Sorted Terms*

Akihisa Yamada

National Institute of Advanced Industrial Science and Technology, Japan

René Thiemann

University of Innsbruck, Austria

June 3, 2024

Abstract

This entry provides a basic library for many-sorted terms and algebras. We view sorted sets just as partial maps from elements to sorts, and define sorted set of terms reusing the data type from the existing library of (unsorted) first order terms. All the existing functionality, such as substitutions and contexts, can be reused without any modifications. We provide predicates stating what substitutions or contexts are considered sorted, and prove facts that they preserve sorts as expected.

Contents

*This research was partly supported by the Austrian Science Fund (FWF) project I 5943.

1 Introduction

This entry extends the First-Order Terms [1] entry with many-sorted terms. Instead of defining a new datatype for sorted terms, we just define sorted sets over the existing datatype of unsorted terms. We do not even introduce our type for sorted sets: we just view sorted sets as partial maps from elements to their sorts.

Part of the entry is presented in [2].

theory *Sorted-Sets*
  imports
    Main
    HOL−Library.FuncSet
    HOL−Library.Monad-Syntax
    Complete-Non-Orders.Binary-Relations
begin

2 Auxiliary Lemmas

lemma *ex-set-conv-ex-nth*:
  $(\exists\, x \in set\ xs.\ P\ x) = (\exists\, i.\ i < length\ xs \wedge P\ (xs\ !\ i))$
  ⟨*proof*⟩

lemma *Ball-Pair-conv*: $(\forall\,(x,y){\in}R.\ P\ x\ y) \longleftrightarrow (\forall\, x\ y.\ (x,y) \in R \longrightarrow P\ x\ y)$ ⟨*proof*⟩

lemma *Some-eq-bind-conv*: $(Some\ x = f \gg\!= g) = (\exists\, y.\ f = Some\ y \wedge g\ y = Some\ x)$
  ⟨*proof*⟩

lemma *length-le-nth-append*: $length\ xs \le n \Longrightarrow (xs@ys)!n = ys!(n{-}length\ xs)$
  ⟨*proof*⟩

lemma *list-all2-same-left*:
$\forall\, a' \in set\ as.\ a' = a \Longrightarrow list\text{-}all2\ r\ as\ bs \longleftrightarrow length\ as = length\ bs \wedge (\forall\, b \in set\ bs.\ r\ a\ b)$

⟨*proof*⟩

lemma *list-all2-same-leftI*:
$\forall a' \in set\ as.\ a' = a \Longrightarrow length\ as = length\ bs \Longrightarrow \forall b \in set\ bs.\ r\ a\ b \Longrightarrow list\text{-}all2\ r\ as\ bs$
⟨*proof*⟩

lemma *list-all2-same-right*:
$\forall b' \in set\ bs.\ b' = b \Longrightarrow list\text{-}all2\ r\ as\ bs \longleftrightarrow length\ as = length\ bs \wedge (\forall a \in set\ as.\ r\ a\ b)$
⟨*proof*⟩

lemma *list-all2-same-rightI*:
$\forall b' \in set\ bs.\ b' = b \Longrightarrow length\ as = length\ bs \Longrightarrow \forall a \in set\ as.\ r\ a\ b \Longrightarrow list\text{-}all2\ r\ as\ bs$
⟨*proof*⟩

lemma *list-all2-all-all*:
$\forall a \in set\ as.\ \forall b \in set\ bs.\ r\ a\ b \Longrightarrow list\text{-}all2\ r\ as\ bs \longleftrightarrow length\ as = length\ bs$
⟨*proof*⟩

lemma *list-all2-indep1*:
$list\text{-}all2\ (\lambda a\ b.\ P\ b)\ as\ bs \longleftrightarrow length\ as = length\ bs \wedge (\forall b \in set\ bs.\ P\ b)$
⟨*proof*⟩

lemma *list-all2-indep2*:
$list\text{-}all2\ (\lambda a\ b.\ P\ a)\ as\ bs \longleftrightarrow length\ as = length\ bs \wedge (\forall a \in set\ as.\ P\ a)$
⟨*proof*⟩

lemma *list-all2-replicate*[*simp*]:
$list\text{-}all2\ r\ (replicate\ n\ x)\ ys \longleftrightarrow length\ ys = n \wedge (\forall y \in set\ ys.\ r\ x\ y)$
$list\text{-}all2\ r\ xs\ (replicate\ n\ y) \longleftrightarrow length\ xs = n \wedge (\forall x \in set\ xs.\ r\ x\ y)$
⟨*proof*⟩

lemma *list-all2-choice-nth*: **assumes** $\forall i < length\ xs.\ \exists y.\ r\ (xs!i)\ y$ **shows** $\exists ys.\ list\text{-}all2\ r\ xs\ ys$
⟨*proof*⟩

