee a = c \wedge b = d)$
 is *linorder-class.Ioc-inj*(proof)

tts-lemma *neqE*:

assumes $x \in U$

```

and y ∈ U
and x ≠ y
and x <ow y ⇒ R
and y <ow x ⇒ R
shows R
is linorder-class.neqE⟨proof⟩

```

```

tts-lemma Ioc-disjoint:
assumes a ∈ U and b ∈ U and c ∈ U and d ∈ U
shows
  ({a<ow..b} ∩ {c<ow..d} = {}) = (b ≤ow a ∨ d ≤ow c ∨ b ≤ow c ∨ d ≤ow a)
is linorder-class.Ioc-disjoint⟨proof⟩

```

```

tts-lemma linorder-cases:
assumes x ∈ U
and y ∈ U
and x <ow y ⇒ P
and x = y ⇒ P
and y <ow x ⇒ P
shows P
is linorder-class.linorder-cases⟨proof⟩

```

```

tts-lemma le-cases3:
assumes x ∈ U
and y ∈ U
and z ∈ U
and [[x ≤ow y; y ≤ow z]] ⇒ P
and [[y ≤ow x; x ≤ow z]] ⇒ P
and [[x ≤ow z; z ≤ow y]] ⇒ P
and [[z ≤ow y; y ≤ow x]] ⇒ P
and [[y ≤ow z; z ≤ow x]] ⇒ P
and [[z ≤ow x; x ≤ow y]] ⇒ P
shows P
is linorder-class.le-cases3⟨proof⟩

```

end

end

3.15.2 Dense linear orders

Definitions and further properties

```

locale dense-linorder-ow = linorder-ow U le ls + dense-order-ow U le ls
for U :: 'ao set and le (infix <sub>ow> 50) and ls (infix <sub>ow> 50)

```

Transfer rules

```

context
includes lifting-syntax
begin

```

```

lemma dense-linorder-transfer[transfer-rule]:
assumes [transfer-rule]: bi-unique A right-total A
shows
  ((A ===> A ===> (=)) ===> (A ===> A ===> (=)) ===> (=))
  (dense-linorder-ow (Collect (Domainp A))) class.dense-linorder
⟨proof⟩

```

end

Relativization

context *dense-linorder-ow*

begin

tts-context

tts: (?*a* to *U*)

rewriting *ctr-simps*

substituting *dense-linorder-ow-axioms*

eliminating through *simp*

begin

tts-lemma *infinite-Icc*:

assumes $a \in U$ and $b \in U$ and $a <_{ow} b$

shows *infinite* $\{a.._{ow}b\}$

is *dense-linorder-class.infinite-Icc* \langle proof \rangle

tts-lemma *infinite-Ico*:

assumes $a \in U$ and $b \in U$ and $a <_{ow} b$

shows *infinite* (on *U* with (\leq_{ow}) $(<_{ow})$: $\{a..<b\}$)

is *dense-linorder-class.infinite-Ico* \langle proof \rangle

tts-lemma *infinite-Ioc*:

assumes $a \in U$ and $b \in U$ and $a <_{ow} b$

shows *infinite* $\{a<_{ow}..b\}$

is *dense-linorder-class.infinite-Ioc* \langle proof \rangle

tts-lemma *infinite-Ioo*:

assumes $a \in U$ and $b \in U$ and $a <_{ow} b$

shows *infinite* $\{a<_{ow}..<_{ow}b\}$

is *dense-linorder-class.infinite-Ioo* \langle proof \rangle

tts-lemma *atLeastLessThan-subseteq-atLeastAtMost-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$

shows

$((\text{on } U \text{ with } (\leq_{ow}) (\lt_{ow}) : \{a..<b\}) \subseteq \{c.._{ow}d\}) =$

$(a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$

is *dense-linorder-class.atLeastLessThan-subseteq-atLeastAtMost-iff* \langle proof \rangle

tts-lemma *greaterThanAtMost-subseteq-atLeastAtMost-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$

shows $(\{a<_{ow}..b\} \subseteq \{c.._{ow}d\}) = (a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$

is *dense-linorder-class.greaterThanAtMost-subseteq-atLeastAtMost-iff* \langle proof \rangle

tts-lemma *greaterThanAtMost-subseteq-atLeastLessThan-iff*:

assumes $a \in U$

and $b \in U$

and $c \in U$

and $d \in U$

shows $(\{a<_{ow}..b\} \subseteq (\text{on } U \text{ with } (\leq_{ow}) (\lt_{ow}) : \{c..<d\})) =$

$(a <_{ow} b \longrightarrow b <_{ow} d \wedge c \leq_{ow} a)$

is *dense-linorder-class.greaterThanAtMost-subseteq-atLeastLessThan-iff* \langle proof \rangle

tts-lemma *greaterThanLessThan-subseteq-atLeastAtMost-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$

shows $(\{a<_{ow}..<_{ow}b\} \subseteq \{c.._{ow}d\}) = (a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$

is *dense-linorder-class.greaterThanLessThan-subseteq-atLeastAtMost-iff*{proof}

tts-lemma *greaterThanLessThan-subseteq-atLeastLessThan-iff*:

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$

shows

$(\{a <_{ow} .. <_{ow} b\} \subseteq (\text{on } U \text{ with } (\leq_{ow}) (<_{ow}) : \{c .. d\})) =$
 $(a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$

is *dense-linorder-class.greaterThanLessThan-subseteq-atLeastLessThan-iff*{proof}

tts-lemma *greaterThanLessThan-subseteq-greaterThanAtMost-iff*:

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$

shows $(\{a <_{ow} .. <_{ow} b\} \subseteq \{c <_{ow} .. d\}) = (a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$

is *dense-linorder-class.greaterThanLessThan-subseteq-greaterThanAtMost-iff*{proof}

tts-lemma *greaterThanLessThan-subseteq-greaterThanLessThan*:

assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$

shows $(\{a <_{ow} .. <_{ow} b\} \subseteq \{c <_{ow} .. <_{ow} d\}) = (a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$

is *dense-linorder-class.greaterThanLessThan-subseteq-greaterThanLessThan*{proof}

end

end

3.16 Relativization of the results about simple topological spaces

3.16.1 Definitions and common properties

Some of the entities that are presented in this subsection were copied from the theory *HOL-Types-To-Sets/Examples/T2-Spaces*.

```

locale topological-space-ow =
  fixes  $U :: 'at\ set$  and  $\tau :: 'at\ set \Rightarrow bool$ 
  assumes open-UNIV[simp, intro]:  $\tau\ U$ 
  assumes open-Int[intro]:
     $[[\ S \subseteq U; T \subseteq U; \tau\ S; \tau\ T\ ]] \Longrightarrow \tau\ (S \cap T)$ 
  assumes open-Union[intro]:
     $[[\ K \subseteq Pow\ U; \forall\ S \in K. \tau\ S\ ]] \Longrightarrow \tau\ (\bigcup K)$ 
begin

context
  includes lifting-syntax
begin

tts-register-sbts  $\tau\ | U$ 
 $\langle proof \rangle$ 

end

end

locale topological-space-pair-ow =
   $ts_1: topological-space-ow\ U_1\ \tau_1 + ts_2: topological-space-ow\ U_2\ \tau_2$ 
  for  $U_1 :: 'at\ set$  and  $\tau_1$  and  $U_2 :: 'bt\ set$  and  $\tau_2$ 

locale topological-space-triple-ow =
   $ts_1: topological-space-ow\ U_1\ \tau_1 +$ 
   $ts_2: topological-space-ow\ U_2\ \tau_2 +$ 
   $ts_3: topological-space-ow\ U_3\ \tau_3$ 
  for  $U_1 :: 'at\ set$  and  $\tau_1$ 
  and  $U_2 :: 'bt\ set$  and  $\tau_2$ 
  and  $U_3 :: 'ct\ set$  and  $\tau_3$ 
begin

sublocale  $tsp_{12}: topological-space-pair-ow\ U_1\ \tau_1\ U_2\ \tau_2\ \langle proof \rangle$ 
sublocale  $tsp_{13}: topological-space-pair-ow\ U_1\ \tau_1\ U_3\ \tau_3\ \langle proof \rangle$ 
sublocale  $tsp_{23}: topological-space-pair-ow\ U_2\ \tau_2\ U_3\ \tau_3\ \langle proof \rangle$ 
sublocale  $tsp_{21}: topological-space-pair-ow\ U_2\ \tau_2\ U_1\ \tau_1\ \langle proof \rangle$ 
sublocale  $tsp_{31}: topological-space-pair-ow\ U_3\ \tau_3\ U_1\ \tau_1\ \langle proof \rangle$ 
sublocale  $tsp_{32}: topological-space-pair-ow\ U_3\ \tau_3\ U_2\ \tau_2\ \langle proof \rangle$ 

end

inductive generate-topology-on :: ['at set set, 'at set, 'at set]  $\Rightarrow bool$ 
  (
     $\langle (in'-topology'-generated'-by\ -\ on\ - : \langle open \rangle\ -) \rangle$ 
    [1000, 1000, 1000] 10
  )
  for  $S :: 'at\ set\ set$ 
  where
    UNIV:  $(in-topology-generated-by\ S\ on\ U : \langle open \rangle\ U)$ 

```

```

| Int: (in-topology-generated-by S on U : «open» (a ∩ b))
  if (in-topology-generated-by S on U : «open» a)
    and (in-topology-generated-by S on U : «open» b)
    and a ⊆ U
    and b ⊆ U
| UN: (in-topology-generated-by S on U : «open» (∪K))
  if K ⊆ Pow U
    and (∧k. k ∈ K ⇒ (in-topology-generated-by S on U : «open» k))
| Basis: (in-topology-generated-by S on U : «open» s)
  if s ∈ S and s ⊆ U

```

lemma *gto-imp-ss*: (in-topology-generated-by S on U : «open» A) ⇒ A ⊆ U
 ⟨proof⟩

lemma *gt-eq-gto*: generate-topology = (λS. generate-topology-on S UNIV)
 ⟨proof⟩

```

ud ⟨topological-space.closed⟩ (⟨(with - : «closed» -)⟩ [1000, 1000] 10)
ud closed' ⟨closed⟩
ud ⟨topological-space.compact⟩ (⟨(with - : «compact» -)⟩ [1000, 1000] 10)
ud compact' ⟨compact⟩
ud ⟨topological-space.connected⟩ (⟨(with - : «connected» -)⟩ [1000, 1000] 10)
ud connected' ⟨connected⟩
ud ⟨topological-space.islimpt⟩ (⟨(with - : - «islimpt» -)⟩ [1000, 1000, 1000] 60)
ud islimpt' ⟨topological-space-class.islimpt⟩
ud ⟨interior⟩ (⟨(with - : «interior» -)⟩ [1000, 1000] 10)
ud ⟨closure⟩ (⟨(with - : «closure» -)⟩ [1000, 1000] 10)
ud ⟨frontier⟩ (⟨(with - : «frontier» -)⟩ [1000, 1000] 10)
ud ⟨countably-compact⟩ (⟨(with - : «countably'-compact» -)⟩ [1000, 1000] 10)

```

definition *topological-basis-with* :: ['a set ⇒ bool, 'a set set] ⇒ bool
 (⟨(with - : «topological'-basis» -)⟩ [1000, 1000] 10)
 where
 (with τ : «topological-basis» B) =
 (∪B = UNIV ∧ (∀ b ∈ B. τ b) ∧ (∀ q. τ q ⇒ (∃ B' ⊆ B. ∪B' = q)))

ctr relativization

```

synthesis ctr-simps
assumes [transfer-domain-rule, transfer-rule]: Domainp A = (λx. x ∈ U)
  and [transfer-rule]: bi-unique A right-total A
trp (?'a A)
in closed-ow: closed.with-def
  (⟨(on - with - : «closed» -)⟩ [1000, 1000] 10)
  and compact-ow: compact.with-def
  (⟨(on - with - : «compact» -)⟩ [1000, 1000, 1000] 10)
  and connected-ow: connected.with-def
  (⟨(on - with - : «connected» -)⟩ [1000, 1000, 1000] 10)
  and islimpt-ow: islimpt.with-def
  (⟨(on - with - : - «islimpt» -)⟩ [1000, 1000, 1000, 1000] 10)
  and interior-ow: interior.with-def
  (⟨(on - with - : «interior» -)⟩ [1000, 1000, 1000] 10)
  and closure-ow: closure.with-def
  (⟨(on - with - : «closure» -)⟩ [1000, 1000, 1000] 10)
  and frontier-ow: frontier.with-def
  (⟨(on - with - : «frontier» -)⟩ [1000, 1000, 1000] 10)
  and countably-compact-ow: countably-compact.with-def
  (⟨(on - with - : «countably'-compact» -)⟩ [1000, 1000, 1000] 10)

```

context *topological-space-ow*

begin

abbreviation *closed* **where** $closed \equiv closed-ow\ U\ \tau$

abbreviation *compact* **where** $compact \equiv compact-ow\ U\ \tau$

abbreviation *connected* **where** $connected \equiv connected-ow\ U\ \tau$

abbreviation *islimpt* (**infixr** $\langle\langle islimpt \rangle\rangle\ 60$)

where $x \langle\langle islimpt \rangle\rangle\ S \equiv on\ U\ with\ \tau : x \langle\langle islimpt \rangle\rangle\ S$

abbreviation *interior* **where** $interior \equiv interior-ow\ U\ \tau$

abbreviation *closure* **where** $closure \equiv closure-ow\ U\ \tau$

abbreviation *frontier* **where** $frontier \equiv frontier-ow\ U\ \tau$

abbreviation *countably-compact*

where $countably-compact \equiv countably-compact-ow\ U\ \tau$

end

context

includes *lifting-syntax*

begin

private lemma *Domainp-fun-rel-eq-subset*:

fixes $A :: ['a, 'c] \Rightarrow bool$

fixes $B :: ['b, 'd] \Rightarrow bool$

assumes *bi-unique A bi-unique B*

shows

$Domainp\ (A\ ==>\ B) =$

$(\lambda f. f\ ' (Collect\ (Domainp\ A)) \subseteq (Collect\ (Domainp\ B)))$

(proof) **lemma** *Ex-rt-bu-transfer[transfer-rule]*:

fixes $A :: ['a, 'c] \Rightarrow bool$

fixes $B :: ['b, 'd] \Rightarrow bool$

assumes *[transfer-rule]: bi-unique A right-total A bi-unique B*

shows

$((B\ ==>\ A)\ ==>\ (=))\ ==>\ (=)$

$(\exists x\ (Collect\ (\lambda f. f\ ' (Collect\ (Domainp\ B)) \subseteq (Collect\ (Domainp\ A))))))$

Ex

(proof)

end

definition *topological-basis-ow* ::

$['a\ set, 'a\ set \Rightarrow bool, 'a\ set\ set] \Rightarrow bool$

$(\langle\langle on\ -\ with\ - : \langle\langle topological'-basis \rangle\rangle\ - \rangle\rangle\ [1000, 1000, 1000]\ 10)$

where

$(on\ U\ with\ \tau : \langle\langle topological-basis \rangle\rangle\ B) =$

$(\cup B = U \wedge (\forall b \in B. \tau\ b) \wedge (\forall q \subseteq U. \tau\ q \longrightarrow (\exists B' \subseteq B. \cup B' = q)))$

context *topological-space*

begin

lemma *topological-basis-with[ud-with]*:

topological-basis = topological-basis-with open

(proof)

end

3.16.2 Transfer rules

Some of the entities that are presented in this subsection were copied from *HOL-Types-To-Sets/Examples/T2-Sp*

context

includes *lifting-syntax*

begin

lemmas *vimage-transfer*[*transfer-rule*] = *vimage-transfer*

lemma *topological-space-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$((\text{rel-set } A \text{ ===> } (=)) \text{ ===> } (=))$

$(\text{topological-space-ow } (\text{Collect } (\text{Domainp } A))) \text{ class.topological-space}$

$\langle \text{proof} \rangle$

lemma *generate-topology-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$((\text{rel-set } (\text{rel-set } A)) \text{ ===> } (\text{rel-set } A \text{ ===> } (=)))$

$(\lambda B. \text{generate-topology-on } B (\text{Collect } (\text{Domainp } A))) \text{ generate-topology}$

$\langle \text{proof} \rangle$

lemma *topological-basis-with-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

$((\text{rel-set } A \text{ ===> } (=)) \text{ ===> } (\text{rel-set } (\text{rel-set } A)) \text{ ===> } (=))$

$(\text{topological-basis-ow } (\text{Collect } (\text{Domainp } A))) \text{ topological-basis-with}$

$\langle \text{proof} \rangle$

end

3.16.3 Relativization

tts-context

tts: (*?'a to* $\langle U_1::'a \text{ set} \rangle$) **and** (*?'b to* $\langle U_2::'b \text{ set} \rangle$)

rewriting *ctr-simps*

begin

tts-lemma *generate-topology-Union*:

assumes $U_1 \neq \{\}$

and $U_2 \neq \{\}$

and $I \subseteq U_1$

and $S \subseteq \text{Pow } U_2$

and $\forall x \in U_1. K(x::'a) \subseteq U_2$

and

$\wedge k. [[k \in U_1; k \in I]] \implies$

$\text{in-topology-generated-by } S \text{ on } U_2 : \langle \text{open} \rangle (K k)$

shows $\text{in-topology-generated-by } S \text{ on } U_2 : \langle \text{open} \rangle (\cup (K 'I))$

is *generate-topology-Union* $\langle \text{proof} \rangle$

end

tts-context

tts: (*?'a to* $\langle U::'a \text{ set} \rangle$)

rewriting *ctr-simps*

eliminating through

$(\text{unfold topological-space-ow-def; auto intro: generate-topology-on.intros})$

begin

tts-lemma *topological-space-generate-topology*:

shows *topological-space-ow U (generate-topology-on S U)*
is *topological-space-generate-topology⟨proof⟩*

end

context *topological-space-ow*
begin

tts-context
tts: (*?'a to U*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating through (*metis open-UNIV*)
begin

tts-lemma *open-empty[simp]:*
shows $\tau \{\}$
is *topological-space-class.open-empty⟨proof⟩*

end

tts-context
tts: (*?'a to U*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating through
 (
 unfold
 closed-ow-def
 compact-ow-def
 connected-ow-def
 interior-ow-def
 closure-ow-def
 frontier-ow-def,
 auto
)
begin

tts-lemma *closed-empty[simp]: closed* $\{\}$
is *topological-space-class.closed-empty⟨proof⟩*

tts-lemma *closed-UNIV[simp]: closed U*
is *topological-space-class.closed-UNIV⟨proof⟩*

tts-lemma *compact-empty[simp]: compact* $\{\}$
is *topological-space-class.compact-empty⟨proof⟩*

tts-lemma *connected-empty[simp]: connected* $\{\}$
is *topological-space-class.connected-empty⟨proof⟩*

tts-lemma *interior-empty[simp]: interior* $\{\} = \{\}$
is *interior-empty⟨proof⟩*

tts-lemma *closure-empty[simp]: closure* $\{\} = \{\}$
is *closure-empty⟨proof⟩*

tts-lemma *closure-UNIV[simp]: closure U = U*
is *closure-UNIV⟨proof⟩*

```

tts-lemma frontier-empty[simp]: frontier {} = {}
  is frontier-empty{proof}

tts-lemma frontier-UNIV[simp]: frontier U = {}
  is frontier-UNIV{proof}

end

tts-context
  tts: (?'a to U)
  rewriting ctr-simps
  substituting topological-space-ow-axioms
  eliminating through (auto simp: UNIV inj-on-def)
begin

tts-lemma connected-Union:
  assumes  $S \subseteq \text{Pow } U$  and  $\bigwedge s. s \in S \implies \text{connected } s$  and  $\bigcap S \cap U \neq \{\}$ 
  shows connected ( $\bigcup S$ )
  given Topological-Spaces.connected-Union
  {proof}

tts-lemma connected-Un:
  assumes  $s \subseteq U$ 
  and  $t \subseteq U$ 
  and connected  $s$ 
  and connected  $t$ 
  and  $s \cap t \neq \{\}$ 
  shows connected ( $s \cup t$ )
  is Topological-Spaces.connected-Un{proof}

end

tts-context
  tts: (?'a to U)
  rewriting ctr-simps
  substituting topological-space-ow-axioms
  eliminating  $\langle ?U \neq \{\} \rangle$  and  $\langle ?A \subseteq ?B \rangle$ 
  through (auto simp: UNIV inj-on-def)
begin

tts-lemma connected-sing:
  assumes  $x \in U$ 
  shows connected { $x$ }
  is topological-space-class.connected-sing{proof}

tts-lemma topological-basisE:
  assumes  $B \subseteq \text{Pow } U$ 
  and  $O' \subseteq U$ 
  and  $x \in U$ 
  and on U with  $\tau : \langle \text{topological-basis} \rangle B$ 
  and  $\tau O'$ 
  and  $x \in O'$ 
  and  $\bigwedge B'. [\![B' \subseteq U; B' \in B; x \in B'; B' \subseteq O'\!] \implies \text{thesis}$ 
  shows thesis
  is topological-space-class.topological-basisE{proof}

tts-lemma islimptE:

```

assumes $x \in U$
and $S \subseteq U$
and $T \subseteq U$
and $x \ll \text{islimpt} \gg S$
and $x \in T$
and τT
and $\bigwedge y. [[y \in U; y \in S; y \in T; y \neq x]] \implies \text{thesis}$
shows *thesis*
is *Elementary-Topology.islimptE* $\langle \text{proof} \rangle$

tts-lemma *islimpt-subset*:
assumes $x \in U$ **and** $T \subseteq U$ **and** $x \ll \text{islimpt} \gg S$ **and** $S \subseteq T$
shows $x \ll \text{islimpt} \gg T$
is *Elementary-Topology.islimpt-subset* $\langle \text{proof} \rangle$

tts-lemma *islimpt-UNIV-iff*:
assumes $x \in U$
shows $x \ll \text{islimpt} \gg U = (\neg \tau \{x\})$
is *Elementary-Topology.islimpt-UNIV-iff* $\langle \text{proof} \rangle$

tts-lemma *islimpt-punctured*:
assumes $x \in U$ **and** $S \subseteq U$
shows $x \ll \text{islimpt} \gg S = x \ll \text{islimpt} \gg S - \{x\}$
is *Elementary-Topology.islimpt-punctured* $\langle \text{proof} \rangle$

tts-lemma *islimpt-EMPTY*:
assumes $x \in U$
shows $\neg x \ll \text{islimpt} \gg \{\}$
is *Elementary-Topology.islimpt-EMPTY* $\langle \text{proof} \rangle$

tts-lemma *islimpt-Un*:
assumes $x \in U$ **and** $S \subseteq U$ **and** $T \subseteq U$
shows $x \ll \text{islimpt} \gg S \cup T = (x \ll \text{islimpt} \gg S \vee x \ll \text{islimpt} \gg T)$
is *Elementary-Topology.islimpt-Un* $\langle \text{proof} \rangle$

tts-lemma *interiorI*:
assumes $x \in U$ **and** $S \subseteq U$ **and** τT **and** $x \in T$ **and** $T \subseteq S$
shows $x \in \text{interior } S$
is *Elementary-Topology.interiorI* $\langle \text{proof} \rangle$

tts-lemma *islimpt-in-closure*:
assumes $x \in U$ **and** $S \subseteq U$
shows $x \ll \text{islimpt} \gg S = (x \in \text{closure } (S - \{x\}))$
is *Elementary-Topology.islimpt-in-closure* $\langle \text{proof} \rangle$

tts-lemma *compact-sing*:
assumes $a \in U$
shows *compact* $\{a\}$
is *Elementary-Topology.compact-sing* $\langle \text{proof} \rangle$

tts-lemma *compact-insert*:
assumes $s \subseteq U$ **and** $x \in U$ **and** *compact* s
shows *compact* $(\text{insert } x s)$
is *Elementary-Topology.compact-insert* $\langle \text{proof} \rangle$

tts-lemma *open-Un*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** τS **and** τT
shows $\tau (S \cup T)$

is *topological-space-class.open-Un*{proof}

tts-lemma *open-Inter*:

assumes $S \subseteq \text{Pow } U$ and finite S and Ball $S \tau$

shows $\tau (\bigcap S \cap U)$

is *topological-space-class.open-Inter*{proof}

tts-lemma *openI*:

assumes $S \subseteq U$ and $\bigwedge x. [[x \in U; x \in S]] \implies \exists y \subseteq U. \tau y \wedge y \subseteq S \wedge x \in y$

shows τS

given *topological-space-class.openI* {proof}

tts-lemma *closed-Un*:

assumes $S \subseteq U$ and $T \subseteq U$ and closed S and closed T

shows closed $(S \cup T)$

is *topological-space-class.closed-Un*{proof}

tts-lemma *closed-Int*:

assumes $S \subseteq U$ and $T \subseteq U$ and closed S and closed T

shows closed $(S \cap T)$

is *topological-space-class.closed-Int*{proof}

tts-lemma *open-Collect-conj*:

assumes $\tau \{x. P x \wedge x \in U\}$ and $\tau \{x. Q x \wedge x \in U\}$

shows $\tau \{x \in U. P x \wedge Q x\}$

is *topological-space-class.open-Collect-conj*{proof}

tts-lemma *open-Collect-disj*:

assumes $\tau \{x. P x \wedge x \in U\}$

and $\tau \{x. Q x \wedge x \in U\}$

shows $\tau \{x \in U. P x \vee Q x\}$

is *topological-space-class.open-Collect-disj*{proof}

tts-lemma *open-Collect-imp*:

assumes closed $\{x. P x \wedge x \in U\}$

and $\tau \{x. Q x \wedge x \in U\}$

shows $\tau \{x \in U. P x \longrightarrow Q x\}$

is *topological-space-class.open-Collect-imp*{proof}

tts-lemma *open-Collect-const*: $\tau \{x. P \wedge x \in U\}$

is *topological-space-class.open-Collect-const*{proof}

tts-lemma *closed-Collect-conj*:

assumes closed $\{x. P x \wedge x \in U\}$ and closed $\{x. Q x \wedge x \in U\}$

shows closed $\{x \in U. P x \wedge Q x\}$

is *topological-space-class.closed-Collect-conj*{proof}

tts-lemma *closed-Collect-disj*:

assumes closed $\{x. P x \wedge x \in U\}$ and closed $\{x. Q x \wedge x \in U\}$

shows closed $\{x \in U. P x \vee Q x\}$

is *topological-space-class.closed-Collect-disj*{proof}

tts-lemma *closed-Collect-imp*:

assumes $\tau \{x. P x \wedge x \in U\}$ and closed $\{x. Q x \wedge x \in U\}$

shows closed $\{x \in U. P x \longrightarrow Q x\}$

is *topological-space-class.closed-Collect-imp*{proof}

tts-lemma *compact-Int-closed*:

assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *compact* S **and** *closed* T
shows *compact* $(S \cap T)$
is *topological-space-class.compact-Int-closed* \langle *proof* \rangle

tts-lemma *compact-diff*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *compact* S **and** τ T
shows *compact* $(S - T)$
is *topological-space-class.compact-diff* \langle *proof* \rangle

tts-lemma *connectedD*:
assumes $U \subseteq U$
and $V \subseteq U$
and *connected* A
and τ U
and τ V
and $U \cap (V \cap A) = \{\}$
and $A \subseteq U \cup V$
shows $U \cap A = \{\} \vee V \cap A = \{\}$
is *topological-space-class.connectedD* \langle *proof* \rangle

tts-lemma *topological-basis-open*:
assumes $B \subseteq \text{Pow } U$ **and** *on* U **with** $\tau : \langle \text{topological-basis} \rangle B$ **and** $X \in B$
shows τX
is *topological-space-class.topological-basis-open* \langle *proof* \rangle

tts-lemma *topological-basis-imp-subbasis*:
assumes $B \subseteq \text{Pow } U$ **and** *on* U **with** $\tau : \langle \text{topological-basis} \rangle B$
shows $\forall s \subseteq U. \tau s = (\text{in-topology-generated-by } B \text{ on } U : \langle \text{open} \rangle s)$
is *topological-space-class.topological-basis-imp-subbasis* \langle *proof* \rangle

tts-lemma *connected-closedD*:
assumes $A \subseteq U$
and $B \subseteq U$
and *connected* s
and $A \cap (B \cap s) = \{\}$
and $s \subseteq A \cup B$
and *closed* A
and *closed* B
shows $A \cap s = \{\} \vee B \cap s = \{\}$
is *Topological-Spaces.connected-closedD* \langle *proof* \rangle

tts-lemma *connected-diff-open-from-closed*:
assumes $u \subseteq U$
and $s \subseteq t$
and $t \subseteq u$
and τ s
and *closed* t
and *connected* u
and *connected* $(t - s)$
shows *connected* $(u - s)$
is *Topological-Spaces.connected-diff-open-from-closed* \langle *proof* \rangle

tts-lemma *interior-maximal*:
assumes $S \subseteq U$ **and** $T \subseteq S$ **and** τ T
shows $T \subseteq \text{interior } S$
is *Elementary-Topology.interior-maximal* \langle *proof* \rangle

tts-lemma *open-subset-interior*:

assumes $S \subseteq U$ **and** $T \subseteq U$ **and** τS
shows $(S \subseteq \text{interior } T) = (S \subseteq T)$
is *Elementary-Topology.open-subset-interior* $\langle \text{proof} \rangle$

tts-lemma *interior-mono*:
assumes $T \subseteq U$ **and** $S \subseteq T$
shows *interior* $S \subseteq \text{interior } T$
is *Elementary-Topology.interior-mono* $\langle \text{proof} \rangle$

tts-lemma *interior-Int*:
assumes $S \subseteq U$ **and** $T \subseteq U$
shows *interior* $(S \cap T) = \text{interior } S \cap \text{interior } T$
is *Elementary-Topology.interior-Int* $\langle \text{proof} \rangle$

tts-lemma *interior-closed-Un-empty-interior*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *closed* S **and** *interior* $T = \{\}$
shows *interior* $(S \cup T) = \text{interior } S$
is *Elementary-Topology.interior-closed-Un-empty-interior* $\langle \text{proof} \rangle$

tts-lemma *countably-compact-imp-acc-point*:
assumes *local.countably-compact* s
and *countable* t
and *infinite* t
and $t \subseteq s$
shows $\exists x \in s. \forall U \in \text{Pow } U. \tau U \wedge x \in U \longrightarrow \text{infinite } (U \cap t)$
is *Elementary-Topology.countably-compact-imp-acc-point* $\langle \text{proof} \rangle$

end

tts-context
tts: $(?a \text{ to } U)$
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$
through $(\text{auto simp: UNIV inj-on-def})$
begin

tts-lemma *first-countableI*:
assumes $\mathcal{A} \subseteq \text{Pow } U$
and $x \in U$
and *countable* \mathcal{A}
and $\bigwedge A. [[A \in \mathcal{A}] \Longrightarrow x \in A$
and $\bigwedge A. [[A \in \mathcal{A}] \Longrightarrow \tau A$
and $\bigwedge S. [[\tau S; x \in S] \Longrightarrow \exists A \in \mathcal{A}. A \subseteq S$
shows $\exists \mathcal{A} \in \{f. \text{range } f \subseteq \text{Pow } U\}.$
 $(\forall i. \tau (\mathcal{A} (i::\text{nat})) \wedge x \in \mathcal{A} i) \wedge$
 $(\forall S \in \text{Pow } U. \tau S \wedge x \in S \longrightarrow (\exists i. \mathcal{A} i \subseteq S))$
given *topological-space-class.first-countableI* $\langle \text{proof} \rangle$

tts-lemma *islimptI*:
assumes $x \in U$
and $S \subseteq U$
and $\bigwedge T. [[x \in T; \tau T] \Longrightarrow \exists y \in S. y \in T \wedge y \neq x$
shows $x \ll \text{islimpt} \gg S$
given *Elementary-Topology.islimptI*
 $\langle \text{proof} \rangle$

tts-lemma *interiorE*:

assumes $x \in U$
and $S \subseteq U$
and $x \in \text{interior } S$
and $\wedge T. [[T \subseteq U; \tau T; x \in T; T \subseteq S]] \implies \text{thesis}$
shows *thesis*
is *Elementary-Topology.interiorE* $\langle \text{proof} \rangle$

tts-lemma *closure-iff-nhds-not-empty*:

assumes $x \in U$ **and** $X \subseteq U$
shows
 $(x \in \text{closure } X) =$
 $(\forall y \subseteq U. \forall z \subseteq U. z \subseteq y \longrightarrow \tau z \longrightarrow x \in z \longrightarrow X \cap y \neq \{\})$
given *Elementary-Topology.closure-iff-nhds-not-empty* $\langle \text{proof} \rangle$

tts-lemma *basis-dense*:

assumes $B \subseteq \text{Pow } U$
and $\forall x \subseteq U. f x \in U$
and *on* U **with** $\tau : \langle \text{topological-basis} \rangle B$
and $\wedge B'. [[B' \subseteq U; B' \neq \{\}]] \implies f B' \in B'$
shows $\forall x \subseteq U. \tau x \longrightarrow x \neq \{\} \longrightarrow (\exists y \in B. f y \in x)$
given *topological-space-class.basis-dense* $\langle \text{proof} \rangle$

tts-lemma *inj-setminus*:

assumes $A \subseteq \text{Pow } U$
shows *inj-on* $(\lambda S. - S \cap U) A$
is *topological-space-class.inj-setminus* $\langle \text{proof} \rangle$

end

tts-context

tts: $(?a \text{ to } U)$
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **and** $\langle ?S \subseteq U \rangle$ **through**
 $($
 unfold
 closed-ow-def
 compact-ow-def
 connected-ow-def
 interior-ow-def
 topological-basis-ow-def
 closure-ow-def
 frontier-ow-def
 countably-compact-ow-def,
 auto
 $)$

begin

tts-lemma *closed-Inter*:

assumes $K \subseteq \text{Pow } U$ **and** *Ball* K *closed*
shows *closed* $(\cap K \cap U)$
is *topological-space-class.closed-Inter* $\langle \text{proof} \rangle$

tts-lemma *closed-Union*:

assumes $S \subseteq \text{Pow } U$ **and** *finite* S **and** *Ball* S *closed*
shows *closed* $(\cup S)$
is *topological-space-class.closed-Union* $\langle \text{proof} \rangle$

tts-lemma *open-closed*:
assumes $S \subseteq U$
shows $\tau S = \text{closed } (- S \cap U)$
is *topological-space-class.open-closed* $\langle \text{proof} \rangle$

tts-lemma *closed-open*:
shows $\text{closed } S = \tau (- S \cap U)$
is *topological-space-class.closed-open* $\langle \text{proof} \rangle$

tts-lemma *open-Diff*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** τS **and** *closed* T
shows $\tau (S - T)$
is *topological-space-class.open-Diff* $\langle \text{proof} \rangle$

tts-lemma *closed-Diff*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *closed* S **and** τT
shows *closed* $(S - T)$
is *topological-space-class.closed-Diff* $\langle \text{proof} \rangle$

tts-lemma *open-Compl*:
assumes *closed* S
shows $\tau (- S \cap U)$
is *topological-space-class.open-Compl* $\langle \text{proof} \rangle$

tts-lemma *closed-Compl*:
assumes $S \subseteq U$ **and** τS
shows *closed* $(- S \cap U)$
is *topological-space-class.closed-Compl* $\langle \text{proof} \rangle$

tts-lemma *open-Collect-neg*:
assumes *closed* $\{x \in U. P x\}$
shows $\tau \{x \in U. \neg P x\}$
given *topological-space-class.open-Collect-neg* $\langle \text{proof} \rangle$

tts-lemma *closed-Collect-neg*:
assumes $\tau \{x \in U. P x\}$
shows *closed* $\{x \in U. \neg P x\}$
given *topological-space-class.closed-Collect-neg* $\langle \text{proof} \rangle$

tts-lemma *closed-Collect-const*: *closed* $\{x \in U. P\}$
given *topological-space-class.closed-Collect-const* $\langle \text{proof} \rangle$

tts-lemma *connectedI*:
assumes
 $\wedge A B.$
 \llbracket
 $A \subseteq U;$
 $B \subseteq U;$
 $\tau A;$
 $\tau B;$
 $A \cap U \neq \{\};$
 $B \cap U \neq \{\};$
 $A \cap (B \cap U) = \{\};$
 $U \subseteq A \cup B$

$\mathbb{I} \implies \text{False}$
shows *connected* U
is *topological-space-class.connectedI*(*proof*)

tts-lemma *topological-basis*:
assumes $B \subseteq \text{Pow } U$
shows (on U with $\tau : \langle\langle \text{topological-basis} \rangle\rangle B =$
 $(\forall x \in \text{Pow } U. \tau x = (\exists B' \in \text{Pow } (U). B' \subseteq B \wedge \cup B' = x))$)
is *topological-space-class.topological-basis*(*proof*)

tts-lemma *topological-basis-iff*:
assumes $B \subseteq \text{Pow } U$ **and** $\wedge B'. \mathbb{I}[B' \subseteq U; B' \in B] \implies \tau B'$
shows (on U with $\tau : \langle\langle \text{topological-basis} \rangle\rangle B =$
 $(\forall O' \in \text{Pow } U. \tau O' \longrightarrow (\forall x \in O'. \exists B' \in B. B' \subseteq O' \wedge x \in B'))$)
is *topological-space-class.topological-basis-iff*(*proof*)

tts-lemma *topological-basisI*:
assumes $B \subseteq \text{Pow } U$
and $\wedge B'. \mathbb{I}[B' \subseteq U; B' \in B] \implies \tau B'$
and $\wedge O' x. \mathbb{I}[O' \subseteq U; x \in U; \tau O'; x \in O'] \implies \exists y \in B. y \subseteq O' \wedge x \in y$
shows on U with $\tau : \langle\langle \text{topological-basis} \rangle\rangle B$
is *topological-space-class.topological-basisI*(*proof*)

tts-lemma *closed-closure*:
assumes $S \subseteq U$
shows *closed* (*closure* S)
is *Elementary-Topology.closed-closure*(*proof*)

tts-lemma *closure-subset*: $S \subseteq \text{closure } S$
is *Elementary-Topology.closure-subset*(*proof*)

tts-lemma *closure-eq*:
assumes $S \subseteq U$
shows (*closure* $S = S$) = *closed* S
is *Elementary-Topology.closure-eq*(*proof*)

tts-lemma *closure-closed*:
assumes $S \subseteq U$ **and** *closed* S
shows *closure* $S = S$
is *Elementary-Topology.closure-closed*(*proof*)

tts-lemma *closure-closure*:
assumes $S \subseteq U$
shows *closure* (*closure* S) = *closure* S
is *Elementary-Topology.closure-closure*(*proof*)

tts-lemma *closure-mono*:
assumes $T \subseteq U$ **and** $S \subseteq T$
shows *closure* $S \subseteq \text{closure } T$
is *Elementary-Topology.closure-mono*(*proof*)

tts-lemma *closure-minimal*:
assumes $T \subseteq U$ **and** $S \subseteq T$ **and** *closed* T
shows *closure* $S \subseteq T$
is *Elementary-Topology.closure-minimal*(*proof*)

tts-lemma *closure-unique*:
assumes $T \subseteq U$

```

    and  $S \subseteq T$ 
    and closed  $T$ 
    and  $\wedge T'. \llbracket T' \subseteq U; S \subseteq T'; \text{closed } T' \rrbracket \implies T \subseteq T'$ 
  shows closure  $S = T$ 
  is Elementary-Topology.closure-unique(proof)

tts-lemma closure-Un:
  assumes  $S \subseteq U$  and  $T \subseteq U$ 
  shows closure  $(S \cup T) = \text{closure } S \cup \text{closure } T$ 
  is Elementary-Topology.closure-Un(proof)

tts-lemma closure-eq-empty:  $(\text{closure } S = \{\}) = (S = \{\})$ 
  is Elementary-Topology.closure-eq-empty(proof)

tts-lemma closure-subset-eq:
  assumes  $S \subseteq U$ 
  shows  $(\text{closure } S \subseteq S) = \text{closed } S$ 
  is Elementary-Topology.closure-subset-eq(proof)

tts-lemma open-Int-closure-eq-empty:
  assumes  $S \subseteq U$  and  $T \subseteq U$  and  $\tau S$ 
  shows  $(S \cap \text{closure } T = \{\}) = (S \cap T = \{\})$ 
  is Elementary-Topology.open-Int-closure-eq-empty(proof)

tts-lemma open-Int-closure-subset:
  assumes  $S \subseteq U$  and  $T \subseteq U$  and  $\tau S$ 
  shows  $S \cap \text{closure } T \subseteq \text{closure } (S \cap T)$ 
  is Elementary-Topology.open-Int-closure-subset(proof)

tts-lemma closure-Un-frontier:  $\text{closure } S = S \cup \text{frontier } S$ 
  is Elementary-Topology.closure-Un-frontier(proof)

tts-lemma compact-imp-countably-compact:
  assumes compact  $U$ 
  shows countably-compact  $U$ 
  is Elementary-Topology.compact-imp-countably-compact(proof)

end

tts-context
  tts: (?a to  $U$ )
  rewriting ctr-simps
  substituting topological-space-ow-axioms
  eliminating through auto
begin

tts-lemma Heine-Borel-imp-Bolzano-Weierstrass:
  assumes  $s \subseteq U$ 
    and compact  $s$ 
    and infinite  $t$ 
    and  $t \subseteq s$ 
  shows  $\exists x \in s. x \ll \text{islimpt} \gg t$ 
  is Elementary-Topology.Heine-Borel-imp-Bolzano-Weierstrass(proof)

end

tts-context

```

```

tts: (?a to U)
rewriting ctr-simps
substituting topological-space-ow-axioms
eliminating ⟨?U ≠ {}⟩ through
  (
    unfold
    closed-ow-def
    compact-ow-def
    connected-ow-def
    interior-ow-def
    topological-basis-ow-def
    closure-ow-def
    frontier-ow-def
    countably-compact-ow-def,
    auto simp: connected-iff-const
  )
begin

tts-lemma connected-closed:
assumes  $s \subseteq U$ 
shows connected  $s =$ 
  (
     $\neg(\exists A \in \text{Pow } U. \exists B \in \text{Pow } U.$ 
      closed  $A \wedge$ 
      closed  $B \wedge$ 
       $s \subseteq A \cup B \wedge$ 
       $A \cap (B \cap s) = \{\}$   $\wedge$ 
       $A \cap s \neq \{\}$   $\wedge$ 
       $B \cap s \neq \{\})$ 
  )
is Topological-Spaces.connected-closed{proof}

tts-lemma closure-complement:
assumes  $S \subseteq U$ 
shows closure  $(- S \cap U) = - \textit{interior } S \cap U$ 
is Elementary-Topology.closure-complement{proof}

tts-lemma interior-complement:
assumes  $S \subseteq U$ 
shows interior  $(- S \cap U) = - \textit{closure } S \cap U$ 
is Elementary-Topology.interior-complement{proof}

tts-lemma interior-diff:
assumes  $S \subseteq U$  and  $T \subseteq U$ 
shows interior  $(S - T) = \textit{interior } S - \textit{closure } T$ 
is Elementary-Topology.interior-diff{proof}

tts-lemma connected-imp-connected-closure:
assumes  $S \subseteq U$  and connected  $S$ 
shows connected  $(\textit{closure } S)$ 
is Elementary-Topology.connected-imp-connected-closure{proof}

tts-lemma frontier-closed:
assumes  $S \subseteq U$ 
shows closed  $(\textit{frontier } S)$ 
is Elementary-Topology.frontier-closed{proof}

tts-lemma frontier-Int:

```

assumes $S \subseteq U$ **and** $T \subseteq U$
shows $\text{frontier } (S \cap T) = \text{closure } (S \cap T) \cap (\text{frontier } S \cup \text{frontier } T)$
is *Elementary-Topology.frontier-Int*{proof}

tts-lemma *frontier-closures*:

assumes $S \subseteq U$
shows $\text{frontier } S = \text{closure } S \cap \text{closure } (- S \cap U)$
is *Elementary-Topology.frontier-closures*{proof}

tts-lemma *frontier-Int-subset*:

assumes $S \subseteq U$ **and** $T \subseteq U$
shows $\text{frontier } (S \cap T) \subseteq \text{frontier } S \cup \text{frontier } T$
is *Elementary-Topology.frontier-Int-subset*{proof}

tts-lemma *frontier-Int-closed*:

assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *closed* S **and** *closed* T
shows $\text{frontier } (S \cap T) = \text{frontier } S \cap T \cup S \cap \text{frontier } T$
is *Elementary-Topology.frontier-Int-closed*{proof}

tts-lemma *frontier-subset-closed*:

assumes $S \subseteq U$ **and** *closed* S
shows $\text{frontier } S \subseteq S$
is *Elementary-Topology.frontier-subset-closed*{proof}

tts-lemma *frontier-subset-eq*:

assumes $S \subseteq U$
shows $(\text{frontier } S \subseteq S) = \text{closed } S$
is *Elementary-Topology.frontier-subset-eq*{proof}

tts-lemma *frontier-complement*:

assumes $S \subseteq U$
shows $\text{frontier } (- S \cap U) = \text{frontier } S$
is *Elementary-Topology.frontier-complement*{proof}

tts-lemma *frontier-Un-subset*:

assumes $S \subseteq U$ **and** $T \subseteq U$
shows $\text{frontier } (S \cup T) \subseteq \text{frontier } S \cup \text{frontier } T$
is *Elementary-Topology.frontier-Un-subset*{proof}

tts-lemma *frontier-disjoint-eq*:

assumes $S \subseteq U$
shows $(\text{frontier } S \cap S = \{\}) = \tau S$
is *Elementary-Topology.frontier-disjoint-eq*{proof}

tts-lemma *frontier-interiors*:

assumes $s \subseteq U$
shows $\text{frontier } s = - \text{interior } s \cap U - \text{interior } (- s \cap U)$
is *Elementary-Topology.frontier-interiors*{proof}

tts-lemma *frontier-interior-subset*:

assumes $S \subseteq U$
shows $\text{frontier } (\text{interior } S) \subseteq \text{frontier } S$
is *Elementary-Topology.frontier-interior-subset*{proof}

tts-lemma *compact-Un*:

assumes $s \subseteq U$ **and** $t \subseteq U$ **and** *compact* s **and** *compact* t
shows *compact* $(s \cup t)$
is *Elementary-Topology.compact-Un*{proof}

tts-lemma *closed-Int-compact*:

assumes $s \subseteq U$ **and** $t \subseteq U$ **and** *closed* s **and** *compact* t
shows *compact* $(s \cap t)$
is *Elementary-Topology.closed-Int-compact* \langle *proof* \rangle

tts-lemma *countably-compact-imp-compact*:

assumes $U \subseteq U$
and $B \subseteq \text{Pow } U$
and *countably-compact* U
and *countable* B
and *Ball* $B \tau$
and $\bigwedge T x. [[T \subseteq U; x \in U; \tau T; x \in T; x \in U]] \implies \exists y \in B. x \in y \wedge y \cap U \subseteq T$
shows *compact* U
is *Elementary-Topology.countably-compact-imp-compact* \langle *proof* \rangle

end

tts-context

tts: $(?a \text{ to } U)$
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through** (*insert closure-eq-empty, blast*)

begin

tts-lemma *closure-interior*:

assumes $S \subseteq U$
shows *closure* $S = - \text{interior } (- S \cap U) \cap U$
is *Elementary-Topology.closure-interior* \langle *proof* \rangle

end

tts-context

tts: $(?a \text{ to } U)$
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$
through (*insert compact-empty, fastforce dest: subset-singletonD*)

begin

tts-lemma *compact-Union*:

assumes $S \subseteq \text{Pow } U$
and *finite* S
and $\bigwedge T. [[T \subseteq U; T \in S]] \implies \text{compact } T$
shows *compact* $(\bigcup S)$
is *Elementary-Topology.compact-Union* \langle *proof* \rangle

end

tts-context

tts: $(?a \text{ to } U)$
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through**
 (
 insert
 interior-empty
 closure-ow-def

```

    closed-UNIV
    compact-empty
    compact-ow-def,
    auto
  )
begin

tts-lemma compactI:
  assumes  $s \subseteq U$ 
  and  $\bigwedge C. [[C \subseteq Pow\ U; Ball\ C\ \tau; s \subseteq \bigcup\ C]] \implies$ 
     $\exists x \subseteq Pow\ U. x \subseteq C \wedge finite\ x \wedge s \subseteq \bigcup\ x$ 
  shows compact  $s$ 
  given topological-space-class.compactI {proof}

tts-lemma compactE:
  assumes  $S \subseteq U$ 
  and  $\mathcal{T} \subseteq Pow\ U$ 
  and compact  $S$ 
  and  $S \subseteq \bigcup\ \mathcal{T}$ 
  and  $\bigwedge B. B \in \mathcal{T} \implies \tau\ B$ 
  and  $\bigwedge \mathcal{T}'. [[\mathcal{T}' \subseteq Pow\ U; \mathcal{T}' \subseteq \mathcal{T}; finite\ \mathcal{T}'; S \subseteq \bigcup\ \mathcal{T}']] \implies thesis$ 
  shows thesis
  given topological-space-class.compactE
  {proof}

tts-lemma compact-fip:
  assumes  $U \subseteq U$ 
  shows compact  $U =$ 
  (
     $\forall x \subseteq Pow\ U.$ 
    Ball  $x$  closed  $\longrightarrow$ 
     $(\forall y \subseteq Pow\ U. y \subseteq x \longrightarrow finite\ y \longrightarrow U \cap (\bigcap\ y \cap U) \neq \{\}) \longrightarrow$ 
     $U \cap (\bigcap\ x \cap U) \neq \{\}$ 
  )
  given topological-space-class.compact-fip {proof}

tts-lemma compact-imp-fip:
  assumes  $S \subseteq U$ 
  and  $Fa \subseteq Pow\ U$ 
  and compact  $S$ 
  and  $\bigwedge T. [[T \subseteq U; T \in Fa]] \implies closed\ T$ 
  and  $\bigwedge F'. [[F' \subseteq Pow\ U; finite\ F'; F' \subseteq Fa]] \implies S \cap (\bigcap\ F' \cap U) \neq \{\}$ 
  shows  $S \cap (\bigcap\ Fa \cap U) \neq \{\}$ 
  is topological-space-class.compact-imp-fip {proof}

tts-lemma closed-limpt:
  assumes  $S \subseteq U$ 
  shows closed  $S = (\forall x \in U. x \ll islimpt \gg S \longrightarrow x \in S)$ 
  is Elementary-Topology.closed-limpt {proof}

tts-lemma open-interior:
  assumes  $S \subseteq U$ 
  shows  $\tau$  (interior  $S$ )
  is Elementary-Topology.open-interior {proof}

tts-lemma interior-subset:
  assumes  $S \subseteq U$ 
  shows interior  $S \subseteq S$ 