lemma *list-all2-choice*: $\forall x \in set\ xs.\ \exists y.\ r\ x\ y \Longrightarrow \exists ys.\ list\text{-}all2\ r\ xs\ ys$
⟨*proof*⟩

lemma *list-all2-concat*:
$list\text{-}all2\ (list\text{-}all2\ r)\ ass\ bss \Longrightarrow list\text{-}all2\ r\ (concat\ ass)\ (concat\ bss)$
⟨*proof*⟩

lemma *those-eq-None*[*simp*]: $those\ as = None \longleftrightarrow None \in set\ as$ ⟨*proof*⟩

lemma *those-eq-Some*[*simp*]: $those\ xos = Some\ xs \longleftrightarrow xos = map\ Some\ xs$
⟨*proof*⟩

lemma *those-map-Some*[*simp*]: *those* (*map Some xs*) = *Some xs* ⟨*proof*⟩

lemma *those-append*:
those (*as* @ *bs*) = *do* {*xs* ← *those as*; *ys* ← *those bs*; *Some* (*xs*@*ys*)}
⟨*proof*⟩

lemma *those-Cons*:
those (*a*#*as*) = *do* {*x* ← *a*; *xs* ← *those as*; *Some* (*x* # *xs*)}
⟨*proof*⟩

lemma *map-singleton-o*[*simp*]: $(\lambda x.\ [x]) \circ f = (\lambda x.\ [f\ x])$ ⟨*proof*⟩

lemmas *list-3-cases* = *remdups-adj.cases*

lemma *in-set-updateD*: $x \in set\ (xs[n := y]) \Longrightarrow x \in set\ xs \lor x = y$
⟨*proof*⟩

lemma *map-nth'*: $length\ xs = n \Longrightarrow map\ (nth\ xs)\ [0..<n] = xs$
⟨*proof*⟩

lemma *product-lists-map-map*: *product-lists* (*map* (*map f*) *xss*) = *map* (*map f*) (*product-lists xss*)
⟨*proof*⟩

lemma (**in** *monoid-add*) *sum-list-concat*: *sum-list* (*concat xs*) = *sum-list* (*map sum-list xs*)
⟨*proof*⟩

context *semiring-1* **begin**

lemma *prod-list-map-sum-list-distrib*:
shows *prod-list* (*map sum-list xss*) = *sum-list* (*map prod-list* (*product-lists xss*))
⟨*proof*⟩

lemma *prod-list-sum-list-distrib*:
$(\prod xs \leftarrow xss.\ \sum x \leftarrow xs.\ f\ x) = (\sum xs \leftarrow product\text{-}lists\ xss.\ \prod x \leftarrow xs.\ f\ x)$
⟨*proof*⟩

end

lemma *ball-set-bex-set-distrib*:
$(\forall xs \in set\ xss.\ \exists x \in set\ xs.\ f\ x) \longleftrightarrow (\exists xs \in set\ (product\text{-}lists\ xss).\ \forall x \in set\ xs.\ f\ x)$
⟨*proof*⟩

lemma *bex-set-ball-set-distrib*:
$(\exists xs \in set\ xss.\ \forall x \in set\ xs.\ f\ x) \longleftrightarrow (\forall xs \in set\ (product\text{-}lists\ xss).\ \exists x \in set\ xs.\ f\ x)$
⟨*proof*⟩

declare *upt-Suc*[*simp del*]

lemma *map-nth-Cons*: $map\ (nth\ (x\#xs))\ [0..<n] = (case\ n\ of\ 0 \Rightarrow [] \mid Suc\ n \Rightarrow x\ \#\ map\ (nth\ xs)\ [0..<n])$
⟨*proof*⟩

lemma *upt-0-Suc-Cons*: $[0..<Suc\ i] = 0\ \#\ map\ Suc\ [0..<i]$
⟨*proof*⟩

lemma *upt-map-add*: $i \le j \Longrightarrow [i..<j] = map\ (\lambda k.\ k + i)\ [0..<j-i]$
⟨*proof*⟩

lemma *map-nth-append*:
$map\ (nth\ (xs\ @\ ys))\ [0..<n] =$
$(if\ n < length\ xs\ then\ map\ (nth\ xs)\ [0..<n]\ else\ xs\ @\ map\ (nth\ ys)\ [0..<n-length\ xs])$
⟨*proof*⟩

lemma *all-dom*: $(\forall x \in dom\ f.\ P\ x) \longleftrightarrow (\forall x\ y.\ f\ x = Some\ y \longrightarrow P\ x)$ ⟨*proof*⟩

lemma *trancl-Collect*: $\{(x,y).\ r\ x\ y\}^+ = \{(x,y).\ tranclp\ r\ x\ y\}$
⟨*proof*⟩

lemma *restrict-submap*[*intro*!]: $A\ |`\ S \subseteq_m A$
⟨*proof*⟩

lemma *restrict-map-mono-left*: $A \subseteq_m A' \Longrightarrow A\ |`\ S \subseteq_m A'\ |`\ S$
and *restrict-map-mono-right*: $S \subseteq S' \Longrightarrow A\ |`\ S \subseteq_m A\ |`\ S'$
⟨*proof*⟩