```

```

    is Elementary-Topology.interior-subset{proof}

tts-lemma interior-open:
  assumes  $S \subseteq U$  and  $\tau S$ 
  shows interior  $S = S$ 
  is Elementary-Topology.interior-open{proof}

tts-lemma interior-eq:
  assumes  $S \subseteq U$ 
  shows  $(\text{interior } S = S) = \tau S$ 
  is Elementary-Topology.interior-eq{proof}

tts-lemma interior-UNIV: interior  $U = U$ 
  is Elementary-Topology.interior-UNIV{proof}

tts-lemma interior-interior:
  assumes  $S \subseteq U$ 
  shows interior (interior  $S$ ) = interior  $S$ 
  is Elementary-Topology.interior-interior{proof}

tts-lemma interior-closure:
  assumes  $S \subseteq U$ 
  shows interior  $S = - \text{closure } (- S \cap U) \cap U$ 
  is Elementary-Topology.interior-closure{proof}

tts-lemma finite-imp-compact:
  assumes  $s \subseteq U$  and finite  $s$ 
  shows compact  $s$ 
  is Elementary-Topology.finite-imp-compact{proof}

tts-lemma countably-compactE:
  assumes  $s \subseteq U$ 
    and  $C \subseteq \text{Pow } U$ 
    and countably-compact  $s$ 
    and Ball  $C \tau$ 
    and  $s \subseteq \bigcup C$ 
    and countable  $C$ 
    and  $\bigwedge C'. [\![C' \subseteq \text{Pow } U; C' \subseteq C; \text{finite } C'; s \subseteq \bigcup C'\!] \implies \text{thesis}$ ]
  shows thesis
  is Elementary-Topology.countably-compactE{proof}

end

tts-context
  tts: (?a to  $U$ )
  rewriting ctr-simps
  substituting topological-space-ow-axioms
  eliminating  $\langle ?U \neq \{\} \rangle$  and  $\langle ?A \subseteq U \rangle$  through (insert interior-empty, auto)
begin

tts-lemma interior-unique:
  assumes  $S \subseteq U$ 
    and  $T \subseteq S$ 
    and  $\tau T$ 
    and  $\bigwedge T'. [\![T' \subseteq S; \tau T'\!] \implies T' \subseteq T$ ]
  shows interior  $S = T$ 
  given Elementary-Topology.interior-unique
  {proof}

```

end

tts-context

tts: (*?a to U*) and (*?b to $\langle U_2::'b \text{ set} \rangle$*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through** (*simp add: subset-iff filterlim-iff*)

begin

tts-lemma *open-UN:*

assumes $A \subseteq U_2$
and $\forall x \in U_2. B x \subseteq U$
and $\forall x \in A. \tau (B x)$
shows $\tau (\bigcup (B \text{ ' } A))$
is *topological-space-class.open-UN*{proof}

tts-lemma *open-Collect-ex:*

assumes $\bigwedge i. i \in U_2 \implies \tau \{x. P i x \wedge x \in U\}$
shows $\tau \{x \in U. \exists i \in U_2. P i x\}$
is *open-Collect-ex*{proof}

end

tts-context

tts: (*?a to U*) and (*?b to $\langle U_2::'b \text{ set} \rangle$*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through** (*unfold closed-ow-def finite-def, auto*)

begin

tts-lemma *open-INT:*

assumes $A \subseteq U_2$ **and** $\forall x \in U_2. B x \subseteq U$ **and** *finite A* **and** $\forall x \in A. \tau (B x)$
shows $\tau (\bigcap (B \text{ ' } A) \cap U)$
is *topological-space-class.open-INT*{proof}

tts-lemma *closed-INT:*

assumes $A \subseteq U_2$ **and** $\forall x \in U_2. B x \subseteq U$ **and** $\forall x \in A. \text{closed } (B x)$
shows $\text{closed } (\bigcap (B \text{ ' } A) \cap U)$
is *topological-space-class.closed-INT*{proof}

tts-lemma *closed-UN:*

assumes $A \subseteq U_2$
and $\forall x \in U_2. B x \subseteq U$
and *finite A*
and $\forall x \in A. \text{closed } (B x)$
shows $\text{closed } (\bigcup (B \text{ ' } A))$
is *topological-space-class.closed-UN*{proof}

end

tts-context

tts: (*?a to U*) and (*?b to $\langle U_2::'b \text{ set} \rangle$*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through** (*insert closed-empty, auto*)

begin

tts-lemma *closed-Collect-all*:

assumes $\bigwedge i. i \in U_2 \implies \text{local.closed } \{x. P \ i \ x \wedge x \in U\}$
shows $\text{local.closed } \{x \in U. \forall i \in U_2. P \ i \ x\}$
is *topological-space-class.closed-Collect-all*\langleproof\rangle

tts-lemma *compactE-image*:

assumes $S \subseteq U$
and $C \subseteq U_2$
and $\forall x \in U_2. f \ x \subseteq U$
and *compact* S
and $\bigwedge T. [[T \in U_2; T \in C]] \implies \tau (f \ T)$
and $S \subseteq \bigcup (f \ ' \ C)$
and $\bigwedge C'. [[C' \subseteq U_2; C' \subseteq C; \text{finite } C'; S \subseteq \bigcup (f \ ' \ C')]] \implies \text{thesis}$
shows *thesis*
is *topological-space-class.compactE-image*\langleproof\rangle

end

tts-context

tts: (*?a* to U) **and** (*?b* to $\langle U_2::'b \ \text{set} \rangle$)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through** (*simp, blast | simp*)

begin

tts-lemma *ne-compact-imp-fip-image*:

assumes $s \subseteq U$
and $I \subseteq U_2$
and $\forall x \in U_2. f \ x \subseteq U$
and *compact* s
and $\bigwedge i. [[i \in U_2; i \in I]] \implies \text{closed } (f \ i)$
and $\bigwedge I'. [[I' \subseteq U_2; \text{finite } I'; I' \subseteq I]] \implies s \cap (\bigcap (f \ ' \ I') \cap U) \neq \{\}$
shows $s \cap (\bigcap (f \ ' \ I) \cap U) \neq \{\}$
is *topological-space-class.compact-imp-fip-image*\langleproof\rangle

end

end

3.16.4 Further results

lemma *topological-basis-closed*:

assumes *topological-basis-ow* $U \ \tau \ B$
shows $B \subseteq \text{Pow } U$
 \langleproof\rangle

lemma *ts-open-eq-ts-open*:

assumes *topological-space-ow* $U \ \tau'$ **and** $\bigwedge s. s \subseteq U \implies \tau' \ s = \tau \ s$
shows *topological-space-ow* $U \ \tau$
 \langleproof\rangle

lemma (*in* *topological-space-ow*) *topological-basis-closed*:

assumes *topological-basis-ow* $U \ \tau \ B$
shows $B \subseteq \text{Pow } U$
 \langleproof\rangle

3.17 Relativization of the results related to the countability properties of topological spaces

3.17.1 First countable topological space

Definitions and common properties

```

locale first-countable-topology-ow =
  topological-space-ow  $U$   $\tau$  for  $U :: 'at$  set and  $\tau +$ 
  assumes first-countable-basis:
  (
     $\forall x \in U.$ 
    (
       $\exists B :: nat \Rightarrow 'at$  set.
       $(\forall i. B\ i \subseteq U \wedge x \in B\ i \wedge \tau (B\ i)) \wedge$ 
       $(\forall S. S \subseteq U \wedge \tau S \wedge x \in S \longrightarrow (\exists i. B\ i \subseteq S))$ 
    )
  )

```

```

locale ts-fct-ow =
  ts: topological-space-ow  $U_1$   $\tau_1 +$  fct: first-countable-topology-ow  $U_2$   $\tau_2$ 
  for  $U_1 :: 'at$  set and  $\tau_1$  and  $U_2 :: 'bt$  set and  $\tau_2$ 
begin

```

```

  sublocale topological-space-pair-ow  $U_1$   $\tau_1$   $U_2$   $\tau_2$  <proof>

```

```

end

```

```

locale first-countable-topology-pair-ow =
  fct1: first-countable-topology-ow  $U_1$   $\tau_1 +$ 
  fct2: first-countable-topology-ow  $U_2$   $\tau_2$ 
  for  $U_1 :: 'at$  set and  $\tau_1$  and  $U_2 :: 'bt$  set and  $\tau_2$ 
begin

```

```

  sublocale ts-fct-ow  $U_1$   $\tau_1$   $U_2$   $\tau_2$  <proof>

```

```

end

```

Transfer rules

```

context
  includes lifting-syntax
begin

```

```

private lemma first-countable-topology-transfer-h:
   $(\forall i. B\ i \subseteq \text{Collect } (\text{Domainp } A) \wedge x \in B\ i \wedge \tau (B\ i)) =$ 
   $(B \text{ ' Collect top } \subseteq \{Aa. Aa \subseteq \text{Collect } (\text{Domainp } A)\} \wedge$ 
   $(\forall i. x \in B\ i \wedge \tau (B\ i)))$ 
  <proof>

```

```

lemma first-countable-topology-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique  $A$  right-total  $A$ 
  shows
   $((\text{rel-set } A \text{ ==> } (=)) \text{ ==> } (=)$ 
   $(\text{first-countable-topology-ow } (\text{Collect } (\text{Domainp } A))))$ 
  class.first-countable-topology
  <proof>

```

end

Relativization

context *first-countable-topology-ow*
begin

tts-context

tts: (?*a* to *U*)

rewriting *ctr-simps*

substituting *first-countable-topology-ow-axioms*

eliminating $\langle ?U \neq \{\} \rangle$ through *simp*

begin

tts-lemma *countable-basis-at-decseq*:

assumes $x \in U$

and $\wedge A.$

[[

$range\ A \subseteq Pow\ U;$

$\wedge i. \tau\ (A\ i);$

$\wedge i. x \in A\ i;$

$\wedge S. [[S \subseteq U; \tau\ S; x \in S]] \implies \forall_F\ i\ in\ sequentially. A\ i \subseteq S$

]] $\implies thesis$

shows *thesis*

is *first-countable-topology-class.countable-basis-at-decseq*{proof}

tts-lemma *first-countable-basisE*:

assumes $x \in U$

and $\wedge A.$

[[

$\mathcal{A} \subseteq Pow\ U;$

countable $\mathcal{A};$

$\wedge A. [[A \subseteq U; A \in \mathcal{A}]] \implies x \in A;$

$\wedge A. [[A \subseteq U; A \in \mathcal{A}]] \implies \tau\ A;$

$\wedge S. [[S \subseteq U; \tau\ S; x \in S]] \implies \exists A \in \mathcal{A}. A \subseteq S]] \implies thesis$

shows *thesis*

is *first-countable-topology-class.first-countable-basisE*{proof}

tts-lemma *first-countable-basis-Int-stableE*:

assumes $x \in U$

and $\wedge A.$

[[

$\mathcal{A} \subseteq Pow\ U;$

countable $\mathcal{A};$

$\wedge A. [[A \subseteq U; A \in \mathcal{A}]] \implies x \in A;$

$\wedge A. [[A \subseteq U; A \in \mathcal{A}]] \implies \tau\ A;$

$\wedge S. [[S \subseteq U; \tau\ S; x \in S]] \implies \exists A \in \mathcal{A}. A \subseteq S;$

$\wedge A\ B. [[A \subseteq U; B \subseteq U; A \in \mathcal{A}; B \in \mathcal{A}]] \implies A \cap B \in \mathcal{A}$

]] $\implies thesis$

shows *thesis*

is *first-countable-topology-class.first-countable-basis-Int-stableE*{proof}

end

end

3.17.2 Topological space with a countable basis

Definitions and common properties

locale *countable-basis-ow* =
topological-space-ow U τ **for** $U :: 'at\ set$ **and** $\tau +$
fixes $B :: 'at\ set\ set$
assumes *is-basis*: *topological-basis-ow* U τ B
and *countable-basis*: *countable* B
begin

lemma *B-ss-PowU[simp]*: $B \subseteq Pow\ U$
<proof>

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *countable-basis-transfer[transfer-rule]*:
assumes [*transfer-rule*]: *bi-unique* A *right-total* A
shows
 $((rel\ set\ A\ ==>\ (=))\ ==>\ rel\ set\ (rel\ set\ A)\ ==>\ (=))$
 $(countable\ basis\ ow\ (Collect\ (Domainp\ A)))\ countable\ basis$
<proof>

end

Relativization

context *countable-basis-ow*
begin

tts-context
tts: (*?a to U*)
rewriting *ctr-simps*
substituting *countable-basis-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through** *auto*
applying [*OF B-ss-PowU*]
begin

tts-lemma *open-countable-basis-ex*:
assumes $X \subseteq U$ **and** $\tau\ X$
shows $\exists B' \in Pow\ (Pow\ U). B' \subseteq B \wedge X = \bigcup B'$
is *countable-basis.open-countable-basis-ex**<proof>*

tts-lemma *countable-dense-exists*:
 $\exists D \in Pow\ U.$
 $countable\ D \wedge$
 $(\forall X \in Pow\ U. \tau\ X \longrightarrow X \neq \{\} \longrightarrow (\exists d \in D. d \in X))$
is *countable-basis.countable-dense-exists**<proof>*

tts-lemma *open-countable-basisE*:
assumes $X \subseteq U$
and $\tau\ X$
and $\bigwedge B'. [[B' \subseteq Pow\ U; B' \subseteq B; X = \bigcup B']] \implies thesis$

shows *thesis*

is *countable-basis.open-countable-basisE* \langle *proof* \rangle

tts-lemma *countable-dense-setE*:

assumes $\wedge D$.

$[[D \subseteq U; \text{countable } D; \wedge X. [[X \subseteq U; \tau X; X \neq \{\}]] \implies \exists x \in D. x \in X]] \implies \text{thesis}$

shows *thesis*

is *countable-basis.countable-dense-setE* \langle *proof* \rangle

end

end

3.17.3 Second countable topological space

Definitions and common properties

locale *second-countable-topology-ow* =

topological-space-ow U τ **for** U :: 'at set **and** τ +

assumes *second-countable-basis*:

$\exists B \subseteq \text{Pow } U. \text{countable } B \wedge (\forall S \subseteq U. \tau S = \text{generate-topology-on } B \cup S)$

Transfer rules

context

includes *lifting-syntax*

begin

lemma *second-countable-topology-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique* A *right-total* A

shows

$((\text{rel-set } A \implies (=)) \implies (=))$

$(\text{second-countable-topology-ow } (\text{Collect } (\text{Domainp } A)))$

class.second-countable-topology

\langle *proof* \rangle

end

Relativization

context *second-countable-topology-ow*

begin

tts-context

tts: (*?a* **to** U)

rewriting *ctr-simps*

substituting *second-countable-topology-ow-axioms*

eliminating $\langle ?U \neq \{\} \rangle$ **through** (*unfold topological-basis-ow-def*, *auto*)

begin

tts-lemma *ex-countable-basis*:

$\exists B \in \text{Pow } (\text{Pow } U). \text{countable } B \wedge (\text{on } U \text{ with } \tau : \langle \text{topological-basis} \rangle B)$

is *Elementary-Topology.ex-countable-basis* \langle *proof* \rangle

end

tts-context

tts: (*?a* **to** U)

rewriting *ctr-simps*

substituting *second-countable-topology-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through** (*auto simp: countable-subset*)
begin

tts-lemma *countable-dense-exists:*

$\exists D \in Pow\ U. \text{countable } D \wedge (\forall X \in Pow\ U. \tau\ X \longrightarrow X \neq \{\} \longrightarrow (\exists d \in D. d \in X))$
is *Elementary-Topology.countable-dense-exists* $\langle \text{proof} \rangle$

tts-lemma *countable-dense-setE:*

assumes $\wedge D.$

\llbracket
 $D \subseteq U;$
countable $D;$
 $\wedge X. \llbracket X \subseteq U; \tau\ X; X \neq \{\} \rrbracket \Longrightarrow \exists d \in D. d \in X$
 $\rrbracket \Longrightarrow \textit{thesis}$

shows *thesis*

is *Elementary-Topology.countable-dense-setE* $\langle \text{proof} \rangle$

tts-lemma *univ-second-countable:*

assumes $\wedge \mathcal{B}.$

\llbracket
 $\mathcal{B} \subseteq Pow\ U;$
countable $\mathcal{B};$
 $\wedge C. \llbracket C \subseteq U; C \in \mathcal{B} \rrbracket \Longrightarrow \tau\ C;$
 $\wedge S. \llbracket S \subseteq U; \tau\ S \rrbracket \Longrightarrow \exists U \in Pow\ (Pow\ U). U \subseteq \mathcal{B} \wedge S = \bigcup U$
 $\rrbracket \Longrightarrow \textit{thesis}$

shows *thesis*

is *Elementary-Topology.univ-second-countable* $\langle \text{proof} \rangle$

tts-lemma *Lindelof:*

assumes $\mathcal{F} \subseteq Pow\ U$

and $\wedge S. \llbracket S \subseteq U; S \in \mathcal{F} \rrbracket \Longrightarrow \tau\ S$

and $\wedge \mathcal{F}'. \llbracket \mathcal{F}' \subseteq Pow\ U; \mathcal{F}' \subseteq \mathcal{F}; \text{countable } \mathcal{F}'; \bigcup \mathcal{F}' = \bigcup \mathcal{F} \rrbracket \Longrightarrow \textit{thesis}$

shows *thesis*

is *Elementary-Topology.Lindelof* $\langle \text{proof} \rangle$

tts-lemma *countable-disjoint-open-subsets:*

assumes $\mathcal{F} \subseteq Pow\ U$ **and** $\wedge S. \llbracket S \subseteq U; S \in \mathcal{F} \rrbracket \Longrightarrow \tau\ S$ **and** *disjoint* \mathcal{F}

shows *countable* \mathcal{F}

is *Elementary-Topology.countable-disjoint-open-subsets* $\langle \text{proof} \rangle$

end

end

3.18 Relativization of the results about ordered topological spaces

3.18.1 Ordered topological space

Definitions and common properties

```

locale order-topology-ow =
  order-ow U le ls for U :: 'at set and le ls +
  fixes  $\tau$  :: 'at set  $\Rightarrow$  bool
  assumes open-generated-order:  $s \subseteq U \Longrightarrow$ 
     $\tau$  s =
    (
      (
        in-topology-generated-by
        (
          ( $\lambda a.$  (on U with ( $\langle_{ow}$ ) :  $\{..< a\}$ ))) ' U  $\cup$ 
          ( $\lambda a.$  (on U with ( $\langle_{ow}$ ) :  $\{a <..\}$ ))) ' U
        )
      )
    )
  begin

  sublocale topological-space-ow
  {proof}

  end

  locale ts-ot-ow =
    ts: topological-space-ow U1  $\tau_1$  + ot: order-topology-ow U2 le2 ls2  $\tau_2$ 
    for U1 :: 'at set and  $\tau_1$  and U2 :: 'bt set and le2 ls2  $\tau_2$ 
  begin

  sublocale topological-space-pair-ow U1  $\tau_1$  U2  $\tau_2$  {proof}

  end

  locale order-topology-pair-ow =
    ot1: order-topology-ow U1 le1 ls1  $\tau_1$  + ot2: order-topology-ow U2 le2 ls2  $\tau_2$ 
    for U1 :: 'at set and le1 ls1  $\tau_1$  and U2 :: 'bt set and le2 ls2  $\tau_2$ 
  begin

  sublocale ts-ot-ow U1  $\tau_1$  U2 le2 ls2  $\tau_2$  {proof}

  end

Transfer rules

context
  includes lifting-syntax
begin

lemma order-topology-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique A right-total A
  shows
  (
    ( $A \Longrightarrow A \Longrightarrow (=)$ )  $\Longrightarrow$ 
  )

```

```

    (A ==> A ==> (=)) ==>
    (rel-set A ==> (=)) ==>
    (=)
  ) (order-topology-ow (Collect (Domainp A))) class.order-topology
  <proof>

```

end

Relativization

context order-topology-ow

begin

tts-context

tts: (?a to U)

substituting order-topology-ow-axioms

eliminating <?U ≠ {}> through simp

begin

tts-lemma open-greaterThan:

assumes $a \in U$

shows $\tau \{a <_{ow} ..\}$

is order-topology-class.open-greaterThan<proof>

tts-lemma open-lessThan:

assumes $a \in U$

shows $\tau \{.. <_{ow} a\}$

is order-topology-class.open-lessThan<proof>

tts-lemma open-greaterThanLessThan:

assumes $a \in U$ and $b \in U$

shows $\tau \{a <_{ow} .. <_{ow} b\}$

is order-topology-class.open-greaterThanLessThan<proof>

end

end

3.18.2 Linearly ordered topological space

Definitions and common properties

locale linorder-topology-ow =

linorder-ow U le ls + order-topology-ow U le ls τ

for U :: 'at set and le ls τ

locale ts-lt-ow =

ts: topological-space-ow U₁ τ_1 + lt: linorder-topology-ow U₂ le₂ ls₂ τ_2

for U₁ :: 'at set and τ_1 and U₂ :: 'bt set and le₂ ls₂ τ_2

begin

sublocale ts-ot-ow U₁ τ_1 U₂ le₂ ls₂ τ_2 <proof>

end

locale ot-lt-ow =

ot: order-topology-ow U₁ le₁ ls₁ τ_1 + lt: linorder-topology-ow U₂ le₂ ls₂ τ_2

for U₁ :: 'at set and le₁ ls₁ τ_1 and U₂ :: 'bt set and le₂ ls₂ τ_2

begin

```

sublocale ts-lt-ow  $U_1 \tau_1 U_2 le_2 ls_2 \tau_2$   $\langle proof \rangle$ 
sublocale order-topology-pair-ow  $U_1 le_1 ls_1 \tau_1 U_2 le_2 ls_2 \tau_2$   $\langle proof \rangle$ 

```

```

end

```

```

locale linorder-topology-pair-ow =
  lt1: linorder-topology-ow  $U_1 le_1 ls_1 \tau_1$  + lt2: linorder-topology-ow  $U_2 le_2 ls_2 \tau_2$ 
  for  $U_1 :: 'at\ set$  and  $le_1\ ls_1\ \tau_1$  and  $U_2 :: 'bt\ set$  and  $le_2\ ls_2\ \tau_2$ 
begin

```

```

sublocale ot-lt-ow  $U_1 le_1 ls_1 \tau_1 U_2 le_2 ls_2 \tau_2$   $\langle proof \rangle$ 

```

```

end

```

Transfer rules

```

context
  includes lifting-syntax
begin

```

```

lemma linorder-topology-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique A right-total A
  shows
    (
      ( $A \implies A \implies (=) \implies$ )
      ( $A \implies A \implies (=) \implies$ )
      (rel-set  $A \implies (=) \implies$ )
      (=)
    )
    (linorder-topology-ow (Collect (Domainp A))) class.linorder-topology
     $\langle proof \rangle$ 

```

```

end

```

Relativization

```

context linorder-topology-ow
begin

```

```

tts-context
  tts: (?a to U)
  rewriting ctr-simps
  substituting linorder-topology-ow-axioms
  eliminating  $\langle ?U \neq \{\} \rangle$  through clarsimp
begin

```

```

tts-lemma open-right:
  assumes  $S \subseteq U$ 
  and  $x \in U$ 
  and  $y \in U$ 
  and  $\tau S$ 
  and  $x \in S$ 
  and  $x <_{ow} y$ 
  shows  $\exists z \in U. x <_{ow} z \wedge (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{x..z\}) \subseteq S$ 
  is linorder-topology-class.open-right $\langle proof \rangle$ 

```

```

tts-lemma open-left:

```

assumes $S \subseteq U$
and $x \in U$
and $y \in U$
and τS
and $x \in S$
and $y <_{ow} x$
shows $\exists z \in U. z <_{ow} x \wedge \{z <_{ow} ..x\} \subseteq S$
is *linorder-topology-class.open-left*{proof}

tts-lemma *connectedD-interval*:

assumes $U \subseteq U$
and $x \in U$
and $y \in U$
and $z \in U$
and *connected* U
and $x \in U$
and $y \in U$
and $x \leq_{ow} z$
and $z \leq_{ow} y$
shows $z \in U$
is *linorder-topology-class.connectedD-interval*{proof}

tts-lemma *connected-contains-Icc*:

assumes $A \subseteq U$
and $a \in U$
and $b \in U$
and *connected* A
and $a \in A$
and $b \in A$
shows $\{a.._{ow} b\} \subseteq A$
is *Topological-Spaces.connected-contains-Icc*{proof}

tts-lemma *connected-contains-Ioo*:

assumes $A \subseteq U$
and $a \in U$
and $b \in U$
and *connected* A
and $a \in A$
and $b \in A$
shows $\{a <_{ow} .. <_{ow} b\} \subseteq A$
is *Topological-Spaces.connected-contains-Ioo*{proof}

end

tts-context

tts: ($?a$ to U)
rewriting *ctr-simps*
substituting *linorder-topology-ow-axioms*
eliminating $\langle ?U \neq \{\} \rangle$ **through** *clarsimp*
begin

tts-lemma *not-in-connected-cases*:

assumes $S \subseteq U$
and $x \in U$
and *connected* S
and $x \notin S$
and $S \neq \{\}$
and $\llbracket \text{bdd-above } S; \wedge y. \llbracket y \in U; y \in S \rrbracket \implies y \leq_{ow} x \rrbracket \implies \textit{thesis}$

and $[[\text{bdd-below } S; \wedge y. [[y \in U; y \in S]] \implies x \leq_{ow} y]] \implies \text{thesis}$
shows *thesis*
is *linorder-topology-class.not-in-connected-cases* $\langle \text{proof} \rangle$

tts-lemma *compact-attains-sup*:

assumes $S \subseteq U$
and *compact* S
and $S \neq \{\}$
shows $\exists x \in S. \forall y \in S. y \leq_{ow} x$
is *linorder-topology-class.compact-attains-sup* $\langle \text{proof} \rangle$

tts-lemma *compact-attains-inf*:

assumes $S \subseteq U$
and *compact* S
and $S \neq \{\}$
shows $\exists x \in S. \text{Ball } S ((\leq_{ow}) x)$
is *linorder-topology-class.compact-attains-inf* $\langle \text{proof} \rangle$

end

end

3.19 Relativization of the results about product topologies

3.19.1 Definitions and common properties

```

ud⟨
  open-prod-inst.open-prod::
    ('a::topological-space × 'b::topological-space) set ⇒ -
  ⟩
ud⟨open::('a::topological-space × 'b::topological-space) set ⇒ bool⟩

ctr relativization
synthesis ctr-simps
assumes [transfer-domain-rule, transfer-rule]:
  Domainp B = (λx. x ∈ U1) Domainp A = (λx. x ∈ U2)
and [transfer-rule]: bi-unique A right-total A
  bi-unique B right-total B
trp (?'a A) and (?'b B)
in open-ow: open-prod.with-def
  (⟨'(/on - - with - - : «open» -/)⟩ [1000, 999, 1000, 999, 1000] 10)

locale product-topology-ow =
  ts1: topological-space-ow U1 τ1 + ts2: topological-space-ow U2 τ2
for U1 :: 'at set and τ1 :: 'at set ⇒ bool
  and U2 :: 'bt set and τ2 :: 'bt set ⇒ bool +
fixes τ :: ('at × 'bt) set ⇒ bool
assumes open-prod[tts-implicit]: τ = open-ow U1 U2 τ2 τ1
begin

sublocale topological-space-ow ⟨U1 × U2⟩ τ
⟨proof⟩

end

```

3.19.2 Transfer rules

```

lemma (in product-topology-ow) open-with-oo-transfer[transfer-rule]:
includes lifting-syntax
fixes A :: ['at, 'a] ⇒ bool
  and B :: ['bt, 'b] ⇒ bool
assumes tdr-U1[transfer-domain-rule]: Domainp A = (λx. x ∈ U1)
  and [transfer-rule]: bi-unique A right-total A
  and tdr-U2[transfer-domain-rule]: Domainp B = (λx. x ∈ U2)
  and [transfer-rule]: bi-unique B right-total B
  and τ1τ1'[transfer-rule]: (rel-set A ==> (=)) τ1 τ1'
  and τ2τ2'[transfer-rule]: (rel-set B ==> (=)) τ2 τ2'
shows (rel-set (rel-prod A B) ==> (=)) τ (open-prod.with τ2' τ1')
⟨proof⟩

```

3.19.3 Relativization

```

context product-topology-ow
begin

tts-context
tts: (?'a to U1) and (?'b to U2)
rewriting ctr-simps
substituting ts1.topological-space-ow-axioms
  and ts2.topological-space-ow-axioms
eliminating ⟨?U ≠ {}⟩ through (fold tts-implicit, insert closed-empty, simp)

```

```

applying [folded tts-implicit]
begin

tts-lemma open-prod-intro:
  assumes  $S \subseteq U_1 \times U_2$ 
  and  $\bigwedge x. \llbracket x \in U_1 \times U_2; x \in S \rrbracket \implies$ 
     $\exists A \in \text{Pow } U_1. \exists B \in \text{Pow } U_2. \tau_1 A \wedge \tau_2 B \wedge A \times B \subseteq S \wedge x \in A \times B$ 
  shows  $\tau S$ 
  is open-prod-intro(proof)

tts-lemma open-Times:
  assumes  $S \subseteq U_1$  and  $T \subseteq U_2$  and  $\tau_1 S$  and  $\tau_2 T$ 
  shows  $\tau (S \times T)$ 
  is open-Times(proof)

tts-lemma open-vimage-fst:
  assumes  $S \subseteq U_1$  and  $\tau_1 S$ 
  shows  $\tau (\text{fst} -' S \cap U_1 \times U_2)$ 
  is open-vimage-fst(proof)

tts-lemma closed-vimage-fst:
  assumes  $S \subseteq U_1$  and  $ts_1.\text{closed } S$ 
  shows closed ( $\text{fst} -' S \cap U_1 \times U_2$ )
  is closed-vimage-fst(proof)

tts-lemma closed-Times:
  assumes  $S \subseteq U_1$  and  $T \subseteq U_2$  and  $ts_1.\text{closed } S$  and  $ts_2.\text{closed } T$ 
  shows closed ( $S \times T$ )
  is closed-Times(proof)

tts-lemma open-image-fst:
  assumes  $S \subseteq U_1 \times U_2$  and  $\tau S$ 
  shows  $\tau_1 (\text{fst} ' S)$ 
  is open-image-fst(proof)

tts-lemma open-image-snd:
  assumes  $S \subseteq U_1 \times U_2$  and  $\tau S$ 
  shows  $\tau_2 (\text{snd} ' S)$ 
  is open-image-snd(proof)

end

tts-context
  tts: (?a to  $U_1$ ) and (?b to  $U_2$ )
  rewriting ctr-simps
  substituting  $ts_1.\text{topological-space-ow-axioms}$ 
  and  $ts_2.\text{topological-space-ow-axioms}$ 
  eliminating  $\langle ?U \neq \{\} \rangle$ 
  through (fold tts-implicit, unfold connected-ow-def, simp)
  applying [folded tts-implicit]
begin

tts-lemma connected-Times:
  assumes  $S \subseteq U_1$  and  $T \subseteq U_2$  and  $ts_1.\text{connected } S$  and  $ts_2.\text{connected } T$ 
  shows connected ( $S \times T$ )
  is connected-Times(proof)

tts-lemma connected-Times-eq:

```

```

assumes  $S \subseteq U_1$  and  $T \subseteq U_2$ 
shows
   $connected (S \times T) = (S = \{\}) \vee T = \{\} \vee ts_1.connected S \wedge ts_2.connected T$ 
is connected-Times-eq{proof}

end

tts-context
tts: (?'b to  $U_1$ ) and (?'a to  $U_2$ )
rewriting ctr-simps
substituting ts1.topological-space-ow-axioms
  and ts2.topological-space-ow-axioms
eliminating  $\langle ?U \neq \{\} \rangle$  through (fold tts-implicit, insert closed-empty, simp)
applying [folded tts-implicit]
begin

tts-lemma open-vimage-snd:
assumes  $S \subseteq U_2$  and  $\tau_2 S$ 
shows  $\tau (snd -' S \cap U_1 \times U_2)$ 
is open-vimage-snd{proof}

tts-lemma closed-vimage-snd:
assumes  $S \subseteq U_2$  and ts2.closed S
shows  $closed (snd -' S \cap U_1 \times U_2)$ 
is closed-vimage-snd{proof}

end

end

```

TTS Vector Spaces

4.1 Introduction

4.1.1 Background

The content of this chapter is an adoption of the applied relativization study presented in [14] to the ETTS. The content of this chapter incorporates many elements of the content of the aforementioned relativization study without an explicit reference. Nonetheless, no attempt was made to ensure that the theorems obtained as a result of this work are identical to the theorems obtained in [14].

4.1.2 Prerequisites

ctr parametricity

in *bij-betw-ow*: *bij-betw-def*

lemma *bij-betw-parametric'*[*transfer-rule*]:

includes *lifting-syntax*

assumes *bi-unique A*

shows $((A \text{ ===> } A) \text{ ===> } \text{rel-set } A \text{ ===> } \text{rel-set } A \text{ ===> } (=))$

bij-betw bij-betw

<proof>

lemma *vimage-transfer*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique B right-total A*

shows

$((A \text{ ===> } B) \text{ ===> } (\text{rel-set } B) \text{ ===> } \text{rel-set } A)$

$(\lambda f s. (\text{vimage } f s) \cap (\text{Collect } (\text{Domainp } A))) (-')$

<proof>

lemma *Eps-unique-transfer-lemma*:

includes *lifting-syntax*

assumes [*transfer-rule*]:

right-total A (A ===> (=)) f g (A ===> (=)) f' g'

and holds: $\exists x. \text{Domainp } A x \wedge f x$

and unique-g: $\bigwedge x y. [\![\ g x; g y \]\!] \implies g' x = g' y$

shows $f' (\text{Eps } (\lambda x. \text{Domainp } A x \wedge f x)) = g' (\text{Eps } g)$

<proof>

4.2 Groups

4.2.1 Definitions and elementary properties

locale *semigroup-add-ow* =

fixes $S :: 'a$ set **and** $pls :: 'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\langle \oplus_{ow} \rangle$ 65)

assumes *add-assoc*:

$\llbracket a \in S; b \in S; c \in S \rrbracket \Longrightarrow (a \oplus_{ow} b) \oplus_{ow} c = a \oplus_{ow} (b \oplus_{ow} c)$

and *add-closed*: $\llbracket a \in S; b \in S \rrbracket \Longrightarrow a \oplus_{ow} b \in S$

begin

lemma *add-closed'*[*simp*]: $\forall x \in S. \forall y \in S. x \oplus_{ow} y \in S$ (*proof*)

end

locale *ab-semigroup-add-ow* = *semigroup-add-ow* +

assumes *add-commute*: $\llbracket a \in S; b \in S \rrbracket \Longrightarrow a \oplus_{ow} b = b \oplus_{ow} a$

locale *comm-monoid-add-ow* = *ab-semigroup-add-ow* +

fixes z

assumes *add-zero*: $a \in S \Longrightarrow z \oplus_{ow} a = a$

and *zero-closed*[*simp*]: $z \in S$

begin

lemma *carrier-ne*[*simp*]: $S \neq \{\}$ (*proof*)

end

definition *sum-with pls z f S* =

(
 if $\exists C. f \text{ ' } S \subseteq C \wedge \text{comm-monoid-add-ow } C \text{ pls } z$
 then $\text{Finite-Set.fold } (pls \circ f) z S$
 else z
)

lemma *sum-with-empty*[*simp*]: $\text{sum-with pls } z f \{\} = z$
 (*proof*)

lemma *sum-with-cases*[*case-names comm zero*]:

assumes $\bigwedge C. \llbracket f \text{ ' } S \subseteq C; \text{comm-monoid-add-ow } C \text{ pls } z \rrbracket \Longrightarrow$

$P (\text{Finite-Set.fold } (pls \circ f) z S)$

and $(\bigwedge C. \text{comm-monoid-add-ow } C \text{ pls } z \Longrightarrow (\exists s \in S. f s \notin C)) \Longrightarrow P z$

shows $P (\text{sum-with pls } z f S)$

(*proof*)

context *comm-monoid-add-ow*

begin

lemma *sum-with-infinite*: $\text{infinite } A \Longrightarrow \text{sum-with } (\oplus_{ow}) z g A = z$
 (*proof*)

context

begin

abbreviation $pls' :: 'a \Rightarrow 'a \Rightarrow 'a$

where $pls' \equiv \lambda x y. (\text{if } x \in S \text{ then } x \text{ else } z) \oplus_{ow} (\text{if } y \in S \text{ then } y \text{ else } z)$

lemma *fold-pls'-closed*: $\text{Finite-Set.fold } (pls' \circ g) z A \in S$ **if** $g \text{ ' } A \subseteq S$

<proof>

lemma *fold-pls'-eq:*

assumes $g \text{ ' } A \subseteq S$

shows $\text{Finite-Set.fold } (pls' \circ g) \ z \ A = \text{Finite-Set.fold } (pls \circ g) \ z \ A$

<proof>

lemma *sum-with-closed:*

assumes $g \text{ ' } A \subseteq S$

shows $\text{sum-with } pls \ z \ g \ A \in S$

<proof>

lemma *sum-with-insert:*

assumes $g\text{-into: } g \ x \in S \ g \text{ ' } A \subseteq S$

and $A: \text{finite } A$

and $x: x \notin A$

shows $\text{sum-with } pls \ z \ g \ (\text{insert } x \ A) = (g \ x) \oplus_{ow} (\text{sum-with } pls \ z \ g \ A)$

<proof>

end

end

locale *ab-group-add-ow = comm-monoid-add-ow +*

fixes $mns \ um$

assumes $ab\text{-left-minus: } a \in S \implies (um \ a) \oplus_{ow} a = z$

and $ab\text{-diff-conv-add-uminus:}$

$[[a \in S; b \in S]] \implies mns \ a \ b = a \oplus_{ow} (um \ b)$

and $uminus\text{-closed: } a \in S \implies um \ a \in S$

4.2.2 Instances (by type class constraints)

lemma *semigroup-add-ow-Ball-def:*

$\text{semigroup-add-ow } S \ pls \longleftrightarrow$

$(\forall a \in S. \forall b \in S. \forall c \in S. pls \ (pls \ a \ b) \ c =$

$pls \ a \ (pls \ b \ c)) \wedge (\forall a \in S. \forall b \in S. pls \ a \ b \in S)$

<proof>

lemma *ab-semigroup-add-ow-Ball-def:*

$ab\text{-semigroup-add-ow } S \ pls \longleftrightarrow$

$\text{semigroup-add-ow } S \ pls \wedge (\forall a \in S. \forall b \in S. pls \ a \ b = pls \ b \ a)$

<proof>

lemma *comm-monoid-add-ow-Ball-def:*

$\text{comm-monoid-add-ow } S \ pls \ z \longleftrightarrow$

$ab\text{-semigroup-add-ow } S \ pls \wedge (\forall a \in S. pls \ z \ a = a) \wedge z \in S$

<proof>

lemma *comm-monoid-add-ow[simp]:*

$\text{comm-monoid-add-ow UNIV } (+) \ (0::'a::\text{comm-monoid-add})$

<proof>

lemma *ab-group-add-ow-Ball-def:*

$ab\text{-group-add-ow } S \ pls \ z \ mns \ um \longleftrightarrow$

$\text{comm-monoid-add-ow } S \ pls \ z \wedge$

$(\forall a \in S. pls \ (um \ a) \ a = z) \wedge$

$(\forall a \in S. \forall b \in S. mns \ a \ b = pls \ a \ (um \ b)) \wedge$

$(\forall a \in S. um \ a \in S)$

<proof>

lemma *sum-with*[*ud-with*]: *sum* = *sum-with* (+) 0

<proof>

lemmas [*tts-implicit*] = *sum-with*[*symmetric*]

4.2.3 Transfer rules

context

includes *lifting-syntax*

begin

lemma *semigroup-add-on-with-transfer*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique* *A*

shows (*rel-set* *A* \implies (*A* \implies *A* \implies *A*) \implies (=))
semigroup-add-ow semigroup-add-ow

<proof>

lemma *Domainp-applyI*:

includes *lifting-syntax*

shows (*A* \implies *B*) *f g* \implies *A x y* \implies *Domainp B (f x)*

<proof>

lemma *Domainp-apply2I*:

includes *lifting-syntax*

shows (*A* \implies *B* \implies *C*) *f g* \implies *A x y* \implies *B x' y'* \implies *Domainp C (f x x')*

<proof>

lemma *ab-semigroup-add-on-with-transfer*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique* *A*

shows

(*rel-set* *A* \implies (*A* \implies *A* \implies *A*) \implies (=))
ab-semigroup-add-ow ab-semigroup-add-ow

<proof>

lemma *right-total-semigroup-add-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total* *A* *bi-unique* *A*

shows ((*A* \implies *A* \implies *A*) \implies (=))

(*semigroup-add-ow (Collect (Domainp A))*) *class.semigroup-add*

<proof>

lemma *comm-monoid-add-on-with-transfer*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique* *A*

shows

(*rel-set* *A* \implies (*A* \implies *A* \implies *A*) \implies *A* \implies (=))
comm-monoid-add-ow comm-monoid-add-ow

<proof>

lemma *right-total-ab-semigroup-add-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total* *A* *bi-unique* *A*

shows

((*A* \implies *A* \implies *A*) \implies (=))

(*ab-semigroup-add-ow (Collect (Domainp A))*) *class.ab-semigroup-add*

<proof>

lemma *right-total-comm-monoid-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique A*
shows $((A \implies A \implies A) \implies A \implies (=))$
(comm-monoid-add-ow (Collect (Domainp A))) class.comm-monoid-add
<proof>

lemma *ab-group-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique A*
shows
 $((A \implies A \implies A) \implies A \implies (A \implies A \implies A) \implies (A \implies A) \implies (=))$
(ab-group-add-ow (Collect (Domainp A))) class.ab-group-add
<proof>

lemma *ex-comm-monoid-add-around-imageE*:
assumes *ex-comm: $\exists C. f ' S \subseteq C \wedge \text{comm-monoid-add-ow } C \text{ pls zero}$*
and transfers:
(A \implies A \implies A) pls pls'
A zero zero'
Domainp (rel-set B) S
and in-dom: $\wedge x. x \in S \implies \text{Domainp } A (f x)$
obtains C where
comm-monoid-add-ow C pls zero f ' S \subseteq C Domainp (rel-set A) C
<proof>

lemma *Domainp-sum-with*:
includes *lifting-syntax*
assumes $\wedge x. x \in t \implies \text{Domainp } A (r x) t \subseteq \text{Collect } (\text{Domainp } A)$
and transfer-rules[*transfer-rule*]: $(A \implies A \implies A) p p' A z z'$
shows *DsI: Domainp A (sum-with p z r t)*
<proof>

lemma *sum-with-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique A bi-unique B*
shows $((A \implies A \implies A) \implies A \implies (B \implies A) \implies \text{rel-set } B \implies A)$
sum-with sum-with
<proof>

end

4.2.4 Relativization.

context *ab-group-add-ow*

begin

tts-context

tts: *(?'a to S)*

rewriting *ctr-simps*

substituting *comm-monoid-add-ow-axioms*

eliminating $\langle S \neq \{ \} \rangle$ **through** *auto*

applying [*OF add-closed' zero-closed*]

begin

tts-lemma *mono-neutral-cong-left*:

assumes *range h \subseteq S*

and *range g \subseteq S*

and *finite T*

and *Sa \subseteq T*

and $\forall x \in T - Sa. h x = z$
and $\wedge x. x \in Sa \implies g x = h x$
shows *sum-with* $(\oplus_{ow}) z g Sa = \text{sum-with } (\oplus_{ow}) z h T$
is *sum.mono-neutral-cong-left*{proof}

end

end

4.3 Modules

4.3.1 module-with

```

locale module-with = ab-group-add plusM zeroM minusM uminusM
for plusM :: ['m, 'm] ⇒ 'm (infixl ‹+M› 65)
and zeroM (‹0M›)
and minusM (infixl ‹-M› 65)
and uminusM (‹-M -› [81] 80) +
fixes scale :: ['cr1::comm-ring-1, 'm] ⇒ 'm (infixr ‹*swith› 75)
assumes scale-right-distrib[algebra-simps]:
  a *swith (x +M y) = a *swith x +M a *swith y
and scale-left-distrib[algebra-simps]:
  (a + b) *swith x = a *swith x +M b *swith x
and scale-scale[simp]: a *swith (b *swith x) = (a * b) *swith x
and scale-one[simp]: 1 *swith x = x

```

```

lemma module-with-overloaded[ud-with]: module = module-with (+) 0 (-) uminus
  ‹proof›

```

```

locale module-pair-with =
  M1: module-with plusM-1 zeroM-1 minusM-1 uminusM-1 scale1 +
  M2: module-with plusM-2 zeroM-2 minusM-2 uminusM-2 scale2
for plusM-1 :: ['m-1, 'm-1] ⇒ 'm-1 (infixl ‹+M'-1› 65)
and zeroM-1 (‹0M'-1›)
and minusM-1 (infixl ‹-M'-1› 65)
and uminusM-1 (‹-M'-1 -› [81] 80)
and scale1 (infixr ‹*swith'-1› 75)
and plusM-2 :: ['m-2, 'm-2] ⇒ 'm-2 (infixl ‹+M'-2› 65)
and zeroM-2 (‹0M'-2›)
and minusM-2 (infixl ‹-M'-2› 65)
and uminusM-2 (‹-M'-2 -› [81] 80)
and scale2 (infixr ‹*swith'-2› 75)

```

```

lemma module-pair-with-overloaded[ud-with]:
  module-pair =
  (
    λscale1 scale2.
      module-pair-with (+) 0 (-) uminus scale1 (+) 0 (-) uminus scale2
  )
  ‹proof›

```

```

locale module-hom-with =
  M1: module-with plusM-1 zeroM-1 minusM-1 uminusM-1 scale1 +
  M2: module-with plusM-2 zeroM-2 minusM-2 uminusM-2 scale2
for plusM-1 :: ['m-1, 'm-1] ⇒ 'm-1 (infixl ‹+M'-1› 65)
and zeroM-1 (‹0M'-1›)
and minusM-1 (infixl ‹-M'-1› 65)
and uminusM-1 (‹-M'-1 -› [81] 80)
and scale1 (infixr ‹*swith'-1› 75)
and plusM-2 :: ['m-2, 'm-2] ⇒ 'm-2 (infixl ‹+M'-2› 65)
and zeroM-2 (‹0M'-2›)
and minusM-2 (infixl ‹-M'-2› 65)
and uminusM-2 (‹-M'-2 -› [81] 80)
and scale2 (infixr ‹*swith'-2› 75) +
fixes f :: 'm-1 ⇒ 'm-2
assumes add: f (b1 +M-1 b2) = f b1 +M-2 f b2
and scale: f (r *swith-1 b) = r *swith-2 f b

```

begin

sublocale *module-pair-with* \langle *proof* \rangle

end

lemma *module-hom-with-overloaded*[*ud-with*]:

```

module-hom =
  (
     $\lambda$ scale1 scale2.
      module-hom-with (+) 0 (-) uminus scale1 (+) 0 (-) uminus scale2
    )
   $\langle$ proof $\rangle$ 
ud  $\langle$ module.subspace $\rangle$  ( $\langle$ (with - - - : «subspace» -) $\rangle$  [1000, 999, 998, 1000] 10)
ud  $\langle$ module.span $\rangle$  ( $\langle$ (with - - - : «span» -) $\rangle$  [1000, 999, 998, 1000] 10)
ud  $\langle$ module.dependent $\rangle$ 
  ( $\langle$ (with - - - - : «dependent» -) $\rangle$  [1000, 999, 998, 997, 1000] 10)
ud  $\langle$ module.representation $\rangle$ 
  (
    (with - - - - : «representation» - -) $\rangle$ 
    [1000, 999, 998, 997, 1000, 999] 10
  )

```

abbreviation *independent-with*

```

( $\langle$ (with - - - - : «independent» -) $\rangle$  [1000, 999, 998, 997, 1000] 10)
where
  (with zeroCR1 zeroM scaleM plusM : «independent» s)  $\equiv$ 
  -(with zeroCR1 zeroM scaleM plusM : «dependent» s)

```

lemma *span-with-transfer*[*transfer-rule*]:

```

includes lifting-syntax
assumes [transfer-rule]: right-total A bi-unique A
shows
  (
    A  $\equiv\equiv\equiv$ 
    (A  $\equiv\equiv\equiv$  A  $\equiv\equiv\equiv$  A)  $\equiv\equiv\equiv$ 
    ((=)  $\equiv\equiv\equiv$  A  $\equiv\equiv\equiv$  A)  $\equiv\equiv\equiv$ 
    rel-set A  $\equiv\equiv\equiv$ 
    rel-set A
  ) span.with span.with
   $\langle$ proof $\rangle$ 