3 Sorted Sets and Maps

declare *domIff*[*iff del*]

We view sorted sets just as partial maps from elements to their sorts. We just introduce the following notation:

definition *hastype* (((-) :/ (-) *in*/ (-)) [*50*,*61*,*51*]*50*)
where $a : \sigma\ in\ A \equiv A\ a = Some\ \sigma$

abbreviation *all-hastype* $\sigma\ A\ P \equiv \forall a.\ a : \sigma\ in\ A \longrightarrow P\ a$
abbreviation *ex-hastype* $\sigma\ A\ P \equiv \exists a.\ a : \sigma\ in\ A \wedge P\ a$

syntax
all-hastype :: *'pttrn* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (∀ - :/ - *in*/ -./ - [*50*,*51*,*51*,*10*]*10*)
ex-hastype :: *'pttrn* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* (∃ - :/ - *in*/ -./ - [*50*,*51*,*51*,*10*]*10*)

translations
$\forall a : \sigma\ in\ A.\ e \rightleftharpoons CONST\ all\text{-}hastype\ \sigma\ A\ (\lambda a.\ e)$
$\exists a : \sigma\ in\ A.\ e \rightleftharpoons CONST\ ex\text{-}hastype\ \sigma\ A\ (\lambda a.\ e)$

lemmas *hastypeI* = *hastype-def*[*unfolded atomize-eq*, *THEN iffD2*]
lemmas *hastypeD*[*dest*] = *hastype-def*[*unfolded atomize-eq*, *THEN iffD1*]
lemmas *eq-Some-iff-hastype* = *hastype-def*[*symmetric*]

lemma *has-same-type*: **assumes** $a : \sigma\ in\ A$ **shows** $a : \sigma'\ in\ A \longleftrightarrow \sigma' = \sigma$
⟨*proof*⟩

lemma *sset-eqI*: **assumes** $(\bigwedge a\ \sigma.\ a : \sigma\ in\ A \longleftrightarrow a : \sigma\ in\ B)$ **shows** $A = B$
⟨*proof*⟩

lemma *in-dom-iff-ex-type*: $a \in dom\ A \longleftrightarrow (\exists\sigma.\ a : \sigma\ in\ A)$ ⟨*proof*⟩

lemma *in-dom-hastypeE*: $a \in dom\ A \Longrightarrow (\bigwedge\sigma.\ a : \sigma\ in\ A \Longrightarrow thesis) \Longrightarrow thesis$
⟨*proof*⟩

lemma *hastype-imp-dom*[*simp*]: $a : \sigma\ in\ A \Longrightarrow a \in dom\ A$ ⟨*proof*⟩

lemma *untyped-imp-not-hastype*: $A\ a = None \Longrightarrow \neg\ a : \sigma\ in\ A$ ⟨*proof*⟩

lemma *nex-hastype-iff*: $(\nexists\sigma.\ a : \sigma\ in\ A) \longleftrightarrow A\ a = None$ ⟨*proof*⟩

lemma *all-dom-iff-all-hastype*: $(\forall x \in dom\ A.\ P\ x) \longleftrightarrow (\forall x\ \sigma.\ x : \sigma\ in\ A \longrightarrow P$
$x)$
⟨*proof*⟩

abbreviation *hastype-list* (((-) $:_l$/ (-) *in*/ (-)) [*50*,*61*,*51*]*50*)
where $as :_l \sigma s\ in\ A \equiv list\text{-}all2\ (\lambda a\ \sigma.\ a : \sigma\ in\ A)\ as\ \sigma s$

lemma *has-same-type-list*:
$as :_l \sigma s\ in\ A \Longrightarrow as :_l \sigma s'\ in\ A \longleftrightarrow \sigma s' = \sigma s$
⟨*proof*⟩

lemma *hastype-list-iff-those*: $as :_l \sigma s\ in\ A \longleftrightarrow those\ (map\ A\ as) = Some\ \sigma s$
⟨*proof*⟩

lemmas *hastype-list-imp-those*[*simp*] = *hastype-list-iff-those*[*THEN iffD1*]

lemma *hastype-list-imp-lists-dom*: $xs :_l \sigma s\ in\ A \Longrightarrow xs \in lists\ (dom\ A)$
⟨*proof*⟩