```

lemma *dependent-with-transfer*[*transfer-rule*]:

```

includes lifting-syntax
assumes [transfer-rule]: right-total A bi-unique A
shows
  (
    (=)  $\equiv\equiv\equiv$ 
    A  $\equiv\equiv\equiv$ 
    (A  $\equiv\equiv\equiv$  A  $\equiv\equiv\equiv$  A)  $\equiv\equiv\equiv$ 
    ((=)  $\equiv\equiv\equiv$  A  $\equiv\equiv\equiv$  A)  $\equiv\equiv\equiv$ 
    rel-set A  $\equiv\equiv\equiv$ 
    (=)
  ) dependent.with dependent.with
   $\langle$ proof $\rangle$ 

```

ctr relativization

synthesis *ctr-simps*

assumes [*transfer-rule*]: *is-equality A bi-unique B*
trp (*?'a A*) **and** (*?'b B*)
in *subspace-with: subspace.with-def*

4.3.2 module-ow

Definitions and common properties

Single module.

locale *module-ow* = *ab-group-add-ow* U_M *plus_M* *zero_M* *minus_M* *uminus_M*
for $U_M :: 'm$ *set*
and *plus_M* (**infixl** $\langle +_M \rangle$ 65)
and *zero_M* ($\langle 0_M \rangle$)
and *minus_M* (**infixl** $\langle -_M \rangle$ 65)
and *uminus_M* ($\langle -_M \rightarrow [81] 80 \rangle$ +)
fixes *scale* :: [*cr1::comm-ring-1, 'm*] \Rightarrow *'m* (**infixr** $\langle *_M \rangle$ 75)
assumes *scale-closed[simp, intro]*: $x \in U_M \Longrightarrow a *_M x \in U_M$
and *scale-right-distrib[algebra-simps]*:
 $\llbracket x \in U_M; y \in U_M \rrbracket \Longrightarrow a *_M (x +_M y) = a *_M x +_M a *_M y$
and *scale-left-distrib[algebra-simps]*:
 $x \in U_M \Longrightarrow (a + b) *_M x = a *_M x +_M b *_M x$
and *scale-scale[simp]*:
 $x \in U_M \Longrightarrow a *_M (b *_M x) = (a * b) *_M x$
and *scale-one[simp]*: $x \in U_M \Longrightarrow 1 *_M x = x$
begin

lemma *scale-closed'[simp]*: $\forall a. \forall x \in U_M. a *_M x \in U_M$ *<proof>*

lemma *minus-closed'[simp]*: $\forall x \in U_M. \forall y \in U_M. x -_M y \in U_M$
<proof>

lemma *uminus-closed'[simp]*: $\forall x \in U_M. -_M x \in U_M$ *<proof>*

tts-register-sbts $\langle *_M \rangle \mid U_M$
<proof>

tts-register-sbts *plus_M* $\mid U_M$
<proof>

tts-register-sbts *zero_M* $\mid U_M$
<proof>

end

Pair of modules.

locale *module-pair-ow* =
 M_1 : *module-ow* U_{M-1} *plus_{M-1}* *zero_{M-1}* *minus_{M-1}* *uminus_{M-1}* *scale₁* +
 M_2 : *module-ow* U_{M-2} *plus_{M-2}* *zero_{M-2}* *minus_{M-2}* *uminus_{M-2}* *scale₂*
for $U_{M-1} :: 'm-1$ *set*
and *plus_{M-1}* (**infixl** $\langle +_{M'-1} \rangle$ 65)
and *zero_{M-1}* ($\langle 0_{M'-1} \rangle$)
and *minus_{M-1}* (**infixl** $\langle -_{M'-1} \rangle$ 65)
and *uminus_{M-1}* ($\langle -_{M'-1} \rightarrow [81] 80 \rangle$)
and *scale₁* :: [*cr1::comm-ring-1, 'm-1*] \Rightarrow *'m-1* (**infixr** $\langle *_M \rangle$ 75)
and $U_{M-2} :: 'm-2$ *set*
and *plus_{M-2}* (**infixl** $\langle +_{M'-2} \rangle$ 65)
and *zero_{M-2}* ($\langle 0_{M'-2} \rangle$)
and *minus_{M-2}* (**infixl** $\langle -_{M'-2} \rangle$ 65)

and $uminus_{M-2}$ ($\langle -_{M'-2} \rightarrow$ [81] 80)
and $scale_2 :: ['cr1::comm-ring-1, 'm-2] \Rightarrow 'm-2$ (**infixr** $\langle *_{SM'-2} \rangle$ 75)

Module homomorphisms.

locale *module-hom-ow* =

M_1 : *module-ow* U_{M-1} $plus_{M-1}$ $zero_{M-1}$ $minus_{M-1}$ $uminus_{M-1}$ $scale_1$ +

M_2 : *module-ow* U_{M-2} $plus_{M-2}$ $zero_{M-2}$ $minus_{M-2}$ $uminus_{M-2}$ $scale_2$

for $U_{M-1} :: 'm-1$ *set*

and $plus_{M-1}$ (**infixl** $\langle +_{M'-1} \rangle$ 65)

and $zero_{M-1}$ ($\langle 0_{M'-1} \rangle$)

and $minus_{M-1}$ (**infixl** $\langle -_{M'-1} \rangle$ 65)

and $uminus_{M-1}$ ($\langle -_{M'-1} \rightarrow$ [81] 80)

and $scale_1 :: ['cr1::comm-ring-1, 'm-1] \Rightarrow 'm-1$ (**infixr** $\langle *_{SM'-1} \rangle$ 75)

and $U_{M-2} :: 'm-2$ *set*

and $plus_{M-2}$ (**infixl** $\langle +_{M'-2} \rangle$ 65)

and $zero_{M-2}$ ($\langle 0_{M'-2} \rangle$)

and $minus_{M-2}$ (**infixl** $\langle -_{M'-2} \rangle$ 65)

and $uminus_{M-2}$ ($\langle -_{M'-2} \rightarrow$ [81] 80)

and $scale_2 :: ['cr1::comm-ring-1, 'm-2] \Rightarrow 'm-2$ (**infixr** $\langle *_{SM'-2} \rangle$ 75) +

fixes $f :: 'm-1 \Rightarrow 'm-2$

assumes *f-closed*[*simp*]: $f \text{ ` } U_{M-1} \subseteq U_{M-2}$

and *add*: $[[b1 \in U_{M-1}; b2 \in U_{M-1}]] \Longrightarrow f (b1 +_{M-1} b2) = f b1 +_{M-2} f b2$

and *scale*: $[[r \in U_{CR1}; b \in U_{M-1}]] \Longrightarrow f (r *_{SM-1} b) = r *_{SM-2} f b$

begin

tts-register-sbts $f \mid \langle U_{M-1} \rangle$ **and** $\langle U_{M-2} \rangle$ *<proof>*

lemma *f-closed*'[*simp*]: $\forall x \in U_{M-1}. f x \in U_{M-2}$ *<proof>*

sublocale *module-pair-ow* *<proof>*

end

Transfer.

lemma *module-with-transfer*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique B right-total B*

fixes *PP lhs*

defines

$PP \equiv$

(
 $(B \Longrightarrow B \Longrightarrow B) \Longrightarrow$
 $B \Longrightarrow$
 $(B \Longrightarrow B \Longrightarrow B) \Longrightarrow$
 $(B \Longrightarrow B) \Longrightarrow$
 $((=) \Longrightarrow B \Longrightarrow B) \Longrightarrow$
 $(=)$
)

and

$lhs \equiv$

(
 $\lambda plus_M zero_M minus_M uminus_M scale.$
 $module-ow (Collect (Domainp B)) plus_M zero_M minus_M uminus_M scale$
)

shows *PP lhs (module-with)*

<proof>

lemma *module-pair-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]:
bi-unique B₁ right-total B₁ bi-unique B₂ right-total B₂
fixes *PP lhs*
defines
PP \equiv
(

(*B₁* \implies *B₁* \implies *B₁*) \implies
B₁ \implies
(*B₁* \implies *B₁* \implies *B₁*) \implies
(*B₁* \implies *B₁*) \implies
((\equiv) \implies *B₁* \implies *B₁*) \implies
(*B₂* \implies *B₂* \implies *B₂*) \implies
B₂ \implies
(*B₂* \implies *B₂* \implies *B₂*) \implies
(*B₂* \implies *B₂*) \implies
((\equiv) \implies *B₂* \implies *B₂*) \implies
(\equiv)
)
and
lhs \equiv
(

 λ
plus_{M-1} zero_{M-1} minus_{M-1} uminus_{M-1} scale₁
plus_{M-2} zero_{M-2} minus_{M-2} uminus_{M-2} scale₂.
module-pair-ow
(*Collect (Domainp B₁)*) *plus_{M-1} zero_{M-1} minus_{M-1} uminus_{M-1} scale₁*
(*Collect (Domainp B₂)*) *plus_{M-2} zero_{M-2} minus_{M-2} uminus_{M-2} scale₂*
)
shows *PP lhs module-pair-with*
(*proof*)

lemma *module-hom-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]:
bi-unique B₁ right-total B₁ bi-unique B₂ right-total B₂
fixes *PP lhs*
defines
PP \equiv
(

(*B₁* \implies *B₁* \implies *B₁*) \implies
B₁ \implies
(*B₁* \implies *B₁* \implies *B₁*) \implies
(*B₁* \implies *B₁*) \implies
((\equiv) \implies *B₁* \implies *B₁*) \implies
(*B₂* \implies *B₂* \implies *B₂*) \implies
B₂ \implies
(*B₂* \implies *B₂* \implies *B₂*) \implies
(*B₂* \implies *B₂*) \implies
((\equiv) \implies *B₂* \implies *B₂*) \implies
(*B₁* \implies *B₂*) \implies
(\equiv)
)
and
lhs \equiv
(

 λ

```

    plusM-1 zeroM-1 minusM-1 uminusM-1 scale1
    plusM-2 zeroM-2 minusM-2 uminusM-2 scale2
    f.
  module-hom-ow
    (Collect (Domainp B1)) plusM-1 zeroM-1 minusM-1 uminusM-1 scale1
    (Collect (Domainp B2)) plusM-2 zeroM-2 minusM-2 uminusM-2 scale2
    f
  )
  shows PP lhs module-hom-with
<proof>

```

4.3.3 module-on

Definitions and common properties

```

locale module-on =
  fixes UM
  and scale :: 'a::comm-ring-1 ⇒ 'b::ab-group-add ⇒ 'b (infixr < * s > 75)
  assumes scale-right-distrib-on[algebra-simps]:
    [[ x ∈ UM; y ∈ UM ]] ⇒ a * s (x + y) = a * s x + a * s y
  and scale-left-distrib-on[algebra-simps]:
    x ∈ UM ⇒ (a + b) * s x = a * s x + b * s x
  and scale-scale-on[simp]: x ∈ UM ⇒ a * s (b * s x) = (a * b) * s x
  and scale-one-on[simp]: x ∈ UM ⇒ 1 * s x = x
  and closed-add: [[ x ∈ UM; y ∈ UM ]] ⇒ x + y ∈ UM
  and closed-zero: 0 ∈ UM
  and closed-scale: x ∈ UM ⇒ a * s x ∈ UM

```

begin

lemma S-ne: $U_M \neq \{\}$ <proof>

```

lemma scale-minus-left-on:
  assumes x ∈ UM
  shows scale (- a) x = - scale a x
<proof>

```

```

lemma closed-uminus:
  assumes x ∈ UM
  shows -x ∈ UM
<proof>

```

```

sublocale implicitM: ab-group-add-ow UM <(+)> 0 <(-)> uminus
<proof>

```

```

sublocale implicitM: module-ow UM <(+)> 0 <(-)> uminus <(*s)>
<proof>

```

```

definition subspace :: 'b set ⇒ bool
  where subspace-on-def: subspace T ↔
    0 ∈ T ∧ (∀ x ∈ T. ∀ y ∈ T. x + y ∈ T) ∧ (∀ c. ∀ x ∈ T. c * s x ∈ T)

```

```

definition span :: 'b set ⇒ 'b set
  where span-on-def: span b = {sum (λa. r a * s a) t | t r. finite t ∧ t ⊆ b}

```

```

definition dependent :: 'b set ⇒ bool
  where dependent-on-def: dependent s ↔
    (∃ t u. finite t ∧ t ⊆ s ∧ (sum (λv. u v * s v) t = 0 ∧ (∃ v ∈ t. u v ≠ 0)))

```

lemma *implicit-subspace-with*[*tts-implicit*]: *subspace.with* (+) 0 (*s) = *subspace*
 ⟨*proof*⟩

lemma *implicit-dependent-with*[*tts-implicit*]:
dependent.with 0 0 (+) (*s) = *dependent*
 ⟨*proof*⟩

lemma *implicit-span-with*[*tts-implicit*]: *span.with* 0 (+) (*s) = *span*
 ⟨*proof*⟩

end

lemma *implicit-module-ow*[*tts-implicit*]:
module-ow U_M (+) 0 (-) *uminus* = *module-on* U_M
 ⟨*proof*⟩

locale *module-pair-on* =
 M_1 : *module-on* U_{M-1} *scale*₁ + M_2 : *module-on* U_{M-2} *scale*₂
for U_{M-1} :: 'b::*ab-group-add set*
and U_{M-2} :: 'c::*ab-group-add set*
and *scale*₁ :: 'a::*comm-ring-1* \Rightarrow - \Rightarrow - (**infixr** <*s₁> 75)
and *scale*₂ :: 'a::*comm-ring-1* \Rightarrow - \Rightarrow - (**infixr** <*s₂> 75)
begin

sublocale *implicit*_M: *module-pair-ow*
 U_{M-1} <(+)> 0 <(-)> *uminus* *scale*₁ U_{M-2} <(+)> 0 <(-)> *uminus* *scale*₂
 ⟨*proof*⟩

end

lemma *implicit-module-pair-ow*[*tts-implicit*]:
module-pair-ow U_{M-1} (+) 0 (-) *uminus* *scale*₁ U_{M-2} (+) 0 (-) *uminus* *scale*₂ =
module-pair-on U_{M-1} U_{M-2} *scale*₁ *scale*₂
 ⟨*proof*⟩

locale *module-hom-on* = M_1 : *module-on* U_{M-1} *scale*₁ + M_2 : *module-on* U_{M-2} *scale*₂
for U_{M-1} :: 'b::*ab-group-add set* **and** U_{M-2} :: 'c::*ab-group-add set*
and *scale*₁ :: 'a::*comm-ring-1* \Rightarrow 'b \Rightarrow 'b (**infixr** <*s₁> 75)
and *scale*₂ :: 'a::*comm-ring-1* \Rightarrow 'c \Rightarrow 'c (**infixr** <*s₂> 75) +
fixes *f* :: 'b \Rightarrow 'c
assumes *hom-closed*: *f* ' $U_{M-1} \subseteq U_{M-2}$
and *add*: $\bigwedge b_1 b_2. [\![b_1 \in U_{M-1}; b_2 \in U_{M-1}]\!] \Longrightarrow f (b_1 + b_2) = f b_1 + f b_2$
and *scale*: $\bigwedge b. b \in U_{M-1} \Longrightarrow f (r *s_1 b) = r *s_2 f b$
begin

sublocale *module-pair-on* U_{M-1} U_{M-2} *scale*₁ *scale*₂ ⟨*proof*⟩

sublocale *implicit*_M: *module-hom-ow*
 U_{M-1} <(+)> 0 <(-)> *uminus* *scale*₁ U_{M-2} <(+)> 0 <(-)> *uminus* *scale*₂
 ⟨*proof*⟩

end

lemma *implicit-module-hom-ow*[*tts-implicit*]:
module-hom-ow U_{M-1} (+) 0 (-) *uminus* *scale*₁ U_{M-2} (+) 0 (-) *uminus* *scale*₂ =
module-hom-on U_{M-1} U_{M-2} *scale*₁ *scale*₂
 ⟨*proof*⟩

4.3.4 Relativization.

context *module-on*
begin

tts-context

tts: (*?b* to $\langle U_M :: 'b \text{ set} \rangle$)
rewriting *ctr-simps*
substituting *implicit_M.module-ow-axioms*
and *implicit_M.ab-group-add-ow-axioms*
eliminating $\langle ?a \in U_M \rangle$ **and** $\langle ?B \subseteq U_M \rangle$ **through** *auto*
applying
 [*OF*
 implicit_M.carrier-ne
 implicit_M.add-closed'
 implicit_M.minus-closed'
 implicit_M.uminus-closed'
 implicit_M.scale-closed',
 unfolded tts-implicit
]

begin

tts-lemma *scale-left-commute:*

assumes $x \in U_M$
shows $a * s b * s x = b * s a * s x$
is *module.scale-left-commute*{proof}

tts-lemma *scale-zero-left:*

assumes $x \in U_M$
shows $0 * s x = 0$
is *module.scale-zero-left*{proof}

tts-lemma *scale-minus-left:*

assumes $x \in U_M$
shows $- a * s x = - (a * s x)$
is *module.scale-minus-left*{proof}

tts-lemma *scale-left-diff-distrib:*

assumes $x \in U_M$
shows $(a - b) * s x = a * s x - b * s x$
is *module.scale-left-diff-distrib*{proof}

tts-lemma *scale-sum-left:*

assumes $x \in U_M$
shows $\text{sum } f A * s x = (\sum a \in A. f a * s x)$
is *module.scale-sum-left*{proof}

tts-lemma *scale-zero-right:* $a * s 0 = 0$

is *module.scale-zero-right*{proof}

tts-lemma *scale-minus-right:*

assumes $x \in U_M$
shows $a * s - x = - (a * s x)$
is *module.scale-minus-right*{proof}

tts-lemma *scale-right-diff-distrib:*

assumes $x \in U_M$ **and** $y \in U_M$

shows $a * s (x - y) = a * s x - a * s y$
is *module.scale-right-diff-distrib*{proof}

tts-lemma *scale-sum-right*:

assumes $\text{range } f \subseteq U_M$
shows $a * s \text{ sum } f A = (\sum x \in A. a * s f x)$
is *module.scale-sum-right*{proof}

tts-lemma *sum-constant-scale*:

assumes $y \in U_M$
shows $(\sum x \in A. y) = \text{of-nat } (\text{card } A) * s y$
is *module.sum-constant-scale*{proof}

tts-lemma *subspace-def*:

assumes $S \subseteq U_M$
shows *subspace* $S =$
 $(0 \in S \wedge (\forall x. \forall y \in S. x * s y \in S) \wedge (\forall x \in S. \forall y \in S. x + y \in S))$
is *module.subspace-def*{proof}

tts-lemma *subspaceI*:

assumes $S \subseteq U_M$
and $0 \in S$
and $\bigwedge x y. [[x \in U_M; y \in U_M; x \in S; y \in S]] \implies x + y \in S$
and $\bigwedge c x. [[x \in U_M; x \in S]] \implies c * s x \in S$
shows *subspace* S
is *module.subspaceI*{proof}

tts-lemma *subspace-single-0*: *subspace* $\{0\}$

is *module.subspace-single-0*{proof}

tts-lemma *subspace-0*:

assumes $S \subseteq U_M$ **and** *subspace* S
shows $0 \in S$
is *module.subspace-0*{proof}

tts-lemma *subspace-add*:

assumes $S \subseteq U_M$ **and** *subspace* S **and** $x \in S$ **and** $y \in S$
shows $x + y \in S$
is *module.subspace-add*{proof}

tts-lemma *subspace-scale*:

assumes $S \subseteq U_M$ **and** *subspace* S **and** $x \in S$
shows $c * s x \in S$
is *module.subspace-scale*{proof}

tts-lemma *subspace-neg*:

assumes $S \subseteq U_M$ **and** *subspace* S **and** $x \in S$
shows $-x \in S$
is *module.subspace-neg*{proof}

tts-lemma *subspace-diff*:

assumes $S \subseteq U_M$ **and** *subspace* S **and** $x \in S$ **and** $y \in S$
shows $x - y \in S$
is *module.subspace-diff*{proof}

tts-lemma *subspace-sum*:

assumes $A \subseteq U_M$
and $\text{range } f \subseteq U_M$

and *subspace* A
and $\wedge x. x \in B \implies f x \in A$
shows $\text{sum } f B \in A$
is *module.subspace-sum*(*proof*)

tts-lemma *subspace-inter*:
assumes $A \subseteq U_M$ **and** $B \subseteq U_M$ **and** *subspace* A **and** *subspace* B
shows *subspace* $(A \cap B)$
is *module.subspace-inter*(*proof*)

tts-lemma *span-explicit'*:
assumes $b \subseteq U_M$
shows $\text{span } b =$
 $\{$
 $x \in U_M. \exists f.$
 $x = (\sum v \in \{x \in U_M. f x \neq 0\}. f v *s v) \wedge$
 $\text{finite } \{x \in U_M. f x \neq 0\} \wedge$
 $(\forall x \in U_M. f x \neq 0 \longrightarrow x \in b)$
 $\}$
is *module.span-explicit'*(*proof*)

tts-lemma *span-finite*:
assumes $S \subseteq U_M$ **and** *finite* S
shows $\text{span } S = \text{range } (\lambda u. \sum v \in S. u v *s v)$
is *module.span-finite*(*proof*)

tts-lemma *span-induct-alt*:
assumes $x \in U_M$
and $S \subseteq U_M$
and $x \in \text{span } S$
and $h 0$
and $\wedge c x y. [\![x \in U_M; y \in U_M; x \in S; h y]\!] \implies h (c *s x + y)$
shows $h x$
is *module.span-induct-alt*(*proof*)

tts-lemma *span-mono*:
assumes $B \subseteq U_M$ **and** $A \subseteq B$
shows $\text{span } A \subseteq \text{span } B$
is *module.span-mono*(*proof*)

tts-lemma *span-base*:
assumes $S \subseteq U_M$ **and** $a \in S$
shows $a \in \text{span } S$
is *module.span-base*(*proof*)

tts-lemma *span-superset*:
assumes $S \subseteq U_M$
shows $S \subseteq \text{span } S$
is *module.span-superset*(*proof*)

tts-lemma *span-zero*:
assumes $S \subseteq U_M$
shows $0 \in \text{span } S$
is *module.span-zero*(*proof*)

tts-lemma *span-add*:
assumes $x \in U_M$
and $S \subseteq U_M$

and $y \in U_M$
and $x \in \text{span } S$
and $y \in \text{span } S$
shows $x + y \in \text{span } S$
is *module.span-add*{proof}

tts-lemma *span-scale*:
assumes $x \in U_M$ **and** $S \subseteq U_M$ **and** $x \in \text{span } S$
shows $c * x \in \text{span } S$
is *module.span-scale*{proof}

tts-lemma *subspace-span*:
assumes $S \subseteq U_M$
shows *subspace* (*span* S)
is *module.subspace-span*{proof}

tts-lemma *span-neg*:
assumes $x \in U_M$ **and** $S \subseteq U_M$ **and** $x \in \text{span } S$
shows $-x \in \text{span } S$
is *module.span-neg*{proof}

tts-lemma *span-diff*:
assumes $x \in U_M$
and $S \subseteq U_M$
and $y \in U_M$
and $x \in \text{span } S$
and $y \in \text{span } S$
shows $x - y \in \text{span } S$
is *module.span-diff*{proof}

tts-lemma *span-sum*:
assumes $\text{range } f \subseteq U_M$ **and** $S \subseteq U_M$ **and** $\bigwedge x. x \in A \implies f x \in \text{span } S$
shows $\text{sum } f A \in \text{span } S$
is *module.span-sum*{proof}

tts-lemma *span-minimal*:
assumes $T \subseteq U_M$ **and** $S \subseteq T$ **and** *subspace* T
shows $\text{span } S \subseteq T$
is *module.span-minimal*{proof}

tts-lemma *span-subspace-induct*:
assumes $x \in U_M$
and $S \subseteq U_M$
and $P \subseteq U_M$
and $x \in \text{span } S$
and *subspace* P
and $\bigwedge x. x \in S \implies x \in P$
shows $x \in P$
given *module.span-subspace-induct*
 {proof}

tts-lemma *span-induct*:
assumes $x \in U_M$
and $S \subseteq U_M$
and $x \in \text{span } S$
and *subspace* $\{x. P x \wedge x \in U_M\}$
and $\bigwedge x. x \in S \implies P x$
shows $P x$

given *module.span-induct* {proof}

tts-lemma *span-empty*: $\text{span } \{\} = \{0\}$
is *module.span-empty*{proof}

tts-lemma *span-subspace*:
assumes $B \subseteq U_M$ **and** $A \subseteq B$ **and** $B \subseteq \text{span } A$ **and** *subspace* B
shows $\text{span } A = B$
is *module.span-subspace*{proof}

tts-lemma *span-span*:
assumes $A \subseteq U_M$
shows $\text{span } (\text{span } A) = \text{span } A$
is *module.span-span*{proof}

tts-lemma *span-add-eq*:
assumes $x \in U_M$ **and** $S \subseteq U_M$ **and** $y \in U_M$ **and** $x \in \text{span } S$
shows $(x + y \in \text{span } S) = (y \in \text{span } S)$
is *module.span-add-eq*{proof}

tts-lemma *span-add-eq2*:
assumes $y \in U_M$ **and** $S \subseteq U_M$ **and** $x \in U_M$ **and** $y \in \text{span } S$
shows $(x + y \in \text{span } S) = (x \in \text{span } S)$
is *module.span-add-eq2*{proof}

tts-lemma *span-singleton*:
assumes $x \in U_M$
shows $\text{span } \{x\} = \text{range } (\lambda k. k *s x)$
is *module.span-singleton*{proof}

tts-lemma *span-Un*:
assumes $S \subseteq U_M$ **and** $T \subseteq U_M$
shows $\text{span } (S \cup T) =$
 $\{x \in U_M. \exists a \in U_M. \exists b \in U_M. x = a + b \wedge a \in \text{span } S \wedge b \in \text{span } T\}$
is *module.span-Un*{proof}

tts-lemma *span-insert*:
assumes $a \in U_M$ **and** $S \subseteq U_M$
shows $\text{span } (\text{insert } a S) = \{x \in U_M. \exists y. x - y *s a \in \text{span } S\}$
is *module.span-insert*{proof}

tts-lemma *span-breakdown*:
assumes $S \subseteq U_M$ **and** $a \in U_M$ **and** $b \in S$ **and** $a \in \text{span } S$
shows $\exists x. a - x *s b \in \text{span } (S - \{b\})$
is *module.span-breakdown*{proof}

tts-lemma *span-breakdown-eq*:
assumes $x \in U_M$ **and** $a \in U_M$ **and** $S \subseteq U_M$
shows $(x \in \text{span } (\text{insert } a S)) = (\exists y. x - y *s a \in \text{span } S)$
is *module.span-breakdown-eq*{proof}

tts-lemma *span-clauses*:
 $\llbracket S \subseteq U_M; a \in S \rrbracket \implies a \in \text{span } S$
 $S \subseteq U_M \implies 0 \in \text{span } S$
 $\llbracket x \in U_M; S \subseteq U_M; y \in U_M; x \in \text{span } S; y \in \text{span } S \rrbracket \implies x + y \in \text{span } S$
 $\llbracket x \in U_M; S \subseteq U_M; x \in \text{span } S \rrbracket \implies c *s x \in \text{span } S$
is *module.span-clauses*{proof}

tts-lemma *span-eq-iff*:

assumes $s \subseteq U_M$

shows $(\text{span } s = s) = \text{subspace } s$

is *module.span-eq-iff*{proof}

tts-lemma *span-eq*:

assumes $S \subseteq U_M$ **and** $T \subseteq U_M$

shows $(\text{span } S = \text{span } T) = (S \subseteq \text{span } T \wedge T \subseteq \text{span } S)$

is *module.span-eq*{proof}

tts-lemma *eq-span-insert-eq*:

assumes $x \in U_M$ **and** $y \in U_M$ **and** $S \subseteq U_M$ **and** $x - y \in \text{span } S$

shows $\text{span } (\text{insert } x S) = \text{span } (\text{insert } y S)$

is *module.eq-span-insert-eq*{proof}

tts-lemma *dependent-mono*:

assumes $A \subseteq U_M$ **and** *dependent* B **and** $B \subseteq A$

shows *dependent* A

is *module.dependent-mono*{proof}

tts-lemma *independent-mono*:

assumes $A \subseteq U_M$ **and** \neg *dependent* A **and** $B \subseteq A$

shows \neg *dependent* B

is *module.independent-mono*{proof}

tts-lemma *dependent-zero*:

assumes $A \subseteq U_M$ **and** $0 \in A$

shows *dependent* A

is *module.dependent-zero*{proof}

tts-lemma *independent-empty*: \neg *dependent* $\{\}$

is *module.independent-empty*{proof}

tts-lemma *independentD*:

assumes $s \subseteq U_M$

and \neg *dependent* s

and *finite* t

and $t \subseteq s$

and $(\sum v \in t. u v * s v) = 0$

and $v \in t$

shows $u v = 0$

is *module.independentD*{proof}

tts-lemma *independent-Union-directed*:

assumes $C \subseteq \text{Pow } U_M$

and $\bigwedge c d. [[c \subseteq U_M; d \subseteq U_M; c \in C; d \in C]] \implies c \subseteq d \vee d \subseteq c$

and $\bigwedge c. [[c \subseteq U_M; c \in C]] \implies \neg$ *dependent* c

shows \neg *dependent* $(\bigcup C)$

is *module.independent-Union-directed*{proof}

tts-lemma *dependent-finite*:

assumes $S \subseteq U_M$ **and** *finite* S

shows *dependent* $S = (\exists x. (\exists y \in S. x y \neq 0) \wedge (\sum v \in S. x v * s v) = 0)$

is *module.dependent-finite*{proof}

tts-lemma *independentD-alt*:

assumes $B \subseteq U_M$

and $x \in U_M$

and \neg dependent B
and finite $\{x \in U_M. X x \neq 0\}$
and $\{x \in U_M. X x \neq 0\} \subseteq B$
and $(\sum x \mid x \in U_M \wedge X x \neq 0. X x *s x) = 0$
shows $X x = 0$
is *module.independentD-alt*{proof}

tts-lemma *spanning-subset-independent*:
assumes $A \subseteq U_M$ **and** $B \subseteq A$ **and** \neg dependent A **and** $A \subseteq \text{span } B$
shows $A = B$
is *module.spanning-subset-independent*{proof}

tts-lemma *unique-representation*:
assumes $\text{basis} \subseteq U_M$
and \neg dependent basis
and $\bigwedge v. [\![v \in U_M; f v \neq 0]\!] \implies v \in \text{basis}$
and $\bigwedge v. [\![v \in U_M; g v \neq 0]\!] \implies v \in \text{basis}$
and finite $\{x \in U_M. f x \neq 0\}$
and finite $\{x \in U_M. g x \neq 0\}$
and
 $(\sum v \in \{x \in U_M. f x \neq 0\}. f v *s v) = (\sum v \in \{x \in U_M. g x \neq 0\}. g v *s v)$
shows $\forall x \in U_M. f x = g x$
is *module.unique-representation[unfolded fun-eq-iff]*{proof}

tts-lemma *independentD-unique*:
assumes $B \subseteq U_M$
and \neg dependent B
and finite $\{x \in U_M. X x \neq 0\}$
and $\{x \in U_M. X x \neq 0\} \subseteq B$
and finite $\{x \in U_M. Y x \neq 0\}$
and $\{x \in U_M. Y x \neq 0\} \subseteq B$
and $(\sum x \mid x \in U_M \wedge X x \neq 0. X x *s x) =$
 $(\sum x \mid x \in U_M \wedge Y x \neq 0. Y x *s x)$
shows $\forall x \in U_M. X x = Y x$
is *module.independentD-unique[unfolded fun-eq-iff]*{proof}

tts-lemma *subspace-UNIV*: *subspace* U_M
is *module.subspace-UNIV*{proof}

tts-lemma *span-UNIV*: $\text{span } U_M = U_M$
is *module.span-UNIV*{proof}

tts-lemma *span-alt*:
assumes $B \subseteq U_M$
shows
 $\text{span } B =$
 $\{$
 $x \in U_M. \exists f.$
 $x = (\sum x \mid x \in U_M \wedge f x \neq 0. f x *s x) \wedge$
 $\text{finite } \{x \in U_M. f x \neq 0\} \wedge$
 $\{x \in U_M. f x \neq 0\} \subseteq B$
 $\}$
is *module.span-alt*{proof}

tts-lemma *dependent-alt*:
assumes $B \subseteq U_M$
shows *dependent* $B =$
 $($

```

     $\exists f.$ 
     $finite \{v \in U_M. f v \neq 0\} \wedge$ 
     $\{v \in U_M. f v \neq 0\} \subseteq B \wedge$ 
     $(\exists v \in U_M. f v \neq 0) \wedge$ 
     $(\sum x \mid x \in U_M \wedge f x \neq 0. f x * s x) = 0$ 
  )
  is module.dependent-alt{proof}

tts-lemma independent-alt:
  assumes  $B \subseteq U_M$ 
  shows
     $(\neg dependent B) =$ 
    (
       $\forall f.$ 
       $finite \{x \in U_M. f x \neq 0\} \longrightarrow$ 
       $\{x \in U_M. f x \neq 0\} \subseteq B \longrightarrow$ 
       $(\sum x \mid x \in U_M \wedge f x \neq 0. f x * s x) = 0 \longrightarrow$ 
       $(\forall x \in U_M. f x = 0)$ 
    )
  is module.independent-alt{proof}

tts-lemma subspace-Int:
  assumes  $range s \subseteq Pow U_M$  and  $\bigwedge i. i \in I \implies subspace (s i)$ 
  shows  $subspace (\bigcap (s ' I) \cap U_M)$ 
  is module.subspace-Int{proof}

tts-lemma subspace-Inter:
  assumes  $f \subseteq Pow U_M$  and  $Ball f subspace$ 
  shows  $subspace (\bigcap f \cap U_M)$ 
  is module.subspace-Inter{proof}

tts-lemma module-hom-scale-self: module-hom-on  $U_M U_M (*s) (*s) ((*s) c)$ 
  is module.module-hom-scale-self{proof}

tts-lemma module-hom-scale-left:
  assumes  $x \in U_M$ 
  shows  $module-hom-on UNIV U_M (*) (*s) (\lambda r. r * s x)$ 
  is module.module-hom-scale-left{proof}

tts-lemma module-hom-id: module-hom-on  $U_M U_M (*s) (*s) id$ 
  is module.module-hom-id{proof}

tts-lemma module-hom-ident: module-hom-on  $U_M U_M (*s) (*s) (\lambda x. x)$ 
  is module.module-hom-ident{proof}

tts-lemma module-hom-uminus: module-hom-on  $U_M U_M (*s) (*s) uminus$ 
  is module.module-hom-uminus{proof}

end

tts-context
  tts: (?b to  $\langle U_M :: 'b set \rangle$ )
  rewriting ctr-simps
  substituting  $implicit_M.module-ow-axioms$ 
  and  $implicit_M.ab-group-add-ow-axioms$ 
  eliminating  $\langle ?a \in U_M \rangle$  and  $\langle ?B \subseteq U_M \rangle$  through  $clarsimp$ 
  applying
  [

```

```

    OF
      implicit_M.carrier-ne
      implicit_M.add-closed'
      implicit_M.minus-closed'
      implicit_M.uminus-closed'
      implicit_M.scale-closed',
    unfolded tts-implicit
  ]
begin

tts-lemma span-explicit:
  assumes  $b \subseteq U_M$ 
  shows  $\text{span } b =$ 
     $\{x \in U_M. \exists y \subseteq U_M. \exists f. (\text{finite } y \wedge y \subseteq b) \wedge x = (\sum a \in y. f a * s a)\}$ 
  given module.span-explicit {proof}

tts-lemma span-unique:
  assumes  $S \subseteq U_M$ 
  and  $T \subseteq U_M$ 
  and  $S \subseteq T$ 
  and subspace  $T$ 
  and  $\wedge T'. [[T' \subseteq U_M; S \subseteq T'; \text{subspace } T']] \implies T \subseteq T'$ 
  shows  $\text{span } S = T$ 
  is module.span-unique{proof}

tts-lemma dependent-explicit:
  assumes  $V \subseteq U_M$ 
  shows dependent  $V =$ 
     $(\exists U \subseteq U_M. \exists f. \text{finite } U \wedge U \subseteq V \wedge (\exists v \in U. f v \neq 0) \wedge (\sum v \in U. f v * s v) = 0)$ 
  given module.dependent-explicit {proof}

tts-lemma independent-explicit-module:
  assumes  $V \subseteq U_M$ 
  shows  $(\neg \text{dependent } V) =$ 
    (
       $\forall U \subseteq U_M. \forall f. \forall v \in U_M.$ 
       $\text{finite } U \longrightarrow$ 
       $U \subseteq V \longrightarrow$ 
       $(\sum u \in U. f u * s u) = 0 \longrightarrow$ 
       $v \in U \longrightarrow$ 
       $f v = 0$ 
    )
  given module.independent-explicit-module {proof}

end

end

context module-pair-on
begin

tts-context
  tts: (?b to  $\langle U_{M-1}::'b \text{ set} \rangle$ ) and (?c to  $\langle U_{M-2}::'c \text{ set} \rangle$ )
  rewriting ctr-simps
  substituting  $M_1.\text{implicit}_M.\text{module-ow-axioms}$ 
  and  $M_2.\text{implicit}_M.\text{module-ow-axioms}$ 
  and  $M_1.\text{implicit}_M.\text{ab-group-add-ow-axioms}$ 
  and  $M_2.\text{implicit}_M.\text{ab-group-add-ow-axioms}$ 

```

and *implicit_M.module-pair-ow-axioms*
 eliminating through *auto*
 applying [*unfolded tts-implicit*]
 begin

tts-lemma *module-hom-zero: module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. 0)$
 is *module-pair.module-hom-zero*{*proof*}

tts-lemma *module-hom-add:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
 and $\forall x \in U_{M-1}. g x \in U_{M-2}$
 and *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
 and *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) g$
 shows *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. f x + g x)$
 is *module-pair.module-hom-add*{*proof*}

tts-lemma *module-hom-sub:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
 and $\forall x \in U_{M-1}. g x \in U_{M-2}$
 and *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
 and *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) g$
 shows *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. f x - g x)$
 is *module-pair.module-hom-sub*{*proof*}

tts-lemma *module-hom-neg:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$ and *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
 shows *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. - f x)$
 is *module-pair.module-hom-neg*{*proof*}

tts-lemma *module-hom-scale:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$ and *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
 shows *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. c *s_2 f x)$
 is *module-pair.module-hom-scale*{*proof*}

tts-lemma *module-hom-compose-scale:*

assumes $c \in U_{M-2}$ and *module-hom-on* $U_{M-1} UNIV (*s_1) (*) f$
 shows *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. f x *s_2 c)$
 is *module-pair.module-hom-compose-scale*{*proof*}

tts-lemma *module-hom-sum:*

assumes $\forall u. \forall v \in U_{M-1}. f u v \in U_{M-2}$
 and $\bigwedge i. i \in I \implies \text{module-hom-on } U_{M-1} U_{M-2} (*s_1) (*s_2) (f i)$
 and $I = \{\} \implies \text{module-on } U_{M-1} (*s_1) \wedge \text{module-on } U_{M-2} (*s_2)$
 shows *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. \sum i \in I. f i x)$
 is *module-pair.module-hom-sum*{*proof*}

tts-lemma *module-hom-eq-on-span:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
 and $\forall x \in U_{M-1}. g x \in U_{M-2}$
 and $B \subseteq U_{M-1}$
 and $x \in U_{M-1}$
 and *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
 and *module-hom-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) g$
 and $\bigwedge x. \llbracket x \in U_{M-1}; x \in B \rrbracket \implies f x = g x$
 and $x \in M_1.\text{span } B$
 shows $f x = g x$
 is *module-pair.module-hom-eq-on-span*{*proof*}

end

end

4.4 Vector spaces

4.4.1 vector-space-with

```

locale vector-space-with = ab-group-add plusVS zeroVS minusVS uminusVS
  for plusVS :: ['vs, 'vs] ⇒ 'vs (infixl ⟨+VS⟩ 65)
    and zeroVS (⟨0VS⟩)
    and minusVS (infixl ⟨-VS⟩ 65)
    and uminusVS (⟨-VS -> [81] 80) +
fixes scale :: ['f::field, 'vs] ⇒ 'vs (infixr ⟨*swith⟩ 75)
assumes scale-right-distrib[algebra-simps]:
  a *swith (x +VS y) = a *swith x +VS a *swith y
  and scale-left-distrib[algebra-simps]:
    (a + b) *swith x = a *swith x +VS b *swith x
  and scale-scale[simp]: a *swith (b *swith x) = (a * b) *swith x
  and scale-one[simp]: 1 *swith x = x
begin

```

```

notation plusVS (infixl ⟨+VS⟩ 65)
  and zeroVS (⟨0VS⟩)
  and minusVS (infixl ⟨-VS⟩ 65)
  and uminusVS (⟨-VS -> [81] 80)
  and scale (infixr ⟨*swith⟩ 75)

```

end

```

lemma vector-space-with-overloaded[ud-with]:
  vector-space = vector-space-with (+) 0 (-) uminus
  ⟨proof⟩

```

```

locale vector-space-pair-with =
  VS1: vector-space-with plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 +
  VS2: vector-space-with plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
for plusVS-1 :: ['vs-1, 'vs-1] ⇒ 'vs-1 (infixl ⟨+VS'-1⟩ 65)
  and zeroVS-1 (⟨0VS'-1⟩)
  and minusVS-1 (infixl ⟨-VS'-1⟩ 65)
  and uminusVS-1 (⟨-VS'-1 -> [81] 80)
  and scale1 :: ['f::field, 'vs-1] ⇒ 'vs-1 (infixr ⟨*swith'-1⟩ 75)
  and plusVS-2 :: ['vs-2, 'vs-2] ⇒ 'vs-2 (infixl ⟨+VS'-2⟩ 65)
  and zeroVS-2 (⟨0VS'-2⟩)
  and minusVS-2 (infixl ⟨-VS'-2⟩ 65)
  and uminusVS-2 (⟨-VS'-2 -> [81] 80)
  and scale2 :: ['f::field, 'vs-2] ⇒ 'vs-2 (infixr ⟨*swith'-2⟩ 75)

```

```

lemma vector-space-pair-with-overloaded[ud-with]:
  vector-space-pair =
  (
    λscale1 scale2.
    vector-space-pair-with (+) 0 (-) uminus scale1 (+) 0 (-) uminus scale2
  )
  ⟨proof⟩

```

```

locale linear-with =
  VS1: vector-space-with plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 +
  VS2: vector-space-with plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2 +
  module-hom-with
  plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1
  plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2

```

```

f
for plusVS-1 :: ['vs-1, 'vs-1] ⇒ 'vs-1 (infixl ⟨+VS'-1⟩ 65)
  and zeroVS-1 (⟨0VS'-1⟩)
  and minusVS-1 (infixl ⟨-VS'-1⟩ 65)
  and uminusVS-1 (⟨-VS'-1 -> [81] 80)
  and scale1 :: ['f::field, 'vs-1] ⇒ 'vs-1 (infixr ⟨*swith'-1⟩ 75)
  and plusVS-2 :: ['vs-2, 'vs-2] ⇒ 'vs-2 (infixl ⟨+VS'-2⟩ 65)
  and zeroVS-2 (⟨0VS'-2⟩)
  and minusVS-2 (infixl ⟨-VS'-2⟩ 65)
  and uminusVS-2 (⟨-VS'-2 -> [81] 80)
  and scale2 :: ['f::field, 'vs-2] ⇒ 'vs-2 (infixr ⟨*swith'-2⟩ 75)
  and f :: 'vs-1 ⇒ 'vs-2

```

lemma *linear-with-overloaded*[*ud-with*]:

```

Vector-Spaces.linear =
(
  λscale1 scale2.
    linear-with (+) 0 (-) uminus scale1 (+) 0 (-) uminus scale2
)
⟨proof⟩

```

locale *finite-dimensional-vector-space-with* =

```

vector-space-with plusVS zeroVS minusVS uminusVS scale
for plusVS :: ['vs, 'vs] ⇒ 'vs
  and zeroVS
  and minusVS
  and uminusVS
  and scale :: ['f::field, 'vs] ⇒ 'vs +
fixes basis :: 'vs set
assumes finite-basis: finite basis
  and independent-basis: independent-with 0 0VS (+VS) (*swith) basis
  and span-basis: span.with 0VS (+VS) (*swith) basis = UNIV

```

lemma *finite-dimensional-vector-space-with-overloaded*[*ud-with*]:

```

finite-dimensional-vector-space =
  finite-dimensional-vector-space-with (+) 0 (-) uminus
⟨proof⟩

```

locale *finite-dimensional-vector-space-pair-1-with* =

```

VS1: finite-dimensional-vector-space-with
  plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 basis1 +
VS2: vector-space-with
  plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
for plusVS-1 :: ['vs-1, 'vs-1] ⇒ 'vs-1 (infixl ⟨+VS'-1⟩ 65)
  and zeroVS-1 (⟨0VS'-1⟩)
  and minusVS-1 (infixl ⟨-VS'-1⟩ 65)
  and uminusVS-1 (⟨-VS'-1 -> [81] 80)
  and scale1 :: ['f::field, 'vs-1] ⇒ 'vs-1 (infixr ⟨*swith'-1⟩ 75)
  and basis1
  and plusVS-2 :: ['vs-2, 'vs-2] ⇒ 'vs-2 (infixl ⟨+VS'-2⟩ 65)
  and zeroVS-2 (⟨0VS'-2⟩)
  and minusVS-2 (infixl ⟨-VS'-2⟩ 65)
  and uminusVS-2 (⟨-VS'-2 -> [81] 80)
  and scale2 :: ['f::field, 'vs-2] ⇒ 'vs-2 (infixr ⟨*swith'-2⟩ 75)

```

lemma *finite-dimensional-vector-space-pair-1-with-overloaded*[*ud-with*]:

```

finite-dimensional-vector-space-pair-1 =
(

```

```

    λscale1 basis1 scale2.
    finite-dimensional-vector-space-pair-1-with
      (+) 0 (-) uminus scale1 basis1 (+) 0 (-) uminus scale2
  )
  ⟨proof⟩

```

```

locale finite-dimensional-vector-space-pair-with =
  VS1: finite-dimensional-vector-space-with
    plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 basis1 +
  VS2: finite-dimensional-vector-space-with
    plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2 basis2
for plusVS-1 :: ['vs-1, 'vs-1] ⇒ 'vs-1 (infixl ⟨+VS'-1⟩ 65)
and zeroVS-1 (⟨0VS'-1⟩)
and minusVS-1 (infixl ⟨-VS'-1⟩ 65)
and uminusVS-1 (⟨-VS'-1 -> [81] 80)
and scale1 :: ['f::field, 'vs-1] ⇒ 'vs-1 (infixr ⟨*swwith'-1⟩ 75)
and basis1
and plusVS-2 :: ['vs-2, 'vs-2] ⇒ 'vs-2 (infixl ⟨+VS'-2⟩ 65)
and zeroVS-2 (⟨0VS'-2⟩)
and minusVS-2 (infixl ⟨-VS'-2⟩ 65)
and uminusVS-2 (⟨-VS'-2 -> [81] 80)
and scale2 :: ['f::field, 'vs-2] ⇒ 'vs-2 (infixr ⟨*swwith'-2⟩ 75)
and basis2

```

```

lemma finite-dimensional-vector-space-pair-with-overloaded[ud-with]:
  finite-dimensional-vector-space-pair =
  (
    λscale1 basis1 scale2 basis2.
    finite-dimensional-vector-space-pair-with
      (+) 0 (-) uminus scale1 basis1 (+) 0 (-) uminus scale2 basis2
  )
  ⟨proof⟩

```

4.4.2 vector-space-ow

Definitions and common properties

Single vector space.

```

locale vector-space-ow = ab-group-add-ow UVS plusVS zeroVS minusVS uminusVS
for UVS :: 'vs set
and plusVS (infixl ⟨+VS⟩ 65)
and zeroVS (⟨0VS⟩)
and minusVS (infixl ⟨-VS⟩ 65)
and uminusVS (⟨-VS -> [81] 80) +
fixes scale :: ['f::field, 'vs] ⇒ 'vs (infixr ⟨*sow⟩ 75)
assumes scale-closed[simp, intro]: x ∈ UVS ⇒ a *sow x ∈ UVS
and scale-right-distrib[algebra-simps]:
  [[ x ∈ UVS; y ∈ UVS ]] ⇒ a *sow (x +VS y) = a *sow x +VS a *sow y
and scale-left-distrib[algebra-simps]:
  x ∈ UVS ⇒ (a + b) *sow x = a *sow x +VS b *sow x
and scale-scale[simp]:
  x ∈ UVS ⇒ a *sow (b *sow x) = (a * b) *sow x
and scale-one[simp]: x ∈ UVS ⇒ 1 *sow x = x
begin

```

```

lemma scale-closed'[simp]: ∀ a. ∀ x ∈ UVS. a *sow x ∈ UVS ⟨proof⟩

```

```

lemma minus-closed'[simp]: ∀ x ∈ UVS. ∀ y ∈ UVS. x -VS y ∈ UVS

```

<proof>

lemma *uminus-closed'*[simp]: $\forall x \in U_{VS}. -_{VS} x \in U_{VS}$ *<proof>*

tts-register-sbts *<(*sow)>* | U_{VS}
<proof>

sublocale *implicit* $_{VS}$: *module-ow* U_{VS} *plus* $_{VS}$ *zero* $_{VS}$ *minus* $_{VS}$ *uminus* $_{VS}$ *scale*
<proof>

end

ud *<vector-space.dim>*
ud *dim'* *<dim>*

Pair of vector spaces.

locale *vector-space-pair-ow* =
 VS_1 : *vector-space-ow* U_{VS-1} *plus* $_{VS-1}$ *zero* $_{VS-1}$ *minus* $_{VS-1}$ *uminus* $_{VS-1}$ *scale* $_1$ +
 VS_2 : *vector-space-ow* U_{VS-2} *plus* $_{VS-2}$ *zero* $_{VS-2}$ *minus* $_{VS-2}$ *uminus* $_{VS-2}$ *scale* $_2$
for U_{VS-1} :: *'vs-1 set*
and *plus* $_{VS-1}$ (**infixl** *<+_{VS'-1}>* 65)
and *zero* $_{VS-1}$ (*<0_{VS'-1}>*)
and *minus* $_{VS-1}$ (**infixl** *<-_{VS'-1}>* 65)
and *uminus* $_{VS-1}$ (*<-_{VS'-1} ->* [81] 80)
and *scale* $_1$:: [*f::field, 'vs-1*] \Rightarrow *'vs-1* (**infixr** *<*sow'-1>* 75)
and U_{VS-2} :: *'vs-2 set*
and *plus* $_{VS-2}$ (**infixl** *<+_{VS'-2}>* 65)
and *zero* $_{VS-2}$ (*<0_{VS'-2}>*)
and *minus* $_{VS-2}$ (**infixl** *<-_{VS'-2}>* 65)
and *uminus* $_{VS-2}$ (*<-_{VS'-2} ->* [81] 80)
and *scale* $_2$:: [*f::field, 'vs-2*] \Rightarrow *'vs-2* (**infixr** *<*sow'-2>* 75)
begin

sublocale *implicit* $_{VS}$: *module-pair-ow*
 U_{VS-1} *plus* $_{VS-1}$ *zero* $_{VS-1}$ *minus* $_{VS-1}$ *uminus* $_{VS-1}$ *scale* $_1$
 U_{VS-2} *plus* $_{VS-2}$ *zero* $_{VS-2}$ *minus* $_{VS-2}$ *uminus* $_{VS-2}$ *scale* $_2$
<proof>

end

Linear map.

locale *linear-ow* =
 VS_1 : *vector-space-ow* U_{VS-1} *plus* $_{VS-1}$ *zero* $_{VS-1}$ *minus* $_{VS-1}$ *uminus* $_{VS-1}$ *scale* $_1$ +
 VS_2 : *vector-space-ow* U_{VS-2} *plus* $_{VS-2}$ *zero* $_{VS-2}$ *minus* $_{VS-2}$ *uminus* $_{VS-2}$ *scale* $_2$ +
module-hom-ow
 U_{VS-1} *plus* $_{VS-1}$ *zero* $_{VS-1}$ *minus* $_{VS-1}$ *uminus* $_{VS-1}$ *scale* $_1$
 U_{VS-2} *plus* $_{VS-2}$ *zero* $_{VS-2}$ *minus* $_{VS-2}$ *uminus* $_{VS-2}$ *scale* $_2$
f
for U_{VS-1} :: *'vs-1 set*
and *plus* $_{VS-1}$ (**infixl** *<+_{VS'-1}>* 65)
and *zero* $_{VS-1}$ (*<0_{VS'-1}>*)
and *minus* $_{VS-1}$ (**infixl** *<-_{VS'-1}>* 65)
and *uminus* $_{VS-1}$ (*<-_{VS'-1} ->* [81] 80)
and *scale* $_1$:: [*f::field, 'vs-1*] \Rightarrow *'vs-1* (**infixr** *<*sow'-1>* 75)
and U_{VS-2} :: *'vs-2 set*
and *plus* $_{VS-2}$ (**infixl** *<+_{VS'-2}>* 65)
and *zero* $_{VS-2}$ (*<0_{VS'-2}>*)
and *minus* $_{VS-2}$ (**infixl** *<-_{VS'-2}>* 65)

```

    and uminusVS-2 (⟨-VS'-2 -⟩ [81] 80)
    and scale2 :: ['f::field, 'vs-2] ⇒ 'vs-2 (infixr ⟨*sow'-2⟩ 75)
    and f :: 'vs-1 ⇒ 'vs-2
begin
sublocale implicitVS: vector-space-pair-ow
  UVS-1 plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1
  UVS-2 plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
  ⟨proof⟩
end

```

Single finite dimensional vector space.

```

locale finite-dimensional-vector-space-ow =
  vector-space-ow UVS plusVS zeroVS minusVS uminusVS scale
  for UVS :: 'vs set
    and plusVS (infixl ⟨+VS⟩ 65)
    and zeroVS (⟨0VS⟩)
    and minusVS (infixl ⟨-VS⟩ 65)
    and uminusVS (⟨-VS -⟩ [81] 80)
    and scale :: ['f::field, 'vs] ⇒ 'vs (infixr ⟨*sow⟩ 75) +
  fixes basis :: 'vs set
  assumes basis-closed: basis ⊆ UVS
    and finite-basis: finite basis
    and independent-basis: independent-with 0 zeroVS plusVS scale basis
    and span-basis: span.with zeroVS plusVS scale basis = UVS

```

Pair of finite dimensional vector spaces.

```

locale finite-dimensional-vector-space-pair-1-ow =
  VS1: finite-dimensional-vector-space-ow
  UVS-1 plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 basis1 +
  VS2: vector-space-ow
  UVS-2 plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
  for UVS-1 :: 'vs-1 set
    and plusVS-1 (infixl ⟨+VS'-1⟩ 65)
    and zeroVS-1 (⟨0VS'-1⟩)
    and minusVS-1 (infixl ⟨-VS'-1⟩ 65)
    and uminusVS-1 (⟨-VS'-1 -⟩ [81] 80)
    and scale1 :: ['f::field, 'vs-1] ⇒ 'vs-1 (infixr ⟨*sow'-1⟩ 75)
    and basis1
    and UVS-2 :: 'vs-2 set
    and plusVS-2 (infixl ⟨+VS'-2⟩ 65)
    and zeroVS-2 (⟨0VS'-2⟩)
    and minusVS-2 (infixl ⟨-VS'-2⟩ 65)
    and uminusVS-2 (⟨-VS'-2 -⟩ [81] 80)
    and scale2 :: ['f::field, 'vs-2] ⇒ 'vs-2 (infixr ⟨*sow'-2⟩ 75)

```

```

locale finite-dimensional-vector-space-pair-ow =
  VS1: finite-dimensional-vector-space-ow
  UVS-1 plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 basis1 +
  VS2: finite-dimensional-vector-space-ow
  UVS-2 plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2 basis2
  for UVS-1 :: 'vs-1 set
    and plusVS-1 (infixl ⟨+VS'-1⟩ 65)
    and zeroVS-1 (⟨0VS'-1⟩)
    and minusVS-1 (infixl ⟨-VS'-1⟩ 65)
    and uminusVS-1 (⟨-VS'-1 -⟩ [81] 80)
    and scale1 :: ['f::field, 'vs-1] ⇒ 'vs-1 (infixr ⟨*sow'-1⟩ 75)

```

```

and basis1
and UVS-2 :: 'vs-2 set
and plusVS-2 (infixl ⟨+VS'-2⟩ 65)
and zeroVS-2 (⟨0VS'-2⟩)
and minusVS-2 (infixl ⟨-VS'-2⟩ 65)
and uminusVS-2 (⟨-VS'-2 -⟩ [81] 80)
and scale2 :: ['f::field, 'vs-2] ⇒ 'vs-2 (infixr ⟨*sow'-2⟩ 75)
and basis2

```

Transfer.

```

lemma vector-space-with-transfer[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: bi-unique B right-total B
  fixes PP lhs
  defines
    PP ≡
    (
      (B ==> B ==> B) ==>
      B ==>
      (B ==> B ==> B) ==>
      (B ==> B) ==>
      ((=) ==> B ==> B) ==>
      (=)
    )
  and
    lhs ≡
    (
      λplusVS zeroVS minusVS uminusVS scale.
      vector-space-ow
      (Collect (Domainp B)) plusVS zeroVS minusVS uminusVS scale
    )
  shows PP lhs vector-space-with
  ⟨proof⟩

```

```

lemma vector-space-pair-with-transfer[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]:
    bi-unique B1 right-total B1 bi-unique B2 right-total B2
  fixes PP lhs
  defines
    PP ≡
    (
      (B1 ==> B1 ==> B1) ==>
      B1 ==>
      (B1 ==> B1 ==> B1) ==>
      (B1 ==> B1) ==>
      ((=) ==> B1 ==> B1) ==>
      (B2 ==> B2 ==> B2) ==>
      B2 ==>
      (B2 ==> B2 ==> B2) ==>
      (B2 ==> B2) ==>
      ((=) ==> B2 ==> B2) ==>
      (=)
    )
  and
    lhs ≡
    (

```

```

    λ
      plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1
      plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2.
    vector-space-pair-ow
      (Collect (Domainp B1)) plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1
      (Collect (Domainp B2)) plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
  )
  shows PP lhs vector-space-pair-with
  ⟨proof⟩

```

lemma *linear-with-transfer*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]:

bi-unique B₁ right-total B₁ bi-unique B₂ right-total B₂

fixes *PP lhs*

defines

PP ≡

```

  (
    (B1 ==> B1 ==> B1) ==>
    B1 ==>
    (B1 ==> B1 ==> B1) ==>
    (B1 ==> B1) ==>
    ((=) ==> B1 ==> B1) ==>
    (B2 ==> B2 ==> B2) ==>
    B2 ==>
    (B2 ==> B2 ==> B2) ==>
    (B2 ==> B2) ==>
    ((=) ==> B2 ==> B2) ==>
    (B1 ==> B2) ==>
    (=)
  )

```

and

lhs ≡

```

  (
    λ
      plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1
      plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
      f.
    linear-ow
      (Collect (Domainp B1)) plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1
      (Collect (Domainp B2)) plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
      f
  )

```

shows *PP lhs linear-with*
 ⟨proof⟩

lemma *linear-with-transfer'*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique B right-total B*

fixes *PP lhs*

defines

PP ≡

```

  (
    (B ==> B ==> B) ==>
    B ==>
    (B ==> B ==> B) ==>
    (B ==> B) ==>
  )

```

```

  ((=) ==> B ==> B) ==>
  (B ==> B ==> B) ==>
  B ==>
  (B ==> B ==> B) ==>
  (B ==> B) ==>
  ((=) ==> B ==> B) ==>
  (B ==> B) ==>
  (=)
)
and
  lhs ≡
  (
    λ
      plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1
      plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
      f.
    linear-ow
      (Collect (Domainp B)) plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1
      (Collect (Domainp B)) plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
      f
  )

```

shows *PP lhs linear-with*

<proof>

lemma *finite-dimensional-vector-space-with-transfer[transfer-rule]:*

includes *lifting-syntax*

assumes *[transfer-rule]: bi-unique B right-total B*

fixes *PP lhs*

defines

```

  PP ≡
  (
    (B ==> B ==> B) ==>
    B ==>
    (B ==> B ==> B) ==>
    (B ==> B) ==>
    ((=) ==> B ==> B) ==>
    rel-set B ==>
    (=)
  )

```

and

```

  lhs ≡
  (
    λplusVS zeroVS minusVS uminusVS scale basis.
    finite-dimensional-vector-space-ow
      (Collect (Domainp B)) plusVS zeroVS minusVS uminusVS scale basis
  )

```

shows *PP lhs finite-dimensional-vector-space-with*

<proof>

lemma *finite-dimensional-vector-space-pair-1-with-transfer[transfer-rule]:*

includes *lifting-syntax*

assumes *[transfer-rule]:*

bi-unique B₁ right-total B₁ bi-unique B₂ right-total B₂

fixes *PP lhs*

defines

```

  PP ≡
  (

```

```

    (B1 ==> B1 ==> B1) ==>
    B1 ==>
    (B1 ==> B1 ==> B1) ==>
    (B1 ==> B1) ==>
    ((=) ==> B1 ==> B1) ==>
    rel-set B1 ==>
    (B2 ==> B2 ==> B2) ==>
    B2 ==>
    (B2 ==> B2 ==> B2) ==>
    (B2 ==> B2) ==>
    ((=) ==> B2 ==> B2) ==>
    (=)
  )
and
  lhs ≡
  (
    λ
      plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 basis1
      plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2.
    finite-dimensional-vector-space-pair-1-ow
    (Collect (Domainp B1))
      plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 basis1
    (Collect (Domainp B2))
      plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2
  )
shows PP lhs finite-dimensional-vector-space-pair-1-with
⟨proof⟩

```

lemma *finite-dimensional-vector-space-pair-with-transfer*[transfer-rule]:

```

includes lifting-syntax
assumes [transfer-rule]:
  bi-unique B1 right-total B1 bi-unique B2 right-total B2
fixes PP lhs
defines
  PP ≡
  (
    (B1 ==> B1 ==> B1) ==>
    B1 ==>
    (B1 ==> B1 ==> B1) ==>
    (B1 ==> B1) ==>
    ((=) ==> B1 ==> B1) ==>
    rel-set B1 ==>
    (B2 ==> B2 ==> B2) ==>
    B2 ==>
    (B2 ==> B2 ==> B2) ==>
    (B2 ==> B2) ==>
    ((=) ==> B2 ==> B2) ==>
    rel-set B2 ==>
    (=)
  )
and
  lhs ≡
  (
    λ
      plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 basis1
      plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2 basis2.
    finite-dimensional-vector-space-pair-ow
    (Collect (Domainp B1))

```

```

      plusVS-1 zeroVS-1 minusVS-1 uminusVS-1 scale1 basis1
    (Collect (Domainp B2))
      plusVS-2 zeroVS-2 minusVS-2 uminusVS-2 scale2 basis2
  )
  shows PP lhs finite-dimensional-vector-space-pair-with
  ⟨proof⟩

```

4.4.3 vector-space-on

```

locale vector-space-on = module-on UVS scale
  for UVS and scale :: 'a::field ⇒ 'b::ab-group-add ⇒ 'b (infixr ⟨*s⟩ 75)
begin

```

```

notation scale (infixr ⟨*s⟩ 75)

```

```

sublocale implicitVS: vector-space-ow UVS ⟨(+⟩ 0 ⟨(-⟩ uminus scale
  ⟨proof⟩

```

```

lemmas ab-group-add-ow-axioms = implicitM.ab-group-add-ow-axioms

```

```

lemmas vector-space-ow-axioms = implicitVS.vector-space-ow-axioms

```

```

definition dim :: 'b set ⇒ nat

```

```

  where dim V = (if ∃ b ⊆ UVS. ¬ dependent b ∧ span b = span V
    then card (SOME b. b ⊆ UVS ∧ ¬ dependent b ∧ span b = span V)
    else 0)

```

```

end

```

```

lemma vector-space-on-alt-def: vector-space-on UVS = module-on UVS
  ⟨proof⟩

```

```

lemma implicit-vector-space-ow[tts-implicit]:

```

```

  vector-space-ow UVS (+) 0 (-) uminus = vector-space-on UVS
  ⟨proof⟩

```

```

locale linear-on =

```

```

  VS1: vector-space-on UM-1 scale1 +
  VS2: vector-space-on UM-2 scale2 +
  module-hom-on UM-1 UM-2 scale1 scale2 f
  for UM-1 UM-2 and scale1::'a::field ⇒ 'b ⇒ 'b::ab-group-add
  and scale2::'a::field ⇒ 'c ⇒ 'c::ab-group-add
  and f

```

```

lemma implicit-linear-on[tts-implicit]:

```

```

  linear-ow UM-1 (+) 0 minus uminus scale1 UM-2 (+) 0 (-) uminus scale2 =
  linear-on UM-1 UM-2 scale1 scale2
  ⟨proof⟩

```

```

locale finite-dimensional-vector-space-on =

```

```

  vector-space-on UVS scale
  for UVS :: 'b::ab-group-add set
  and scale :: 'a::field ⇒ 'b ⇒ 'b +
  fixes basis :: 'b set
  assumes finite-basis: finite basis
  and independent-basis: ¬ dependent basis
  and span-basis: span basis = UVS
  and basis-subset: basis ⊆ UVS

```

```

begin

```

sublocale *implicit*_{VS}:

finite-dimensional-vector-space-ow U_{VS} $\langle (+) \rangle$ 0 $\langle (-) \rangle$ *uminus scale basis*
 $\langle \text{proof} \rangle$

end

lemma *implicit-finite-dimensional-vector-space-on*[*tts-implicit*]:

finite-dimensional-vector-space-ow U_{VS} (+) 0 *minus uminus scale basis* =
finite-dimensional-vector-space-on U_{VS} *scale basis*
 $\langle \text{proof} \rangle$

locale *vector-space-pair-on* =

VS_1 : *vector-space-on* U_{M-1} *scale*₁ +

VS_2 : *vector-space-on* U_{M-2} *scale*₂

for U_{M-1} :: 'b::ab-group-add set **and** U_{M-2} :: 'c::ab-group-add set
and $scale_1$:: 'a::field \Rightarrow - \Rightarrow - (**infixr** $\langle *s_1 \rangle$ 75)
and $scale_2$:: 'a \Rightarrow - \Rightarrow - (**infixr** $\langle *s_2 \rangle$ 75)

begin

notation $scale_1$ (**infixr** $\langle *s_1 \rangle$ 75)

notation $scale_2$ (**infixr** $\langle *s_2 \rangle$ 75)

sublocale *module-pair-on* U_{M-1} U_{M-2} $scale_1$ $scale_2$ $\langle \text{proof} \rangle$

sublocale *implicit*_{VS}:

vector-space-pair-ow

U_{M-1} $\langle (+) \rangle$ 0 $\langle (-) \rangle$ *uminus scale*₁

U_{M-2} $\langle (+) \rangle$ 0 $\langle (-) \rangle$ *uminus scale*₂

$\langle \text{proof} \rangle$

end

lemma *implicit-vector-space-pair-on*[*tts-implicit*]:

vector-space-pair-ow

U_{M-1} (+) 0 (-) *uminus scale*₁

U_{M-2} (+) 0 (-) *uminus scale*₂ =

vector-space-pair-on U_{M-1} U_{M-2} $scale_1$ $scale_2$

$\langle \text{proof} \rangle$

locale *finite-dimensional-vector-space-pair-1-on* =

VS_1 : *finite-dimensional-vector-space-on* U_{M-1} $scale_1$ *basis*₁ +

VS_2 : *vector-space-on* U_{M-2} $scale_2$

for U_{M-1} U_{M-2}

and $scale_1$:: 'a::field \Rightarrow 'b::ab-group-add \Rightarrow 'b

and $scale_2$:: 'a::field \Rightarrow 'c::ab-group-add \Rightarrow 'c

and *basis*₁

begin

sublocale *vector-space-pair-on* U_{M-1} U_{M-2} $scale_1$ $scale_2$ $\langle \text{proof} \rangle$

sublocale *implicit*_{VS}:

finite-dimensional-vector-space-pair-1-ow

U_{M-1} $\langle (+) \rangle$ 0 $\langle (-) \rangle$ *uminus scale*₁ *basis*₁ U_{M-2} $\langle (+) \rangle$ 0 $\langle (-) \rangle$ *uminus scale*₂

$\langle \text{proof} \rangle$

end

lemma *implicit-finite-dimensional-vector-space-pair-1-on*[*tts-implicit*]:
finite-dimensional-vector-space-pair-1-ow
 $U_{M-1} (+) 0 \text{ minus } \text{uminus } \text{scale}_1 \text{ basis1 } U_{M-2} (+) 0 (-) \text{ uminus } \text{scale}_2 =$
finite-dimensional-vector-space-pair-1-on $U_{M-1} U_{M-2} \text{ scale}_1 \text{ scale}_2 \text{ basis1}$
 ⟨*proof*⟩

locale *finite-dimensional-vector-space-pair-on* =
 VS_1 : *finite-dimensional-vector-space-on* $U_{M-1} \text{ scale}_1 \text{ basis1} +$
 VS_2 : *finite-dimensional-vector-space-on* $U_{M-2} \text{ scale}_2 \text{ basis2}$
for $U_{M-1} U_{M-2}$
and $\text{scale}_1 :: 'a :: \text{field} \Rightarrow 'b :: \text{ab-group-add} \Rightarrow 'b$
and $\text{scale}_2 :: 'a :: \text{field} \Rightarrow 'c :: \text{ab-group-add} \Rightarrow 'c$
and $\text{basis1} \text{ basis2}$
begin

sublocale *finite-dimensional-vector-space-pair-1-on* $U_{M-1} U_{M-2} \text{ scale}_1 \text{ scale}_2$
 ⟨*proof*⟩

sublocale *implicit_{VS}*:
finite-dimensional-vector-space-pair-ow
 $U_{M-1} \langle (+) \rangle 0 \langle (-) \rangle \text{ uminus } \text{scale}_1 \text{ basis1}$
 $U_{M-2} \langle (+) \rangle 0 \langle (-) \rangle \text{ uminus } \text{scale}_2 \text{ basis2}$
 ⟨*proof*⟩

end

lemma *implicit-finite-dimensional-vector-space-pair-on*[*tts-implicit*]:
finite-dimensional-vector-space-pair-ow
 $U_{M-1} (+) 0 \text{ minus } \text{uminus } \text{scale}_1 \text{ basis1}$
 $U_{M-2} (+) 0 (-) \text{ uminus } \text{scale}_2 \text{ basis2} =$
finite-dimensional-vector-space-pair-on
 $U_{M-1} U_{M-2} \text{ scale}_1 \text{ scale}_2 \text{ basis1} \text{ basis2}$
 ⟨*proof*⟩

4.4.4 Relativization : part I

context *vector-space-on*
begin

tts-context
tts: (*?b to* $\langle U_{VS} :: 'b \text{ set} \rangle$)
rewriting *ctr-simps*
substituting *ab-group-add-ow-axioms*
and *vector-space-ow-axioms*
and *implicit_M.module-ow-axioms*
applying
 [*OF*
implicit_M.carrier-ne
implicit_M.add-closed'
implicit_M.zero-closed
implicit_{VS}.minus-closed'
implicit_{VS}.uminus-closed'
implicit_{VS}.scale-closed',
unfolded tts-implicit
]
begin

tts-lemma *linear-id*: *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ *id*
is *vector-space.linear-id*{*proof*}

tts-lemma *linear-ident*: *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ $(\lambda x. x)$
is *vector-space.linear-ident*{*proof*}

tts-lemma *linear-scale-self*: *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ $((*s) c)$
is *vector-space.linear-scale-self*{*proof*}

tts-lemma *linear-scale-left*:
assumes $x \in U_{VS}$
shows *linear-on UNIV* U_{VS} $(*)$ $(*s)$ $(\lambda r. r *s x)$
is *vector-space.linear-scale-left*{*proof*}

tts-lemma *linear-uminus*: *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ *uminus*
is *vector-space.linear-uminus*{*proof*}

tts-lemma *linear-imp-scale*[*consumes* - 1, *case-names* 1]:
assumes *range* $D \subseteq U_{VS}$
and *linear-on UNIV* U_{VS} $(*)$ $(*s)$ D
and $\bigwedge d. \llbracket d \in U_{VS}; D = (\lambda x. x *s d) \rrbracket \implies$ *thesis*
shows *thesis*
is *vector-space.linear-imp-scale*{*proof*}

tts-lemma *scale-eq-0-iff*:
assumes $x \in U_{VS}$
shows $(a *s x = 0) = (a = 0 \vee x = 0)$
is *vector-space.scale-eq-0-iff*{*proof*}

tts-lemma *scale-left-imp-eq*:
assumes $x \in U_{VS}$ **and** $y \in U_{VS}$ **and** $a \neq 0$ **and** $a *s x = a *s y$
shows $x = y$
is *vector-space.scale-left-imp-eq*{*proof*}

tts-lemma *scale-right-imp-eq*:
assumes $x \in U_{VS}$ **and** $x \neq 0$ **and** $a *s x = b *s x$
shows $a = b$
is *vector-space.scale-right-imp-eq*{*proof*}

tts-lemma *scale-cancel-left*:
assumes $x \in U_{VS}$ **and** $y \in U_{VS}$
shows $(a *s x = a *s y) = (x = y \vee a = 0)$
is *vector-space.scale-cancel-left*{*proof*}

tts-lemma *scale-cancel-right*:
assumes $x \in U_{VS}$
shows $(a *s x = b *s x) = (a = b \vee x = 0)$
is *vector-space.scale-cancel-right*{*proof*}

tts-lemma *injective-scale*:
assumes $c \neq 0$
shows *inj-on* $((*s) c)$ U_{VS}
is *vector-space.injective-scale*{*proof*}

tts-lemma *dependent-def*:
assumes $P \subseteq U_{VS}$
shows *dependent* $P = (\exists x \in P. x \in \text{span}(P - \{x\}))$
is *vector-space.dependent-def*{*proof*}

tts-lemma *dependent-single*:

assumes $x \in U_{VS}$
shows *dependent* $\{x\} = (x = 0)$
is *vector-space.dependent-single*{proof}

tts-lemma *in-span-insert*:

assumes $a \in U_{VS}$
and $b \in U_{VS}$
and $S \subseteq U_{VS}$
and $a \in \text{span } (\text{insert } b \ S)$
and $a \notin \text{span } S$
shows $b \in \text{span } (\text{insert } a \ S)$
is *vector-space.in-span-insert*{proof}

tts-lemma *dependent-insertD*:

assumes $a \in U_{VS}$ **and** $S \subseteq U_{VS}$ **and** $a \notin \text{span } S$ **and** *dependent* $(\text{insert } a \ S)$
shows *dependent* S
is *vector-space.dependent-insertD*{proof}

tts-lemma *independent-insertI*:

assumes $a \in U_{VS}$ **and** $S \subseteq U_{VS}$ **and** $a \notin \text{span } S$ **and** \neg *dependent* S
shows \neg *dependent* $(\text{insert } a \ S)$
is *vector-space.independent-insertI*{proof}

tts-lemma *independent-insert*:

assumes $a \in U_{VS}$ **and** $S \subseteq U_{VS}$
shows $(\neg$ *dependent* $(\text{insert } a \ S)) =$
(if $a \in S$ *then* \neg *dependent* S *else* \neg *dependent* $S \wedge a \notin \text{span } S)$
is *vector-space.independent-insert*{proof}

tts-lemma *maximal-independent-subset-extend*[*consumes* - 1, *case-names* 1]:

assumes $S \subseteq U_{VS}$
and $V \subseteq U_{VS}$
and $S \subseteq V$
and \neg *dependent* S
and $\bigwedge B. [[B \subseteq U_{VS}; S \subseteq B; B \subseteq V; \neg$ *dependent* $B; V \subseteq \text{span } B]] \implies$ *thesis*
shows *thesis*
is *vector-space.maximal-independent-subset-extend*{proof}

tts-lemma *maximal-independent-subset*[*consumes* - 1, *case-names* 1]:

assumes $V \subseteq U_{VS}$
and $\bigwedge B. [[B \subseteq U_{VS}; B \subseteq V; \neg$ *dependent* $B; V \subseteq \text{span } B]] \implies$ *thesis*
shows *thesis*
is *vector-space.maximal-independent-subset*{proof}

tts-lemma *in-span-delete*:

assumes $a \in U_{VS}$
and $S \subseteq U_{VS}$
and $b \in U_{VS}$
and $a \in \text{span } S$
and $a \notin \text{span } (S - \{b\})$
shows $b \in \text{span } (\text{insert } a \ (S - \{b\}))$
is *vector-space.in-span-delete*{proof}

tts-lemma *span-redundant*:

assumes $x \in U_{VS}$ **and** $S \subseteq U_{VS}$ **and** $x \in \text{span } S$
shows $\text{span } (\text{insert } x \ S) = \text{span } S$

is *vector-space.span-redundant*(proof)

tts-lemma *span-trans*:

assumes $x \in U_{VS}$
and $S \subseteq U_{VS}$
and $y \in U_{VS}$
and $x \in \text{span } S$
and $y \in \text{span } (\text{insert } x \ S)$
shows $y \in \text{span } S$
is *vector-space.span-trans*(proof)

tts-lemma *span-insert-0*:

assumes $S \subseteq U_{VS}$
shows $\text{span } (\text{insert } 0 \ S) = \text{span } S$
is *vector-space.span-insert-0*(proof)

tts-lemma *span-delete-0*:

assumes $S \subseteq U_{VS}$
shows $\text{span } (S - \{0\}) = \text{span } S$
is *vector-space.span-delete-0*(proof)

tts-lemma *span-image-scale*:

assumes $S \subseteq U_{VS}$ **and** *finite* S **and** $\bigwedge x. [[x \in U_{VS}; x \in S]] \implies c \ x \neq 0$
shows $\text{span } ((\lambda x. c \ x \ *s \ x) \ ` \ S) = \text{span } S$
is *vector-space.span-image-scale*(proof)

tts-lemma *exchange-lemma*:

assumes $T \subseteq U_{VS}$
and $S \subseteq U_{VS}$
and *finite* T
and \neg *dependent* S
and $S \subseteq \text{span } T$
shows $\exists t' \in \text{Pow } U_{VS}.$
 $\text{card } t' = \text{card } T \wedge \text{finite } t' \wedge S \subseteq t' \wedge t' \subseteq S \cup T \wedge S \subseteq \text{span } t'$
is *vector-space.exchange-lemma*(proof)

tts-lemma *independent-span-bound*:

assumes $T \subseteq U_{VS}$
and $S \subseteq U_{VS}$
and *finite* T
and \neg *dependent* S
and $S \subseteq \text{span } T$
shows $\text{finite } S \wedge \text{card } S \leq \text{card } T$
is *vector-space.independent-span-bound*(proof)

tts-lemma *independent-explicit-finite-subsets*:

assumes $A \subseteq U_{VS}$
shows $(\neg$ *dependent* $A) =$
 (
 $\forall x \subseteq U_{VS}.$
 $x \subseteq A \longrightarrow$
 $\text{finite } x \longrightarrow$
 $(\forall f. (\sum v \in x. f \ v \ *s \ v) = 0 \longrightarrow (\forall x \in x. f \ x = 0))$
)
given *vector-space.independent-explicit-finite-subsets* (proof)

tts-lemma *independent-if-scalars-zero*:

assumes $A \subseteq U_{VS}$

and *finite* A
and $\bigwedge x. [\![x \in U_{VS}; (\sum_{x \in A}. f x * s x) = 0; x \in A]\!] \implies f x = 0$
shows \neg *dependent* A
is *vector-space.independent-if-scalars-zero* {*proof*}

tts-lemma *subspace-sums*:

assumes $S \subseteq U_{VS}$ **and** $T \subseteq U_{VS}$ **and** *subspace* S **and** *subspace* T
shows *subspace* $\{x \in U_{VS}. \exists a \in U_{VS}. \exists b \in U_{VS}. x = a + b \wedge a \in S \wedge b \in T\}$
is *vector-space.subspace-sums* {*proof*}

tts-lemma *bij-if-span-eq-span-bases*:

assumes $B \subseteq U_{VS}$
and $C \subseteq U_{VS}$
and \neg *dependent* B
and \neg *dependent* C
and *span* $B = \text{span } C$
shows $\exists x. \text{bij-betw } x B C \wedge (\forall a \in U_{VS}. x a \in U_{VS})$
given *vector-space.bij-if-span-eq-span-bases* {*proof*}

end

end

4.4.5 Transfer: *dim*

context *vector-space-on*

begin

lemma *dim-eq-card*:

assumes $B \subseteq U_{VS}$
and $V \subseteq U_{VS}$
and BV : *span* $B = \text{span } V$
and B : \neg *dependent* B
shows *dim* $V = \text{card } B$

{*proof*}

lemma *dim-transfer*[*transfer-rule*]:

includes *lifting-syntax*
assumes [*transfer-domain-rule*]: *Domain* $p A = (\lambda x. x \in U_{VS})$
and [*transfer-rule*]: *right-total* A *bi-unique* A
and [*transfer-rule*]: $(A \implies A \implies A)$ *plus* *plus'*
and [*transfer-rule*]: $(=) \implies A \implies A)$ *scale* *scale'*
and [*transfer-rule*]: A *0* *zero'*

shows (*rel-set* $A \implies (=)$) *dim* (*dim.with plus' zero' 0 scale'*)

{*proof*}

end

4.4.6 Relativization: part II

context *vector-space-on*

begin

tts-context

tts: (*?'b* **to** $\langle U_{VS}::'b \text{ set} \rangle$)

sbterms: ($\langle (+)::'b::\text{ab-group-add} \implies ?'b \implies ?'b \rangle$ **to** $\langle (+)::'b \implies 'b \implies 'b \rangle$)

and

(

```

    ⟨?scale::?'a::field ⇒?'b::ab-group-add⇒?'b::ab-group-add⟩ to
    ⟨(*s)::'a⇒'b⇒'b⟩
  )
  and ⟨0::?'b::ab-group-add⟩ to ⟨0::'b⟩
  rewriting ctr-simps
  substituting ab-group-add-ow-axioms
  and vector-space-ow-axioms
  and implicitM.module-ow-axioms
  eliminating ⟨?a ∈ ?A⟩ and ⟨?B ⊆ ?C⟩ through auto
  applying
  [
    OF
    implicitM.carrier-ne
    implicitV.S.minus-closed'
    implicitV.S.uminus-closed',
    unfolded tts-implicit
  ]
begin

tts-lemma dim-unique:
  assumes V ⊆ UV.S
  and B ⊆ V
  and V ⊆ span B
  and ¬ dependent B
  and card B = n
  shows dim V = n
  is vector-space.dim-unique{proof}

tts-lemma basis-card-eq-dim:
  assumes V ⊆ UV.S and B ⊆ V and V ⊆ span B and ¬ dependent B
  shows card B = dim V
  is vector-space.basis-card-eq-dim{proof}

tts-lemma dim-eq-card-independent:
  assumes B ⊆ UV.S and ¬ dependent B
  shows dim B = card B
  is vector-space.dim-eq-card-independent{proof}

tts-lemma dim-span:
  assumes S ⊆ UV.S
  shows dim (span S) = dim S
  is vector-space.dim-span{proof}

tts-lemma dim-span-eq-card-independent:
  assumes B ⊆ UV.S and ¬ dependent B
  shows dim (span B) = card B
  is vector-space.dim-span-eq-card-independent{proof}

tts-lemma dim-le-card:
  assumes V ⊆ UV.S and W ⊆ UV.S and V ⊆ span W and finite W
  shows dim V ≤ card W
  is vector-space.dim-le-card{proof}

tts-lemma span-eq-dim:
  assumes S ⊆ UV.S and T ⊆ UV.S and span S = span T
  shows dim S = dim T
  is vector-space.span-eq-dim{proof}

```

```

tts-lemma dim-le-card':
  assumes  $s \subseteq U_{VS}$  and finite s
  shows  $\dim s \leq \text{card } s$ 
  is vector-space.dim-le-card'{proof}

tts-lemma span-card-ge-dim:
  assumes  $V \subseteq U_{VS}$  and  $B \subseteq V$  and  $V \subseteq \text{span } B$  and finite B
  shows  $\dim V \leq \text{card } B$ 
  is vector-space.span-card-ge-dim{proof}

end

tts-context
  tts: (?b to  $\langle U_{VS}::'b \text{ set} \rangle$ )
  sbterms: ( $\langle (+)::?'b::\text{ab-group-add} \Rightarrow ?'b \Rightarrow ?'b \rangle$  to  $\langle (+)::'b \Rightarrow 'b \Rightarrow 'b \rangle$ )
    and ( $\langle ?scale::?'a::\text{field} \Rightarrow ?'b::\text{ab-group-add} \Rightarrow ?'b \rangle$  to  $\langle (*s)::'a \Rightarrow 'b \Rightarrow 'b \rangle$ )
    and ( $\langle 0::?'b::\text{ab-group-add} \rangle$  to  $\langle 0::'b \rangle$ )
  rewriting ctr-simps
  substituting ab-group-add-ow-axioms
    and vector-space-ow-axioms
    and implicitM.module-ow-axioms
  applying
  [
    OF
    implicitM.carrier-ne
    implicitVS.minus-closed'
    implicitVS.uminus-closed',
    unfolded tts-implicit
  ]
begin

tts-lemma basis-exists:
  assumes  $V \subseteq U_{VS}$ 
  and  $\wedge B.$ 
  [[
     $B \subseteq U_{VS};$ 
     $B \subseteq V;$ 
     $\neg \text{dependent } B;$ 
     $V \subseteq \text{span } B;$ 
     $\text{card } B = \dim V$ 
  ]]  $\implies$  thesis
  shows thesis
  is vector-space.basis-exists{proof}

end

end

context finite-dimensional-vector-space-on
begin

tts-context
  tts: (?b to  $\langle U_{VS}::'b \text{ set} \rangle$ )
  sbterms: ( $\langle (+)::?'b::\text{ab-group-add} \Rightarrow ?'b \Rightarrow ?'b \rangle$  to  $\langle (+)::'b \Rightarrow 'b \Rightarrow 'b \rangle$ )
    and ( $\langle ?scale::?'a::\text{field} \Rightarrow ?'b::\text{ab-group-add} \Rightarrow ?'b \rangle$  to  $\langle (*s)::'a \Rightarrow 'b \Rightarrow 'b \rangle$ )
    and ( $\langle 0::?'b::\text{ab-group-add} \rangle$  to  $\langle 0::'b \rangle$ )
  rewriting ctr-simps
  substituting ab-group-add-ow-axioms

```

```

    and vector-space-ow-axioms
    and implicitVS.finite-dimensional-vector-space-ow-axioms
    and implicitM.module-ow-axioms
eliminating ⟨?a ∈ ?A⟩ and ⟨?B ⊆ ?C⟩ through auto
applying
[
  OF
  implicitM.carrier-ne
  implicitVS.minus-closed'
  implicitVS.uminus-closed'
  basis-subset,
  unfolded tts-implicit
]
begin

tts-lemma finiteI-independent:
  assumes B ⊆ UVS and ¬ dependent B
  shows finite B
  is finite-dimensional-vector-space.finiteI-independent{proof}

tts-lemma dim-empty: dim {} = 0
  is finite-dimensional-vector-space.dim-empty{proof}

tts-lemma dim-insert:
  assumes x ∈ UVS and S ⊆ UVS
  shows dim (insert x S) = (if x ∈ span S then dim S else dim S + 1)
  is finite-dimensional-vector-space.dim-insert{proof}

tts-lemma dim-singleton:
  assumes x ∈ UVS
  shows dim {x} = (if x = 0 then 0 else 1)
  is finite-dimensional-vector-space.dim-singleton{proof}

tts-lemma choose-subspace-of-subspace[consumes - 1, case-names 1]:
  assumes S ⊆ UVS
  and n ≤ dim S
  and ∧T. [[T ⊆ UVS; subspace T; T ⊆ span S; dim T = n]] ⇒ thesis
  shows thesis
  is finite-dimensional-vector-space.choose-subspace-of-subspace{proof}

tts-lemma basis-subspace-exists[consumes - 1, case-names 1]:
  assumes S ⊆ UVS
  and subspace S
  and ∧B.
  [[
    B ⊆ UVS;
    finite B;
    B ⊆ S;
    ¬ dependent B;
    span B = S;
    card B = dim S
  ]] ⇒ thesis
  shows thesis
  is finite-dimensional-vector-space.basis-subspace-exists{proof}

tts-lemma dim-mono:
  assumes V ⊆ UVS and W ⊆ UVS and V ⊆ span W
  shows dim V ≤ dim W

```

is *finite-dimensional-vector-space.dim-mono*{proof}

tts-lemma *dim-subset*:
 assumes $T \subseteq U_{VS}$ and $S \subseteq T$
 shows $\dim S \leq \dim T$
 is *finite-dimensional-vector-space.dim-subset*{proof}

tts-lemma *dim-eq-0*:
 assumes $S \subseteq U_{VS}$
 shows $(\dim S = 0) = (S \subseteq \{0\})$
 is *finite-dimensional-vector-space.dim-eq-0*{proof}

tts-lemma *dim-UNIV*: $\dim U_{VS} = \text{card basis}$
 is *finite-dimensional-vector-space.dim-UNIV*{proof}

tts-lemma *independent-card-le-dim*:
 assumes $V \subseteq U_{VS}$ and $B \subseteq V$ and $\neg \text{dependent } B$
 shows $\text{card } B \leq \dim V$
 is *finite-dimensional-vector-space.independent-card-le-dim*{proof}

tts-lemma *card-ge-dim-independent*:
 assumes $V \subseteq U_{VS}$ and $B \subseteq V$ and $\neg \text{dependent } B$ and $\dim V \leq \text{card } B$
 shows $V \subseteq \text{span } B$
 is *finite-dimensional-vector-space.card-ge-dim-independent*{proof}

tts-lemma *card-le-dim-spanning*:
 assumes $V \subseteq U_{VS}$
 and $B \subseteq V$
 and $V \subseteq \text{span } B$
 and *finite* B
 and $\text{card } B \leq \dim V$
 shows $\neg \text{dependent } B$
 is *finite-dimensional-vector-space.card-le-dim-spanning*{proof}

tts-lemma *card-eq-dim*:
 assumes $V \subseteq U_{VS}$ and $B \subseteq V$ and $\text{card } B = \dim V$ and *finite* B
 shows $(\neg \text{dependent } B) = (V \subseteq \text{span } B)$
 is *finite-dimensional-vector-space.card-eq-dim*{proof}

tts-lemma *subspace-dim-equal*:
 assumes $T \subseteq U_{VS}$
 and *subspace* S
 and *subspace* T
 and $S \subseteq T$
 and $\dim T \leq \dim S$
 shows $S = T$
 is *finite-dimensional-vector-space.subspace-dim-equal*{proof}

tts-lemma *dim-eq-span*:
 assumes $T \subseteq U_{VS}$ and $S \subseteq T$ and $\dim T \leq \dim S$
 shows $\text{span } S = \text{span } T$
 is *finite-dimensional-vector-space.dim-eq-span*{proof}

tts-lemma *dim-psubset*:
 assumes $S \subseteq U_{VS}$ and $T \subseteq U_{VS}$ and $\text{span } S \subset \text{span } T$
 shows $\dim S < \dim T$
 is *finite-dimensional-vector-space.dim-psubset*{proof}

tts-lemma *indep-card-eq-dim-span*:

assumes $B \subseteq U_{VS}$ **and** \neg *dependent* B
shows $\text{finite } B \wedge \text{card } B = \text{dim } (\text{span } B)$
is *finite-dimensional-vector-space.indep-card-eq-dim-span*(*proof*)

tts-lemma *independent-bound-general*:

assumes $S \subseteq U_{VS}$ **and** \neg *dependent* S
shows $\text{finite } S \wedge \text{card } S \leq \text{dim } S$
is *finite-dimensional-vector-space.independent-bound-general*(*proof*)

tts-lemma *independent-explicit*:

assumes $B \subseteq U_{VS}$
shows $(\neg \text{dependent } B) =$
 $(\text{finite } B \wedge (\forall x. (\sum_{v \in B} x v * s v) = 0 \longrightarrow (\forall a \in B. x a = 0)))$
is *finite-dimensional-vector-space.independent-explicit*(*proof*)

tts-lemma *dim-sums-Int*:

assumes $S \subseteq U_{VS}$ **and** $T \subseteq U_{VS}$ **and** *subspace* S **and** *subspace* T
shows
 $\text{dim } \{x \in U_{VS}. \exists y \in U_{VS}. \exists z \in U_{VS}. x = y + z \wedge y \in S \wedge z \in T\} + \text{dim } (S \cap T) =$
 $\text{dim } S + \text{dim } T$
is *finite-dimensional-vector-space.dim-sums-Int*(*proof*)

tts-lemma *dependent-biggerset-general*:

assumes $S \subseteq U_{VS}$ **and** *finite* $S \implies \text{dim } S < \text{card } S$
shows *dependent* S
is *finite-dimensional-vector-space.dependent-biggerset-general*(*proof*)

tts-lemma *subset-le-dim*:

assumes $S \subseteq U_{VS}$ **and** $T \subseteq U_{VS}$ **and** $S \subseteq \text{span } T$
shows $\text{dim } S \leq \text{dim } T$
is *finite-dimensional-vector-space.subset-le-dim*(*proof*)

tts-lemma *linear-inj-imp-surj*:

assumes $\forall x \in U_{VS}. f x \in U_{VS}$
and *linear-on* $U_{VS} U_{VS} (*s) (*s) f$
and *inj-on* $f U_{VS}$
shows $f ' U_{VS} = U_{VS}$
is *finite-dimensional-vector-space.linear-inj-imp-surj*(*proof*)

tts-lemma *linear-surj-imp-inj*:

assumes $\forall x \in U_{VS}. f x \in U_{VS}$
and *linear-on* $U_{VS} U_{VS} (*s) (*s) f$
and $f ' U_{VS} = U_{VS}$
shows *inj-on* $f U_{VS}$
is *finite-dimensional-vector-space.linear-surj-imp-inj*(*proof*)

tts-lemma *linear-inverse-left*:

assumes $\forall x \in U_{VS}. f x \in U_{VS}$
and $\forall x \in U_{VS}. f' x \in U_{VS}$
and *linear-on* $U_{VS} U_{VS} (*s) (*s) f$
and *linear-on* $U_{VS} U_{VS} (*s) (*s) f'$
shows $(\forall x \in U_{VS}. (f \circ f') x = \text{id } x) = (\forall x \in U_{VS}. (f' \circ f) x = \text{id } x)$
is *finite-dimensional-vector-space.linear-inverse-left[unfolding fun-eq-iff]*(*proof*)

tts-lemma *left-inverse-linear*:

assumes $\forall x \in U_{VS}. f x \in U_{VS}$
and $\forall x \in U_{VS}. g x \in U_{VS}$

```

    and linear-on  $U_{VS} U_{VS} (*s) (*s) f$ 
    and  $\forall x \in U_{VS}. (g \circ f) x = id x$ 
  shows linear-on  $U_{VS} U_{VS} (*s) (*s) g$ 
  is finite-dimensional-vector-space.left-inverse-linear[unfolded fun-eq-iff]{proof}

tts-lemma right-inverse-linear:
  assumes  $\forall x \in U_{VS}. f x \in U_{VS}$ 
  and  $\forall x \in U_{VS}. g x \in U_{VS}$ 
  and linear-on  $U_{VS} U_{VS} (*s) (*s) f$ 
  and  $\forall x \in U_{VS}. (f \circ g) x = id x$ 
  shows linear-on  $U_{VS} U_{VS} (*s) (*s) g$ 
  is finite-dimensional-vector-space.right-inverse-linear[unfolded fun-eq-iff]{proof}

end

end

context vector-space-pair-on
begin

tts-context
  tts: ( $?b$  to  $\langle U_{M-1}::'b$  set  $\rangle$ ) and ( $?c$  to  $\langle U_{M-2}::'c$  set  $\rangle$ )
  sbterms: ( $\langle (+)::?'b::ab-group-add \Rightarrow ?'b \Rightarrow ?'b \rangle$  to  $\langle (+)::?'b \Rightarrow ?'b \Rightarrow ?'b \rangle$ )
    and ( $\langle ?s1.0::?'a::field \Rightarrow ?'b::ab-group-add \Rightarrow ?'b \rangle$  to  $\langle (*s_1)::?'a \Rightarrow ?'b \Rightarrow ?'b \rangle$ )
    and ( $\langle 0::?'b::ab-group-add \rangle$  to  $\langle 0::?'b \rangle$ )
    and ( $\langle (+)::?'c::ab-group-add \Rightarrow ?'c \Rightarrow ?'c \rangle$  to  $\langle (+)::?'c \Rightarrow ?'c \Rightarrow ?'c \rangle$ )
    and ( $\langle ?s2.0::?'a::field \Rightarrow ?'c::ab-group-add \Rightarrow ?'c \rangle$  to  $\langle (*s_2)::?'a \Rightarrow ?'c \Rightarrow ?'c \rangle$ )
    and ( $\langle 0::?'c::ab-group-add \rangle$  to  $\langle 0::?'c \rangle$ )
  rewriting ctr-simps
  substituting  $VS_1.ab-group-add-ow-axioms$ 
  and  $VS_1.vector-space-ow-axioms$ 
  and  $VS_2.ab-group-add-ow-axioms$ 
  and  $VS_2.vector-space-ow-axioms$ 
  and  $implicit_{VS}.vector-space-pair-ow-axioms$ 
  and  $VS_1.implicit_M.module-ow-axioms$ 
  and  $VS_2.implicit_M.module-ow-axioms$ 
  eliminating  $\langle ?a \in U_{M-1} \rangle$  and  $\langle ?B \subseteq U_{M-1} \rangle$  and  $\langle ?a \in U_{M-2} \rangle$  and  $\langle ?B \subseteq U_{M-2} \rangle$ 
  through auto
  applying
  [
    OF
     $VS_1.implicit_M.carrier-ne$   $VS_2.implicit_M.carrier-ne$ 
     $VS_1.implicit_M.minus-closed'$   $VS_1.implicit_M.uminus-closed'$ 
     $VS_2.implicit_{VS}.minus-closed'$   $VS_2.implicit_{VS}.uminus-closed'$ ,
    unfolded tts-implicit
  ]
begin

tts-lemma linear-add:
  assumes  $\forall x \in U_{M-1}. f x \in U_{M-2}$ 
  and  $b1 \in U_{M-1}$ 
  and  $b2 \in U_{M-1}$ 
  and linear-on  $U_{M-1} U_{M-2} (*s_1) (*s_2) f$ 
  shows  $f (b1 + b2) = f b1 + f b2$ 
  is vector-space-pair.linear-add{proof}

tts-lemma linear-scale:
  assumes  $\forall x \in U_{M-1}. f x \in U_{M-2}$ 