lemma *subsset*: $A \subseteq_m A' \longleftrightarrow (\forall a\ \sigma.\ a : \sigma\ in\ A \longrightarrow a : \sigma\ in\ A')$
⟨*proof*⟩

lemmas *subssetI* = *subsset*[*THEN iffD2*, *rule-format*]
lemmas *subssetD* = *subsset*[*THEN iffD1*, *rule-format*]

lemma *subsset-hastype-listD*: $A \subseteq_m A' \Longrightarrow as :_l \sigma s\ in\ A \Longrightarrow as :_l \sigma s\ in\ A'$
⟨*proof*⟩

lemma *has-same-type-in-subsset*:
$a : \sigma\ in\ A' \Longrightarrow A \subseteq_m A' \Longrightarrow a : \sigma'\ in\ A \Longrightarrow \sigma' = \sigma$
⟨*proof*⟩

lemma *has-same-type-in-dom-subsset*:
$a : \sigma\ in\ A' \Longrightarrow A \subseteq_m A' \Longrightarrow a \in dom\ A \longleftrightarrow a : \sigma\ in\ A$
⟨*proof*⟩

lemma *hastype-restrict*: $a : \sigma\ in\ A\ |`\ S \longleftrightarrow a \in S \wedge a : \sigma\ in\ A$
⟨*proof*⟩

lemma *hastype-the-simp*[*simp*]: $a : \sigma\ in\ A \Longrightarrow the\ (A\ a) = \sigma$
⟨*proof*⟩

lemma *hastype-in-Some*[*simp*]: $x : \sigma\ in\ Some \longleftrightarrow x = \sigma$ ⟨*proof*⟩

lemma *hastype-in-upd*[*simp*]: $x : \sigma\ in\ A(y \mapsto \tau) \longleftrightarrow (if\ x = y\ then\ \sigma = \tau\ else\ x$
$: \sigma\ in\ A)$
⟨*proof*⟩

lemma *all-set-hastype-iff-those*: $\forall\, a \in set\ as.\ a : \sigma\ in\ A \Longrightarrow$
$those\ (map\ A\ as) = Some\ (replicate\ (length\ as)\ \sigma)$
⟨*proof*⟩

The partial version of list nth:

primrec *safe-nth* **where**
safe-nth [] - = *None*
| *safe-nth* (*a*#*as*) *n* = (*case n of 0* ⇒ *Some a* | *Suc n* ⇒ *safe-nth as n*)

lemma *safe-nth-simp*[*simp*]: $i < length\ as \Longrightarrow safe\text{-}nth\ as\ i = Some\ (as\ !\ i)$
⟨*proof*⟩

lemma *safe-nth-None*[*simp*]:
$length\ as \leq i \Longrightarrow safe\text{-}nth\ as\ i = None$
⟨*proof*⟩

lemma *safe-nth*: $safe\text{-}nth\ as\ i = (if\ i < length\ as\ then\ Some\ (as\ !\ i)\ else\ None)$
⟨*proof*⟩

lemma *safe-nth-eq-SomeE*:
$safe\text{-}nth\ as\ i = Some\ a \Longrightarrow (i < length\ as \Longrightarrow as\ !\ i = a \Longrightarrow thesis) \Longrightarrow thesis$
⟨*proof*⟩

lemma *dom-safe-nth*[*simp*]: $dom\ (safe\text{-}nth\ as) = \{0..<length\ as\}$
⟨*proof*⟩

lemma *safe-nth-replicate*[*simp*]:
$safe\text{-}nth\ (replicate\ n\ a)\ i = (if\ i < n\ then\ Some\ a\ else\ None)$

⟨*proof*⟩

lemma *safe-nth-append*:
safe-nth (*ls*@*rs*) *i* = (*if i* < *length ls then Some* (*ls*!*i*) *else safe-nth rs* (*i* − *length ls*))
⟨*proof*⟩

lemma *hastype-in-safe-nth*[*simp*]: $i : \sigma$ *in safe-nth* $\sigma s \longleftrightarrow i <$ *length* $\sigma s \wedge \sigma = \sigma s!i$
⟨*proof*⟩

lemmas *hastype-in-safe-nthE* = *safe-nth-eq-SomeE*[*folded hastype-def*]

lemma *hastype-in-o*[*simp*]: $a : \sigma$ *in* $A \circ f \longleftrightarrow f\ a : \sigma$ *in A* ⟨*proof*⟩

definition *o-sset* (**infix** ∘*s 55*) **where**
f ∘*s A* ≡ *map-option f* ∘ *A*

lemma *hastype-in-o-sset*: $a : \sigma'$ *in f* ∘*s A* ⟷ (∃σ. $a : \sigma$ *in A* ∧ $\sigma' = f\ \sigma$)
⟨*proof*⟩