```

and $b \in U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
shows $f (r *s_1 b) = r *s_2 f b$
is *vector-space-pair.linear-scale* \langle *proof* \rangle

tts-lemma *linear-neg*:
assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $x \in U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
shows $f (- x) = - f x$
is *vector-space-pair.linear-neg* \langle *proof* \rangle

tts-lemma *linear-diff*:
assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $x \in U_{M-1}$
and $y \in U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
shows $f (x - y) = f x - f y$
is *vector-space-pair.linear-diff* \langle *proof* \rangle

tts-lemma *linear-sum*:
assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and *range* $g \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
shows $f (sum g S) = (\sum a \in S. f (g a))$
is *vector-space-pair.linear-sum* \langle *proof* \rangle

tts-lemma *linear-inj-on-iff-eq-0*:
assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $s \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and *VS₁.subspace* s
shows *inj-on* $f s = (\forall x \in s. f x = 0 \longrightarrow x = 0)$
is *vector-space-pair.linear-inj-on-iff-eq-0* \langle *proof* \rangle

tts-lemma *linear-inj-iff-eq-0*:
assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
shows *inj-on* $f U_{M-1} = (\forall x \in U_{M-1}. f x = 0 \longrightarrow x = 0)$
is *vector-space-pair.linear-inj-iff-eq-0* \langle *proof* \rangle

tts-lemma *linear-subspace-image*:
assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $S \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and *VS₁.subspace* S
shows *VS₂.subspace* $(f ' S)$
is *vector-space-pair.linear-subspace-image* \langle *proof* \rangle

tts-lemma *linear-subspace-kernel*:
assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
shows *VS₁.subspace* $\{x \in U_{M-1}. f x = 0\}$
is *vector-space-pair.linear-subspace-kernel* \langle *proof* \rangle

tts-lemma *linear-span-image*:
assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $S \subseteq U_{M-1}$

and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
shows $VS_2.\text{span} (f \text{ ' } S) = f \text{ ' } VS_1.\text{span} S$
is *vector-space-pair.linear-span-image* $\langle\text{proof}\rangle$

tts-lemma *linear-dependent-inj-imageD*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $s \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $VS_2.\text{dependent} (f \text{ ' } s)$
and *inj-on* $f (VS_1.\text{span} s)$
shows $VS_1.\text{dependent} s$
is *vector-space-pair.linear-dependent-inj-imageD* $\langle\text{proof}\rangle$

tts-lemma *linear-eq-0-on-span*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $b \subseteq U_{M-1}$
and $x \in U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $\bigwedge x. [[x \in U_{M-1}; x \in b]] \implies f x = 0$
and $x \in VS_1.\text{span} b$
shows $f x = 0$
is *vector-space-pair.linear-eq-0-on-span* $\langle\text{proof}\rangle$

tts-lemma *linear-independent-injective-image*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $s \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $\neg VS_1.\text{dependent} s$
and *inj-on* $f (VS_1.\text{span} s)$
shows $\neg VS_2.\text{dependent} (f \text{ ' } s)$
is *vector-space-pair.linear-independent-injective-image* $\langle\text{proof}\rangle$

tts-lemma *linear-inj-on-span-independent-image*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $B \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $\neg VS_2.\text{dependent} (f \text{ ' } B)$
and *inj-on* $f B$
shows *inj-on* $f (VS_1.\text{span} B)$
is *vector-space-pair.linear-inj-on-span-independent-image* $\langle\text{proof}\rangle$

tts-lemma *linear-inj-on-span-iff-independent-image*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $B \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $\neg VS_2.\text{dependent} (f \text{ ' } B)$
shows *inj-on* $f (VS_1.\text{span} B) = \text{inj-on } f B$
is *vector-space-pair.linear-inj-on-span-iff-independent-image* $\langle\text{proof}\rangle$

tts-lemma *linear-subspace-linear-preimage*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $S \subseteq U_{M-2}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $VS_2.\text{subspace} S$
shows $VS_1.\text{subspace} \{x \in U_{M-1}. f x \in S\}$
is *vector-space-pair.linear-subspace-linear-preimage* $\langle\text{proof}\rangle$

tts-lemma *linear-spans-image*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $V \subseteq U_{M-1}$
and $B \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $V \subseteq VS_1.\text{span } B$
shows $f' V \subseteq VS_2.\text{span } (f' B)$
is *vector-space-pair.linear-spans-image**{proof}*

tts-lemma *linear-spanning-surjective-image:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $S \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $U_{M-1} \subseteq VS_1.\text{span } S$
and $f' U_{M-1} = U_{M-2}$
shows $U_{M-2} \subseteq VS_2.\text{span } (f' S)$
is *vector-space-pair.linear-spanning-surjective-image**{proof}*

tts-lemma *linear-eq-on-span:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $\forall x \in U_{M-1}. g x \in U_{M-2}$
and $B \subseteq U_{M-1}$
and $x \in U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) g$
and $\bigwedge x. [[x \in U_{M-1}; x \in B]] \implies f x = g x$
and $x \in VS_1.\text{span } B$
shows $f x = g x$
is *vector-space-pair.linear-eq-on-span**{proof}*

tts-lemma *linear-compose-scale-right:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
shows *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. c *s_2 f x)$
is *vector-space-pair.linear-compose-scale-right**{proof}*

tts-lemma *linear-compose-add:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $\forall x \in U_{M-1}. g x \in U_{M-2}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) g$
shows *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. f x + g x)$
is *vector-space-pair.linear-compose-add**{proof}*

tts-lemma *linear-zero:*

shows *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. 0)$
is *vector-space-pair.linear-zero**{proof}*

tts-lemma *linear-compose-sub:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $\forall x \in U_{M-1}. g x \in U_{M-2}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) g$
shows *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. f x - g x)$
is *vector-space-pair.linear-compose-sub**{proof}*

tts-lemma *linear-compose-neg:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$

shows *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. - f x)$
is *vector-space-pair.linear-compose-neg*{proof}

tts-lemma *linear-compose-scale*:

assumes $c \in U_{M-2}$
and *linear-on* $U_{M-1} UNIV (*s_1) (*) f$
shows *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. f x *s_2 c)$
is *vector-space-pair.linear-compose-scale*{proof}

tts-lemma *linear-indep-image-lemma*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $B \subseteq U_{M-1}$
and $x \in U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and *finite* B
and $\neg VS_2.dependent (f \text{ ` } B)$
and *inj-on* $f B$
and $x \in VS_1.span B$
and $f x = 0$
shows $x = 0$
is *vector-space-pair.linear-indep-image-lemma*{proof}

tts-lemma *linear-eq-on*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $\forall x \in U_{M-1}. g x \in U_{M-2}$
and $x \in U_{M-1}$
and $B \subseteq U_{M-1}$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) g$
and $x \in VS_1.span B$
and $\bigwedge b. [[b \in U_{M-1}; b \in B]] \implies f b = g b$
shows $f x = g x$
is *vector-space-pair.linear-eq-on*{proof}

tts-lemma *linear-compose-sum*:

assumes $\forall x. \forall y \in U_{M-1}. f x y \in U_{M-2}$
and $\forall x \in S. \text{linear-on } U_{M-1} U_{M-2} (*s_1) (*s_2) (f x)$
shows *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) (\lambda x. \sum a \in S. f a x)$
is *vector-space-pair.linear-compose-sum*{proof}

tts-lemma *linear-independent-extend-subspace*:

assumes $B \subseteq U_{M-1}$
and $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $\neg VS_1.dependent B$
shows
 $\exists x.$
 $(\forall a \in U_{M-1}. x a \in U_{M-2}) \wedge$
 $\text{linear-on } U_{M-1} U_{M-2} (*s_1) (*s_2) x \wedge$
 $(\forall a \in B. x a = f a) \wedge$
 $x \text{ ` } U_{M-1} = VS_2.span (f \text{ ` } B)$
given *vector-space-pair.linear-independent-extend-subspace* {proof}

tts-lemma *linear-independent-extend*:

assumes $B \subseteq U_{M-1}$
and $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $\neg VS_1.dependent B$
shows
 $\exists x.$

$(\forall a \in U_{M-1}. x a \in U_{M-2}) \wedge$
 $(\forall a \in B. x a = f a) \wedge$
 $\text{linear-on } U_{M-1} U_{M-2} (*s_1) (*s_2) x$
given *vector-space-pair.linear-independent-extend* $\langle \text{proof} \rangle$

tts-lemma *linear-exists-left-inverse-on:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $V \subseteq U_{M-1}$
and $\text{linear-on } U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $VS_1.\text{subspace } V$
and $\text{inj-on } f V$
shows
 $\exists x.$
 $(\forall a \in U_{M-2}. x a \in U_{M-1}) \wedge$
 $x \text{ ' } U_{M-2} \subseteq V \wedge$
 $(\forall a \in V. x (f a) = a) \wedge$
 $\text{linear-on } U_{M-2} U_{M-1} (*s_2) (*s_1) x$
given *vector-space-pair.linear-exists-left-inverse-on* $\langle \text{proof} \rangle$

tts-lemma *linear-exists-right-inverse-on:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $V \subseteq U_{M-1}$
and $\text{linear-on } U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $VS_1.\text{subspace } V$
shows
 $\exists x.$
 $(\forall a \in U_{M-2}. x a \in U_{M-1}) \wedge$
 $x \text{ ' } U_{M-2} \subseteq V \wedge$
 $(\forall a \in f \text{ ' } V. f (x a) = a) \wedge$
 $\text{linear-on } U_{M-2} U_{M-1} (*s_2) (*s_1) x$
given *vector-space-pair.linear-exists-right-inverse-on* $\langle \text{proof} \rangle$

tts-lemma *linear-inj-on-left-inverse:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $S \subseteq U_{M-1}$
and $\text{linear-on } U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $\text{inj-on } f (VS_1.\text{span } S)$
shows
 $\exists x.$
 $(\forall a \in U_{M-2}. x a \in U_{M-1}) \wedge$
 $x \text{ ' } U_{M-2} \subseteq VS_1.\text{span } S \wedge$
 $(\forall a \in VS_1.\text{span } S. x (f a) = a) \wedge$
 $\text{linear-on } U_{M-2} U_{M-1} (*s_2) (*s_1) x$
given *vector-space-pair.linear-inj-on-left-inverse* $\langle \text{proof} \rangle$

tts-lemma *linear-surj-right-inverse:*

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$
and $T \subseteq U_{M-2}$
and $S \subseteq U_{M-1}$
and $\text{linear-on } U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $VS_2.\text{span } T \subseteq f \text{ ' } VS_1.\text{span } S$
shows
 $\exists x.$
 $(\forall a \in U_{M-2}. x a \in U_{M-1}) \wedge$
 $x \text{ ' } U_{M-2} \subseteq VS_1.\text{span } S \wedge$
 $(\forall a \in VS_2.\text{span } T. f (x a) = a) \wedge$
 $\text{linear-on } U_{M-2} U_{M-1} (*s_2) (*s_1) x$
given *vector-space-pair.linear-surj-right-inverse* $\langle \text{proof} \rangle$

tts-lemma *finite-basis-to-basis-subspace-isomorphism:*

assumes $S \subseteq U_{M-1}$
and $T \subseteq U_{M-2}$
and $VS_1.\text{subspace } S$
and $VS_2.\text{subspace } T$
and $VS_1.\text{dim } S = VS_2.\text{dim } T$
and *finite* B
and $B \subseteq S$
and $\neg VS_1.\text{dependent } B$
and $S \subseteq VS_1.\text{span } B$
and $\text{card } B = VS_1.\text{dim } S$
and *finite* C
and $C \subseteq T$
and $\neg VS_2.\text{dependent } C$
and $T \subseteq VS_2.\text{span } C$
and $\text{card } C = VS_2.\text{dim } T$
shows
 $\exists x.$
 $(\forall a \in U_{M-1}. x \ a \in U_{M-2}) \wedge$
 $\text{linear-on } U_{M-1} \ U_{M-2} \ (*s_1) \ (*s_2) \ x \wedge$
 $x \ ' \ B = C \wedge$
 $\text{inj-on } x \ S \wedge x \ ' \ S = T$
given *vector-space-pair.finite-basis-to-basis-subspace-isomorphism* $\langle \text{proof} \rangle$

tts-lemma *linear-subspace-vimage:*

assumes $\forall x \in U_{M-1}. f \ x \in U_{M-2}$
and $S \subseteq U_{M-2}$
and $\text{linear-on } U_{M-1} \ U_{M-2} \ (*s_1) \ (*s_2) \ f$
and $VS_2.\text{subspace } S$
shows $VS_1.\text{subspace } (f \ - \ ' \ S \cap U_{M-1})$
is *vector-space-pair.linear-subspace-vimage* $\langle \text{proof} \rangle$

tts-lemma *linear-injective-left-inverse:*

assumes $\forall x \in U_{M-1}. f \ x \in U_{M-2}$
and $\text{linear-on } U_{M-1} \ U_{M-2} \ (*s_1) \ (*s_2) \ f$
and $\text{inj-on } f \ U_{M-1}$
shows
 $\exists x.$
 $(\forall a \in U_{M-2}. x \ a \in U_{M-1}) \wedge$
 $(\forall a \in U_{M-1}. (x \circ f) \ a = \text{id } a) \wedge$
 $\text{linear-on } U_{M-2} \ U_{M-1} \ (*s_2) \ (*s_1) \ x$
given *vector-space-pair.linear-injective-left-inverse* $[\text{unfolded fun-eq-iff}]$
 $\langle \text{proof} \rangle$

tts-lemma *linear-surjective-right-inverse:*

assumes $\forall x \in U_{M-1}. f \ x \in U_{M-2}$
and $\text{linear-on } U_{M-1} \ U_{M-2} \ (*s_1) \ (*s_2) \ f$
and $f \ ' \ U_{M-1} = U_{M-2}$
shows
 $\exists x.$
 $(\forall a \in U_{M-2}. x \ a \in U_{M-1}) \wedge$
 $(\forall a \in U_{M-2}. (f \circ x) \ a = \text{id } a) \wedge$
 $\text{linear-on } U_{M-2} \ U_{M-1} \ (*s_2) \ (*s_1) \ x$
given *vector-space-pair.linear-surjective-right-inverse* $[\text{unfolded fun-eq-iff}]$
 $\langle \text{proof} \rangle$

end

end

context *finite-dimensional-vector-space-pair-1-on*
begin

tts-context

tts: ($?b$ to $\langle U_{M-1}::'b \text{ set} \rangle$) and ($?c$ to $\langle U_{M-2}::'c \text{ set} \rangle$)
 sbterms: ($\langle (+)::?'b::ab\text{-group-add} \Rightarrow ?'b \Rightarrow ?'b \rangle$ to $\langle (+)::'b \Rightarrow 'b \Rightarrow 'b \rangle$)
 and ($\langle ?s1.0::?'a::field \Rightarrow ?'b::ab\text{-group-add} \Rightarrow ?'b \rangle$ to $\langle (*s_1)::'a \Rightarrow 'b \Rightarrow 'b \rangle$)
 and ($\langle 0::?'b::ab\text{-group-add} \rangle$ to $\langle 0::'b \rangle$)
 and ($\langle (+)::?'c::ab\text{-group-add} \Rightarrow ?'c \Rightarrow ?'c \rangle$ to $\langle (+)::'c \Rightarrow 'c \Rightarrow 'c \rangle$)
 and ($\langle ?s2.0::?'a::field \Rightarrow ?'c::ab\text{-group-add} \Rightarrow ?'c \rangle$ to $\langle (*s_2)::'a \Rightarrow 'c \Rightarrow 'c \rangle$)
 and ($\langle 0::?'c::ab\text{-group-add} \rangle$ to $\langle 0::'c \rangle$)

rewriting *ctr-simps*

substituting *VS₁.ab-group-add-ow-axioms*

and *VS₁.vector-space-ow-axioms*

and *VS₂.ab-group-add-ow-axioms*

and *VS₂.vector-space-ow-axioms*

and *implicit_{VS}.vector-space-pair-ow-axioms*

and *VS₁.implicit_M.module-ow-axioms*

and *VS₂.implicit_M.module-ow-axioms*

and *implicit_{VS}.finite-dimensional-vector-space-pair-1-ow-axioms*

applying

[
OF
VS₁.implicit_M.carrier-ne VS₂.implicit_M.carrier-ne
VS₁.implicit_{VS}.minus-closed' VS₁.implicit_{VS}.uminus-closed'
VS₂.implicit_{VS}.minus-closed' VS₂.implicit_{VS}.uminus-closed'
VS₁.basis-subset,
unfolded tts-implicit
]

begin

tts-lemma *lt-dim-image-eq*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$

and $S \subseteq U_{M-1}$

and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$

and *inj-on* $f (VS_1.\text{span } S)$

shows $VS_2.\text{dim } (f \text{ ' } S) = VS_1.\text{dim } S$

is *finite-dimensional-vector-space-pair-1.dim-image-eq*(proof)

tts-lemma *lt-dim-image-le*:

assumes $\forall x \in U_{M-1}. f x \in U_{M-2}$

and $S \subseteq U_{M-1}$

and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$

shows $VS_2.\text{dim } (f \text{ ' } S) \leq VS_1.\text{dim } S$

is *finite-dimensional-vector-space-pair-1.dim-image-le*(proof)

end

end

context *finite-dimensional-vector-space-pair-on*
begin

tts-context

tts: ($?b$ to $\langle U_{M-1}::'b \text{ set} \rangle$) and ($?c$ to $\langle U_{M-2}::'c \text{ set} \rangle$)

```

sbterms: (⟨(+)::?'b::ab-group-add⇒?'b⇒?'b⟩ to ⟨(+)::'b⇒'b⇒'b⟩)
  and (⟨?s1.0::?'a::field ⇒?'b::ab-group-add⇒?'b⟩ to ⟨(*s1)::'a⇒'b⇒'b⟩)
  and (⟨0::?'b::ab-group-add⟩ to ⟨0::'b⟩)
  and (⟨(+)::?'c::ab-group-add⇒?'c⇒?'c⟩ to ⟨(+)::'c⇒'c⇒'c⟩)
  and (⟨?s2.0::?'a::field ⇒?'c::ab-group-add⇒?'c⟩ to ⟨(*s2)::'a⇒'c⇒'c⟩)
  and (⟨0::?'c::ab-group-add⟩ to ⟨0::'c⟩)
rewriting ctr-simps
substituting VS1.ab-group-add-ow-axioms
  and VS1.vector-space-ow-axioms
  and VS2.ab-group-add-ow-axioms
  and VS2.vector-space-ow-axioms
  and implicitVS.vector-space-pair-ow-axioms
  and VS1.implicitM.module-ow-axioms
  and VS2.implicitM.module-ow-axioms
  and implicitVS.finite-dimensional-vector-space-pair-ow-axioms
applying
  [
    OF
    VS1.implicitM.carrier-ne VS2.implicitM.carrier-ne
    VS1.implicitVS.minus-closed' VS1.implicitVS.uminus-closed'
    VS2.implicitVS.minus-closed' VS2.implicitVS.uminus-closed'
    VS1.basis-subset VS2.basis-subset,
    unfolded tts-implicit
  ]
begin

tts-lemma linear-surjective-imp-injective:
assumes ∀ x∈UM-1. f x ∈ UM-2
  and linear-on UM-1 UM-2 (*s1) (*s2) f
  and f ' UM-1 = UM-2
  and VS2.dim UM-2 = VS1.dim UM-1
shows inj-on f UM-1
  is finite-dimensional-vector-space-pair.linear-surjective-imp-injective{proof}

tts-lemma linear-injective-imp-surjective:
assumes ∀ x∈UM-1. f x ∈ UM-2
  and linear-on UM-1 UM-2 (*s1) (*s2) f
  and inj-on f UM-1
  and VS2.dim UM-2 = VS1.dim UM-1
shows f ' UM-1 = UM-2
  is finite-dimensional-vector-space-pair.linear-injective-imp-surjective{proof}

tts-lemma linear-injective-isomorphism:
assumes ∀ x∈UM-1. f x ∈ UM-2
  and linear-on UM-1 UM-2 (*s1) (*s2) f
  and inj-on f UM-1
  and VS2.dim UM-2 = VS1.dim UM-1
shows
  ∃ x.
    (∀ a∈UM-2. x a ∈ UM-1) ∧
    linear-on UM-2 UM-1 (*s2) (*s1) x ∧
    (∀ a∈UM-1. x (f a) = a) ∧
    (∀ a∈UM-2. f (x a) = a)
  given finite-dimensional-vector-space-pair.linear-injective-isomorphism
  {proof}

tts-lemma linear-surjective-isomorphism:
assumes ∀ x∈UM-1. f x ∈ UM-2

```

and *linear-on* $U_{M-1} U_{M-2} (*s_1) (*s_2) f$
and $f \text{ ' } U_{M-1} = U_{M-2}$
and $VS_2.dim U_{M-2} = VS_1.dim U_{M-1}$
shows
 $\exists x.$
 $(\forall a \in U_{M-2}. x a \in U_{M-1}) \wedge$
linear-on $U_{M-2} U_{M-1} (*s_2) (*s_1) x \wedge$
 $(\forall a \in U_{M-1}. x (f a) = a) \wedge$
 $(\forall a \in U_{M-2}. f (x a) = a)$
given *finite-dimensional-vector-space-pair.linear-surjective-isomorphism*
<proof>

tts-lemma *basis-to-basis-subspace-isomorphism:*

assumes $S \subseteq U_{M-1}$
and $T \subseteq U_{M-2}$
and $B \subseteq U_{M-1}$
and $C \subseteq U_{M-2}$
and $VS_1.subspace S$
and $VS_2.subspace T$
and $VS_1.dim S = VS_2.dim T$
and $B \subseteq S$
and $\neg VS_1.dependent B$
and $S \subseteq VS_1.span B$
and $card B = VS_1.dim S$
and $C \subseteq T$
and $\neg VS_2.dependent C$
and $T \subseteq VS_2.span C$
and $card C = VS_2.dim T$
shows
 $\exists x.$
 $(\forall a \in U_{M-1}. x a \in U_{M-2}) \wedge$
linear-on $U_{M-1} U_{M-2} (*s_1) (*s_2) x \wedge$
 $x \text{ ' } B = C \wedge$
inj-on $x S \wedge$
 $x \text{ ' } S = T$
given *finite-dimensional-vector-space-pair.basis-to-basis-subspace-isomorphism*
<proof>

tts-lemma *subspace-isomorphism:*

assumes $S \subseteq U_{M-1}$
and $T \subseteq U_{M-2}$
and $VS_1.subspace S$
and $VS_2.subspace T$
and $VS_1.dim S = VS_2.dim T$
shows $\exists x.$
 $(\forall a \in U_{M-1}. x a \in U_{M-2}) \wedge$
 $(inj-on x S \wedge x \text{ ' } S = T) \wedge$
linear-on $U_{M-1} U_{M-2} (*s_1) (*s_2) x$
given *finite-dimensional-vector-space-pair.subspace-isomorphism* *<proof>*

end

end

TTS Foundations

5.1 Extension of the theory *Set*

```

lemma Ex1-transfer[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: bi-unique A right-total A
  shows ((A ==> (=)) ==> (=)) ( $\lambda P. (\exists!x \in (\text{Collect } (\text{Domain } p A)). P x)$ ) Ex1
  <proof>

```

5.2 Definite description operator

5.2.1 Definition and common properties

definition *The-on*

where *The-on* $U P =$
(if $\exists!x. x \in U \wedge P x$ *then* *Some* $(THE\ x. x \in U \wedge P x)$ *else* *None*)

syntax

-The-on $:: ptnrn \Rightarrow 'a\ set \Rightarrow bool \Rightarrow 'a\ option$
 $\langle (THE\ -\ on\ -./\ -) \rangle [0, 0, 10] 10$

syntax-consts

-The-on $\Rightarrow The-on$

translations $THE\ x\ on\ U. P \Rightarrow CONST\ The-on\ U\ (\lambda x. P)$

print-translation \langle

```
[
  (
    const_syntax<The_on>,
    fn ctxt => fn [Ut, Abs abs] =>
      let val (x, t) = Syntax_Trans.atomic_abs_tr' ctxt abs
      in Syntax.const syntax_const<_The_on> $ x $ Ut $ t end
  )
]
```

$\rangle \langle ML \rangle$

lemma *The-on-UNIV-eq-The:*

assumes $\exists!x. P x$

obtains x **where** $(THE\ x\ on\ UNIV. P x) = Some\ x$ **and** $(THE\ x. P x) = x$

$\langle proof \rangle$

lemma *The-on-UNIV-None:*

assumes $\neg(\exists!x. P x)$

shows $(THE\ x\ on\ UNIV. P x) = None$

$\langle proof \rangle$

lemma *The-on-eq-The:*

assumes $\exists!x. x \in U \wedge P x$

obtains x **where** $(THE\ x\ on\ U. P x) = Some\ x$ **and** $(THE\ x. x \in U \wedge P x) = x$

$\langle proof \rangle$

lemma *The-on-None:*

assumes $\neg(\exists!x. x \in U \wedge P x)$

shows $(THE\ x\ on\ U. P x) = None$

$\langle proof \rangle$

lemma *The-on-Some-equality[intro]:*

assumes $a \in U$ **and** $P a$ **and** $\bigwedge x. x \in U \Longrightarrow P x \Longrightarrow x = a$

shows $(THE\ x\ on\ U. P x) = Some\ a$

$\langle proof \rangle$

lemma *The-on-equality[intro]:*

assumes $a \in U$ **and** $P a$ **and** $\bigwedge x. x \in U \Longrightarrow P x \Longrightarrow x = a$

shows $the\ (THE\ x\ on\ U. P x) = a$

$\langle proof \rangle$

lemma *The-on-SomeI:*

assumes $a \in U$ **and** $P a$ **and** $\bigwedge x. x \in U \Longrightarrow P x \Longrightarrow x = a$

obtains x **where** $(THE\ x\ on\ U.\ P\ x) = Some\ x$ **and** $P\ x$
 $\langle proof \rangle$

lemma *The-onI*:

assumes $a \in U$ **and** $P\ a$ **and** $\bigwedge x. x \in U \implies P\ x \implies x = a$
shows P (*the* $(THE\ x\ on\ U.\ P\ x)$)
 $\langle proof \rangle$

lemma *The-on-SomeI'*:

assumes $\exists!x. x \in U \wedge P\ x$
obtains x **where** $(THE\ x\ on\ U.\ P\ x) = Some\ x$ **and** $P\ x$
 $\langle proof \rangle$

lemma *The-onI'*:

assumes $\exists!x. x \in U \wedge P\ x$
shows P (*the* $(THE\ x\ on\ U.\ P\ x)$)
 $\langle proof \rangle$

lemma *The-on-SomeI2*:

assumes $a \in U$
and $P\ a$
and $\bigwedge x. x \in U \implies P\ x \implies x = a$
and $\bigwedge x. x \in U \implies P\ x \implies Q\ x$
obtains x **where** $(THE\ x\ on\ U.\ P\ x) = Some\ x$ **and** $Q\ x$
 $\langle proof \rangle$

lemma *The-on-I2*:

assumes $a \in U$
and $P\ a$
and $\bigwedge x. x \in U \implies P\ x \implies x = a$
and $\bigwedge x. x \in U \implies P\ x \implies Q\ x$
shows Q (*the* $(THE\ x\ on\ U.\ P\ x)$)
 $\langle proof \rangle$

lemma *The-on-SomeI2*:

assumes $\exists!x. x \in U \wedge P\ x$ **and** $\bigwedge x. x \in U \implies P\ x \implies Q\ x$
obtains x **where** $(THE\ x\ on\ U.\ P\ x) = Some\ x$ **and** $Q\ x$
 $\langle proof \rangle$

lemma *The-onI2*:

assumes $\exists!x. x \in U \wedge P\ x$ **and** $\bigwedge x. x \in U \implies P\ x \implies Q\ x$
shows Q (*the* $(THE\ x\ on\ U.\ P\ x)$)
 $\langle proof \rangle$

lemma *The-on1-equality* [*elim?*]:

assumes $\exists!x. P\ x$ **and** $a \in U$ **and** $P\ a$
shows $(THE\ x\ on\ U.\ P\ x) = Some\ a$
 $\langle proof \rangle$

lemma *the-sym-eq-trivial*:

assumes $x \in U$
shows $(THE\ y\ on\ U.\ x = y) = Some\ x$
 $\langle proof \rangle$

5.2.2 Transfer rules

lemma *The-on-transfer*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique A right-total A*
shows (*rel-set A* \implies (*A* \implies (=)) \implies *rel-option A*) *The-on The-on*
{*proof*}

5.3 Auxiliary

5.3.1 Methods

```
method ow-locale-transfer uses locale-defs =  
(  
  unfold locale-defs,  
  (  
    (simp only: all-simps(6) all-comm, fold Ball-def)  
    | (fold Ball-def)  
    | tactic⟨all-tac⟩  
  ),  
  transfer-prover-start,  
  transfer-step+,  
  rule refl  
)
```

5.4 Abstract orders on types

5.4.1 Background

The results presented in this section were ported (with amendments and additions) from the theories *Orderings* and *Set-Interval* in the main library of Isabelle/HOL.

5.4.2 Order operations

Abstract order operations.

```

locale ord =
  fixes le ls :: ['a, 'a] ⇒ bool

locale ord-syntax = ord le ls for le ls :: ['a, 'a] ⇒ bool
begin

```

notation

```

le (⟨'(<sub>a</sub>')⟩) and
le (infix <sub>a</sub> 50) and
ls (⟨'(<sub>a</sub>')⟩) and
ls (infix <sub>a</sub> 50)

```

abbreviation (*input*) ge (**infix** _a 50)

where $x \geq_a y \equiv y \leq_a x$

abbreviation (*input*) gt (**infix** _a 50)

where $x >_a y \equiv y <_a x$

notation

```

ge (⟨'(>sub>a</sub>')⟩) and
ge (infix <sub>a</sub> 50) and
gt (⟨'(>sub>a</sub>')⟩) and
gt (infix <sub>a</sub> 50)

```

end

```

locale ord-dual = ord le ls for le ls :: ['a, 'a] ⇒ bool
begin

```

interpretation ord-syntax ⟨*proof*⟩

sublocale dual: ord ge gt ⟨*proof*⟩

end

Pairs.

```

locale ord-pair = orda: ord lea lsa + ordb: ord leb lsb
  for lea lsa :: ['a, 'a] ⇒ bool and leb lsb :: ['b, 'b] ⇒ bool
begin

```

sublocale rev: ord-pair le_b ls_b le_a ls_a ⟨*proof*⟩

end

```

locale ord-pair-syntax = ord-pair lea lsa leb lsb
  for lea lsa :: ['a, 'a] ⇒ bool and leb lsb :: ['b, 'b] ⇒ bool
begin

```

sublocale ord_a: ord-syntax le_a ls_a + ord_b: ord-syntax le_b ls_b ⟨*proof*⟩

notation le_a ($\langle'(\leq_a)'\rangle$)
and le_a (**infix** $\langle\leq_a\rangle$ 50)
and ls_a ($\langle'(<_a)'\rangle$)
and ls_a (**infix** $\langle<_a\rangle$ 50)
and le_b ($\langle'(\leq_b)'\rangle$)
and le_b (**infix** $\langle\leq_b\rangle$ 50)
and ls_b ($\langle'(<_b)'\rangle$)
and ls_b (**infix** $\langle<_b\rangle$ 50)

notation $ord_{a.ge}$ ($\langle'(\geq_a)'\rangle$)
and $ord_{a.ge}$ (**infix** $\langle\geq_a\rangle$ 50)
and $ord_{a.gt}$ ($\langle'(>_a)'\rangle$)
and $ord_{a.gt}$ (**infix** $\langle>_a\rangle$ 50)
and $ord_{b.ge}$ ($\langle'(\geq_b)'\rangle$)
and $ord_{b.ge}$ (**infix** $\langle\geq_b\rangle$ 50)
and $ord_{b.gt}$ ($\langle'(>_b)'\rangle$)
and $ord_{b.gt}$ (**infix** $\langle>_b\rangle$ 50)

end

locale $ord\text{-}pair\text{-}dual = ord\text{-}pair$ le_a ls_a le_b ls_b
for le_a $ls_a :: [a, 'a] \Rightarrow bool$ **and** le_b $ls_b :: [b, 'b] \Rightarrow bool$
begin

interpretation $ord\text{-}pair\text{-}syntax$ $\langle proof \rangle$

sublocale $ord\text{-}dual$: $ord\text{-}pair$ $\langle(\leq_a)\rangle$ $\langle(<_a)\rangle$ $\langle(\geq_b)\rangle$ $\langle(>_b)\rangle$ $\langle proof \rangle$
sublocale $dual\text{-}ord$: $ord\text{-}pair$ $\langle(\geq_a)\rangle$ $\langle(>_a)\rangle$ $\langle(\leq_b)\rangle$ $\langle(<_b)\rangle$ $\langle proof \rangle$
sublocale $dual\text{-}dual$: $ord\text{-}pair$ $\langle(\geq_a)\rangle$ $\langle(>_a)\rangle$ $\langle(\geq_b)\rangle$ $\langle(>_b)\rangle$ $\langle proof \rangle$

end

5.4.3 Preorders

Definitions

Abstract preorders.

locale $preorder = ord$ le ls **for** le $ls :: [a, 'a] \Rightarrow bool$ +
assumes $less\text{-}le\text{-}not\text{-}le$: ls x $y \longleftrightarrow le$ x $y \wedge \neg (le$ y $x)$
and $order\text{-}refl[iff]$: le x x
and $order\text{-}trans$: le x $y \Longrightarrow le$ y $z \Longrightarrow le$ x z

locale $preorder\text{-}dual = preorder$ le ls **for** le $ls :: [a, 'a] \Rightarrow bool$
begin

interpretation $ord\text{-}syntax$ $\langle proof \rangle$

sublocale $ord\text{-}dual$ $\langle proof \rangle$

sublocale $dual$: $preorder$ ge gt
 $\langle proof \rangle$

end

Pairs.

locale $ord\text{-}preorder = ord\text{-}pair$ le_a ls_a le_b ls_b + ord_b : $preorder$ le_b ls_b
for le_a $ls_a :: [a, 'a] \Rightarrow bool$ **and** le_b $ls_b :: [b, 'b] \Rightarrow bool$

```

locale ord-preorder-dual = ord-preorder  $le_a$   $ls_a$   $le_b$   $ls_b$ 
  for  $le_a$   $ls_a$  :: [ $'a$ ,  $'a$ ]  $\Rightarrow$  bool and  $le_b$   $ls_b$  :: [ $'b$ ,  $'b$ ]  $\Rightarrow$  bool
begin

interpretation ord-pair-syntax  $\langle$ proof $\rangle$ 

sublocale ord-pair-dual  $\langle$ proof $\rangle$ 
sublocale ord-dual: ord-preorder  $\langle$  $(\leq_a)$  $\rangle$   $\langle$  $(<_a)$  $\rangle$   $\langle$  $(\geq_b)$  $\rangle$   $\langle$  $(>_b)$  $\rangle$ 
   $\langle$ proof $\rangle$ 
sublocale dual-ord: ord-preorder  $\langle$  $(\geq_a)$  $\rangle$   $\langle$  $(>_a)$  $\rangle$   $\langle$  $(\leq_b)$  $\rangle$   $\langle$  $(<_b)$  $\rangle$ 
   $\langle$ proof $\rangle$ 
sublocale dual-dual: ord-preorder  $\langle$  $(\geq_a)$  $\rangle$   $\langle$  $(>_a)$  $\rangle$   $\langle$  $(\geq_b)$  $\rangle$   $\langle$  $(>_b)$  $\rangle$ 
   $\langle$ proof $\rangle$ 

end

locale preorder-pair = ord-preorder  $le_a$   $ls_a$   $le_b$   $ls_b$  + orda: preorder  $le_a$   $ls_a$ 
  for  $le_a$   $ls_a$  :: [ $'a$ ,  $'a$ ]  $\Rightarrow$  bool and  $le_b$   $ls_b$  :: [ $'b$ ,  $'b$ ]  $\Rightarrow$  bool
begin

sublocale rev: preorder-pair  $le_b$   $ls_b$   $le_a$   $ls_a$   $\langle$ proof $\rangle$ 

end

locale preorder-pair-dual = preorder-pair  $le_a$   $ls_a$   $le_b$   $ls_b$ 
  for  $le_a$   $ls_a$  :: [ $'a$ ,  $'a$ ]  $\Rightarrow$  bool and  $le_b$   $ls_b$  :: [ $'b$ ,  $'b$ ]  $\Rightarrow$  bool
begin

interpretation ord-pair-syntax  $\langle$ proof $\rangle$ 

sublocale ord-preorder-dual  $\langle$ proof $\rangle$ 
sublocale ord-dual: preorder-pair  $\langle$  $(\leq_a)$  $\rangle$   $\langle$  $(<_a)$  $\rangle$   $\langle$  $(\geq_b)$  $\rangle$   $\langle$  $(>_b)$  $\rangle$   $\langle$ proof $\rangle$ 
sublocale dual-ord: preorder-pair  $\langle$  $(\geq_a)$  $\rangle$   $\langle$  $(>_a)$  $\rangle$   $\langle$  $(\leq_b)$  $\rangle$   $\langle$  $(<_b)$  $\rangle$ 
   $\langle$ proof $\rangle$ 
sublocale dual-dual: preorder-pair  $\langle$  $(\geq_a)$  $\rangle$   $\langle$  $(>_a)$  $\rangle$   $\langle$  $(\geq_b)$  $\rangle$   $\langle$  $(>_b)$  $\rangle$   $\langle$ proof $\rangle$ 

end

Results

context preorder
begin

interpretation ord-syntax  $\langle$ proof $\rangle$ 

Reflexivity.

lemma eq-refl:
  assumes  $x = y$ 
  shows  $x \leq_a y$ 
   $\langle$ proof $\rangle$ 

lemma less-irrefl[iff]:  $\neg x <_a x$   $\langle$ proof $\rangle$ 

lemma less-imp-le:
  assumes  $x <_a y$ 
  shows  $x \leq_a y$ 
   $\langle$ proof $\rangle$ 

```

lemma *strict-implies-not-eq*:

assumes $a <_a b$

shows $a \neq b$

<proof>

Asymmetry.

lemma *less-not-sym*:

assumes $x <_a y$

shows $\neg (y <_a x)$

<proof>

lemma *asym*:

assumes $a <_a b$ **and** $b <_a a$

shows *False*

<proof>

lemma *less-asym*:

assumes $x <_a y$ **and** $(\neg P \implies y <_a x)$

shows P

<proof>

Transitivity.

lemma *less-trans*:

assumes $x <_a y$ **and** $y <_a z$

shows $x <_a z$

<proof>

lemma *le-less-trans*:

assumes $x \leq_a y$ **and** $y <_a z$

shows $x <_a z$

<proof>

lemma *less-le-trans*:

assumes $x <_a y$ **and** $y \leq_a z$

shows $x <_a z$

<proof>

lemma *less-imp-not-less*:

assumes $x <_a y$

shows $(\neg y <_a x) \longleftrightarrow \text{True}$

<proof>

lemma *less-imp-triv*:

assumes $x <_a y$

shows $(y <_a x \longrightarrow P) \longleftrightarrow \text{True}$

<proof>

lemma *less-asym'*:

assumes $a <_a b$ **and** $b <_a a$

shows P

<proof>

end

5.4.4 Partial orders

Definitions

Abstract partial orders.

locale *order* = *preorder* *le* *ls* **for** *le* *ls* :: [*'a*, *'a*] ⇒ *bool* +
assumes *antisym*: *le* *x* *y* ⇒ *le* *y* *x* ⇒ *x* = *y*

locale *order-dual* = *order* *le* *ls* **for** *le* *ls* :: [*'a*, *'a*] ⇒ *bool*
begin

interpretation *ord-syntax* ⟨*proof*⟩

sublocale *preorder-dual* ⟨*proof*⟩

sublocale *dual*: *order* *ge* *gt*
 ⟨*proof*⟩

end

Pairs.

locale *ord-order* = *ord-preorder* *le_a* *ls_a* *le_b* *ls_b* + *ord_b*: *order* *le_b* *ls_b*
for *le_a* *ls_a* :: [*'a* ⇒ *'a*] ⇒ *bool* **and** *le_b* *ls_b* :: [*'b* ⇒ *'b*] ⇒ *bool*

locale *ord-order-dual* = *ord-order* *le_a* *ls_a* *le_b* *ls_b*
for *le_a* *ls_a* :: [*'a* ⇒ *'a*] ⇒ *bool* **and** *le_b* *ls_b* :: [*'b* ⇒ *'b*] ⇒ *bool*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-preorder-dual* ⟨*proof*⟩

sublocale *ord-dual*: *ord-order* ⟨(*≤_a*)⟩ ⟨(*<_a*)⟩ ⟨(*≥_b*)⟩ ⟨(*>_b*)⟩
 ⟨*proof*⟩

sublocale *dual-ord*: *ord-order* ⟨(*≥_a*)⟩ ⟨(*>_a*)⟩ ⟨(*≤_b*)⟩ ⟨(*<_b*)⟩
 ⟨*proof*⟩

sublocale *dual-dual*: *ord-order* ⟨(*≥_a*)⟩ ⟨(*>_a*)⟩ ⟨(*≥_b*)⟩ ⟨(*>_b*)⟩
 ⟨*proof*⟩

end

locale *preorder-order* = *ord-order* *le_a* *ls_a* *le_b* *ls_b* + *ord_a*: *preorder* *le_a* *ls_a*
for *le_a* *ls_a* :: [*'a*, *'a*] ⇒ *bool* **and** *le_b* *ls_b* :: [*'b*, *'b*] ⇒ *bool*

begin

sublocale *preorder-pair* ⟨*proof*⟩

end

locale *preorder-order-dual* = *preorder-order* *le_a* *ls_a* *le_b* *ls_b*
for *le_a* *ls_a* :: [*'a*, *'a*] ⇒ *bool* **and** *le_b* *ls_b* :: [*'b*, *'b*] ⇒ *bool*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-order-dual* ⟨*proof*⟩

sublocale *preorder-pair-dual* ⟨*proof*⟩

sublocale *ord-dual*: *preorder-order* ⟨(*≤_a*)⟩ ⟨(*<_a*)⟩ ⟨(*≥_b*)⟩ ⟨(*>_b*)⟩ ⟨*proof*⟩

sublocale *dual-ord*: *preorder-order* ⟨(*≥_a*)⟩ ⟨(*>_a*)⟩ ⟨(*≤_b*)⟩ ⟨(*<_b*)⟩ ⟨*proof*⟩

sublocale *dual-dual*: *preorder-order* $\langle(\geq_a)\rangle \langle(>_a)\rangle \langle(\geq_b)\rangle \langle(>_b)\rangle \langle\text{proof}\rangle$

end

locale *order-pair* = *preorder-order* $le_a\ ls_a\ le_b\ ls_b + ord_a$: *order* $le_a\ ls_a$

for $le_a\ ls_a :: [!a, 'a] \Rightarrow \text{bool}$ **and** $le_b\ ls_b :: [!b, 'b] \Rightarrow \text{bool}$

begin

sublocale *rev*: *order-pair* $le_b\ ls_b\ le_a\ ls_a \langle\text{proof}\rangle$

end

locale *order-pair-dual* = *order-pair* $le_a\ ls_a\ le_b\ ls_b$

for $le_a\ ls_a :: [!a, 'a] \Rightarrow \text{bool}$ **and** $le_b\ ls_b :: [!b, 'b] \Rightarrow \text{bool}$

begin

interpretation *ord-pair-syntax* $\langle\text{proof}\rangle$

sublocale *preorder-order-dual* $\langle\text{proof}\rangle$

sublocale *ord-dual*: *order-pair* $\langle(\leq_a)\rangle \langle(<_a)\rangle \langle(\geq_b)\rangle \langle(>_b)\rangle \langle\text{proof}\rangle$

sublocale *dual-ord*: *order-pair* $\langle(\geq_a)\rangle \langle(>_a)\rangle \langle(\leq_b)\rangle \langle(<_b)\rangle$

$\langle\text{proof}\rangle$

sublocale *dual-dual*: *order-pair* $\langle(\geq_a)\rangle \langle(>_a)\rangle \langle(\geq_b)\rangle \langle(>_b)\rangle \langle\text{proof}\rangle$

end

Results

context *order*

begin

interpretation *ord-syntax* $\langle\text{proof}\rangle$

Reflexivity.

lemma *less-le*: $x <_a y \leftrightarrow x \leq_a y \wedge x \neq y$

$\langle\text{proof}\rangle$

lemma *le-less*: $x \leq_a y \leftrightarrow x <_a y \vee x = y \langle\text{proof}\rangle$

lemma *le-imp-less-or-eq*:

assumes $x \leq_a y$

shows $x <_a y \vee x = y$

$\langle\text{proof}\rangle$

lemma *less-imp-not-eq*:

assumes $x <_a y$

shows $(x = y) \leftrightarrow \text{False}$

$\langle\text{proof}\rangle$

lemma *less-imp-not-eq2*:

assumes $x <_a y$

shows $(y = x) \leftrightarrow \text{False}$

$\langle\text{proof}\rangle$

Transitivity.

lemma *neq-le-trans*:

assumes $a \neq b$ **and** $a \leq_a b$

shows $a <_a b$

<proof>

lemma *le-neq-trans*:

assumes $a \leq_a b$ **and** $a \neq b$

shows $a <_a b$

<proof>

Asymmetry.

lemma *eq-iff*: $x = y \leftrightarrow x \leq_a y \wedge y \leq_a x$ *<proof>*

lemma *antisym-conv*:

assumes $y \leq_a x$

shows $x \leq_a y \leftrightarrow x = y$

<proof>

Other results.

lemma *antisym-conv1*:

assumes $\neg x <_a y$

shows $x \leq_a y \leftrightarrow x = y$

<proof>

lemma *antisym-conv2*:

assumes $x \leq_a y$

shows $\neg x <_a y \leftrightarrow x = y$

<proof>

lemma *leD*:

assumes $y \leq_a x$

shows $\neg x <_a y$

<proof>

end

5.4.5 Dense orders

Abstract dense orders.

locale *dense-order* = *order le ls* **for** $le\ ls :: ['a, 'a] \Rightarrow bool$ +
assumes *dense*: $ls\ x\ y \Longrightarrow (\exists z. ls\ x\ z \wedge ls\ z\ y)$

locale *dense-order-dual* = *dense-order le ls* **for** $le\ ls :: ['a, 'a] \Rightarrow bool$
begin

interpretation *ord-syntax* *<proof>*

sublocale *order-dual* *<proof>*

sublocale *dual: dense-order ge gt*
<proof>

end

Pairs.

locale *ord-dense-order* = *ord-order le_a ls_a le_b ls_b* + *ord_b: dense-order le_b ls_b*
for $le_a\ ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b\ ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

locale *ord-dense-order-dual* = *ord-dense-order le_a ls_a le_b ls_b*
for $le_a\ ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b\ ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

begin

interpretation *ord-pair-syntax* $\langle proof \rangle$

sublocale *ord-order-dual* $\langle proof \rangle$

sublocale *ord-dual: ord-dense-order* $\langle (\leq_a) \rangle \langle (<_a) \rangle \langle (\geq_b) \rangle \langle (>_b) \rangle$
 $\langle proof \rangle$

sublocale *dual-ord: ord-dense-order* $\langle (\geq_a) \rangle \langle (>_a) \rangle \langle (\leq_b) \rangle \langle (<_b) \rangle$
 $\langle proof \rangle$

sublocale *dual-dual: ord-dense-order* $\langle (\geq_a) \rangle \langle (>_a) \rangle \langle (\geq_b) \rangle \langle (>_b) \rangle$
 $\langle proof \rangle$

end

locale *preorder-dense-order* =

ord-dense-order $le_a ls_a le_b ls_b + ord_a: preorder le_a ls_a$

for $le_a ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

begin

sublocale *preorder-order* $\langle proof \rangle$

end

locale *preorder-dense-order-dual* = *preorder-dense-order* $le_a ls_a le_b ls_b$

for $le_a ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

begin

interpretation *ord-pair-syntax* $\langle proof \rangle$

sublocale *ord-dense-order-dual* $\langle proof \rangle$

sublocale *preorder-order-dual* $\langle proof \rangle$

sublocale *ord-dual: preorder-dense-order* $\langle (\leq_a) \rangle \langle (<_a) \rangle \langle (\geq_b) \rangle \langle (>_b) \rangle \langle proof \rangle$

sublocale *dual-ord: preorder-dense-order* $\langle (\geq_a) \rangle \langle (>_a) \rangle \langle (\leq_b) \rangle \langle (<_b) \rangle \langle proof \rangle$

sublocale *dual-dual: preorder-dense-order* $\langle (\geq_a) \rangle \langle (>_a) \rangle \langle (\geq_b) \rangle \langle (>_b) \rangle \langle proof \rangle$

end

locale *order-dense-order* =

preorder-dense-order $le_a ls_a le_b ls_b + ord_a: order le_a ls_a$

for $le_a ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

begin

sublocale *order-pair* $\langle proof \rangle$

end

locale *order-dense-order-dual* = *order-dense-order* $le_a ls_a le_b ls_b$

for $le_a ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

begin

interpretation *ord-pair-syntax* $\langle proof \rangle$

sublocale *preorder-dense-order-dual* $\langle proof \rangle$

sublocale *order-pair-dual* $\langle proof \rangle$

sublocale *ord-dual: order-dense-order* $\langle (\leq_a) \rangle \langle (<_a) \rangle \langle (\geq_b) \rangle \langle (>_b) \rangle \langle proof \rangle$

sublocale *dual-ord: order-dense-order* $\langle (\geq_a) \rangle \langle (>_a) \rangle \langle (\leq_b) \rangle \langle (<_b) \rangle \langle proof \rangle$

sublocale *dual-dual: order-dense-order* $\langle (\geq_a) \rangle \langle (>_a) \rangle \langle (\geq_b) \rangle \langle (>_b) \rangle \langle proof \rangle$

end

locale *dense-order-pair* =
order-dense-order $le_a\ ls_a\ le_b\ ls_b + ord_a$: *dense-order* $le_a\ ls_a$
for $le_a\ ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b\ ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

locale *dense-order-pair-dual* = *dense-order-pair* $le_a\ ls_a\ le_b\ ls_b$
for $le_a\ ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b\ ls_b :: 'b \Rightarrow 'b \Rightarrow bool$
begin

interpretation *ord-pair-syntax* $\langle proof \rangle$

sublocale *order-dense-order-dual* $\langle proof \rangle$

sublocale *ord-dual*: *dense-order-pair* $\langle (\leq_a) \rangle \langle (<_a) \rangle \langle (\geq_b) \rangle \langle (>_b) \rangle \langle proof \rangle$

sublocale *dual-ord*: *dense-order-pair* $\langle (\geq_a) \rangle \langle (>_a) \rangle \langle (\leq_b) \rangle \langle (<_b) \rangle$
 $\langle proof \rangle$

sublocale *dual-dual*: *dense-order-pair* $\langle (\geq_a) \rangle \langle (>_a) \rangle \langle (\geq_b) \rangle \langle (>_b) \rangle \langle proof \rangle$

end

5.4.6 (Unique) top and bottom elements

Abstract extremum.