lemma *hastype-in-o-ssetI*: $a : \sigma$ *in A* ⟹ $f\ \sigma = \sigma'$ ⟹ $a : \sigma'$ *in f* ∘*s A*
⟨*proof*⟩

lemma *hastype-in-o-ssetD*: $a : \tau$ *in f* ∘*s A* ⟹ ∃σ. $a : \sigma$ *in A* ∧ $\tau = f\ \sigma$
⟨*proof*⟩

lemma *hastype-in-o-ssetE*: $a : \tau$ *in f* ∘*s A* ⟹ (⋀σ. $a : \sigma$ *in A* ⟹ $\tau = f\ \sigma$ ⟹ *thesis*) ⟹ *thesis*
⟨*proof*⟩

lemma *o-sset-restrict-sset-assoc*[*simp*]: *f* ∘*s* (*A* |‘ *X*) = (*f* ∘*s A*) |‘ *X*
⟨*proof*⟩

lemma *id-o-sset*[*simp*]: *id* ∘*s A* = *A*
and *identity-o-sset*[*simp*]: (λ*x*. *x*) ∘*s A* = *A*
⟨*proof*⟩

lemma *o-ssetI*: *A x* = *Some y* ⟹ *z* = *f y* ⟹ (*f* ∘*s A*) *x* = *Some z* ⟨*proof*⟩

lemma *o-ssetE*: (*f* ∘*s A*) *x* = *Some z* ⟹ (⋀*y*. *A x* = *Some y* ⟹ *z* = *f y* ⟹ *thesis*) ⟹ *thesis*
⟨*proof*⟩

lemma *dom-o-sset*[*simp*]: *dom* (*f* ∘*s A*) = *dom A*
⟨*proof*⟩

lemma *safe-nth-map*: *safe-nth* (*map f as*) = *f* ∘*s safe-nth as*
⟨*proof*⟩

notation *Map.empty* (∅)
lemma *safe-nth-Nil*[*simp*]: *safe-nth* [] = ∅ ⟨*proof*⟩

lemma *o-sset-empty*[*simp*]: *f* ∘*s* ∅ = ∅ ⟨*proof*⟩

lemma *hastype-in-empty*[*simp*]: ¬*x* : *σ in* ∅ ⟨*proof*⟩

3.1 Maps between Sorted Sets

locale *sort-preserving* = **fixes** *f* :: *'a* ⇒ *'b* **and** *A* :: *'a* ⇀ *'s*
assumes *same-value-imp-same-type*: *a* : *σ in A* ⟹ *b* : *τ in A* ⟹ *f a* = *f b* ⟹ *σ* = *τ*
begin

lemma *same-value-imp-in-dom-iff*:
assumes *fafa'*: *f a* = *f a'* **and** *a*: *a* : *σ in A* **shows** *a'*: *a'* ∈ *dom A* ⟷ *a'* : *σ in A*
⟨*proof*⟩

end

lemma *sort-preserving-cong*:
A = *A'* ⟹ (⋀*a σ*. *a* : *σ in A* ⟹ *f a* = *f' a*) ⟹ *sort-preserving f A* ⟷ *sort-preserving f' A'*
⟨*proof*⟩

lemma *inj-on-dom-imp-sort-preserving*:
assumes *inj-on f* (*dom A*) **shows** *sort-preserving f A*
⟨*proof*⟩

lemma *inj-imp-sort-preserving*:
assumes *inj f* **shows** *sort-preserving f A*
⟨*proof*⟩

locale *sorted-map* =
fixes *f* :: *'a* ⇒ *'b* **and** *A* :: *'a* ⇀ *'s* **and** *B* :: *'b* ⇀ *'s*
assumes *sorted-map*: ⋀*a σ*. *a* : *σ in A* ⟹ *f a* : *σ in B*
begin

lemma *target-has-same-type*: *a* : *σ in A* ⟹ *f a* : *τ in B* ⟷ *σ* = *τ*
⟨*proof*⟩

lemma *target-dom-iff-hastype*:
a : *σ in A* ⟹ *f a* ∈ *dom B* ⟷ *f a* : *σ in B*
⟨*proof*⟩

lemma *source-dom-iff-hastype*:
f a : *σ in B* ⟹ *a* ∈ *dom A* ⟷ *a* : *σ in A*

⟨*proof*⟩

lemma *elim*:
assumes *a*: $(\bigwedge a\ \sigma.\ a : \sigma\ in\ A \Longrightarrow f\ a : \sigma\ in\ B) \Longrightarrow P$
shows *P*
⟨*proof*⟩

sublocale *sort-preserving*
⟨*proof*⟩

lemma *funcset-dom*: $f : dom\ A \to dom\ B$
⟨*proof*⟩

lemma *sorted-map-list*: $as :_l \sigma s\ in\ A \Longrightarrow map\ f\ as :_l \sigma s\ in\ B$
⟨*proof*⟩