locale *extremum* =
fixes *extremum* :: 'a

locale *ord-extremum* = *ord* $le\ ls + extremum\ extremum$
for $le\ ls :: 'a \Rightarrow 'a \Rightarrow bool$ **and** *extremum* :: 'a

Concrete syntax.

locale *bot* = *extremum* *bot* **for** *bot* :: 'a
begin

notation *bot* $\langle \perp \rangle$

end

locale *top* = *extremum* *top* **for** *top* :: 'a
begin

notation *top* $\langle \top \rangle$

end

5.4.7 (Unique) top and bottom elements for partial orders

Definitions

Abstract partial order with extremum.

locale *order-extremum* = *ord-extremum* $le\ ls\ extremum + order\ le\ ls$
for $le\ ls :: 'a \Rightarrow 'a \Rightarrow bool$
and *extremum* :: 'a +
assumes *extremum[simp]*: $le\ a\ extremum$

Concrete syntax.

locale *order-bot* =
order-dual $le\ ls +$

dual: order-extremum $\langle \lambda x y. le\ y\ x \rangle \langle \lambda x y. ls\ y\ x \rangle\ bot +$
bot bot
for $le\ ls :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $bot :: 'a$

locale *order-top* = *order-dual le ls* + *order-extremum le ls top* + *top top*
for $le\ ls :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $top :: 'a$

Results

context *order-extremum*
begin

interpretation *ord-syntax* $\langle proof \rangle$

lemma *extremum-uniqueI*:
assumes $extremum \leq_a a$
shows $a = extremum$
 $\langle proof \rangle$

lemma *extremum-unique*: $extremum \leq_a a \longleftrightarrow a = extremum$
 $\langle proof \rangle$

lemma *extremum-strict[simp]*: $\neg (extremum <_a a)$
 $\langle proof \rangle$

lemma *not-eq-extremum*: $a \neq extremum \longleftrightarrow a <_a extremum$
 $\langle proof \rangle$

end

5.4.8 Partial orders without top or bottom elements

Abstract partial orders without top or bottom elements.

locale *no-extremum* = *order le ls* **for** $le\ ls :: 'a \Rightarrow 'a \Rightarrow bool$ +
assumes *gt-ex*: $\exists y. ls\ x\ y$

Concrete syntax.

locale *no-top* = *order-dual le ls* + *no-extremum le ls*
for $le\ ls :: 'a \Rightarrow 'a \Rightarrow bool$

locale *no-bot* =
order-dual le ls +
dual: no-extremum $\langle \lambda x y. le\ y\ x \rangle \langle \lambda x y. ls\ y\ x \rangle$
for $le\ ls :: 'a \Rightarrow 'a \Rightarrow bool$

5.4.9 Least and greatest operators

definition *Least* :: $['a\ set, ['a, 'a] \Rightarrow bool, 'a \Rightarrow bool] \Rightarrow 'a\ option$
 $\langle \langle on - with - : \langle \langle Least \rangle - \rangle [1000, 1000, 1000] 10 \rangle$

where

$on\ U\ with\ op : \langle \langle Least \rangle P \equiv (THE\ x\ on\ U. P\ x \wedge (\forall y \in U. P\ y \longrightarrow op\ x\ y))$

ctr relativization

synthesis *ctr-simps*

assumes $[transfer-domain-rule, transfer-rule]: Domainp\ A = (\lambda x. x \in U)$

and $[transfer-rule]: bi-unique\ A\ right-total\ A$

trp $(? 'a\ A)$

in *Least-def*

context *ord-syntax*
begin

abbreviation *Least* **where** $Least \equiv Type\text{-}Simple\text{-}Orders.Least\ UNIV (\leq_a)$

abbreviation *Greatest* **where** $Greatest \equiv Type\text{-}Simple\text{-}Orders.Least\ UNIV (\geq_a)$

lemmas $Least\text{-}def = Least\text{-}def[of\ UNIV \langle(\leq_a)\rangle]$

end

context *order*
begin

interpretation *ord-syntax* $\langle proof \rangle$

lemma *Least-equality*:

assumes $P\ x$ **and** $\bigwedge y. P\ y \implies x \leq_a y$

shows $Least\ P = Some\ x$

$\langle proof \rangle$

lemma *LeastI2-order*:

assumes $P\ x$

and $\bigwedge y. P\ y \implies x \leq_a y$

and $\bigwedge x. P\ x \implies \forall y. P\ y \longrightarrow x \leq_a y \implies Q\ x$

obtains z **where** $Least\ P = Some\ z$ **and** $Q\ z$

$\langle proof \rangle$

lemma *Least-ex1*:

assumes $\exists! x. P\ x \wedge (\forall y. P\ y \longrightarrow x \leq_a y)$

obtains x **where** $Least\ P = Some\ x$ **and** $P\ x$ **and** $P\ z \implies x \leq_a z$

$\langle proof \rangle$

end

5.4.10 min and max

definition $min :: [['a, 'a] \Rightarrow bool, 'a, 'a] \Rightarrow 'a$ **where**

$min\ le\ a\ b = (if\ le\ a\ b\ then\ a\ else\ b)$

ctr *parametricity*

in *min-def*

context *ord-syntax*
begin

abbreviation *min* **where** $min \equiv Type\text{-}Simple\text{-}Orders.min (\leq_a)$

abbreviation *max* **where** $max \equiv Type\text{-}Simple\text{-}Orders.min (\geq_a)$

end

context *ord*
begin

interpretation *ord-syntax* $\langle proof \rangle$

lemma *min-absorb1*: $x \leq_a y \implies min\ x\ y = x$

$\langle proof \rangle$

end

context *order*
begin

interpretation *ord-syntax* $\langle proof \rangle$

lemma *min-absorb2*:
 assumes $y \leq_a x$
 shows $\min x y = y$
 $\langle proof \rangle$

end

context *order-extremum*
begin

interpretation *ord-syntax* $\langle proof \rangle$

lemma *max-top[simp]*: $\max \text{extremum } x = \text{extremum}$
 $\langle proof \rangle$

lemma *max-top2[simp]*: $\max x \text{extremum} = \text{extremum}$
 $\langle proof \rangle$

lemma *min-top[simp]*: $\min \text{extremum } x = x$ $\langle proof \rangle$

lemma *min-top2[simp]*: $\min x \text{extremum} = x$ $\langle proof \rangle$

end

5.4.11 Monotonicity

definition *mono* ::
 $['a \text{ set}, ['a, 'a] \Rightarrow \text{bool}, ['b, 'b] \Rightarrow \text{bool}, 'a \Rightarrow 'b] \Rightarrow \text{bool}$
 $\langle \langle \text{on - with - - : } \langle \text{mono} \rangle \text{ -} \rangle [1000, 1000, 999, 1000] 10 \rangle$

where

 on U_a with $op_1 \ op_2 : \langle \text{mono} \rangle f \equiv \forall x \in U_a. \forall y \in U_a. op_1 \ x \ y \longrightarrow op_2 \ (f \ x) \ (f \ y)$

ctr parametricity

 in *mono-def*

context *ord-pair-syntax*
begin

abbreviation *mono_{ab}*

 where $\text{mono}_{ab} \equiv \text{Type-Simple-Orders.mono UNIV } (\leq_a) (\leq_b)$

abbreviation *mono_{ba}*

 where $\text{mono}_{ba} \equiv \text{Type-Simple-Orders.mono UNIV } (\leq_b) (\leq_a)$

abbreviation *antimono_{ab}*

 where $\text{antimono}_{ab} \equiv \text{Type-Simple-Orders.mono UNIV } (\leq_a) (\geq_b)$

abbreviation *antimono_{ba}*

 where $\text{antimono}_{ba} \equiv \text{Type-Simple-Orders.mono UNIV } (\leq_b) (\geq_a)$

abbreviation *strict-mono_{ab}*

 where $\text{strict-mono}_{ab} \equiv \text{Type-Simple-Orders.mono UNIV } (<_a) (<_b)$

abbreviation *strict-mono_{ba}*

where $strict-mono_{ba} \equiv Type\text{-}Simple\text{-}Orders.mono\ UNIV\ (<_b)\ (<_a)$
abbreviation $strict-antimono_{ab}$
where $strict-antimono_{ab} \equiv Type\text{-}Simple\text{-}Orders.mono\ UNIV\ (<_a)\ (>_b)$
abbreviation $strict-antimono_{ba}$
where $strict-antimono_{ba} \equiv Type\text{-}Simple\text{-}Orders.mono\ UNIV\ (<_b)\ (>_a)$

end

context *ord-pair*
begin

interpretation *ord-pair-syntax* $\langle proof \rangle$

lemma *monoI*[*intro?*]:
assumes $\wedge x\ y. x \leq_a y \implies f\ x \leq_b f\ y$
shows $mono_{ab}\ f$
 $\langle proof \rangle$

lemma *monoD*[*dest?*]:
assumes $mono_{ab}\ f$ **and** $x \leq_a y$
shows $f\ x \leq_b f\ y$
 $\langle proof \rangle$

lemma *monoE*:
assumes $mono_{ab}\ f$ **and** $x \leq_a y$
obtains $f\ x \leq_b f\ y$
 $\langle proof \rangle$

lemma *strict-monoI*[*intro?*]:
assumes $\wedge x\ y. x <_a y \implies f\ x <_b f\ y$
shows $strict-mono_{ab}\ f$
 $\langle proof \rangle$

lemma *strict-monoD*[*dest?*]:
assumes $strict-mono_{ab}\ f$ **and** $x <_a y$
shows $f\ x <_b f\ y$
 $\langle proof \rangle$

lemma *strict-monoE*:
assumes $strict-mono_{ab}\ f$ **and** $x <_a y$
obtains $f\ x <_b f\ y$
 $\langle proof \rangle$

end

context *order-pair*
begin

interpretation *ord-pair-syntax* $\langle proof \rangle$

lemma *strict-mono-mono*[*dest?*]:
assumes $strict-mono_{ab}\ f$
shows $mono_{ab}\ f$
 $\langle proof \rangle$

end

5.4.12 Set intervals

definition *ray* :: [*'a set*, [*'a, 'a*] \Rightarrow *bool*, *'a*] \Rightarrow *'a set*

(\langle (*on - with -* : {*..c-*}) \rangle [1000, 1000, 1000] 10)

where *on U with op* : {*..cu*} \equiv {*x* \in *U*. *op x u*}

definition *interval* ::

[*'a set*, [*'a, 'a*] \Rightarrow *bool*, [*'a, 'a*] \Rightarrow *bool*, *'a, 'a*] \Rightarrow *'a set*

(\langle (*on - with - -* : {*-c..c-*}) \rangle [1000, 1000, 999, 1000, 1000] 10)

where *on U with op₁ op₂* : {*lc..cu*} \equiv

(*on U with* ($\lambda x y$. *op₁ y x*) : {*..cl*}) \cap (*on U with op₂* : {*..cu*})

lemma *ray-transfer*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows (*rel-set A* \implies (*A* \implies *A* \implies (=)) \implies *A* \implies *rel-set A*) *ray ray*

\langle *proof* \rangle

ctr relativization

assumes [*transfer-rule*]: *right-total A bi-unique A*

trp (*?'a A*)

in *interval-def*

lemma *interval-ge-le*:

(*on UNIV with* ($\lambda x y$. *le_a y x*) ($\lambda x y$. *le_b y x*) : {*lc..ch*}) =

(*on UNIV with le_b le_a* : {*hc..cl*})

\langle *proof* \rangle

context *ord-syntax*

begin

abbreviation *lessThan* (\langle {*..<_a-*}) \rangle)

where {*..<_au*} \equiv *on UNIV with* (*<_a*) : {*..cu*}

abbreviation *atMost* (\langle {*..≤_a-*}) \rangle)

where {*..≤_au*} \equiv *on UNIV with* (*≤_a*) : {*..cu*}

abbreviation *greaterThan* (\langle {*-<_a..*}) \rangle)

where {*l<_a..*} \equiv *on UNIV with* (*>_a*) : {*..cl*}

abbreviation *atLeast* (\langle {*-≤_a..*}) \rangle)

where {*l≤_a..*} \equiv *on UNIV with* (*≥_a*) : {*..cl*}

abbreviation *greaterThanLessThan* (\langle {*-<_a..<_a-*}) \rangle)

where {*l<_a..<_au*} \equiv *on UNIV with* (*<_a*) (*<_a*) : {*lc..cu*}

abbreviation *atLeastLessThan* (\langle {*-≤_a..<_a-*}) \rangle)

where {*l≤_a..<_au*} \equiv *on UNIV with* (*≤_a*) (*<_a*) : {*lc..cu*}

abbreviation *greaterThanAtMost* (\langle {*-<_a..≤_a-*}) \rangle)

where {*l<_a..≤_au*} \equiv *on UNIV with* (*<_a*) (*≤_a*) : {*lc..cu*}

abbreviation *atLeastAtMost* (\langle {*-≤_a..≤_a-*}) \rangle)

where {*l≤_a..≤_au*} \equiv *on UNIV with* (*≤_a*) (*≤_a*) : {*lc..cu*}

abbreviation *lessThanGreaterThan* (\langle {*->_a..>_a-*}) \rangle)

where {*l>_a..>_au*} \equiv *on UNIV with* (*>_a*) (*>_a*) : {*lc..cu*}

abbreviation *lessThanAtLeast* (\langle {*-≥_a..>_a-*}) \rangle)

where {*l≥_a..>_au*} \equiv *on UNIV with* (*≥_a*) (*>_a*) : {*lc..cu*}

abbreviation *atMostGreaterThan* (\langle {*->_a..≥_a-*}) \rangle)

where {*l>_a..≥_au*} \equiv *on UNIV with* (*>_a*) (*≥_a*) : {*lc..cu*}

abbreviation *atMostAtLeast* (\langle {*-≥_a..≥_a-*}) \rangle)

where {*l≥_a..≥_au*} \equiv *on UNIV with* (*≥_a*) (*≥_a*) : {*lc..cu*}

end

context *ord*

begin

interpretation *ord-syntax* $\langle proof \rangle$

lemma *lessThan-iff*[*iff*]: $(i \in \{..<_a k\}) = (i <_a k)$
 $\langle proof \rangle$

lemma *atLeast-iff*[*iff*]: $(i \in \{k \leq_a ..\}) = (k \leq_a i)$
 $\langle proof \rangle$

lemma *greaterThanLessThan-iff*[*simp*]: $(i \in \{l <_{a..} <_a u\}) = (l <_a i \wedge i <_a u)$
 $\langle proof \rangle$

lemma *atLeastLessThan-iff*[*simp*]: $(i \in \{l \leq_{a..} <_a u\}) = (l \leq_a i \wedge i <_a u)$
 $\langle proof \rangle$

lemma *greaterThanAtMost-iff*[*simp*]: $(i \in \{l <_{a..} \leq_a u\}) = (l <_a i \wedge i \leq_a u)$
 $\langle proof \rangle$

lemma *atLeastAtMost-iff*[*simp*]: $(i \in \{l \leq_{a..} \leq_a u\}) = (l \leq_a i \wedge i \leq_a u)$
 $\langle proof \rangle$

lemma *greaterThanLessThan-eq*: $\{a <_{a..} <_a b\} = \{a <_{a..}\} \cap \{.. <_a b\}$
 $\langle proof \rangle$

end

context *ord-pair-syntax*

begin

notation *ord_a.lessThan* ($\langle \{.. <_a -\} \rangle$)
and *ord_a.atMost* ($\langle \{.. \leq_a -\} \rangle$)
and *ord_a.greaterThan* ($\langle \{<_{a..} -\} \rangle$)
and *ord_a.atLeast* ($\langle \{- \leq_{a..}\} \rangle$)
and *ord_a.greaterThanLessThan* ($\langle \{<_{a..} <_a -\} \rangle$)
and *ord_a.atLeastLessThan* ($\langle \{- \leq_{a..} <_a -\} \rangle$)
and *ord_a.greaterThanAtMost* ($\langle \{<_{a..} \leq_a -\} \rangle$)
and *ord_a.atLeastAtMost* ($\langle \{- \leq_{a..} \leq_a -\} \rangle$)
and *ord_a.lessThanGreaterThan* ($\langle \{>_{a..} >_a -\} \rangle$)
and *ord_a.lessThanAtLeast* ($\langle \{- \geq_{a..} >_a -\} \rangle$)
and *ord_a.atMostGreaterThan* ($\langle \{>_{a..} \geq_a -\} \rangle$)
and *ord_a.atMostAtLeast* ($\langle \{- \geq_{a..} \geq_a -\} \rangle$)
and *ord_b.lessThan* ($\langle \{.. <_b -\} \rangle$)
and *ord_b.atMost* ($\langle \{.. \leq_b -\} \rangle$)
and *ord_b.greaterThan* ($\langle \{<_{b..} -\} \rangle$)
and *ord_b.atLeast* ($\langle \{- \leq_{b..}\} \rangle$)
and *ord_b.greaterThanLessThan* ($\langle \{<_{b..} <_b -\} \rangle$)
and *ord_b.atLeastLessThan* ($\langle \{- \leq_{b..} <_b -\} \rangle$)
and *ord_b.greaterThanAtMost* ($\langle \{<_{b..} \leq_b -\} \rangle$)
and *ord_b.atLeastAtMost* ($\langle \{- \leq_{b..} \leq_b -\} \rangle$)
and *ord_b.lessThanGreaterThan* ($\langle \{>_{b..} >_b -\} \rangle$)
and *ord_b.lessThanAtLeast* ($\langle \{- \geq_{b..} >_b -\} \rangle$)
and *ord_b.atMostGreaterThan* ($\langle \{>_{b..} \geq_b -\} \rangle$)
and *ord_b.atMostAtLeast* ($\langle \{- \geq_{b..} \geq_b -\} \rangle$)

end

context *preorder*

begin

interpretation *ord-syntax* $\langle proof \rangle$

lemma *Ioi-le-Ico*: $\{a <_{a..}\} \subseteq \{a \leq_{a..}\}$
 $\langle proof \rangle$

end

context *preorder*

begin

interpretation *ord-syntax* $\langle proof \rangle$

interpretation *preorder-dual le ls*
 $\langle proof \rangle$

lemma *single-Diff-lessThan*[*simp*]: $\{k\} - \{..<_a k\} = \{k\}$ $\langle proof \rangle$

lemma *atLeast-subset-iff*[*iff*]: $(\{x \leq_{a..}\} \subseteq \{y \leq_{a..}\}) = (y \leq_a x)$
 $\langle proof \rangle$

lemma *atLeastatMost-empty*[*simp*]:

assumes $b <_a a$

shows $\{a \leq_{a..} \leq_a b\} = \{\}$

$\langle proof \rangle$

lemma *atLeastatMost-empty-iff*[*simp*]: $\{a \leq_{a..} \leq_a b\} = \{\} \leftrightarrow (\neg a \leq_a b)$
 $\langle proof \rangle$

lemma *atLeastatMost-empty-iff2*[*simp*]: $\{\} = \{a \leq_{a..} \leq_a b\} \leftrightarrow (\neg a \leq_a b)$
 $\langle proof \rangle$

lemma *atLeastLessThan-empty*[*simp*]:

assumes $b \leq_a a$

shows $\{a \leq_{a..} <_a b\} = \{\}$

$\langle proof \rangle$

lemma *atLeastLessThan-empty-iff*[*simp*]: $\{a \leq_{a..} <_a b\} = \{\} \leftrightarrow (\neg a <_a b)$
 $\langle proof \rangle$

lemma *atLeastLessThan-empty-iff2*[*simp*]: $\{\} = \{a \leq_{a..} <_a b\} \leftrightarrow (\neg a <_a b)$
 $\langle proof \rangle$

lemma *greaterThanAtMost-empty*[*simp*]:

assumes $l \leq_a k$

shows $\{k <_{a..} \leq_a l\} = \{\}$

$\langle proof \rangle$

lemma *greaterThanAtMost-empty-iff*[*simp*]: $\{k <_{a..} \leq_a l\} = \{\} \leftrightarrow \neg k <_a l$
 $\langle proof \rangle$

lemma *greaterThanAtMost-empty-iff2*[*simp*]: $\{\} = \{k <_{a..} \leq_a l\} \leftrightarrow \neg k <_a l$
 $\langle proof \rangle$

lemma *greaterThanLessThan-empty*[*simp*]:

assumes $l \leq_a k$

shows $\{k <_{a..} <_a l\} = \{\}$

$\langle \text{proof} \rangle$

lemma *atLeastatMost-subset-iff[simp]*:

$$\{a \leq_{a..} \leq_a b\} \subseteq \{c \leq_{a..} \leq_a d\} \longleftrightarrow (\neg a \leq_a b) \vee c \leq_a a \wedge b \leq_a d$$

$\langle \text{proof} \rangle$

lemma *atLeastatMost-psubset-iff*:

$$\{a \leq_{a..} \leq_a b\} < \{c \leq_{a..} \leq_a d\} \longleftrightarrow$$

$$((\neg a \leq_a b) \vee c \leq_a a \wedge b \leq_a d \wedge (c <_a a \vee b <_a d)) \wedge c \leq_a d$$

$\langle \text{proof} \rangle$

lemma *Icc-subset-Ici-iff[simp]*:

$$\{l \leq_{a..} \leq_a h\} \subseteq \{l' \leq_{a..}\} = (\neg l \leq_a h \vee l \geq_a l')$$

$\langle \text{proof} \rangle$

lemma *Icc-subset-Iic-iff[simp]*:

$$\{l \leq_{a..} \leq_a h\} \subseteq \{.. \leq_a h'\} = (\neg l \leq_a h \vee h \leq_a h')$$

$\langle \text{proof} \rangle$

lemma *not-Ici-eq-empty[simp]*: $\{l \leq_{a..}\} \neq \{\}$ $\langle \text{proof} \rangle$

lemmas *not-empty-eq-Ici-eq-empty[simp]* = *not-Ici-eq-empty[symmetric]*

lemma *Iio-Int-singleton*: $\{.. <_a k\} \cap \{x\} = (\text{if } x <_a k \text{ then } \{x\} \text{ else } \{\})$ $\langle \text{proof} \rangle$

lemma *ivl-disj-int-one*:

$$\{.. \leq_a l\} \cap \{l <_{a..} <_a u\} = \{\}$$

$$\{.. <_a l\} \cap \{l \leq_{a..} <_a u\} = \{\}$$

$$\{.. \leq_a l\} \cap \{l <_{a..} \leq_a u\} = \{\}$$

$$\{.. <_a l\} \cap \{l \leq_{a..} \leq_a u\} = \{\}$$

$$\{l <_{a..} \leq_a u\} \cap \{u <_{a..}\} = \{\}$$

$$\{l <_{a..} <_a u\} \cap \{u \leq_{a..}\} = \{\}$$

$$\{l \leq_{a..} \leq_a u\} \cap \{u <_{a..}\} = \{\}$$

$$\{l \leq_{a..} <_a u\} \cap \{u \leq_{a..}\} = \{\}$$

$\langle \text{proof} \rangle$

lemma *ivl-disj-int-two*:

$$\{l <_{a..} <_a m\} \cap \{m \leq_{a..} <_a u\} = \{\}$$

$$\{l <_{a..} \leq_a m\} \cap \{m <_{a..} <_a u\} = \{\}$$

$$\{l \leq_{a..} <_a m\} \cap \{m \leq_{a..} <_a u\} = \{\}$$

$$\{l \leq_{a..} \leq_a m\} \cap \{m <_{a..} <_a u\} = \{\}$$

$$\{l <_{a..} <_a m\} \cap \{m \leq_{a..} \leq_a u\} = \{\}$$

$$\{l <_{a..} \leq_a m\} \cap \{m <_{a..} \leq_a u\} = \{\}$$

$$\{l \leq_{a..} <_a m\} \cap \{m \leq_{a..} \leq_a u\} = \{\}$$

$$\{l \leq_{a..} \leq_a m\} \cap \{m <_{a..} \leq_a u\} = \{\}$$

$\langle \text{proof} \rangle$

end

context *order*

begin

interpretation *ord-syntax* $\langle \text{proof} \rangle$

interpretation *order-dual* *le ls*

$\langle \text{proof} \rangle$

lemma *atMost-Int-atLeast*: $\{.. \leq_a n\} \cap \{n \leq_{a..}\} = \{n\}$

$\langle proof \rangle$

lemma *atLeast-eq-iff[iff]*: $(\{x \leq_a \dots\} = \{y \leq_a \dots\}) = (x = y)$

$\langle proof \rangle$

lemma *atLeastLessThan-eq-atLeastAtMost-diff*: $\{a \leq_a \dots <_a b\} = \{a \leq_a \dots \leq_a b\} - \{b\}$

$\langle proof \rangle$

lemma *greaterThanAtMost-eq-atLeastAtMost-diff*: $\{a <_a \dots \leq_a b\} = \{a \leq_a \dots \leq_a b\} - \{a\}$

$\langle proof \rangle$

lemma *atLeastAtMost-singleton[simp]*: $\{a \leq_a \dots \leq_a a\} = \{a\}$

$\langle proof \rangle$

lemma *atLeastAtMost-singleton'*:

assumes $a = b$

shows $\{a \leq_a \dots \leq_a b\} = \{a\}$

$\langle proof \rangle$

lemma *Icc-eq-Icc[simp]*:

$\{l \leq_a \dots \leq_a h\} = \{l' \leq_a \dots \leq_a h'\} = (l = l' \wedge h = h' \vee \neg l \leq_a h \wedge \neg l' \leq_a h')$

$\langle proof \rangle$

lemma *atLeastAtMost-singleton-iff[simp]*: $\{a \leq_a \dots \leq_a b\} = \{c\} \longleftrightarrow a = b \wedge b = c$

$\langle proof \rangle$

end

context *order-extremum*

begin

interpretation *ord-syntax* $\langle proof \rangle$

lemma *atMost-eq-UNIV-iff*: $\{\dots \leq_a x\} = UNIV \longleftrightarrow x = extremum$

$\langle proof \rangle$

end

context *no-extremum*

begin

interpretation *ord-syntax* $\langle proof \rangle$

interpretation *order-dual le ls*

$\langle proof \rangle$

lemma *not-UNIV-le-Icc[simp]*: $\neg UNIV \subseteq \{l \leq_a \dots \leq_a h\}$

$\langle proof \rangle$

lemma *not-UNIV-le-Iic[simp]*: $\neg UNIV \subseteq \{\dots \leq_a h\}$

$\langle proof \rangle$

lemma *not-Ici-le-Icc[simp]*: $\neg \{l \leq_a \dots\} \subseteq \{l' \leq_a \dots \leq_a h'\}$

$\langle proof \rangle$

lemma *not-Ici-le-Iic[simp]*: $\neg \{l \leq_a \dots\} \subseteq \{\dots \leq_a h'\}$

$\langle proof \rangle$

lemma *not-UNIV-eq-Icc*[simp]: $UNIV \neq \{l' \leq_a .. \leq_a h'\}$
 ⟨proof⟩

lemmas *not-Icc-eq-UNIV*[simp] = *not-UNIV-eq-Icc*[symmetric]

lemma *not-UNIV-eq-Iic*[simp]: $UNIV \neq \{.. \leq_a h'\}$
 ⟨proof⟩

lemmas *not-Iic-eq-UNIV*[simp] = *not-UNIV-eq-Iic*[symmetric]

lemma *not-Icc-eq-Ici*[simp]: $\{l \leq_a .. \leq_a h\} \neq \{l' \leq_a ..\}$
 ⟨proof⟩

lemmas *not-Ici-eq-Icc*[simp] = *not-Icc-eq-Ici*[symmetric]

lemma *not-Iic-eq-Ici*[simp]: $\{.. \leq_a h\} \neq \{l' \leq_a ..\}$
 ⟨proof⟩

lemmas *not-Ici-eq-Iic*[simp] = *not-Iic-eq-Ici*[symmetric]

lemma *greaterThan-non-empty*[simp]: $\{x <_a ..\} \neq \{\}$
 ⟨proof⟩

end

context *order*
begin

interpretation *ord-syntax* ⟨proof⟩

interpretation *order-pair* *le ls le ls* ⟨proof⟩

interpretation *ord-pair-syntax* *le ls le ls* ⟨proof⟩

lemma *mono-image-least*:

assumes *f-mono*: $mono_{ab} f$

and *f-img*: $f \cdot \{m \leq_a .. <_a n\} = \{m' \leq_a .. <_a n'\}$

and $m <_a n$

shows $f m = m'$

⟨proof⟩

end

5.4.13 Bounded sets

definition *bdd* :: $['a \text{ set}, ['a, 'a] \Rightarrow \text{bool}, 'a \text{ set}] \Rightarrow \text{bool}$
 (⟨(on - with - : «bdd» -)⟩ [1000, 1000, 1000] 10)
where $bdd U \text{ op } A \longleftrightarrow (\exists M \in U. \forall x \in A. \text{op } x M)$

ctr *parametricity*
in *bdd-def*

context *ord-syntax*
begin

abbreviation *bdd-above* **where** $bdd\text{-above} \equiv bdd \text{ UNIV } (\leq_a)$

abbreviation *bdd-below* **where** $bdd\text{-below} \equiv bdd \text{ UNIV } (\geq_a)$

end

context *preorder*

begin

interpretation *ord-syntax* $\langle proof \rangle$

interpretation *preorder-dual* $\langle proof \rangle$

lemma *bdd-aboveI[intro]*:

assumes $\bigwedge x. x \in A \implies x \leq_a M$

shows *bdd-above* A

$\langle proof \rangle$

lemma *bdd-belowI[intro]*:

assumes $\bigwedge x. x \in A \implies m \leq_a x$

shows *bdd-below* A

$\langle proof \rangle$

lemma *bdd-aboveI2*:

assumes $\bigwedge x. x \in A \implies f x \leq_a M$

shows *bdd-above* $(f \text{ ` } A)$

$\langle proof \rangle$

lemma *bdd-belowI2*:

assumes $\bigwedge x. x \in A \implies m \leq_a f x$

shows *bdd-below* $(f \text{ ` } A)$

$\langle proof \rangle$

lemma *bdd-above-empty[simp, intro]*: *bdd-above* $\{\}$

$\langle proof \rangle$

lemma *bdd-below-empty[simp, intro]*: *bdd-below* $\{\}$

$\langle proof \rangle$

lemma *bdd-above-mono*:

assumes *bdd-above* B **and** $A \subseteq B$

shows *bdd-above* A

$\langle proof \rangle$

lemma *bdd-below-mono*:

assumes *bdd-below* B **and** $A \subseteq B$

shows *bdd-below* A

$\langle proof \rangle$

lemma *bdd-above-Int1[simp]*:

assumes *bdd-above* A

shows *bdd-above* $(A \cap B)$

$\langle proof \rangle$

lemma *bdd-above-Int2[simp]*:

assumes *bdd-above* B

shows *bdd-above* $(A \cap B)$

$\langle proof \rangle$

lemma *bdd-below-Int1[simp]*:

assumes *bdd-below* A

shows *bdd-below* $(A \cap B)$

$\langle proof \rangle$

lemma *bdd-below-Int2[simp]*:

assumes *bdd-below B*

shows *bdd-below (A ∩ B)*

<proof>

lemma *bdd-above-Ioo[simp, intro]*: *bdd-above {a <_a .. <_a b}*

<proof>

lemma *bdd-above-Ico[simp, intro]*: *bdd-above {a ≤_a .. <_a b}*

<proof>

lemma *bdd-above-Iio[simp, intro]*: *bdd-above {.. <_a b}*

<proof>

lemma *bdd-above-Ioc[simp, intro]*: *bdd-above {a <_a .. ≤_a b}* *<proof>*

lemma *bdd-above-Icc[simp, intro]*: *bdd-above {a ≤_a .. ≤_a b}*

<proof>

lemma *bdd-above-Iic[simp, intro]*: *bdd-above {.. ≤_a b}*

<proof>

lemma *bdd-below-Ioo[simp, intro]*: *bdd-below {a <_a .. <_a b}*

<proof>

lemma *bdd-below-Ioc[simp, intro]*: *bdd-below {a <_a .. ≤_a b}*

<proof>

lemma *bdd-below-Ioi[simp, intro]*: *bdd-below {a <_a ..}*

<proof>

lemma *bdd-below-Ico[simp, intro]*: *bdd-below {a ≤_a .. <_a b}* *<proof>*

lemma *bdd-below-Icc[simp, intro]*: *bdd-below {a ≤_a .. ≤_a b}* *<proof>*

lemma *bdd-below-Ici[simp, intro]*: *bdd-below {a ≤_a ..}*

<proof>

end

context *order-pair*

begin

interpretation *ord-pair-syntax* *<proof>*

lemma *bdd-above-image-mono*:

assumes *mono_{a b} f* **and** *ord_a.bdd-above A*

shows *ord_b.bdd-above (f ‘ A)*

<proof>

lemma *bdd-below-image-mono*:

assumes *mono_{a b} f* **and** *ord_a.bdd-below A*

shows *ord_b.bdd-below (f ‘ A)*

<proof>

lemma *bdd-above-image-antimono*:

assumes *antimono_{a b} f* **and** *ord_a.bdd-below A*

shows *ord_b.bdd-above (f ‘ A)*

<proof>

lemma *bdd-below-image-antimono*:
 assumes *antimono_{a b} f* **and** *ord_a.bdd-above A*
 shows *ord_b.bdd-below (f ‘ A)*
 <proof>

end

context *order-extremum*
begin

interpretation *ord-syntax* *<proof>*

interpretation *order-dual* *<proof>*

lemma *bdd-above-top[simp, intro!]*: *bdd-above A*
 <proof>

end

5.5 Abstract orders on explicit sets

5.5.1 Background

Some of the results presented in this section were ported (with amendments and additions) from the theories *Orderings* and *Set-Interval* in the main library of Isabelle/HOL.

5.5.2 Order operations

locale *ord-ow* =

fixes $U :: 'a \text{ set}$ **and** $le\ ls :: ['a, 'a] \Rightarrow \text{bool}$

begin

tts-register-sbts $le \mid U$

<proof>

tts-register-sbts $ls \mid U$

<proof>

end

locale *ord-syntax-ow* = *ord-ow* $U\ le\ ls$

for $U :: 'a \text{ set}$ **and** $le\ ls :: ['a, 'a] \Rightarrow \text{bool}$

begin

notation

le ($\langle'(\leq_a)'\rangle$) **and**

le (**infix** $\langle\leq_a\rangle\ 50$) **and**

ls ($\langle'(<_a)'\rangle$) **and**

ls (**infix** $\langle<_a\rangle\ 50$)

abbreviation (*input*) ge (**infix** $\langle\geq_a\rangle\ 50$)

where $x \geq_a y \equiv y \leq_a x$

abbreviation (*input*) gt (**infix** $\langle>_a\rangle\ 50$)

where $x >_a y \equiv y <_a x$

notation

ge ($\langle'(\geq_a)'\rangle$) **and**

ge (**infix** $\langle\geq_a\rangle\ 50$) **and**

gt ($\langle'(>_a)'\rangle$) **and**

gt (**infix** $\langle>_a\rangle\ 50$)

abbreviation *Least* **where** $Least \equiv \text{Type-Simple-Orders.Least } U (\leq_a)$

abbreviation *Greatest* **where** $Greatest \equiv \text{Type-Simple-Orders.Least } U (\geq_a)$

abbreviation *min* **where** $min \equiv \text{Type-Simple-Orders.min } (\leq_a)$

abbreviation *max* **where** $max \equiv \text{Type-Simple-Orders.min } (\geq_a)$

abbreviation *lessThan* ($\langle\{\cdot\langle\cdot\langle\cdot\rangle\}\rangle$) **where** $\{\cdot\langle\cdot\langle\cdot\rangle\} \equiv \text{on } U \text{ with } (\langle\cdot\rangle) : \{\cdot\langle\cdot\rangle\}$

abbreviation *atMost* ($\langle\{\cdot\langle\cdot\langle\cdot\rangle\}\rangle$) **where** $\{\cdot\langle\cdot\langle\cdot\rangle\} \equiv \text{on } U \text{ with } (\leq_a) : \{\cdot\langle\cdot\rangle\}$

abbreviation *greaterThan* ($\langle\{\langle\cdot\langle\cdot\rangle\}\rangle$) **where** $\{\langle\cdot\langle\cdot\rangle\} \equiv \text{on } U \text{ with } (\rangle_a) : \{\langle\cdot\rangle\}$

abbreviation *atLeast* ($\langle\{\langle\cdot\langle\cdot\rangle\}\rangle$) **where** $\{\langle\cdot\langle\cdot\rangle\} \equiv \text{on } U \text{ with } (\geq_a) : \{\langle\cdot\rangle\}$

abbreviation *greaterThanLessThan* ($\langle\{\langle\cdot\langle\cdot\rangle\langle\cdot\rangle\}\rangle$)

where $\{\langle\cdot\langle\cdot\rangle\langle\cdot\rangle\} \equiv \text{on } U \text{ with } (\langle\cdot\rangle) (\langle\cdot\rangle) : \{\langle\cdot\rangle\langle\cdot\rangle\}$

abbreviation *atLeastLessThan* ($\langle\{\langle\cdot\langle\cdot\rangle\langle\cdot\rangle\}\rangle$)

where $\{\langle\cdot\langle\cdot\rangle\langle\cdot\rangle\} \equiv \text{on } U \text{ with } (\leq_a) (\langle\cdot\rangle) : \{\langle\cdot\rangle\langle\cdot\rangle\}$

abbreviation *greaterThanAtMost* ($\langle\{\langle\cdot\langle\cdot\rangle\langle\cdot\rangle\}\rangle$)

where $\{\langle\cdot\langle\cdot\rangle\langle\cdot\rangle\} \equiv \text{on } U \text{ with } (\langle\cdot\rangle) (\leq_a) : \{\langle\cdot\rangle\langle\cdot\rangle\}$

abbreviation *atLeastAtMost* ($\langle \{ \leq_a \dots \leq_a \} \rangle$)
where $\{ \leq_a \dots \leq_a u \} \equiv$ on U with (\leq_a) $(\leq_a) : \{ l \sqsubset \dots \sqsubset u \}$
abbreviation *lessThanGreaterThan* ($\langle \{ >_a \dots >_a \} \rangle$)
where $\{ >_a \dots >_a u \} \equiv$ on U with $(>_a)$ $(>_a) : \{ l \sqsubset \dots \sqsubset u \}$
abbreviation *lessThanAtLeast* ($\langle \{ \geq_a \dots \geq_a \} \rangle$)
where $\{ \geq_a \dots \geq_a u \} \equiv$ on U with (\geq_a) $(>_a) : \{ l \sqsubset \dots \sqsubset u \}$
abbreviation *atMostGreaterThan* ($\langle \{ >_a \dots \geq_a \} \rangle$)
where $\{ >_a \dots \geq_a u \} \equiv$ on U with $(>_a)$ $(\geq_a) : \{ l \sqsubset \dots \sqsubset u \}$
abbreviation *atMostAtLeast* ($\langle \{ \geq_a \dots \geq_a \} \rangle$)
where $\{ \geq_a \dots \geq_a u \} \equiv$ on U with (\geq_a) $(\geq_a) : \{ l \sqsubset \dots \sqsubset u \}$

abbreviation *bdd-above* **where** *bdd-above* \equiv *bdd* U (\leq_a)
abbreviation *bdd-below* **where** *bdd-below* \equiv *bdd* U (\geq_a)

end

locale *ord-dual-ow* = *ord-syntax-ow* U *le* *ls*
for $U :: 'a$ set **and** le $ls :: ['a, 'a] \Rightarrow bool$
begin

sublocale *dual*: *ord-ow* U *ge* *gt* $\langle proof \rangle$

end

locale *ord-pair-ow* = *ord_a*: *ord-ow* U_a *le_a* *ls_a* + *ord_b*: *ord-ow* U_b *le_b* *ls_b*
for $U_a :: 'a$ set **and** le_a ls_a **and** $U_b :: 'b$ set **and** le_b ls_b
begin

sublocale *rev*: *ord-pair-ow* U_b *le_b* *ls_b* U_a *le_a* *ls_a* $\langle proof \rangle$

end

locale *ord-pair-syntax-ow* = *ord-pair-ow* U_a *le_a* *ls_a* U_b *le_b* *ls_b*
for $U_a :: 'a$ set **and** le_a ls_a **and** $U_b :: 'b$ set **and** le_b ls_b
begin

sublocale *ord_a*: *ord-syntax-ow* U_a *le_a* *ls_a* + *ord_b*: *ord-syntax-ow* U_b *le_b* *ls_b* $\langle proof \rangle$

notation *le_a* ($\langle '(\leq_a) \rangle$)
and *le_a* (**infix** $\langle \leq_a \rangle$ 50)
and *ls_a* ($\langle '(<_a) \rangle$)
and *ls_a* (**infix** $\langle <_a \rangle$ 50)
and *le_b* ($\langle '(\leq_b) \rangle$)
and *le_b* (**infix** $\langle \leq_b \rangle$ 50)
and *ls_b* ($\langle '(<_b) \rangle$)
and *ls_b* (**infix** $\langle <_b \rangle$ 50)

notation *ord_a.ge* ($\langle '(\geq_a) \rangle$)
and *ord_a.ge* (**infix** $\langle \geq_a \rangle$ 50)
and *ord_a.gt* ($\langle '(>_a) \rangle$)
and *ord_a.gt* (**infix** $\langle >_a \rangle$ 50)
and *ord_b.ge* ($\langle '(\geq_b) \rangle$)
and *ord_b.ge* (**infix** $\langle \geq_b \rangle$ 50)
and *ord_b.gt* ($\langle '(>_b) \rangle$)
and *ord_b.gt* (**infix** $\langle >_b \rangle$ 50)

abbreviation *mono_{ab}*
where *mono_{ab}* \equiv *Type-Simple-Orders.mono* U_a (\leq_a) (\leq_b)

abbreviation $mono_{b_a}$

where $mono_{b_a} \equiv Type\text{-Simple-Orders.mono } U_b (\leq_b) (\leq_a)$

abbreviation $antimono_{a_b}$

where $antimono_{a_b} \equiv Type\text{-Simple-Orders.mono } U_a (\leq_a) (\geq_b)$

abbreviation $antimono_{b_a}$

where $antimono_{b_a} \equiv Type\text{-Simple-Orders.mono } U_b (\leq_b) (\geq_a)$

abbreviation $strict\text{-mono}_{a_b}$

where $strict\text{-mono}_{a_b} \equiv Type\text{-Simple-Orders.mono } U_a (<_a) (<_b)$

abbreviation $strict\text{-mono}_{b_a}$

where $strict\text{-mono}_{b_a} \equiv Type\text{-Simple-Orders.mono } U_b (<_b) (<_a)$

abbreviation $strict\text{-antimono}_{a_b}$

where $strict\text{-antimono}_{a_b} \equiv Type\text{-Simple-Orders.mono } U_a (<_a) (>_b)$

abbreviation $strict\text{-antimono}_{b_a}$

where $strict\text{-antimono}_{b_a} \equiv Type\text{-Simple-Orders.mono } U_b (<_b) (>_a)$

notation $ord_a.lessThan (\langle \{..<_a-\} \rangle)$

and $ord_a.atMost (\langle \{.. \leq_a-\} \rangle)$

and $ord_a.greaterThan (\langle \{<_a..\} \rangle)$

and $ord_a.atLeast (\langle \{-\leq_a..\} \rangle)$

and $ord_a.greaterThanLessThan (\langle \{<_a..\<_a-\} \rangle)$

and $ord_a.atLeastLessThan (\langle \{-\leq_a..\<_a-\} \rangle)$

and $ord_a.greaterThanAtMost (\langle \{<_a..\leq_a-\} \rangle)$

and $ord_a.atLeastAtMost (\langle \{-\leq_a..\leq_a-\} \rangle)$

and $ord_a.lessThanGreaterThan (\langle \{>_a..\>_a-\} \rangle)$

and $ord_a.lessThanAtLeast (\langle \{-\geq_a..\>_a-\} \rangle)$

and $ord_a.atMostGreaterThan (\langle \{>_a..\geq_a-\} \rangle)$

and $ord_a.atMostAtLeast (\langle \{-\geq_a..\geq_a-\} \rangle)$

and $ord_b.lessThan (\langle \{..<_b-\} \rangle)$

and $ord_b.atMost (\langle \{.. \leq_b-\} \rangle)$

and $ord_b.greaterThan (\langle \{<_b..\} \rangle)$

and $ord_b.atLeast (\langle \{-\leq_b..\} \rangle)$

and $ord_b.greaterThanLessThan (\langle \{<_b..\<_b-\} \rangle)$

and $ord_b.atLeastLessThan (\langle \{-\leq_b..\<_b-\} \rangle)$

and $ord_b.greaterThanAtMost (\langle \{<_b..\leq_b-\} \rangle)$

and $ord_b.atLeastAtMost (\langle \{-\leq_b..\leq_b-\} \rangle)$

and $ord_b.lessThanGreaterThan (\langle \{>_b..\>_b-\} \rangle)$

and $ord_b.lessThanAtLeast (\langle \{-\geq_b..\>_b-\} \rangle)$

and $ord_b.atMostGreaterThan (\langle \{>_b..\geq_b-\} \rangle)$

and $ord_b.atMostAtLeast (\langle \{-\geq_b..\geq_b-\} \rangle)$

end

locale $ord\text{-pair-dual-ow} = ord\text{-pair-syntax-ow } U_a le_a ls_a U_b le_b ls_b$

for $U_a :: 'a$ set and $le_a ls_a$ and $U_b :: 'b$ set and $le_b ls_b$

context $ord\text{-pair-dual-ow}$

begin

sublocale $ord\text{-dual}: ord\text{-pair-ow } U_a \langle (\leq_a) \rangle \langle (<_a) \rangle U_b \langle (\geq_b) \rangle \langle (>_b) \rangle \langle proof \rangle$

sublocale $dual\text{-ord}: ord\text{-pair-ow } U_a \langle (\geq_a) \rangle \langle (>_a) \rangle U_b \langle (\leq_b) \rangle \langle (<_b) \rangle \langle proof \rangle$

sublocale $dual\text{-dual}: ord\text{-pair-ow } U_a \langle (\geq_a) \rangle \langle (>_a) \rangle U_b \langle (\geq_b) \rangle \langle (>_b) \rangle \langle proof \rangle$

end

Relativization

context $ord\text{-ow}$

begin

interpretation *ord-syntax-ow* $\langle proof \rangle$

tts-context

tts: ($?a$ to U)

sbterms: ($\langle ?ls::?a \Rightarrow ?a \Rightarrow bool \rangle$ to ls)

rewriting *ctr-simps*

eliminating through *auto*

begin

tts-lemma *lessThan-iff:*

assumes $i \in U$ **and** $k \in U$

shows $(i \in \{..<_a k\}) = (i <_a k)$

is *ord.lessThan-iff* $\langle proof \rangle$

tts-lemma *greaterThanLessThan-iff:*

assumes $i \in U$ **and** $l \in U$ **and** $u \in U$

shows $(i \in \{l <_a .. <_a u\}) = (i <_a u \wedge l <_a i)$

is *ord.greaterThanLessThan-iff* $\langle proof \rangle$

tts-lemma *greaterThanLessThan-eq:*

assumes $a \in U$ **and** $b \in U$

shows $\{a <_a .. <_a b\} = \{a <_a ..\} \cap \{.. <_a b\}$

is *ord.greaterThanLessThan-eq* $\langle proof \rangle$

end

tts-context

tts: ($?a$ to U)

sbterms: ($\langle ?le::?a \Rightarrow ?a \Rightarrow bool \rangle$ to le)

rewriting *ctr-simps*

eliminating through *auto*

begin

tts-lemma *min-absorb1:*

assumes $x \in U$ **and** $y \in U$ **and** $x \leq_a y$

shows $\min x y = x$

is *ord.min-absorb1* $\langle proof \rangle$

tts-lemma *atLeast-iff:*

assumes $i \in U$ **and** $k \in U$

shows $(i \in \{k \leq_a ..\}) = (k \leq_a i)$

is *ord.atLeast-iff* $\langle proof \rangle$

tts-lemma *atLeastAtMost-iff:*

assumes $i \in U$ **and** $l \in U$ **and** $u \in U$

shows $(i \in \{l \leq_a .. \leq_a u\}) = (i \leq_a u \wedge l \leq_a i)$

is *ord.atLeastAtMost-iff* $\langle proof \rangle$

end

tts-context

tts: ($?a$ to U)

sbterms: ($\langle ?le::?a \Rightarrow ?a \Rightarrow bool \rangle$ to le)

and ($\langle ?ls::?a \Rightarrow ?a \Rightarrow bool \rangle$ to ls)

rewriting *ctr-simps*

eliminating through *auto*

begin

tts-lemma *atLeastLessThan-iff*:

assumes $i \in U$ and $l \in U$ and $u \in U$
 shows $(i \in \{l \leq_a \cdot <_a u\}) = (l \leq_a i \wedge i <_a u)$
 is *ord.atLeastLessThan-iff*{proof}

tts-lemma *greaterThanAtMost-iff*:

assumes $i \in U$ and $l \in U$ and $u \in U$
 shows $(i \in \{l <_a \cdot \leq_a u\}) = (i \leq_a u \wedge l <_a i)$
 is *ord.greaterThanAtMost-iff*{proof}

end

end

context *ord-pair-ow*

begin

interpretation *ord-pair-syntax-ow* {proof}

tts-context

tts: ($?a$ to U_a) and ($?b$ to U_b)
 sbterms: ($\langle ?le_a :: ?'a \Rightarrow ?'a \Rightarrow bool \rangle$ to le_a) and ($\langle ?le_b :: ?'b \Rightarrow ?'b \Rightarrow bool \rangle$ to le_b)
 rewriting *ctr-simps*
 eliminating through (*simp add: Type-Simple-Orders.mono-def*)

begin

tts-lemma *monoD*:

assumes $x \in U_a$ and $y \in U_a$ and $mono_{ab} f$ and $x \leq_a y$
 shows $f x \leq_b f y$
 is *ord-pair.monoD*{proof}

tts-lemma *monoI*:

assumes $\bigwedge x y. \llbracket x \in U_a; y \in U_a; x \leq_a y \rrbracket \Longrightarrow f x \leq_b f y$
 shows $mono_{ab} f$
 is *ord-pair.monoI*{proof}

tts-lemma *monoE*:

assumes $x \in U_a$
 and $y \in U_a$
 and $mono_{ab} f$
 and $x \leq_a y$
 and $f x \leq_b f y \Longrightarrow thesis$
 shows *thesis*
 is *ord-pair.monoE*{proof}

end

tts-context

tts: ($?a$ to U_a) and ($?b$ to U_b)
 sbterms: ($\langle ?ls_a :: ?'a \Rightarrow ?'a \Rightarrow bool \rangle$ to le_a) and ($\langle ?ls_b :: ?'b \Rightarrow ?'b \Rightarrow bool \rangle$ to le_b)
 rewriting *ctr-simps*
 eliminating through (*simp add: Type-Simple-Orders.mono-def*)

begin

tts-lemma *strict-monoD*:

assumes $x \in U_a$
 and $y \in U_a$

and $mono_{ab} f$
and $x \leq_a y$
shows $f x \leq_b f y$
is $ord\text{-}pair.strict\text{-}monoD\langle proof \rangle$

tts-lemma $strict\text{-}monoI$:
assumes $\bigwedge x y. \llbracket x \in U_a; y \in U_a; x \leq_a y \rrbracket \implies f x \leq_b f y$
shows $mono_{ab} f$
is $ord\text{-}pair.strict\text{-}monoI\langle proof \rangle$

tts-lemma $strict\text{-}monoE$:
assumes $x \in U_a$
and $y \in U_a$
and $mono_{ab} f$
and $x \leq_a y$
and $f x \leq_b f y \implies thesis$
shows $thesis$
is $ord\text{-}pair.strict\text{-}monoE\langle proof \rangle$

end

end

5.5.3 Preorders

Definitions and common properties

locale $preorder\text{-}ow = ord\text{-}ow U le ls$
for $U :: 'a set$ **and** $le ls +$
assumes $less\text{-}le\text{-}not\text{-}le: \llbracket x \in U; y \in U \rrbracket \implies ls x y \longleftrightarrow le x y \wedge \neg (le y x)$
and $order\text{-}refl[iff]: x \in U \implies le x x$
and $order\text{-}trans: \llbracket x \in U; y \in U; z \in U; le x y; le y z \rrbracket \implies le x z$

locale $preorder\text{-}dual\text{-}ow = preorder\text{-}ow U le ls$ **for** $U :: 'a set$ **and** $le ls$
begin

sublocale $ord\text{-}dual\text{-}ow \langle proof \rangle$

sublocale $dual: preorder\text{-}ow U ge gt$
 $\langle proof \rangle$

end

locale $ord\text{-}preorder\text{-}ow =$
 $ord\text{-}pair\text{-}ow U_a le_a ls_a U_b le_b ls_b + ord_b: preorder\text{-}ow U_b le_b ls_b$
for $U_a :: 'a set$ **and** $le_a ls_a$ **and** $U_b :: 'b set$ **and** $le_b ls_b$

locale $ord\text{-}preorder\text{-}dual\text{-}ow = ord\text{-}preorder\text{-}ow U_a le_a ls_a U_b le_b ls_b$
for $U_a :: 'a set$ **and** $le_a ls_a$ **and** $U_b :: 'b set$ **and** $le_b ls_b$
begin

sublocale $ord\text{-}pair\text{-}dual\text{-}ow \langle proof \rangle$

sublocale $ord\text{-}dual: ord\text{-}preorder\text{-}ow U_a \langle (\leq_a) \rangle \langle (<_a) \rangle U_b \langle (\geq_b) \rangle \langle (>_b) \rangle$
 $\langle proof \rangle$

sublocale $dual\text{-}ord: ord\text{-}preorder\text{-}ow U_a \langle (\geq_a) \rangle \langle (>_a) \rangle U_b \langle (\leq_b) \rangle \langle (<_b) \rangle$
 $\langle proof \rangle$

sublocale $dual\text{-}dual: ord\text{-}preorder\text{-}ow U_a \langle (\geq_a) \rangle \langle (>_a) \rangle U_b \langle (\geq_b) \rangle \langle (>_b) \rangle$
 $\langle proof \rangle$

end

locale *preorder-pair-ow* =
 ord-preorder-ow U_a le_a ls_a U_b le_b ls_b + ord_a: preorder-ow U_a le_a ls_a
 for U_a :: 'a set and le_a ls_a and U_b :: 'b set and le_b ls_b
begin

sublocale *rev*: preorder-pair-ow U_b le_b ls_b U_a le_a ls_a *<proof>*

end

locale *preorder-pair-dual-ow* = preorder-pair-ow U_a le_a ls_a U_b le_b ls_b
 for U_a :: 'a set and le_a ls_a and U_b :: 'b set and le_b ls_b
begin

sublocale *ord-preorder-dual-ow* *<proof>*

sublocale *ord-dual*: preorder-pair-ow U_a $\langle(\leq_a)\rangle$ $\langle(<_a)\rangle$ U_b $\langle(\geq_b)\rangle$ $\langle(>_b)\rangle$ *<proof>*

sublocale *dual-ord*: preorder-pair-ow U_a $\langle(\geq_a)\rangle$ $\langle(>_a)\rangle$ U_b $\langle(\leq_b)\rangle$ $\langle(<_b)\rangle$
<proof>

sublocale *dual-dual*: preorder-pair-ow U_a $\langle(\geq_a)\rangle$ $\langle(>_a)\rangle$ U_b $\langle(\geq_b)\rangle$ $\langle(>_b)\rangle$ *<proof>*

end

Transfer rules

lemma *preorder-ow*[*ud-with*]: preorder = preorder-ow UNIV
<proof>

lemma *ord-preorder-ow*[*ud-with*]: ord-preorder = ord-preorder-ow UNIV
<proof>

lemma *preorder-pair-ow*[*ud-with*]:
 preorder-pair =
 (λle_a ls_a le_b ls_b . preorder-pair-ow UNIV le_a ls_a UNIV le_b ls_b)
<proof>

context

includes *lifting-syntax*

begin

lemma *preorder-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: right-total A

shows

(rel-set A \implies ($A \implies A \implies (=)$) \implies ($A \implies A \implies (=)$) \implies (=))
 preorder-ow preorder-ow

<proof>

lemma *ord-preorder-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: right-total A

shows

(rel-set A \implies ($A \implies A \implies (=)$) \implies ($A \implies A \implies (=)$) \implies (=))
 ord-preorder-ow ord-preorder-ow

<proof>

lemma *preorder-pair-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: right-total A right-total B

shows

```

(
  rel-set A ==> (A ==> A ==> (=)) ==> (A ==> A ==> (=)) ==>
  rel-set B ==> (B ==> B ==> (=)) ==> (B ==> B ==> (=)) ==>
  (=)
) preorder-pair-ow preorder-pair-ow
⟨proof⟩

```

end

Relativization

context *preorder-ow*

begin

interpretation *ord-syntax-ow* ⟨proof⟩

tts-context

tts: (*?a to U*)

sbterms: (*⟨?ls::?a ⇒ ?a ⇒ bool⟩ to ls*)

and (*⟨?le::?a ⇒ ?a ⇒ bool⟩ to le*)

rewriting *ctr-simps*

substituting *preorder-ow-axioms*

eliminating through *auto*

begin

tts-lemma *less-irrefl*:

assumes $x \in U$

shows $\neg x <_a x$

is *preorder.less-irrefl*⟨proof⟩

tts-lemma *eq-refl*:

assumes $y \in U$ and $x = y$

shows $x \leq_a y$

is *preorder.eq-refl*⟨proof⟩

tts-lemma *less-imp-le*:

assumes $x \in U$ and $y \in U$ and $x <_a y$

shows $x \leq_a y$

is *preorder.less-imp-le*⟨proof⟩

tts-lemma *strict-implies-not-eq*:

assumes $b \in U$ and $a <_a b$

shows $a \neq b$

is *preorder.strict-implies-not-eq*⟨proof⟩

tts-lemma *less-not-sym*:

assumes $x \in U$ and $y \in U$ and $x <_a y$

shows $\neg y <_a x$

is *preorder.less-not-sym*⟨proof⟩

tts-lemma *not-empty-eq-Ici-eq-empty*:

assumes $l \in U$

shows $\{ \} \neq \{ l \leq_a \cdot \}$

is *preorder.not-empty-eq-Ici-eq-empty*⟨proof⟩

tts-lemma *not-Ici-eq-empty*:

assumes $l \in U$

shows $\{ l \leq_a \cdot \} \neq \{ \}$

is *preorder.not-Ici-eq-empty*(proof)

tts-lemma *asym*:

assumes $a \in U$ and $b \in U$ and $a <_a b$ and $b <_a a$

shows *False*

is *preorder.asym*(proof)

tts-lemma *less-asym'*:

assumes $a \in U$ and $b \in U$ and $a <_a b$ and $b <_a a$

shows *P*

is *preorder.less-asym'*(proof)

tts-lemma *less-imp-not-less*:

assumes $x \in U$ and $y \in U$ and $x <_a y$

shows $(\neg y <_a x) = \text{True}$

is *preorder.less-imp-not-less*(proof)

tts-lemma *single-Diff-lessThan*:

assumes $k \in U$

shows $\{k\} - \{..<_a k\} = \{k\}$

is *preorder.single-Diff-lessThan*(proof)

tts-lemma *less-imp-triv*:

assumes $x \in U$ and $y \in U$ and $x <_a y$

shows $(y <_a x \longrightarrow P) = \text{True}$

is *preorder.less-imp-triv*(proof)

tts-lemma *ivl-disj-int-one*:

assumes $l \in U$ and $u \in U$

shows

$\{.. \leq_a l\} \cap \{l <_{a..} <_a u\} = \{\}$

$\{.. <_a l\} \cap \{l \leq_{a..} <_a u\} = \{\}$

$\{.. \leq_a l\} \cap \{l <_{a..} \leq_a u\} = \{\}$

$\{.. <_a l\} \cap \{l \leq_{a..} \leq_a u\} = \{\}$

$\{l <_{a..} \leq_a u\} \cap \{u <_{a..}\} = \{\}$

$\{l <_{a..} <_a u\} \cap \{u \leq_{a..}\} = \{\}$

$\{l \leq_{a..} \leq_a u\} \cap \{u <_{a..}\} = \{\}$

$\{l \leq_{a..} <_a u\} \cap \{u \leq_{a..}\} = \{\}$

is *preorder.ivl-disj-int-one*(proof)

tts-lemma *atLeastatMost-empty-iff2*:

assumes $a \in U$ and $b \in U$

shows $(\{\} = \{a \leq_{a..} \leq_a b\}) = (\neg a \leq_a b)$

is *preorder.atLeastatMost-empty-iff2*(proof)

tts-lemma *atLeastLessThan-empty-iff2*:

assumes $a \in U$ and $b \in U$

shows $(\{\} = \{a \leq_{a..} <_a b\}) = (\neg a <_a b)$

is *preorder.atLeastLessThan-empty-iff2*(proof)

tts-lemma *greaterThanAtMost-empty-iff2*:

assumes $k \in U$ and $l \in U$

shows $(\{\} = \{k <_{a..} \leq_a l\}) = (\neg k <_a l)$

is *preorder.greaterThanAtMost-empty-iff2*(proof)

tts-lemma *atLeastatMost-empty-iff*:

assumes $a \in U$ and $b \in U$

shows $(\{a \leq_{a..} \leq_a b\} = \{\}) = (\neg a \leq_a b)$

is *preorder.atLeastatMost-empty-iff*(proof)

tts-lemma *atLeastLessThan-empty-iff*:

assumes $a \in U$ and $b \in U$

shows $(\{a \leq_a \dots <_a b\} = \{\}) = (\neg a <_a b)$

is *preorder.atLeastLessThan-empty-iff*(proof)

tts-lemma *greaterThanAtMost-empty-iff*:

assumes $k \in U$ and $l \in U$

shows $(\{k <_a \dots \leq_a l\} = \{\}) = (\neg k <_a l)$

is *preorder.greaterThanAtMost-empty-iff*(proof)

tts-lemma *atLeastLessThan-empty*:

assumes $b \in U$ and $a \in U$ and $b \leq_a a$

shows $\{a \leq_a \dots <_a b\} = \{\}$

is *preorder.atLeastLessThan-empty*(proof)

tts-lemma *greaterThanAtMost-empty*:

assumes $l \in U$ and $k \in U$ and $l \leq_a k$

shows $\{k <_a \dots \leq_a l\} = \{\}$

is *preorder.greaterThanAtMost-empty*(proof)

tts-lemma *greaterThanLessThan-empty*:

assumes $l \in U$ and $k \in U$ and $l \leq_a k$

shows $\{k <_a \dots <_a l\} = \{\}$

is *preorder.greaterThanLessThan-empty*(proof)

tts-lemma *le-less-trans*:

assumes $x \in U$ and $y \in U$ and $z \in U$ and $x \leq_a y$ and $y <_a z$

shows $x <_a z$

is *preorder.le-less-trans*(proof)

tts-lemma *atLeastatMost-empty*:

assumes $b \in U$ and $a \in U$ and $b <_a a$

shows $\{a \leq_a \dots \leq_a b\} = \{\}$

is *preorder.atLeastatMost-empty*(proof)

tts-lemma *less-le-trans*:

assumes $x \in U$ and $y \in U$ and $z \in U$ and $x <_a y$ and $y \leq_a z$

shows $x <_a z$

is *preorder.less-le-trans*(proof)

tts-lemma *less-trans*:

assumes $x \in U$ and $y \in U$ and $z \in U$ and $x <_a y$ and $y <_a z$

shows $x <_a z$

is *preorder.less-trans*(proof)

tts-lemma *ivl-disj-int-two*:

assumes $l \in U$ and $m \in U$ and $u \in U$

shows

$$\{l <_a \dots <_a m\} \cap \{m \leq_a \dots <_a u\} = \{\}$$

$$\{l <_a \dots \leq_a m\} \cap \{m <_a \dots <_a u\} = \{\}$$

$$\{l \leq_a \dots <_a m\} \cap \{m \leq_a \dots <_a u\} = \{\}$$

$$\{l \leq_a \dots \leq_a m\} \cap \{m <_a \dots <_a u\} = \{\}$$

$$\{l <_a \dots <_a m\} \cap \{m \leq_a \dots \leq_a u\} = \{\}$$

$$\{l <_a \dots \leq_a m\} \cap \{m <_a \dots \leq_a u\} = \{\}$$

$$\{l \leq_a \dots <_a m\} \cap \{m \leq_a \dots \leq_a u\} = \{\}$$

$$\{l \leq_a \dots \leq_a m\} \cap \{m <_a \dots \leq_a u\} = \{\}$$

is *preorder.ivl-disj-int-two*(proof)

tts-lemma *less-asm*:

assumes $x \in U$ and $y \in U$ and $x <_a y$ and $\neg P \implies y <_a x$

shows P

is *preorder.less-asm*(proof)

tts-lemma *Iio-Int-singleton*:

assumes $k \in U$ and $x \in U$

shows $\{..<_a k\} \cap \{x\} = (\text{if } x <_a k \text{ then } \{x\} \text{ else } \{\})$

is *preorder.Iio-Int-singleton*(proof)

tts-lemma *Ioi-le-Ico*:

assumes $a \in U$

shows $\{a <_{a..}\} \subseteq \{a \leq_{a..}\}$

is *preorder.Ioi-le-Ico*(proof)

tts-lemma *Icc-subset-Iic-iff*:

assumes $l \in U$ and $h \in U$ and $h' \in U$

shows $(\{l \leq_{a..} \leq_a h\} \subseteq \{.. \leq_a h'\}) = (\neg l \leq_a h \vee h \leq_a h')$

is *preorder.Icc-subset-Iic-iff*(proof)

tts-lemma *atLeast-subset-iff*:

assumes $x \in U$ and $y \in U$

shows $(\{x \leq_{a..}\} \subseteq \{y \leq_{a..}\}) = (y \leq_a x)$

is *preorder.atLeast-subset-iff*(proof)

tts-lemma *Icc-subset-Ici-iff*:

assumes $l \in U$ and $h \in U$ and $l' \in U$

shows $(\{l \leq_{a..} \leq_a h\} \subseteq \{l' \leq_{a..}\}) = (\neg l \leq_a h \vee l' \leq_a l)$

is *preorder.Icc-subset-Ici-iff*(proof)

tts-lemma *atLeastatMost-subset-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$

shows $(\{a \leq_{a..} \leq_a b\} \subseteq \{c \leq_{a..} \leq_a d\}) = (\neg a \leq_a b \vee b \leq_a d \wedge c \leq_a a)$

is *preorder.atLeastatMost-subset-iff*(proof)

tts-lemma *atLeastatMost-psubset-iff*:

assumes $a \in U$ and $b \in U$ and $c \in U$ and $d \in U$

shows $(\{a \leq_{a..} \leq_a b\} \subseteq \{c \leq_{a..} \leq_a d\}) =$

$(c \leq_a d \wedge (\neg a \leq_a b \vee c \leq_a a \wedge b \leq_a d \wedge (c <_a a \vee b <_a d)))$

is *preorder.atLeastatMost-psubset-iff*(proof)

tts-lemma *bdd-above-empty*:

assumes $U \neq \{\}$

shows *bdd-above* $\{\}$

is *preorder.bdd-above-empty*(proof)

tts-lemma *bdd-above-Iic*:

assumes $b \in U$

shows *bdd-above* $\{.. \leq_a b\}$

is *preorder.bdd-above-Iic*(proof)

tts-lemma *bdd-above-Iio*:

assumes $b \in U$

shows *bdd-above* $\{.. <_a b\}$

is *preorder.bdd-above-Iio*(proof)

tts-lemma *bdd-below-empty*:
assumes $U \neq \{\}$
shows *bdd-below* $\{\}$
is *preorder.bdd-below-empty*(*proof*)

tts-lemma *bdd-above-Icc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a \leq_a .. \leq_a b\}$
is *preorder.bdd-above-Icc*(*proof*)

tts-lemma *bdd-above-Ico*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a \leq_a .. <_a b\}$
is *preorder.bdd-above-Ico*(*proof*)

tts-lemma *bdd-above-Ioc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a <_a .. \leq_a b\}$
is *preorder.bdd-above-Ioc*(*proof*)

tts-lemma *bdd-above-Ioo*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a <_a .. <_a b\}$
is *preorder.bdd-above-Ioo*(*proof*)

tts-lemma *bdd-above-Int1*:
assumes $A \subseteq U$ **and** $B \subseteq U$ **and** *bdd-above* A
shows *bdd-above* $(A \cap B)$
is *preorder.bdd-above-Int1*(*proof*)

tts-lemma *bdd-above-Int2*:
assumes $B \subseteq U$ **and** $A \subseteq U$ **and** *bdd-above* B
shows *bdd-above* $(A \cap B)$
is *preorder.bdd-above-Int2*(*proof*)

tts-lemma *bdd-below-Icc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a \leq_a .. \leq_a b\}$
is *preorder.bdd-below-Icc*(*proof*)

tts-lemma *bdd-below-Ico*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a \leq_a .. <_a b\}$
is *preorder.bdd-below-Ico*(*proof*)

tts-lemma *bdd-below-Ioc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a <_a .. \leq_a b\}$
is *preorder.bdd-below-Ioc*(*proof*)

tts-lemma *bdd-below-Ioo*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a <_a .. <_a b\}$
is *preorder.bdd-below-Ioo*(*proof*)

tts-lemma *bdd-below-Ici*:
assumes $a \in U$
shows *bdd-below* $\{a \leq_a ..\}$

is *preorder.bdd-below-Ici*{proof}

tts-lemma *bdd-below-Ioi*:
assumes $a \in U$
shows *bdd-below* $\{a <_{a..}\}$
is *preorder.bdd-below-Ioi*{proof}

tts-lemma *bdd-above-mono*:
assumes $B \subseteq U$ **and** *bdd-above* B **and** $A \subseteq B$
shows *bdd-above* A
is *preorder.bdd-above-mono*{proof}

tts-lemma *bdd-aboveI*:
assumes $A \subseteq U$ **and** $M \in U$ **and** $\bigwedge x. \llbracket x \in U; x \in A \rrbracket \implies x \leq_a M$
shows *bdd-above* A
is *preorder.bdd-aboveI*{proof}

tts-lemma *bdd-aboveI2*:
assumes $\text{range } f \subseteq U$ **and** $M \in U$ **and** $\bigwedge x. x \in A \implies f x \leq_a M$
shows *bdd-above* $(f \text{ ` } A)$
is *preorder.bdd-aboveI2*{proof}

tts-lemma *bdd-below-Int1*:
assumes $A \subseteq U$ **and** $B \subseteq U$ **and** *bdd-below* A
shows *bdd-below* $(A \cap B)$
is *preorder.bdd-below-Int1*{proof}

tts-lemma *bdd-below-Int2*:
assumes $B \subseteq U$ **and** $A \subseteq U$ **and** *bdd-below* B
shows *bdd-below* $(A \cap B)$
is *preorder.bdd-below-Int2*{proof}

tts-lemma *bdd-belowI*:
assumes $A \subseteq U$ **and** $m \in U$ **and** $\bigwedge x. \llbracket x \in U; x \in A \rrbracket \implies m \leq_a x$
shows *bdd-below* A
is *preorder.bdd-belowI*{proof}

tts-lemma *bdd-below-mono*:
assumes $B \subseteq U$ **and** *bdd-below* B **and** $A \subseteq B$
shows *bdd-below* A
is *preorder.bdd-below-mono*{proof}

tts-lemma *bdd-belowI2*:
assumes $m \in U$ **and** $\text{range } f \subseteq U$ **and** $\bigwedge x. x \in A \implies m \leq_a f x$
shows *bdd-below* $(f \text{ ` } A)$
is *preorder.bdd-belowI2*{**where** 'b='d }{proof}

end

end

5.5.4 Partial orders

locale *order-ow* = *preorder-ow* U *le* ls
for $U :: \text{'a set}$ **and** le ls +
assumes *antisym*: $x \in U \implies y \in U \implies le\ x\ y \implies le\ y\ x \implies x = y$

locale *order-dual-ow* = *order-ow* U *le* ls

```

for  $U :: 'a \text{ set}$  and  $le \text{ } ls$ 
begin

sublocale preorder-dual-ow  $\langle \text{proof} \rangle$ 

sublocale dual: order-ow  $U \text{ } ge \text{ } gt$ 
 $\langle \text{proof} \rangle$ 

end

locale ord-order-ow =
  ord-preorder-ow  $U_a \text{ } le_a \text{ } ls_a \text{ } U_b \text{ } le_b \text{ } ls_b$  + ordb: order-ow  $U_b \text{ } le_b \text{ } ls_b$ 
  for  $U_a :: 'a \text{ set}$  and  $le_a \text{ } ls_a$  and  $U_b :: 'b \text{ set}$  and  $le_b \text{ } ls_b$ 

locale ord-order-dual-ow = ord-order-ow  $U_a \text{ } le_a \text{ } ls_a \text{ } U_b \text{ } le_b \text{ } ls_b$ 
  for  $U_a :: 'a \text{ set}$  and  $le_a \text{ } ls_a$  and  $U_b :: 'b \text{ set}$  and  $le_b \text{ } ls_b$ 
begin

sublocale ord-preorder-dual-ow  $\langle \text{proof} \rangle$ 
sublocale ord-dual: ord-order-ow  $U_a \langle (\leq_a) \rangle \langle (<_a) \rangle U_b \langle (\geq_b) \rangle \langle (>_b) \rangle$ 
 $\langle \text{proof} \rangle$ 
sublocale dual-ord: ord-order-ow  $U_a \langle (\geq_a) \rangle \langle (>_a) \rangle U_b \langle (\leq_b) \rangle \langle (<_b) \rangle$ 
 $\langle \text{proof} \rangle$ 
sublocale dual-dual: ord-order-ow  $U_a \langle (\geq_a) \rangle \langle (>_a) \rangle U_b \langle (\geq_b) \rangle \langle (>_b) \rangle$ 
 $\langle \text{proof} \rangle$ 

end

locale preorder-order-ow =
  ord-order-ow  $U_a \text{ } le_a \text{ } ls_a \text{ } U_b \text{ } le_b \text{ } ls_b$  + orda: preorder-ow  $U_a \text{ } le_a \text{ } ls_a$ 
  for  $U_a :: 'a \text{ set}$  and  $le_a \text{ } ls_a$  and  $U_b :: 'b \text{ set}$  and  $le_b \text{ } ls_b$ 
begin

sublocale preorder-pair-ow  $\langle \text{proof} \rangle$ 

end

locale preorder-order-dual-ow = preorder-order-ow  $U_a \text{ } le_a \text{ } ls_a \text{ } U_b \text{ } le_b \text{ } ls_b$ 
  for  $U_a :: 'a \text{ set}$  and  $le_a \text{ } ls_a$  and  $U_b :: 'b \text{ set}$  and  $le_b \text{ } ls_b$ 
begin

sublocale ord-order-dual-ow  $\langle \text{proof} \rangle$ 
sublocale preorder-pair-dual-ow  $\langle \text{proof} \rangle$ 
sublocale ord-dual: preorder-order-ow  $U_a \langle (\leq_a) \rangle \langle (<_a) \rangle U_b \langle (\geq_b) \rangle \langle (>_b) \rangle$   $\langle \text{proof} \rangle$ 
sublocale dual-ord: preorder-order-ow  $U_a \langle (\geq_a) \rangle \langle (>_a) \rangle U_b \langle (\leq_b) \rangle \langle (<_b) \rangle$   $\langle \text{proof} \rangle$ 
sublocale dual-dual: preorder-order-ow  $U_a \langle (\geq_a) \rangle \langle (>_a) \rangle U_b \langle (\geq_b) \rangle \langle (>_b) \rangle$   $\langle \text{proof} \rangle$ 

end

locale order-pair-ow =
  preorder-order-ow  $U_a \text{ } le_a \text{ } ls_a \text{ } U_b \text{ } le_b \text{ } ls_b$  + orda: order-ow  $U_a \text{ } le_a \text{ } ls_a$ 
  for  $U_a :: 'a \text{ set}$  and  $le_a \text{ } ls_a$  and  $U_b :: 'b \text{ set}$  and  $le_b \text{ } ls_b$ 
begin

sublocale rev: order-pair-ow  $U_b \text{ } le_b \text{ } ls_b \text{ } U_a \text{ } le_a \text{ } ls_a$   $\langle \text{proof} \rangle$ 

end

```

```

locale order-pair-dual-ow = order-pair-ow  $U_a$   $le_a$   $ls_a$   $U_b$   $le_b$   $ls_b$ 
  for  $U_a :: 'a$  set and  $le_a$   $ls_a$  and  $U_b :: 'b$  set and  $le_b$   $ls_b$ 
begin

sublocale preorder-order-dual-ow ⟨proof⟩
sublocale ord-dual: order-pair-ow  $U_a$  ⟨ $\leq_a$ ⟩ ⟨ $<_a$ ⟩  $U_b$  ⟨ $\geq_b$ ⟩ ⟨ $>_b$ ⟩ ⟨proof⟩
sublocale dual-ord: order-pair-ow  $U_a$  ⟨ $\geq_a$ ⟩ ⟨ $>_a$ ⟩  $U_b$  ⟨ $\leq_b$ ⟩ ⟨ $<_b$ ⟩
  ⟨proof⟩
sublocale dual-dual: order-pair-ow  $U_a$  ⟨ $\geq_a$ ⟩ ⟨ $>_a$ ⟩  $U_b$  ⟨ $\geq_b$ ⟩ ⟨ $>_b$ ⟩ ⟨proof⟩

end

```

Transfer rules

```

lemma order-ow[ud-with]: order = order-ow UNIV
  ⟨proof⟩

```

```

lemma ord-order-ow[ud-with]: ord-order = ord-order-ow UNIV
  ⟨proof⟩

```

```

lemma preorder-order-ow[ud-with]:
  preorder-order =
    ( $\lambda le_a$   $ls_a$   $le_b$   $ls_b$ . preorder-order-ow UNIV  $le_a$   $ls_a$  UNIV  $le_b$   $ls_b$ )
  ⟨proof⟩

```

```

lemma order-pair-ow[ud-with]:
  order-pair = ( $\lambda le_a$   $ls_a$   $le_b$   $ls_b$ . order-pair-ow UNIV  $le_a$   $ls_a$  UNIV  $le_b$   $ls_b$ )
  ⟨proof⟩

```

context

```

includes lifting-syntax
begin

```

```

lemma order-ow-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique  $A$  right-total  $A$ 
  shows
    (rel-set  $A$   $\implies$  ( $A$   $\implies$   $A$   $\implies$  (=))  $\implies$  ( $A$   $\implies$   $A$   $\implies$  (=))  $\implies$  (=))
    order-ow order-ow
  ⟨proof⟩

```

```

lemma ord-order-ow-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique  $A$  right-total  $A$ 
  shows
    (
      rel-set  $A$   $\implies$  ( $A$   $\implies$   $A$   $\implies$  (=))  $\implies$  ( $A$   $\implies$   $A$   $\implies$  (=))  $\implies$ 
      (=)
    ) ord-order-ow ord-order-ow
  ⟨proof⟩

```

```

lemma preorder-order-ow-transfer[transfer-rule]:
  assumes [transfer-rule]: right-total  $A$  bi-unique  $B$  right-total  $B$ 
  shows
    (
      rel-set  $A$   $\implies$  ( $A$   $\implies$   $A$   $\implies$  (=))  $\implies$  ( $A$   $\implies$   $A$   $\implies$  (=))  $\implies$ 
      rel-set  $B$   $\implies$  ( $B$   $\implies$   $B$   $\implies$  (=))  $\implies$  ( $B$   $\implies$   $B$   $\implies$  (=))  $\implies$ 
      (=)
    ) preorder-order-ow preorder-order-ow
  ⟨proof⟩

```

```

lemma order-pair-ow-transfer[transfer-rule]:
  assumes [transfer-rule]:
    bi-unique A right-total A bi-unique B right-total B
  shows
    (
      rel-set A  $\implies$  (A  $\implies$  A  $\implies$  (=))  $\implies$  (A  $\implies$  A  $\implies$  (=))  $\implies$ 
      rel-set B  $\implies$  (B  $\implies$  B  $\implies$  (=))  $\implies$  (B  $\implies$  B  $\implies$  (=))  $\implies$ 
      (=)
    ) order-pair-ow order-pair-ow
  <proof>

end

```

Relativization

```

context order-ow
begin

```

```

interpretation ord-syntax-ow <proof>

```

```

tts-context

```

```

  tts: (?'a to U)
  sbterms: (?ls::?'a  $\Rightarrow$  ?'a  $\Rightarrow$  bool to ls)
    and (?le::?'a  $\Rightarrow$  ?'a  $\Rightarrow$  bool to le)
  rewriting ctr-simps
  substituting order-ow-axioms
  eliminating through auto
begin

```

```

tts-lemma atLeastAtMost-singleton:

```

```

  assumes a  $\in$  U
  shows  $\{a \leq_a \dots \leq_a a\} = \{a\}$ 
  is order.atLeastAtMost-singleton<proof>

```

```

tts-lemma less-imp-not-eq:

```

```

  assumes y  $\in$  U and x <_a y
  shows  $(x = y) = \text{False}$ 
  is order.less-imp-not-eq<proof>

```

```

tts-lemma less-imp-not-eq2:

```

```

  assumes y  $\in$  U and x <_a y
  shows  $(y = x) = \text{False}$ 
  is order.less-imp-not-eq2<proof>

```

```

tts-lemma eq-iff:

```

```

  assumes x  $\in$  U and y  $\in$  U
  shows  $(x = y) = (x \leq_a y \wedge y \leq_a x)$ 
  is order.eq-iff<proof>

```

```

tts-lemma le-less:

```

```

  assumes x  $\in$  U and y  $\in$  U
  shows  $(x \leq_a y) = (x <_a y \vee x = y)$ 
  is order.le-less<proof>

```

```

tts-lemma min-absorb2:

```

```

  assumes y  $\in$  U and x  $\in$  U and y  $\leq_a$  x
  shows min x y = y

```

is *order.min-absorb2*(proof)

tts-lemma *less-le*:

assumes $x \in U$ and $y \in U$
 shows $(x <_a y) = (x \leq_a y \wedge x \neq y)$
 is *order.less-le*(proof)

tts-lemma *le-imp-less-or-eq*:

assumes $x \in U$ and $y \in U$ and $x \leq_a y$
 shows $x <_a y \vee x = y$
 is *order.le-imp-less-or-eq*(proof)

tts-lemma *antisym-conv*:

assumes $y \in U$ and $x \in U$ and $y \leq_a x$
 shows $(x \leq_a y) = (x = y)$
 is *order.antisym-conv*(proof)

tts-lemma *le-neq-trans*:

assumes $a \in U$ and $b \in U$ and $a \leq_a b$ and $a \neq b$
 shows $a <_a b$
 is *order.le-neq-trans*(proof)

tts-lemma *neq-le-trans*:

assumes $a \in U$ and $b \in U$ and $a \neq b$ and $a \leq_a b$
 shows $a <_a b$
 is *order.neq-le-trans*(proof)

tts-lemma *atLeastAtMost-singleton'*:

assumes $b \in U$ and $a = b$
 shows $\{a \leq_a .. \leq_a b\} = \{a\}$
 is *order.atLeastAtMost-singleton'*(proof)

tts-lemma *atLeastLessThan-eq-atLeastAtMost-diff*:

assumes $a \in U$ and $b \in U$
 shows $\{a \leq_a .. <_a b\} = \{a \leq_a .. \leq_a b\} - \{b\}$
 is *order.atLeastLessThan-eq-atLeastAtMost-diff*(proof)

tts-lemma *greaterThanAtMost-eq-atLeastAtMost-diff*:

assumes $a \in U$ and $b \in U$
 shows $\{a <_a .. \leq_a b\} = \{a \leq_a .. \leq_a b\} - \{a\}$
 is *order.greaterThanAtMost-eq-atLeastAtMost-diff*(proof)

tts-lemma *atMost-Int-atLeast*:

assumes $n \in U$
 shows $\{.. \leq_a n\} \cap \{n \leq_a ..\} = \{n\}$
 is *order.atMost-Int-atLeast*(proof)

tts-lemma *atLeast-eq-iff*:

assumes $x \in U$ and $y \in U$
 shows $(\{x \leq_a ..\} = \{y \leq_a ..\}) = (x = y)$
 is *order.atLeast-eq-iff*(proof)

tts-lemma *Least-equality*:

assumes $x \in U$ and $P x$
 and $\wedge y. \llbracket y \in U; P y \rrbracket \implies x \leq_a y$
 shows *Least P = Some x*
 is *order.Least-equality*(proof)

tts-lemma *Icc-eq-Icc*:

assumes $l \in U$ **and** $h \in U$ **and** $l' \in U$ **and** $h' \in U$
shows $(\{l \leq_a \dots \leq_a h\} = \{l' \leq_a \dots \leq_a h'\}) =$
 $(h = h' \wedge l = l' \vee \neg l' \leq_a h' \wedge \neg l \leq_a h)$
is *order.Icc-eq-Icc* \langle *proof* \rangle

tts-lemma *LeastI2-order*:

assumes $x \in U$
and $P x$
and $\bigwedge y. \llbracket y \in U; P y \rrbracket \implies x \leq_a y$
and $\bigwedge x. \llbracket x \in U; P x; \forall y \in U. P y \longrightarrow x \leq_a y \rrbracket \implies Q x$
and $\bigwedge z. \llbracket z \in U; \text{Least } P = \text{Some } z; Q z \rrbracket \implies \textit{thesis}$
shows *thesis*
is *order.LeastI2-order* \langle *proof* \rangle

tts-lemma *mono-image-least*:

assumes $\forall x \in U. f x \in U$
and $m \in U$
and $n \in U$
and $m' \in U$
and $n' \in U$
and *on* U *with* (\leq_a) $(\leq_a) : \langle \textit{mono} \rangle f$
and $f' \{m \leq_a \dots \leq_a n\} = \{m' \leq_a \dots \leq_a n'\}$
and $m <_a n$
shows $f m = m'$
is *order.mono-image-least* \langle *proof* \rangle

tts-lemma *antisym-conv1*:

assumes $x \in U$ **and** $y \in U$ **and** $\neg x <_a y$
shows $(x \leq_a y) = (x = y)$
is *order.antisym-conv1* \langle *proof* \rangle

tts-lemma *antisym-conv2*:

assumes $x \in U$ **and** $y \in U$ **and** $x \leq_a y$
shows $(\neg x <_a y) = (x = y)$
is *order.antisym-conv2* \langle *proof* \rangle

tts-lemma *leD*:

assumes $y \in U$ **and** $x \in U$ **and** $y \leq_a x$
shows $\neg x <_a y$
is *order.leD* \langle *proof* \rangle

end

tts-context

tts: $(?a \text{ to } U)$
rewriting *ctr-simps*
substituting *order-ow-axioms*
eliminating $\langle ?A \neq \{\} \rangle$ **through** *auto*
begin

tts-lemma *atLeastAtMost-singleton-iff*:

assumes $a \in U$
and $b \in U$
and $c \in U$
shows $(\{a \leq_a \dots \leq_a b\} = \{c\}) = (a = b \wedge b = c)$
is *order.atLeastAtMost-singleton-iff* \langle *proof* \rangle

end

tts-context

tts: ($?'a$ to U)
sbterms: ($\langle ?'ls::?'a \Rightarrow ?'a \Rightarrow \text{bool} \rangle$ to ls)
 and ($\langle ?'le::?'a \Rightarrow ?'a \Rightarrow \text{bool} \rangle$ to le)
rewriting *ctr-simps*
substituting *order-ow-axioms*
eliminating through *auto*

begin

tts-lemma *Least-ex1:*

assumes $z \in U$
 and $\exists!x. x \in U \wedge P x \wedge (\forall y \in U. P y \longrightarrow x \leq_a y)$
 and $\wedge x. \llbracket x \in U; \text{Least } P = \text{Some } x; P x; P z \Longrightarrow x \leq_a z \rrbracket \Longrightarrow \textit{thesis}$
shows *thesis*
is *order.Least-ex1{proof}*

end

end

context *order-pair-ow*

begin

interpretation *ord-pair-syntax-ow* {proof}

tts-context

tts: ($?'a$ to U_a) and ($?'b$ to U_b)
sbterms: ($\langle ?'ls_a::?'a \Rightarrow ?'a \Rightarrow \text{bool} \rangle$ to ls_a)
 and ($\langle ?'le_a::?'a \Rightarrow ?'a \Rightarrow \text{bool} \rangle$ to le_a)
 and ($\langle ?'ls_b::?'b \Rightarrow ?'b \Rightarrow \text{bool} \rangle$ to ls_b)
 and ($\langle ?'le_b::?'b \Rightarrow ?'b \Rightarrow \text{bool} \rangle$ to le_b)
rewriting *ctr-simps*
substituting *order-pair-ow-axioms*
eliminating through (*auto simp: mono-def bdd-def*)

begin

tts-lemma *strict-mono-mono:*

assumes $\forall x \in U_a. f x \in U_b$ and *strict-mono_{ab} f*
shows *mono_{ab} f*
is *order-pair.strict-mono-mono{proof}*

tts-lemma *bdd-above-image-mono:*

assumes $\forall x \in U_a. f x \in U_b$ and $A \subseteq U_a$ and *mono_{ab} f* and *ord_a.bdd-above A*
shows *ord_b.bdd-above (f ' A)*
is *order-pair.bdd-above-image-mono{proof}*

tts-lemma *bdd-below-image-mono:*

assumes $\forall x \in U_a. f x \in U_b$ and $A \subseteq U_a$ and *mono_{ab} f* and *ord_a.bdd-below A*
shows *ord_b.bdd-below (f ' A)*
is *order-pair.bdd-below-image-mono{proof}*

tts-lemma *bdd-below-image-antimono:*

assumes $\forall x \in U_a. f x \in U_b$
 and $A \subseteq U_a$
 and *antimono_{ab} f*
 and *ord_a.bdd-above A*

shows *ord_b.bdd-below* (*f* ‘ *A*)
is *order-pair.bdd-below-image-antimono*{*proof*}

tts-lemma *bdd-above-image-antimono*:
assumes $\forall x \in U_a. f x \in U_b$
and $A \subseteq U_a$
and *antimono_{ab}* *f*
and *ord_a.bdd-below* *A*
shows *ord_b.bdd-above* (*f* ‘ *A*)
is *order-pair.bdd-above-image-antimono*{*proof*}

end

end

5.5.5 Dense orders

Definitions and common properties

locale *dense-order-ow* = *order-ow* *U* *le* *ls*
for *U* :: ‘*a* set **and** *le* *ls* +
assumes *dense*: $[[x \in U; y \in U; ls\ x\ y]] \implies (\exists z \in U. ls\ x\ z \wedge ls\ z\ y)$

locale *dense-order-dual-ow* = *dense-order-ow* *U* *le* *ls*
for *U* :: ‘*a* set **and** *le* *ls*
begin

interpretation *ord-syntax-ow* {*proof*}

sublocale *order-dual-ow* {*proof*}

sublocale *dual*: *dense-order-ow* *U* *ge* *gt*
{*proof*}

end

locale *ord-dense-order-ow* =
ord-order-ow *U_a* *le_a* *ls_a* *U_b* *le_b* *ls_b* + *ord_b*: *dense-order-ow* *U_b* *le_b* *ls_b*
for *U_a* :: ‘*a* set **and** *le_a* *ls_a* **and** *U_b* :: ‘*b* set **and** *le_b* *ls_b*

locale *ord-dense-order-dual-ow* = *ord-dense-order-ow* *U_a* *le_a* *ls_a* *U_b* *le_b* *ls_b*
for *U_a* :: ‘*a* set **and** *le_a* *ls_a* **and** *U_b* :: ‘*b* set **and** *le_b* *ls_b*
begin

sublocale *ord-order-dual-ow* {*proof*}

sublocale *ord-dual*: *ord-dense-order-ow* *U_a* ⟨(*≤_a*)⟩ ⟨(*<_a*)⟩ *U_b* ⟨(*≥_b*)⟩ ⟨(*>_b*)⟩
{*proof*}

sublocale *dual-ord*: *ord-dense-order-ow* *U_a* ⟨(*≥_a*)⟩ ⟨(*>_a*)⟩ *U_b* ⟨(*≤_b*)⟩ ⟨(*<_b*)⟩
{*proof*}

sublocale *dual-dual*: *ord-dense-order-ow* *U_a* ⟨(*≥_a*)⟩ ⟨(*>_a*)⟩ *U_b* ⟨(*≥_b*)⟩ ⟨(*>_b*)⟩
{*proof*}

end

locale *preorder-dense-order-ow* =
ord-dense-order-ow *U_a* *le_a* *ls_a* *U_b* *le_b* *ls_b* + *ord_a*: *preorder-ow* *U_a* *le_a* *ls_a*
for *U_a* :: ‘*a* set **and** *le_a* *ls_a* **and** *U_b* :: ‘*b* set **and** *le_b* *ls_b*
begin

sublocale *preorder-order-ow* $\langle \text{proof} \rangle$

end

locale *preorder-dense-order-dual-ow* =

preorder-dense-order-ow U_a le_a ls_a U_b le_b ls_b

for $U_a :: 'a$ set **and** le_a ls_a **and** $U_b :: 'b$ set **and** le_b ls_b

begin

sublocale *ord-dense-order-dual-ow* $\langle \text{proof} \rangle$

sublocale *preorder-order-dual-ow* $\langle \text{proof} \rangle$

sublocale *ord-dual: preorder-dense-order-ow* U_a $\langle (\leq_a) \rangle$ $\langle (<_a) \rangle$ U_b $\langle (\geq_b) \rangle$ $\langle (>_b) \rangle$
 $\langle \text{proof} \rangle$

sublocale *dual-ord: preorder-dense-order-ow* U_a $\langle (\geq_a) \rangle$ $\langle (>_a) \rangle$ U_b $\langle (\leq_b) \rangle$ $\langle (<_b) \rangle$
 $\langle \text{proof} \rangle$

sublocale *dual-dual: preorder-dense-order-ow* U_a $\langle (\geq_a) \rangle$ $\langle (>_a) \rangle$ U_b $\langle (\geq_b) \rangle$ $\langle (>_b) \rangle$
 $\langle \text{proof} \rangle$

end

locale *order-dense-order-ow* =

preorder-dense-order-ow U_a le_a ls_a U_b le_b ls_b + *ord_a: order-ow* U_a le_a ls_a

for $U_a :: 'a$ set **and** le_a ls_a **and** $U_b :: 'b$ set **and** le_b ls_b

begin

sublocale *order-pair-ow* $\langle \text{proof} \rangle$

end

locale *order-dense-order-dual-ow* = *order-dense-order-ow* U_a le_a ls_a U_b le_b ls_b

for $U_a :: 'a$ set **and** le_a ls_a **and** $U_b :: 'b$ set **and** le_b ls_b

begin

sublocale *preorder-dense-order-dual-ow* $\langle \text{proof} \rangle$

sublocale *order-pair-dual-ow* $\langle \text{proof} \rangle$

sublocale *ord-dual: order-dense-order-ow* U_a $\langle (\leq_a) \rangle$ $\langle (<_a) \rangle$ U_b $\langle (\geq_b) \rangle$ $\langle (>_b) \rangle$ $\langle \text{proof} \rangle$

sublocale *dual-ord: order-dense-order-ow* U_a $\langle (\geq_a) \rangle$ $\langle (>_a) \rangle$ U_b $\langle (\leq_b) \rangle$ $\langle (<_b) \rangle$ $\langle \text{proof} \rangle$

sublocale *dual-dual: order-dense-order-ow* U_a $\langle (\geq_a) \rangle$ $\langle (>_a) \rangle$ U_b $\langle (\geq_b) \rangle$ $\langle (>_b) \rangle$ $\langle \text{proof} \rangle$

end

locale *dense-order-pair-ow* =

order-dense-order-ow U_a le_a ls_a U_b le_b ls_b + *ord_a: dense-order-ow* U_a le_a ls_a

for $U_a :: 'a$ set **and** le_a ls_a **and** $U_b :: 'b$ set **and** le_b ls_b

locale *dense-order-pair-dual-ow* = *dense-order-pair-ow* U_a le_a ls_a U_b le_b ls_b

for $U_a :: 'a$ set **and** le_a ls_a **and** $U_b :: 'b$ set **and** le_b ls_b

begin

sublocale *order-dense-order-dual-ow* $\langle \text{proof} \rangle$

sublocale *ord-dual: dense-order-pair-ow* U_a $\langle (\leq_a) \rangle$ $\langle (<_a) \rangle$ U_b $\langle (\geq_b) \rangle$ $\langle (>_b) \rangle$ $\langle \text{proof} \rangle$

sublocale *dual-ord: dense-order-pair-ow* U_a $\langle (\geq_a) \rangle$ $\langle (>_a) \rangle$ U_b $\langle (\leq_b) \rangle$ $\langle (<_b) \rangle$

$\langle \text{proof} \rangle$

sublocale *dual-dual: dense-order-pair-ow* U_a $\langle (\geq_a) \rangle$ $\langle (>_a) \rangle$ U_b $\langle (\geq_b) \rangle$ $\langle (>_b) \rangle$ $\langle \text{proof} \rangle$

end

Transfer rules

lemma *dense-order-ow*[*ud-with*]: *dense-order* = *dense-order-ow* *UNIV*
 ⟨*proof*⟩

lemma *ord-dense-order-ow*[*ud-with*]: *ord-dense-order* = *ord-dense-order-ow* *UNIV*
 ⟨*proof*⟩

lemma *preorder-dense-order-ow*[*ud-with*]:
preorder-dense-order =
 (λ*le_a ls_a le_b ls_b*. *preorder-dense-order-ow* *UNIV le_a ls_a UNIV le_b ls_b*)
 ⟨*proof*⟩

lemma *order-dense-order-ow*[*ud-with*]:
order-dense-order =
 (λ*le_a ls_a le_b ls_b*. *order-dense-order-ow* *UNIV le_a ls_a UNIV le_b ls_b*)
 ⟨*proof*⟩

lemma *dense-order-pair-ow*[*ud-with*]:
dense-order-pair =
 (λ*le_a ls_a le_b ls_b*. *dense-order-pair-ow* *UNIV le_a ls_a UNIV le_b ls_b*)
 ⟨*proof*⟩

context

includes *lifting-syntax*

begin

lemma *desne-order-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
 (*rel-set* *A* \implies (*A* \implies *A* \implies (=)) \implies (*A* \implies *A* \implies (=)) \implies (=))
dense-order-ow *dense-order-ow*
 ⟨*proof*⟩

lemma *ord-dense-order-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique* *A* *right-total* *A*
shows
 (
 rel-set *A* \implies (*A* \implies *A* \implies (=)) \implies (*A* \implies *A* \implies (=)) \implies
 (=)
) *ord-dense-order-ow* *ord-dense-order-ow*
 ⟨*proof*⟩

lemma *preorder-dense-order-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total* *A* *bi-unique* *B* *right-total* *B*
shows
 (
 rel-set *A* \implies (*A* \implies *A* \implies (=)) \implies (*A* \implies *A* \implies (=)) \implies
 rel-set *B* \implies (*B* \implies *B* \implies (=)) \implies (*B* \implies *B* \implies (=)) \implies
 (=)
) *preorder-dense-order-ow* *preorder-dense-order-ow*
 ⟨*proof*⟩

lemma *order-dense-order-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]:
bi-unique *A* *right-total* *A* *bi-unique* *B* *right-total* *B*
shows
 (