lemma *in-dom*: $a \in dom\ A \Longrightarrow f\ a \in dom\ B$ ⟨*proof*⟩

end

notation *sorted-map* (- $:_s$(/ - →/ -) [*50*,*51*,*51*]*50*)

abbreviation *all-sorted-map* $A\ B\ P \equiv \forall f.\ f :_s A \to B \longrightarrow P\ f$
abbreviation *ex-sorted-map* $A\ B\ P \equiv \exists f.\ f :_s A \to B \wedge P\ f$

syntax
all-sorted-map :: ′*pttrn* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* (∀ - $:_s$(/ - →/ -)./ - [*50*,*51*,*51*,*10*]*10*)
ex-sorted-map :: ′*pttrn* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* (∃ - $:_s$(/ - →/ -)./ - [*50*,*51*,*51*,*10*]*10*)

translations
$\forall f :_s A \to B.\ e \rightleftharpoons CONST\ all\text{-}sorted\text{-}map\ A\ B\ (\lambda f.\ e)$
$\exists f :_s A \to B.\ e \rightleftharpoons CONST\ ex\text{-}sorted\text{-}map\ A\ B\ (\lambda f.\ e)$

lemmas *sorted-mapI* = *sorted-map.intro*

lemma *sorted-mapD*: $f :_s A \to B \Longrightarrow a : \sigma\ in\ A \Longrightarrow f\ a : \sigma\ in\ B$
⟨*proof*⟩

lemmas *sorted-mapE* = *sorted-map.elim*

lemma assumes $f :_s A \to B$
shows *sorted-map-o*: $g :_s B \to C \Longrightarrow g \circ f :_s A \to C$
and *sorted-map-cmono*: $A' \subseteq_m A \Longrightarrow f :_s A' \to B$
and *sorted-map-mono*: $B \subseteq_m B' \Longrightarrow f :_s A \to B'$
⟨*proof*⟩

lemma *sorted-map-cong*:
$(\bigwedge a\ \sigma.\ a : \sigma\ in\ A \Longrightarrow f\ a = f'\ a) \Longrightarrow$
$A = A' \Longrightarrow$

$(\bigwedge a\ \sigma.\ a : \sigma\ in\ A \Longrightarrow f\ a : \sigma\ in\ B \longleftrightarrow f\ a : \sigma\ in\ B') \Longrightarrow$
$f :_s A \to B \longleftrightarrow f' :_s A' \to B'$
⟨*proof*⟩

lemma *sorted-choice*:
assumes $\forall\, a\ \sigma.\ a : \sigma\ in\ A \longrightarrow (\exists\, b : \sigma\ in\ B.\ P\ a\ b)$
shows $\exists f :_s A \to B.\ (\forall\, a \in dom\ A.\ P\ a\ (f\ a))$
⟨*proof*⟩

lemma *sorted-map-empty*[*simp*]: $f :_s \emptyset \to A$
⟨*proof*⟩

lemma *sorted-map-comp-nth*:
$\alpha :_s (f \circ s\ safe\text{-}nth\ (a\#as)) \to A \longleftrightarrow \alpha\ 0 : f\ a\ in\ A \wedge (\alpha \circ Suc :_s (f \circ s\ safe\text{-}nth\ as) \to A)$
(**is** $?l \longleftrightarrow ?r$)
⟨*proof*⟩

locale *inhabited* = **fixes** *A*
assumes *inhabited*: $\bigwedge\sigma.\ \exists\, a.\ a : \sigma\ in\ A$
begin

lemma *ex-sorted-map*: $\exists\,\alpha.\ \alpha :_s V \to A$
⟨*proof*⟩

end

3.2 Sorted Images

The partial version of *The* operator.

definition $safe\text{-}The\ P \equiv if\ \exists!x.\ P\ x\ then\ Some\ (The\ P)\ else\ None$

lemma *safe-The-eq-Some*: $safe\text{-}The\ P = Some\ x \longleftrightarrow P\ x \wedge (\forall\, x'.\ P\ x' \longrightarrow x' = x)$
⟨*proof*⟩

lemma *safe-The-eq-None*: $safe\text{-}The\ P = None \longleftrightarrow \neg(\exists!x.\ P\ x)$
⟨*proof*⟩

lemma *safe-The-False*[*simp*]: $safe\text{-}The\ (\lambda x.\ False) = None$
⟨*proof*⟩

definition *sorted-image* :: $('a \Rightarrow 'b) \Rightarrow ('a \rightharpoonup 's) \Rightarrow 'b \rightharpoonup 's$ (**infixr** ‘s 90) **where**
$(f\ `^s\ A)\ b \equiv safe\text{-}The\ (\lambda\sigma.\ \exists\, a : \sigma\ in\ A.\ f\ a = b)$

lemma *hastype-in-imageE*:
assumes $fx : \sigma\ in\ f\ `^s\ X$
and $\bigwedge x.\ x : \sigma\ in\ X \Longrightarrow fx = f\ x \Longrightarrow thesis$
shows *thesis*