$rel\text{-}set\ A \implies (A \implies A \implies (=)) \implies (A \implies A \implies (=)) \implies$
 $rel\text{-}set\ B \implies (B \implies B \implies (=)) \implies (B \implies B \implies (=)) \implies$
 $(=)$
) *order-dense-order-ow order-dense-order-ow*
 <proof>

lemma *dense-order-pair-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]:

bi-unique A right-total A bi-unique B right-total B

shows

(
 $rel\text{-}set\ A \implies (A \implies A \implies (=)) \implies (A \implies A \implies (=)) \implies$
 $rel\text{-}set\ B \implies (B \implies B \implies (=)) \implies (B \implies B \implies (=)) \implies$
 $(=)$
) *dense-order-pair-ow dense-order-pair-ow*
 <proof>

end

5.5.6 (Unique) top and bottom elements

locale *extremum-ow* =

fixes $U :: 'a\ set$ **and** *extremum*

assumes *extremum-closed*[*simp*]: $extremum \in U$

locale *bot-ow* = *extremum-ow U bot* **for** $U :: 'a\ set$ **and** *bot*

begin

notation *bot* ($\langle \perp \rangle$)

end

locale *top-ow* = *extremum-ow U top* **for** $U :: 'a\ set$ **and** *top*

begin

notation *top* ($\langle \top \rangle$)

end

locale *ord-extremum-ow* = *ord-ow U le ls + extremum-ow U extremum*

for $U :: 'a\ set$ **and** *le ls extremum*

locale *order-extremum-ow* = *ord-extremum-ow U le ls extremum + order-ow U le ls*

for $U :: 'a\ set$ **and** *le ls extremum +*

assumes *extremum*[*simp*]: $a \in U \implies le\ a\ extremum$

locale *order-bot-ow* =

order-dual-ow U le ls + dual: order-extremum-ow U ge gt bot + bot-ow U bot

for $U :: 'a\ set$ **and** *le ls bot*

locale *order-top* =

order-dual-ow U le ls + order-extremum-ow U le ls top + top-ow U top

for $U :: 'a\ set$ **and** *le ls top*

Transfer rules

lemma *order-extremum-ow*[*ud-with*]: *order-extremum* = *order-extremum-ow UNIV*

<proof>

context

includes *lifting-syntax*

begin

lemma *extremum-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A*

shows (*rel-set A* \implies *A* \implies (=)) *extremum-ow extremum-ow*
 \langle *proof* \rangle

lemma *ord-extremum-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A*

shows (*rel-set A* \implies *A* \implies (=)) *ord-extremum-ow ord-extremum-ow*
 \langle *proof* \rangle

lemma *order-extremum-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A*

shows

(
rel-set A \implies (*A* \implies *A* \implies (=)) \implies (*A* \implies *A* \implies (=)) \implies *A* \implies
 (=)

) *order-extremum-ow order-extremum-ow*

\langle *proof* \rangle

end

Relativization

context *order-extremum-ow*

begin

interpretation *ord-syntax-ow* \langle *proof* \rangle

tts-context

tts: (*?a to U*)

sbterms: (\langle *?le::?a \Rightarrow ?a \Rightarrow bool* \rangle **to** *le*)

and (\langle *?ls::?a \Rightarrow ?a \Rightarrow bool* \rangle **to** *ls*)

rewriting *ctr-simps*

substituting *order-extremum-ow-axioms*

eliminating through *force*

begin

tts-lemma *extremum-strict*:

assumes $a \in U$

shows \neg *extremum* $<_a$ *a*

is *order-extremum.extremum-strict* \langle *proof* \rangle

tts-lemma *bdd-above-top*:

assumes $A \subseteq U$

shows *bdd-above A*

is *order-extremum.bdd-above-top* \langle *proof* \rangle

tts-lemma *min-top*:

assumes $x \in U$

shows *min extremum* $x = x$

is *order-extremum.min-top* \langle *proof* \rangle

tts-lemma *min-top2*:

assumes $x \in U$
shows $\min x \text{ extremum} = x$
is *order-extremum.min-top2*{proof}

tts-lemma *extremum-unique*:
assumes $a \in U$
shows $(\text{extremum} \leq_a a) = (a = \text{extremum})$
is *order-extremum.extremum-unique*{proof}

tts-lemma *not-eq-extremum*:
assumes $a \in U$
shows $(a \neq \text{extremum}) = (a <_a \text{extremum})$
is *order-extremum.not-eq-extremum*{proof}

tts-lemma *extremum-uniqueI*:
assumes $a \in U$ **and** $\text{extremum} \leq_a a$
shows $a = \text{extremum}$
is *order-extremum.extremum-uniqueI*{proof}

tts-lemma *max-top*:
assumes $x \in U$
shows $\max \text{ extremum } x = \text{extremum}$
is *order-extremum.max-top*{proof}

tts-lemma *max-top2*:
assumes $x \in U$
shows $\max x \text{ extremum} = \text{extremum}$
is *order-extremum.max-top2*{proof}

tts-lemma *atMost-eq-UNIV-iff*:
assumes $x \in U$
shows $(\{.. \leq_a x\} = U) = (x = \text{extremum})$
is *order-extremum.atMost-eq-UNIV-iff*{proof}

end

end

5.5.7 Absence of top or bottom elements

locale *no-extremum-ow* = *order-ow* U *le* ls **for** $U :: 'a \text{ set}$ **and** *le* ls +
assumes *gt-ex*: $x \in U \implies \exists y \in U. ls \ x \ y$

locale *no-top-ow* = *order-dual-ow* U *le* ls + *no-extremum-ow* U *le* ls
for $U :: 'a \text{ set}$ **and** *le* ls

locale *no-bot-ow* = *order-dual-ow* U *le* ls + *dual: no-extremum-ow* U *ge* gt
for $U :: 'a \text{ set}$ **and** *le* ls

Transfer rules

lemma *no-extremum-ow*[*ud-with*]: *no-extremum* = *no-extremum-ow* *UNIV*
 {proof}

context
includes *lifting-syntax*
begin

lemma *no-extremum-ow-axioms-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 (*rel-set A* \implies (*A* \implies *A* \implies (=)) \implies (=))
no-extremum-ow-axioms no-extremum-ow-axioms
 ⟨*proof*⟩

lemma *no-extremum-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
 (*rel-set A* \implies (*A* \implies *A* \implies (=)) \implies (*A* \implies *A* \implies (=)) \implies (=))
no-extremum-ow no-extremum-ow
 ⟨*proof*⟩

end

Relativization

lemma *right-total-UNIV-transfer'*[*transfer-rule*]:
assumes *right-total A* **and** *Domainp A = ($\lambda x. x \in U$)*
shows *rel-set A U UNIV*
 ⟨*proof*⟩

context *no-extremum-ow*
begin

interpretation *ord-syntax-ow* ⟨*proof*⟩

tts-context

tts: (*?a to U*)
sbterms: (*!le::?a \Rightarrow ?a \Rightarrow bool* **to** *le*)
and (*!ls::?a \Rightarrow ?a \Rightarrow bool* **to** *ls*)
rewriting *ctr-simps*
substituting *no-extremum-ow-axioms*
eliminating through *force*

begin

tts-lemma *not-UNIV-eq-Iic*:
assumes *h' \in U*
shows *U \neq $\{..\leq_a h'\}$*
is *no-extremum.not-UNIV-eq-Iic*⟨*proof*⟩

tts-lemma *not-Iic-eq-UNIV*:
assumes *h' \in U*
shows *$\{..\leq_a h'\} \neq U$*
is *no-extremum.not-Iic-eq-UNIV*⟨*proof*⟩

tts-lemma *not-UNIV-le-Iic*:
assumes *h \in U*
shows $\neg U \subseteq \{..\leq_a h\}$
is *no-extremum.not-UNIV-le-Iic*⟨*proof*⟩

tts-lemma *not-UNIV-eq-Icc*:
assumes *l' \in U* **and** *h' \in U*
shows *U \neq $\{l' \leq_a .. \leq_a h'\}$*
is *no-extremum.not-UNIV-eq-Icc*⟨*proof*⟩

tts-lemma *not-Icc-eq-UNIV*:

assumes $l' \in U$ **and** $h' \in U$
shows $\{l' \leq_{a..} \leq_a h'\} \neq U$
is *no-extremum.not-Icc-eq-UNIV*(proof)

tts-lemma *not-UNIV-le-Icc*:
assumes $l \in U$ **and** $h \in U$
shows $\neg U \subseteq \{l \leq_{a..} \leq_a h\}$
is *no-extremum.not-UNIV-le-Icc*(proof)

tts-lemma *greaterThan-non-empty*:
assumes $x \in U$
shows $\{x <_{a..}\} \neq \{\}$
is *no-extremum.greaterThan-non-empty*(proof)

tts-lemma *not-Iic-eq-Ici*:
assumes $h \in U$ **and** $l' \in U$
shows $\{.. \leq_a h\} \neq \{l' \leq_{a..}\}$
is *no-extremum.not-Iic-eq-Ici*(proof)

tts-lemma *not-Ici-eq-Iic*:
assumes $l' \in U$ **and** $h \in U$
shows $\{l' \leq_{a..}\} \neq \{.. \leq_a h\}$
is *no-extremum.not-Ici-eq-Iic*(proof)

tts-lemma *not-Ici-le-Iic*:
assumes $l \in U$ **and** $h' \in U$
shows $\neg \{l \leq_{a..}\} \subseteq \{.. \leq_a h'\}$
is *no-extremum.not-Ici-le-Iic*(proof)

tts-lemma *not-Icc-eq-Ici*:
assumes $l \in U$ **and** $h \in U$ **and** $l' \in U$
shows $\{l \leq_{a..} \leq_a h\} \neq \{l' \leq_{a..}\}$
is *no-extremum.not-Icc-eq-Ici*(proof)

tts-lemma *not-Ici-eq-Icc*:
assumes $l' \in U$ **and** $l \in U$ **and** $h \in U$
shows $\{l' \leq_{a..}\} \neq \{l \leq_{a..} \leq_a h\}$
is *no-extremum.not-Ici-eq-Icc*(proof)

tts-lemma *not-Ici-le-Icc*:
assumes $l \in U$ **and** $l' \in U$ **and** $h' \in U$
shows $\neg \{l \leq_{a..}\} \subseteq \{l' \leq_{a..} \leq_a h'\}$
is *no-extremum.not-Ici-le-Icc*(proof)

end

end

declare *right-total-UNIV-transfer*[*transfer-rule del*]

5.6 Abstract semigroups on types

5.6.1 Background

The results presented in this section were ported (with amendments and additions) from the theory *Groups* in the main library of Isabelle/HOL.

5.6.2 Preliminaries

named-theorems *tts-ac-simps assoc. and comm. simplification rules*
and *tts-algebra-simps algebra simplification rules*
and *tts-field-simps algebra simplification rules for fields*

5.6.3 Binary operations

Abstract operation.

locale *binary-op* =
fixes $f :: 'a \Rightarrow 'a \Rightarrow 'a$

locale *binary-op-syntax* = *binary-op f* **for** $f :: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation f (**infixl** $\langle \oplus_a \rangle$ 65)

end

Concrete syntax.

locale *plus* = *binary-op plus* **for** $plus :: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation *plus* (**infixl** $\langle +_a \rangle$ 65)

end

locale *minus* = *binary-op minus* **for** $minus :: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation *minus* (**infixl** $\langle -_a \rangle$ 65)

end

locale *times* = *binary-op times* **for** $times :: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation *times* (**infixl** $\langle *_a \rangle$ 70)

end

locale *divide* = *binary-op divide* **for** $divide :: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation *divide* (**infixl** $\langle '/_a \rangle$ 70)

end

Pairs.

locale *binary-op-pair* = $alg_a: binary-op f_a + alg_b: binary-op f_b$

```

for  $f_a :: 'a \Rightarrow 'a \Rightarrow 'a$  and  $f_b :: 'b \Rightarrow 'b \Rightarrow 'b$ 

locale binary-op-pair-syntax = binary-op-pair  $f_a$   $f_b$ 
  for  $f_a :: 'a \Rightarrow 'a \Rightarrow 'a$  and  $f_b :: 'b \Rightarrow 'b \Rightarrow 'b$ 
begin

notation  $f_a$  (infixl  $\langle \oplus_a \rangle$  65)
notation  $f_b$  (infixl  $\langle \oplus_b \rangle$  65)

end

```

5.6.4 Simple semigroups

Definitions

Abstract semigroups.

```

locale semigroup = binary-op  $f$  for  $f :: 'a \Rightarrow 'a \Rightarrow 'a$  +
  assumes assoc[tts-ac-simps, tts-algebra-simps]:  $f (f a b) c = f a (f b c)$ 

```

```

locale semigroup-syntax = binary-op-syntax  $f$  for  $f :: 'a \Rightarrow 'a \Rightarrow 'a$ 

```

Concrete syntax.

```

locale semigroup-add = semigroup plus for plus ::  $'a \Rightarrow 'a \Rightarrow 'a$ 
begin

```

```

  sublocale plus plus  $\langle$ proof $\rangle$ 

```

```

end

```

```

locale semigroup-mult = semigroup times for times ::  $'a \Rightarrow 'a \Rightarrow 'a$ 
begin

```

```

  sublocale times times  $\langle$ proof $\rangle$ 

```

```

end

```

Pairs.

```

locale semigroup-pair = alg $a$ : semigroup  $f_a$  + alg $b$ : semigroup  $f_b$ 
  for  $f_a :: 'a \Rightarrow 'a \Rightarrow 'a$  and  $f_b :: 'b \Rightarrow 'b \Rightarrow 'b$ 
begin

```

```

  sublocale binary-op-pair  $f_a$   $f_b$   $\langle$ proof $\rangle$ 

```

```

  sublocale rev: semigroup-pair  $f_b$   $f_a$   $\langle$ proof $\rangle$ 

```

```

end

```

```

locale semigroup-pair-syntax = binary-op-pair-syntax

```

5.6.5 Commutative semigroups

Definitions

Abstract commutative semigroup.

```

locale comm-semigroup = semigroup  $f$  for  $f :: 'a \Rightarrow 'a \Rightarrow 'a$  +
  assumes commute[tts-ac-simps, tts-algebra-simps]:  $f a b = f b a$ 

```

```

locale comm-semigroup-syntax = semigroup-syntax

```

Concrete syntax.

locale *comm-semigroup-add* = *comm-semigroup plus* **for** *plus* :: 'a ⇒ 'a ⇒ 'a
begin

sublocale *semigroup-add plus* ⟨*proof*⟩

end

locale *comm-semigroup-mult* = *comm-semigroup times* **for** *times* :: 'a ⇒ 'a ⇒ 'a
begin

sublocale *semigroup-mult times* ⟨*proof*⟩

end

Pairs.

locale *comm-semigroup-pair* = *alg_a: comm-semigroup f_a* + *alg_b: comm-semigroup f_b*
for *f_a* :: 'a ⇒ 'a ⇒ 'a **and** *f_b* :: 'b ⇒ 'b ⇒ 'b
begin

sublocale *semigroup-pair f_a f_b* ⟨*proof*⟩

sublocale *rev: comm-semigroup-pair f_b f_a* ⟨*proof*⟩

end

locale *comm-semigroup-pair-syntax* = *semigroup-pair-syntax*

Results

context *comm-semigroup*
begin

interpretation *comm-semigroup-syntax f* ⟨*proof*⟩

lemma *left-commute[tts-ac-simps, tts-algebra-simps, field-simps]*:
 $b \oplus_a (a \oplus_a c) = a \oplus_a (b \oplus_a c)$
⟨*proof*⟩

end

5.6.6 Cancellative semigroups

Definitions

Abstract cancellative semigroup.

locale *cancel-semigroup* = *semigroup f* **for** *f* :: 'a ⇒ 'a ⇒ 'a +
assumes *add-left-imp-eq: f a b = f a c ⇒ b = c*
assumes *add-right-imp-eq: f b a = f c a ⇒ b = c*

locale *cancel-semigroup-syntax* = *semigroup-syntax f* **for** *f* :: 'a ⇒ 'a ⇒ 'a

Concrete syntax.

locale *cancel-semigroup-add* = *cancel-semigroup plus*
for *plus* :: 'a ⇒ 'a ⇒ 'a
begin

sublocale *semigroup-add plus* ⟨*proof*⟩

end

locale *cancel-semigroup-mult* = *cancel-semigroup times*
 for *times* :: 'a ⇒ 'a ⇒ 'a
begin

sublocale *semigroup-mult times* ⟨*proof*⟩

end

Pairs.

locale *cancel-semigroup-pair* =
 alg_a: *cancel-semigroup f_a* + alg_b: *cancel-semigroup f_b*
 for *f_a* :: 'a ⇒ 'a ⇒ 'a **and** *f_b* :: 'b ⇒ 'b ⇒ 'b
begin

sublocale *semigroup-pair f_a f_b* ⟨*proof*⟩

sublocale *rev: cancel-semigroup-pair f_b f_a* ⟨*proof*⟩

end

locale *cancel-semigroup-pair-syntax* = *semigroup-pair-syntax f_a f_b*
 for *f_a* :: 'a ⇒ 'a ⇒ 'a **and** *f_b* :: 'b ⇒ 'b ⇒ 'b

Results

context *cancel-semigroup*
begin

interpretation *cancel-semigroup-syntax f* ⟨*proof*⟩

lemma *add-left-cancel[simp]*: $a \oplus_a b = a \oplus_a c \longleftrightarrow b = c$
 ⟨*proof*⟩

lemma *add-right-cancel[simp]*: $b \oplus_a a = c \oplus_a a \longleftrightarrow b = c$
 ⟨*proof*⟩

lemma *inj-on-add[simp]*: *inj-on* ((\oplus_a) *a*) *A* ⟨*proof*⟩

lemma *inj-on-add'[simp]*: *inj-on* ($\lambda b. b \oplus_a a$) *A* ⟨*proof*⟩

lemma *bij-betw-add[simp]*: *bij-betw* ((\oplus_a) *a*) *A B* \longleftrightarrow (\oplus_a) *a* ' *A = B*
 ⟨*proof*⟩

end

5.6.7 Cancellative commutative semigroups

Definitions

Abstract cancellative commutative semigroups.

locale *cancel-comm-semigroup* = *comm: comm-semigroup f* + *binary-op fi*
 for *f fi* :: 'a ⇒ 'a ⇒ 'a +
assumes *add-diff-cancel-left'[simp]*: *fi* (*f a b*) *a* = *b*
and *diff-diff-add[tts-algebra-simps, tts-field-simps]*:
fi (*fi a b*) *c* = *fi a* (*f b c*)

```

locale cancel-comm-semigroup-syntax = comm-semigroup-syntax f + binary-op fi
  for f fi :: 'a ⇒ 'a ⇒ 'a
begin

```

```

notation fi (infixl ⟨ $\ominus_a$ ⟩ 65)

```

```

end

```

Concrete syntax.

```

locale cancel-comm-semigroup-add = cancel-comm-semigroup plus minus
  for plus minus :: 'a ⇒ 'a ⇒ 'a
begin

```

```

sublocale comm-semigroup-add plus ⟨proof⟩

```

```

sublocale minus minus ⟨proof⟩

```

```

end

```

```

locale cancel-comm-semigroup-mult = cancel-comm-semigroup times divide
  for times divide :: 'a ⇒ 'a ⇒ 'a
begin

```

```

sublocale comm-semigroup-mult times ⟨proof⟩

```

```

sublocale divide divide ⟨proof⟩

```

```

end

```

Pairs.

```

locale cancel-comm-semigroup-pair =
  alga: cancel-comm-semigroup fa fia + algb: cancel-comm-semigroup fb fib
  for fa fia :: 'a ⇒ 'a ⇒ 'a and fb fib :: 'b ⇒ 'b ⇒ 'b
begin

```

```

sublocale comm-semigroup-pair fa fb ⟨proof⟩

```

```

sublocale rev: cancel-comm-semigroup-pair fb fib fa fia ⟨proof⟩

```

```

end

```

```

locale cancel-comm-semigroup-pair-syntax =
  comm-semigroup-pair-syntax fa fb + binary-op fia + binary-op fib
  for fa fia fb fib
begin

```

```

notation fia (infixl ⟨ $\ominus_a$ ⟩ 65)

```

```

notation fib (infixl ⟨ $\ominus_b$ ⟩ 65)

```

```

end

```

Results

```

context cancel-comm-semigroup
begin

```

```

interpretation cancel-comm-semigroup-syntax ⟨proof⟩

```

```

lemma add-diff-cancel-right'[simp]: (a  $\oplus_a$  b)  $\ominus_a$  b = a
  ⟨proof⟩

```

sublocale *cancel*: *cancel-semigroup*

<proof>

lemmas *cancel-semigroup-axioms* = *cancel.cancel-semigroup-axioms*

lemma *add-diff-cancel-left[simp]*: $(c \oplus_a a) \ominus_a (c \oplus_a b) = a \ominus_a b$

<proof>

lemma *add-diff-cancel-right[simp]*: $(a \oplus_a c) \ominus_a (b \oplus_a c) = a \ominus_a b$

<proof>

lemma *diff-right-commute*: $a \ominus_a c \ominus_a b = a \ominus_a b \ominus_a c$

<proof>

end

context *cancel-comm-semigroup-pair*

begin

sublocale *cancel*: *cancel-semigroup-pair* *<proof>*

lemmas *cancel-semigroup-pair-axioms* = *cancel.cancel-semigroup-pair-axioms*

end

5.7 Extension of the theory *Lifting-Set*

context

includes *lifting-syntax*

begin

lemma *set-pred-eq-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total A*

shows

$((\text{rel-set } A \text{ ===> } (=)) \text{ ===> } (\text{rel-set } A \text{ ===> } (=)) \text{ ===> } (=))$

$(\lambda X Y. \forall s \subseteq \text{Collect } (\text{Domainp } A). X s = Y s)$

$((=)::['b \text{ set} \Rightarrow \text{bool}, 'b \text{ set} \Rightarrow \text{bool}] \Rightarrow \text{bool})$

<proof> **lemma** *vimage-fst-transfer-h*:

pred-prod (*Domainp A*) (*Domainp B*) *x =*

$(x \in \text{Collect } (\text{Domainp } A) \times \text{Collect } (\text{Domainp } B))$

<proof>

lemma *vimage-fst-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A right-total A right-total B*

shows

$((\text{rel-prod } A B \text{ ===> } A) \text{ ===> } \text{rel-set } A \text{ ===> } \text{rel-set } (\text{rel-prod } A B))$

$(\lambda f S. (f - ' S) \cap ((\text{Collect } (\text{Domainp } A)) \times (\text{Collect } (\text{Domainp } B))))$

vimage

<proof>

lemma *vimage-snd-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *right-total A bi-unique B right-total B*

shows

$((\text{rel-prod } A B \text{ ===> } B) \text{ ===> } \text{rel-set } B \text{ ===> } \text{rel-set } (\text{rel-prod } A B))$

$(\lambda f S. (f - ' S) \cap ((\text{Collect } (\text{Domainp } A)) \times (\text{Collect } (\text{Domainp } B))))$

vimage

<proof>

lemma *vimage-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique B right-total A*

shows

$((A \text{ ===> } B) \text{ ===> } (\text{rel-set } B) \text{ ===> } \text{rel-set } A)$

$(\lambda f s. (\text{vimage } f s) \cap (\text{Collect } (\text{Domainp } A))) (-')$

<proof>

lemma *pairwise-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A*

shows $((A \text{ ===> } A \text{ ===> } (=)) \text{ ===> } \text{rel-set } A \text{ ===> } (=))$ *pairwise pairwise*

<proof>

lemma *disjnt-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A*

shows $(\text{rel-set } A \text{ ===> } \text{rel-set } A \text{ ===> } (=))$ *disjnt disjnt*

<proof>

lemma *bij-betw-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique A bi-unique B*

shows $((A \text{ ===> } B) \text{ ===> } \text{rel-set } A \text{ ===> } \text{rel-set } B \text{ ===> } (=))$ *bij-betw bij-betw*

<proof>

end

5.8 Abstract semigroups on sets

5.8.1 Background

The results presented in this section were ported (with amendments and additions) from the theory *Groups* in the main library of Isabelle/HOL.

5.8.2 Binary operations

Abstract binary operation.

locale *binary-op-base-ow* =

fixes $U :: 'a \text{ set}$ **and** $f :: 'a \Rightarrow 'a \Rightarrow 'a$

locale *binary-op-ow* = *binary-op-base-ow* U **for** $U :: 'a \text{ set}$ **and** $f +$

assumes *op-closed*: $x \in U \Longrightarrow y \in U \Longrightarrow f x y \in U$

locale *binary-op-syntax-ow* = *binary-op-base-ow* U **for** $U :: 'a \text{ set}$ **and** f
begin

notation f (**infixl** $\langle \oplus_a \rangle$ 70)

end

Concrete syntax.

locale *plus-ow* = *binary-op-ow* U **plus** **for** $U :: 'a \text{ set}$ **and** *plus*

begin

notation *plus* (**infixl** $\langle +_a \rangle$ 65)

end

locale *minus-ow* = *binary-op-ow* U **minus** **for** $U :: 'a \text{ set}$ **and** *minus*

begin

notation *minus* (**infixl** $\langle -_a \rangle$ 65)

end

locale *times-ow* = *binary-op-ow* U **times** **for** $U :: 'a \text{ set}$ **and** *times*

begin

notation *times* (**infixl** $\langle *_a \rangle$ 70)

end

locale *divide-ow* = *binary-op-ow* U **divide** **for** $U :: 'a \text{ set}$ **and** *divide*

begin

notation *divide* (**infixl** $\langle '/_a \rangle$ 70)

end

Pairs.

locale *binary-op-base-pair-ow* =

alg_a: *binary-op-base-ow* U_a f_a + *alg_b*: *binary-op-base-ow* U_b f_b

for $U_a :: 'a \text{ set}$ **and** f_a **and** $U_b :: 'b \text{ set}$ **and** f_b

```

locale binary-op-pair-ow = alga: binary-op-ow Ua fa + algb: binary-op-ow Ub fb
  for Ua :: 'a set and fa and Ub :: 'b set and fb
begin

  sublocale binary-op-base-pair-ow Ua fa Ub fb <proof>
  sublocale rev: binary-op-base-pair-ow Ub fb Ua fa <proof>

end

locale binary-op-pair-syntax-ow = binary-op-base-pair-ow Ua fa Ub fb
  for Ua :: 'a set and fa and Ub :: 'b set and fb
begin

  notation fa (infixl <⊕a> 70)
  notation fb (infixl <⊕b> 70)

```

end

Results

```

context binary-op-ow

```

```

begin

```

```

interpretation binary-op-syntax-ow <proof>

```

```

lemma op-closed'[simp]:  $\forall x \in U. \forall y \in U. x \oplus_a y \in U$  <proof>

```

```

tts-register-sbts <⊕a> | U <proof>

```

end

5.8.3 Simple semigroups

Definitions

Abstract semigroup.

```

locale semigroup-ow = binary-op-ow U f for U :: 'a set and f +
  assumes assoc[tts-ac-simps]:
   $[[ a \in U; b \in U; c \in U ]] \implies f (f a b) c = f a (f b c)$ 

```

```

locale semigroup-syntax-ow = binary-op-syntax-ow U f for U :: 'a set and f

```

Concrete syntax.

```

locale semigroup-add-ow = semigroup-ow U plus for U :: 'a set and plus
begin

```

```

  sublocale plus-ow U plus <proof>

```

end

```

locale semigroup-mult-ow = semigroup-ow U times for U :: 'a set and times
begin

```

```

  sublocale times-ow U times <proof>

```

end

Pairs.

```

locale semigroup-pair-ow = alga: semigroup-ow Ua fa + algb: semigroup-ow Ub fb
  for Ua :: 'a set and fa and Ub :: 'b set and fb
begin

```

```

sublocale binary-op-pair-ow Ua fa Ub fb <proof>
sublocale rev: semigroup-pair-ow Ub fb Ua fa <proof>

```

```

end

```

```

locale semigroup-pair-syntax-ow = binary-op-pair-syntax-ow Ua fa Ub fb
  for Ua :: 'a set and fa and Ub :: 'b set and fb

```

Transfer rules

```

lemma semigroup-ow[ud-with]: semigroup = semigroup-ow UNIV
  <proof>

```

```

lemma semigroup-pair-ow[ud-with]:
  semigroup-pair = (λfa fb. semigroup-pair-ow UNIV fa UNIV fb)
  <proof>

```

```

context
  includes lifting-syntax
begin

```

```

lemma semigroup-ow-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique A right-total A
  shows
    (rel-set A ==> (A ==> A ==> A) ==> (=)) semigroup-ow semigroup-ow
  <proof>

```

```

lemma semigroup-pair-ow-transfer[transfer-rule]:
  assumes [transfer-rule]:
    bi-unique A right-total A bi-unique B right-total B
  shows
    (
      rel-set A ==> (A ==> A ==> A) ==>
      rel-set B ==> (B ==> B ==> B) ==>
      (=)
    )
    semigroup-pair-ow semigroup-pair-ow
  <proof>

```

```

end

```

5.8.4 Commutative semigroups

Definitions

Abstract commutative semigroup.

```

locale comm-semigroup-ow = semigroup-ow U f for U :: 'a set and f +
  assumes commute[tts-ac-simps]: a ∈ U ==> b ∈ U ==> f a b = f b a

```

```

locale comm-semigroup-syntax-ow = semigroup-syntax-ow U f
  for U :: 'a set and f

```

Concrete syntax.

```

locale comm-semigroup-add-ow = comm-semigroup-ow U plus

```

```

for  $U :: 'a$  set and plus
begin

sublocale semigroup-add-ow  $U$  plus  $\langle$ proof $\rangle$ 

end

locale comm-semigroup-mult-ow = comm-semigroup-ow  $U$  times
  for  $U :: 'a$  set and times
begin

sublocale semigroup-mult-ow  $U$  times  $\langle$ proof $\rangle$ 

end

```

Pairs.

```

locale comm-semigroup-pair-ow =
  alga: comm-semigroup-ow  $U_a$   $f_a$  + algb: comm-semigroup-ow  $U_b$   $f_b$ 
  for  $U_a :: 'a$  set and  $f_a$  and  $U_b :: 'b$  set and  $f_b$ 
begin

sublocale semigroup-pair-ow  $U_a$   $f_a$   $U_b$   $f_b$   $\langle$ proof $\rangle$ 
sublocale rev: comm-semigroup-pair-ow  $U_b$   $f_b$   $U_a$   $f_a$   $\langle$ proof $\rangle$ 

end

locale comm-semigroup-pair-syntax-ow = semigroup-pair-syntax-ow  $U_a$   $f_a$   $U_b$   $f_b$ 
  for  $U_a :: 'a$  set and  $f_a$  and  $U_b :: 'b$  set and  $f_b$ 

```

Transfer rules

```

lemma comm-semigroup-ow[ud-with]: comm-semigroup = comm-semigroup-ow UNIV
   $\langle$ proof $\rangle$ 

```

```

lemma comm-semigroup-pair-ow[ud-with]:
  comm-semigroup-pair =  $(\lambda f_a f_b. \text{comm-semigroup-pair-ow UNIV } f_a \text{ UNIV } f_b)$ 
   $\langle$ proof $\rangle$ 

```

context

```

includes lifting-syntax
begin

```

```

lemma comm-semigroup-ow-transfer[transfer-rule]:
  assumes [transfer-rule]: bi-unique  $A$  right-total  $A$ 
  shows
    (rel-set  $A$   $\implies$  ( $A \implies A \implies A$ )  $\implies$  (=))
    comm-semigroup-ow comm-semigroup-ow
   $\langle$ proof $\rangle$ 

```

```

lemma comm-semigroup-pair-ow-transfer[transfer-rule]:
  assumes [transfer-rule]:
    bi-unique  $A$  right-total  $A$  bi-unique  $B$  right-total  $B$ 
  shows
    (
      rel-set  $A$   $\implies$  ( $A \implies A \implies A$ )  $\implies$ 
      rel-set  $B$   $\implies$  ( $B \implies B \implies B$ )  $\implies$ 
      (=)
    )

```

comm-semigroup-pair-ow comm-semigroup-pair-ow
 ⟨proof⟩

end

Relativization

context *comm-semigroup-ow*
begin

interpretation *comm-semigroup-syntax-ow* ⟨proof⟩

tts-context

tts: (?'a to U)

substituting *comm-semigroup-ow-axioms*

eliminating through *auto*

begin

tts-lemma *left-commute:*

assumes $b \in U$

and $a \in U$

and $c \in U$

shows $b \oplus_a (a \oplus_a c) = a \oplus_a (b \oplus_a c)$

is *comm-semigroup.left-commute*⟨proof⟩

end

end

5.8.5 Cancellative semigroups

Definitions

Abstract cancellative semigroup.

locale *cancel-semigroup-ow* = *semigroup-ow* U f **for** U :: 'a set **and** f +

assumes *add-left-imp-eq:*

$\llbracket a \in U; b \in U; c \in U; f a b = f a c \rrbracket \implies b = c$

assumes *add-right-imp-eq:*

$\llbracket b \in U; a \in U; c \in U; f b a = f c a \rrbracket \implies b = c$

locale *cancel-semigroup-syntax-ow* = *semigroup-syntax-ow* U f

for U :: 'a set **and** f

Concrete syntax.

locale *cancel-semigroup-add-ow* = *cancel-semigroup-ow* U *plus*

for U :: 'a set **and** *plus*

begin

sublocale *semigroup-add-ow* U *plus* ⟨proof⟩

end

locale *cancel-semigroup-mult-ow* = *cancel-semigroup-ow* U *times*

for U :: 'a set **and** *times*

begin

sublocale *semigroup-mult-ow* U *times* ⟨proof⟩

end

Pairs.

locale *cancel-semigroup-pair-ow* =
alg_a: *cancel-semigroup-ow* $U_a f_a$ + *alg_b*: *cancel-semigroup-ow* $U_b f_b$
for $U_a :: 'a$ set **and** f_a **and** $U_b :: 'b$ set **and** f_b
begin

sublocale *semigroup-pair-ow* $U_a f_a$ $U_b f_b$ *<proof>*
sublocale *rev: cancel-semigroup-pair-ow* $U_b f_b$ $U_a f_a$ *<proof>*

end

locale *cancel-semigroup-pair-syntax-ow* = *semigroup-pair-syntax-ow* $U_a f_a$ $U_b f_b$
for $U_a :: 'a$ set **and** f_a **and** $U_b :: 'b$ set **and** f_b

Transfer rules

lemma *cancel-semigroup-ow[ud-with]*:
cancel-semigroup = *cancel-semigroup-ow* *UNIV*
<proof>

lemma *cancel-semigroup-pair-ow[ud-with]*:
cancel-semigroup-pair = $(\lambda f_a f_b. \text{cancel-semigroup-pair-ow } UNIV f_a UNIV f_b)$
<proof>

context

includes *lifting-syntax*

begin

lemma *cancel-semigroup-ow-transfer[transfer-rule]*:
assumes [*transfer-rule*]: *bi-unique* A *right-total* A
shows
 $(\text{rel-set } A \implies (A \implies A \implies A) \implies (=))$
cancel-semigroup-ow cancel-semigroup-ow
<proof>

lemma *cancel-semigroup-pair-ow-transfer[transfer-rule]*:
assumes [*transfer-rule*]:
bi-unique A *right-total* A *bi-unique* B *right-total* B
shows
 $($
 $\text{rel-set } A \implies (A \implies A \implies A) \implies$
 $\text{rel-set } B \implies (B \implies B \implies B) \implies$
 $(=)$
 $) \text{cancel-semigroup-pair-ow cancel-semigroup-pair-ow}$
<proof>

end

Relativization

context *cancel-semigroup-ow*

begin

interpretation *cancel-semigroup-syntax-ow* *<proof>*

tts-context

```

tts: (?'a to U)
rewriting ctr-simps
substituting cancel-semigroup-ow-axioms
eliminating through auto
begin

tts-lemma add-right-cancel:
  assumes b ∈ U and a ∈ U and c ∈ U
  shows (b ⊕a a = c ⊕a a) = (b = c)
  is cancel-semigroup.add-right-cancel{proof}

tts-lemma add-left-cancel:
  assumes a ∈ U and b ∈ U and c ∈ U
  shows (a ⊕a b = a ⊕a c) = (b = c)
  is cancel-semigroup.add-left-cancel{proof}

tts-lemma inj-on-add':
  assumes a ∈ U and A ⊆ U
  shows inj-on (λb. b ⊕a a) A
  is cancel-semigroup.inj-on-add'{proof}

tts-lemma inj-on-add:
  assumes a ∈ U and A ⊆ U
  shows inj-on ((⊕a) a) A
  is cancel-semigroup.inj-on-add{proof}

tts-lemma bij-betw-add:
  assumes a ∈ U and A ⊆ U and B ⊆ U
  shows bij-betw ((⊕a) a) A B = ((⊕a) a ' A = B)
  is cancel-semigroup.bij-betw-add{proof}

end

end

```

5.8.6 Cancellative commutative semigroups

Definitions

Abstract cancellative commutative semigroups.

```

locale cancel-comm-semigroup-ow = comm-semigroup-ow U f + binary-op-ow U fi
  for U :: 'a set and f fi +
  assumes add-diff-cancel-left'[simp]: [[ a ∈ U; b ∈ U ]] ⇒ fi (f a b) a = b
  and diff-diff-add[tts-algebra-simps, tts-field-simps]:
  [[ a ∈ U; b ∈ U; c ∈ U ]] ⇒ fi (fi a b) c = fi a (f b c)

```

```

locale cancel-comm-semigroup-syntax-ow =
  comm-semigroup-syntax-ow U f + binary-op-base-ow U fi
  for U :: 'a set and f fi
begin

```

```

notation fi (infixl <⊕a> 65)

```

```

end

```

Concrete syntax.

```

locale cancel-comm-semigroup-add-ow = cancel-comm-semigroup-ow U plus minus
  for U :: 'a set and plus minus

```

begin

sublocale *comm-semigroup-add-ow* U *plus* \langle *proof* \rangle

sublocale *minus-ow* U *minus* \langle *proof* \rangle

end

locale *cancel-comm-semigroup-mult* = *cancel-comm-semigroup-ow* U *times* *divide*

for $U :: 'a$ set **and** *times* *divide*

begin

sublocale *comm-semigroup-mult-ow* U *times* \langle *proof* \rangle

sublocale *divide-ow* U *divide* \langle *proof* \rangle

end

Pairs.

locale *cancel-comm-semigroup-pair-ow* =

alg_a: *cancel-comm-semigroup-ow* U_a f_a f_{i_a} +

alg_b: *cancel-comm-semigroup-ow* U_b f_b f_{i_b}

for $U_a :: 'a$ set **and** f_a f_{i_a} **and** $U_b :: 'b$ set **and** f_b f_{i_b}

begin

sublocale *comm-semigroup-pair-ow* U_a f_a U_b f_b \langle *proof* \rangle

sublocale *rev*: *cancel-comm-semigroup-pair-ow* U_b f_b f_{i_b} U_a f_a f_{i_a} \langle *proof* \rangle

end

locale *cancel-comm-semigroup-pair-syntax-ow* =

comm-semigroup-pair-syntax-ow U_a f_a U_b f_b +

binary-op-ow U_a f_{i_a} +

binary-op-ow U_b f_{i_b}

for $U_a :: 'a$ set **and** f_a f_{i_a} **and** $U_b :: 'b$ set **and** f_b f_{i_b}

begin

notation f_{i_a} (**infixl** $\langle \ominus_a \rangle$ 65)

notation f_{i_b} (**infixl** $\langle \ominus_b \rangle$ 65)

end

Transfer rules

lemma *cancel-comm-semigroup-ow*[*ud-with*]:

cancel-comm-semigroup = *cancel-comm-semigroup-ow* *UNIV*

\langle *proof* \rangle

lemma *cancel-comm-semigroup-pair-ow*[*ud-with*]:

cancel-comm-semigroup-pair =

$(\lambda f_a f_b f_{i_b}. \text{cancel-comm-semigroup-pair-ow } \text{UNIV } f_a f_{i_a} \text{ UNIV } f_b f_{i_b})$

\langle *proof* \rangle

context

includes *lifting-syntax*

begin

lemma *cancel-comm-semigroup-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]: *bi-unique* A *right-total* A

shows

(*rel-set* $A \implies (A \implies A \implies A) \implies (A \implies A \implies A) \implies (=)$)
cancel-comm-semigroup-ow cancel-comm-semigroup-ow
 ⟨*proof*⟩

lemma *cancel-comm-semigroup-pair-ow-transfer*[*transfer-rule*]:

assumes [*transfer-rule*]:

bi-unique A right-total A bi-unique B right-total B

shows

(
rel-set A \implies (A \implies A \implies A) \implies (A \implies A \implies A) \implies
rel-set B \implies (B \implies B \implies B) \implies (B \implies B \implies B) \implies
 (=)
) *cancel-comm-semigroup-pair-ow cancel-comm-semigroup-pair-ow*
 ⟨*proof*⟩

end

Relativization

context *cancel-comm-semigroup-ow*

begin

interpretation *cancel-comm-semigroup-syntax-ow* ⟨*proof*⟩

tts-context

tts: (*?'a to U*)

sbterms: (*(?f::?'a \Rightarrow ?'a \Rightarrow ?'a) to f*)

and (*(?fi::?'a \Rightarrow ?'a \Rightarrow ?'a) to fi*)

rewriting *ctr-simps*

substituting *cancel-comm-semigroup-ow-axioms*

eliminating through *auto*

begin

tts-lemma *add-diff-cancel-right'*:

assumes *a \in U and b \in U*

shows *a \oplus_a b \ominus_a b = a*

is *cancel-comm-semigroup.add-diff-cancel-right'*⟨*proof*⟩

tts-lemma *add-diff-cancel-right*:

assumes *a \in U and c \in U and b \in U*

shows *a \oplus_a c \ominus_a b \oplus_a c = a \ominus_a b*

is *cancel-comm-semigroup.add-diff-cancel-right*⟨*proof*⟩

tts-lemma *add-diff-cancel-left*:

assumes *c \in U and a \in U and b \in U*

shows *c \oplus_a a \ominus_a c \oplus_a b = a \ominus_a b*

is *cancel-comm-semigroup.add-diff-cancel-left*⟨*proof*⟩

tts-lemma *diff-right-commute*:

assumes *a \in U and c \in U and b \in U*

shows *a \ominus_a c \ominus_a b = a \ominus_a b \ominus_a c*

is *cancel-comm-semigroup.diff-right-commute*⟨*proof*⟩

tts-lemma *cancel-semigroup-axioms*:

assumes *U \neq {}*

shows *cancel-semigroup-ow U (\oplus_a)*

is *cancel-comm-semigroup.cancel-semigroup-axioms*⟨*proof*⟩

end

sublocale *cancel-semigroup-ow*
 \langle *proof* \rangle

end

context *cancel-comm-semigroup-pair-ow*
begin

sublocale *cancel-semigroup-pair-ow* \langle *proof* \rangle

end

Bibliography

- [1] Isabelle/HOL Standard Library, 2020. URL <https://isabelle.in.tum.de/website-Isabelle2020/dist/library/HOL/HOL/index.html>.
- [2] Isabelle/HOL Analysis, 2020. URL <https://isabelle.in.tum.de/website-Isabelle2020/dist/library/HOL/HOL-Analysis/index.html>.
- [3] C. Ballarin. Locales and Locale Expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs*, volume 3085, pages 34–50. Springer, Heidelberg, 2004. ISBN 978-3-540-22164-7.
- [4] C. Ballarin. Locales: A Module System for Mathematical Theories. *Journal of Automated Reasoning*, 52(2):123–153, 2014.
- [5] C. Ballarin, editor. *The Isabelle/HOL Algebra Library*. 2020. URL <https://isabelle.in.tum.de/website-Isabelle2020/dist/library/HOL/HOL-Algebra/document.pdf>.
- [6] A. J. Cain. *Nine Chapters on the Semigroup Art*. Lisbon, 2019.
- [7] A. Chaieb and M. Wenzel. Context Aware Calculation and Deduction: Ring Equalities Via Gröbner Bases in Isabelle. In M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, editors, *Towards Mechanized Mathematical Assistants*, volume 4573, pages 27–39. Springer, Heidelberg, 2007. ISBN 978-3-540-73083-5.
- [8] J. Divasón, O. Kunčar, R. Thiemann, and A. Yamada. Perron-Frobenius Theorem for Spectral Radius Analysis. *Archive of Formal Proofs*, 2016.
- [9] J. Gilcher, A. Lochbihler, and D. Traytel. Conditional Parametricity in Isabelle/HOL. In *TABLEAUX/FroCoS/ITP*, Brasília, Brazil, 2017.
- [10] F. Haftmann and M. Wenzel. Constructive Type Classes in Isabelle. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs*, volume 4502, pages 160–174. Springer, Heidelberg, 2007. ISBN 978-3-540-74464-1.
- [11] B. Huffman and O. Kunčar. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In G. Gonthier and M. Norrish, editors, *Certified Programs and Proofs*, volume 8307, pages 131–146. Springer, Heidelberg, 2013. ISBN 978-3-319-03545-1.
- [12] F. Immler. Automation for unloading definitions, 2019. URL <http://isabelle.in.tum.de/repos/isabelle/rev/ab5a8a2519b0>.
- [13] F. Immler. Re: [isabelle] Several questions in relation to a use case for "types to sets", 2019. URL <https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2019-January/msg00072.html>.
- [14] F. Immler and B. Zhan. Smooth Manifolds and Types to Sets for Linear Algebra in Isabelle/HOL. In A. Mahboubi and M. O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal*, CPP 2019, pages 65–77. ACM, New York, 2019. ISBN 978-1-4503-6222-1.

- [15] F. Kammüller, M. Wenzel, and L. C. Paulson. Locales A Sectioning Concept for Isabelle. In Y. Bertot, G. Dowek, L. Théry, A. Hirschowitz, and C. Paulin-Mohring, editors, *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, pages 149–165, Heidelberg, 1999. Springer. ISBN 978-3-540-48256-7.
- [16] K. Kappelmann, L. Bulwahn, and S. Willenbrink. SpecCheck - Specification-Based Testing for Isabelle/ML. *Archive of Formal Proofs*, 2021.
- [17] A. Krauss and A. Schropp. A Mechanized Translation from Higher-Order Logic to Set Theory. In M. Kaufmann and L. C. Paulson, editors, *Interactive Theorem Proving*, volume 6172, pages 323–338. Springer, Heidelberg, 2010. ISBN 978-3-642-14051-8.
- [18] O. Kunčar. *Types, Abstraction and Parametric Polymorphism in Higher-Order Logic*. PhD thesis, Technische Universität München, Munich, 2015.
- [19] O. Kunčar and A. Popescu. From Types to Sets by Local Type Definitions in Higher-Order Logic. In J. C. Blanchette and S. Merz, editors, *Interactive Theorem Proving*, volume 9807, pages 200–218. Springer, Heidelberg, 2016. ISBN 978-3-319-43144-4.
- [20] O. Kunčar and A. Popescu. Comprehending Isabelle/HOL’s Consistency. In H. Yang, editor, *Programming Languages and Systems*, volume 10201, pages 724–749. Springer, Heidelberg, 2017. ISBN 978-3-662-54433-4.
- [21] O. Kunčar and A. Popescu. From Types to Sets by Local Type Definition in Higher-Order Logic. *Journal of Automated Reasoning*, 62(2):237–260, 2019.
- [22] P. Lammich. Automatic Data Refinement. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving*, pages 84–99, Heidelberg, 2013. Springer. ISBN 978-3-642-39634-2.
- [23] A. Maletzky. Hilbert’s Nullstellensatz. *Archive of Formal Proofs*, 2019.
- [24] M. Milehins. Conditional Transfer Rule. *Archive of Formal Proofs*, 2021.
- [25] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (revised)*. MIT Press, Cambridge, Massachusetts, 1997. ISBN 978-0-262-63181-5.
- [26] T. Nipkow and G. Snelting. Type Classes and Overloading Resolution via Order-Sorted Unification. In J. Hughes, editor, *Functional Programming Languages and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 1–14. Springer, Heidelberg, 1991. ISBN 978-3-540-47599-6.
- [27] L. C. Paulson. Natural Deduction as Higher-Order Resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986.
- [28] L. C. Paulson. The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.
- [29] C. Urban. *The Isabelle Cookbook: A Gentle Tutorial for Programming Isabelle/ML*. 2019.
- [30] M. Wenzel. Type Classes and Overloading in Higher-Order Logic. In E. L. Gunter and A. P. Felty, editors, *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, pages 307–322. Springer, Heidelberg, 1997. ISBN 978-3-540-69526-4.
- [31] M. Wenzel. Isar — A Generic Interpretative Approach to Readable Formal Proof Documents. In Y. Bertot, G. Dowek, L. Théry, A. Hirschowitz, and C. Paulin-Mohring, editors, *Theorem Proving in Higher Order Logics*, volume 1690, pages 167–183. Springer, Heidelberg, 1999. ISBN 978-3-540-66463-5.

- [32] M. Wenzel. Isabelle/Isar — a Generic Framework for Human-Readable Proof Documents. *Studies in Logic, Grammar and Rhetoric*, 10(23):277–297, 2007.
- [33] M. Wenzel. The Isabelle/Isar Implementation. 2019.
- [34] M. Wenzel. The Isabelle/Isar Reference Manual. 2019.
- [35] M. M. Wenzel. *Isabelle/Isar — A Versatile Environment for Human-Readable Formal Proof Documents*. PhD thesis, Technische Universität München, Munich, 2001.