⟨*proof*⟩

lemma *in-dom-imageE*:
$b \in dom\ (f\ {}^{`s}\ A) \Longrightarrow (\bigwedge a\ \sigma.\ a : \sigma\ in\ A \Longrightarrow b = f\ a \Longrightarrow thesis) \Longrightarrow thesis$
⟨*proof*⟩

context *sort-preserving* **begin**

lemma *hastype-in-imageI*: $a : \sigma\ in\ A \Longrightarrow b = f\ a \Longrightarrow b : \sigma\ in\ f\ {}^{`s}\ A$
⟨*proof*⟩

lemma *hastype-in-imageI2*: $a : \sigma\ in\ A \Longrightarrow f\ a : \sigma\ in\ f\ {}^{`s}\ A$
⟨*proof*⟩

lemma *hastype-in-image*: $b : \sigma\ in\ f\ {}^{`s}\ A \longleftrightarrow (\exists\, a : \sigma\ in\ A.\ f\ a = b)$
⟨*proof*⟩

lemma *in-dom-imageI*: $a \in dom\ A \Longrightarrow b = f\ a \Longrightarrow b \in dom\ (f\ {}^{`s}\ A)$
⟨*proof*⟩

lemma *in-dom-imageI2*: $a \in dom\ A \Longrightarrow f\ a \in dom\ (f\ {}^{`s}\ A)$
⟨*proof*⟩

lemma *hastype-list-in-image*: $bs :_l \sigma s\ in\ f\ {}^{`s}\ A \longleftrightarrow (\exists\, as.\ as :_l \sigma s\ in\ A \wedge map\ f\ as = bs)$
⟨*proof*⟩

lemma *dom-image*[*simp*]: $dom\ (f\ {}^{`s}\ A) = f\ `\ dom\ A$
⟨*proof*⟩

sublocale *to-image*: $sorted\text{-}map\ f\ A\ f\ {}^{`s}\ A$
⟨*proof*⟩

lemma *sorted-map-iff-image-subset*:
$f :_s A \rightarrow B \longleftrightarrow f\ {}^{`s}\ A \subseteq_m B$
⟨*proof*⟩

end

lemma *sort-preserving-o*:
assumes *f*: *sort-preserving f A* **and** *g*: *sort-preserving g* $(f\ {}^{`s}\ A)$
shows *sort-preserving* $(g \circ f)\ A$
⟨*proof*⟩

lemma *sorted-image-image*:
assumes *f*: *sort-preserving f A* **and** *g*: *sort-preserving g* $(f\ {}^{`s}\ A)$
shows $g\ {}^{`s}\ f\ {}^{`s}\ A = (g \circ f)\ {}^{`s}\ A$
⟨*proof*⟩

context *sorted-map* **begin**

lemma *image-subsset*[*intro*!]: f ‘^{s} $A \subseteq_m B$
⟨*proof*⟩

lemma *dom-image-subset*[*intro*!]: f ‘ *dom* $A \subseteq$ *dom* B
⟨*proof*⟩

end

lemma *sorted-image-cong*: $(\bigwedge a\ \sigma.\ a : \sigma\ in\ A \Longrightarrow f\ a = f'\ a) \Longrightarrow f$ ‘^{s} $A = f'$ ‘^{s} A
⟨*proof*⟩

lemma *inj-on-dom-imp-sort-preserving-inv-into*:
assumes *inj*: *inj-on* f (*dom* A) **shows** *sort-preserving* (*inv-into* (*dom* A) f) (f ‘^{s} A)
⟨*proof*⟩

lemma *inj-imp-sort-preserving-inv*:
assumes *inj*: *inj* f **shows** *sort-preserving* (*inv* f) (f ‘^{s} A)
⟨*proof*⟩

lemma *inj-on-dom-imp-inv-into-image-cancel*:
assumes *inj*: *inj-on* f (*dom* A)
shows *inv-into* (*dom* A) f ‘^{s} f ‘^{s} $A = A$
⟨*proof*⟩

lemma *inj-imp-inv-image-cancel*:
assumes *inj*: *inj* f
shows *inv* f ‘^{s} f ‘^{s} $A = A$
⟨*proof*⟩

definition *sorted-Imagep* (**infixr** ‘‘^{s} *90*)
where $((\sqsubseteq)$ ‘‘^{s} $A)\ b \equiv$ *safe-The* $(\lambda\sigma.\ \exists\, a : \sigma\ in\ A.\ a \sqsubseteq b)$ **for** r (**infix** $\sqsubseteq$ *50*)

lemma *untyped-hastypeE*: $A\ a = None \Longrightarrow a : \sigma\ in\ A \Longrightarrow thesis$
⟨*proof*⟩

end

4 Sorted Terms

theory *Sorted-Terms*
imports *Sorted-Sets First-Order-Terms.Term*
begin

4.1 Overloaded Notations

consts *vars* :: $'a \Rightarrow 'b\ set$

adhoc-overloading *vars vars-term*

consts *map-vars* :: $('a \Rightarrow 'b) \Rightarrow 'c \Rightarrow 'd$

adhoc-overloading *map-vars map-term* $(\lambda x.\ x)$

lemma *map-term-eq-Var*: $\textit{map-term}\ F\ V\ s = \textit{Var}\ y \longleftrightarrow (\exists x.\ s = \textit{Var}\ x \wedge y = V\ x)$
⟨*proof*⟩

lemma *map-vars-id-iff*: $\textit{map-vars}\ f\ s = s \longleftrightarrow (\forall x \in \textit{vars-term}\ s.\ f\ x = x)$
⟨*proof*⟩

lemma *map-var-term-id*[*simp*]: $\textit{map-term}\ (\lambda x.\ x)\ \textit{id} = \textit{id}$ ⟨*proof*⟩

lemma *map-term-eq-Fun*:
$\textit{map-term}\ F\ V\ s = \textit{Fun}\ g\ ts \longleftrightarrow (\exists f\ ss.\ s = \textit{Fun}\ f\ ss \wedge g = F\ f \wedge ts = \textit{map}\ (\textit{map-term}\ F\ V)\ ss)$
⟨*proof*⟩

declare *domIff*[*iff del*]

4.2 Sorted Signatures and Sorted Sets of Terms

We view a sorted signature as a partial map that assigns an output sort to the pair of a function symbol and a list of input sorts.

type-synonym $('f,'s)\ \textit{ssig} = 'f \times 's\ \textit{list} \rightharpoonup 's$

definition *hastype-in-ssig* :: $'f \Rightarrow 's\ \textit{list} \Rightarrow 's \Rightarrow ('f,'s)\ \textit{ssig} \Rightarrow \textit{bool}$
(- : - → - *in* - [*50*,*61*,*61*,*50*]*50*)
where $f : \sigma s \rightarrow \tau\ \textit{in}\ F \equiv F\ (f,\sigma s) = \textit{Some}\ \tau$

lemmas *hastype-in-ssigI* = *hastype-in-ssig-def*[*unfolded atomize-eq*, *THEN iffD2*]
lemmas *hastype-in-ssigD* = *hastype-in-ssig-def*[*unfolded atomize-eq*, *THEN iffD1*]

lemma *hastype-in-ssig-imp-dom*:
assumes $f : \sigma s \rightarrow \tau\ \textit{in}\ F$ **shows** $(f,\sigma s) \in \textit{dom}\ F$
⟨*proof*⟩

lemma *has-same-type-ssig*:
assumes $f : \sigma s \rightarrow \tau\ \textit{in}\ F$ **and** $f : \sigma s \rightarrow \tau'\ \textit{in}\ F$ **shows** $\tau = \tau'$
⟨*proof*⟩

lemma *hastype-restrict-ssig*: $f : \sigma s \rightarrow \tau\ \textit{in}\ F\ |`\ S \longleftrightarrow (f,\sigma s) \in S \wedge f : \sigma s \rightarrow \tau\ \textit{in}\ F$
⟨*proof*⟩

lemma *subssigI*: **assumes** $\bigwedge f\ \sigma s\ \tau.\ f : \sigma s \rightarrow \tau\ \textit{in}\ F \Longrightarrow f : \sigma s \rightarrow \tau\ \textit{in}\ F'$

shows $F \subseteq_m F'$
⟨*proof*⟩

lemma *subssigD*: **assumes** *FF*: $F \subseteq_m F'$ **and** $f : \sigma s \to \tau$ *in* F **shows** $f : \sigma s \to \tau$ *in* F'
⟨*proof*⟩

The sorted set of terms:

primrec *Term* ($\mathcal{T}'$(-,-')) **where**
$\mathcal{T}(F,V)$ (*Var v*) = *V v*
| $\mathcal{T}(F,V)$ (*Fun f ss*) =
(*case those* (*map* $\mathcal{T}(F,V)$ *ss*) *of None* ⇒ *None* | *Some* σs ⇒ *F* (f,σs))

lemma *Var-hastype*[*simp*]: *Var v* : σ *in* $\mathcal{T}(F,V) \longleftrightarrow v : \sigma$ *in* V
⟨*proof*⟩

lemma *Fun-hastype*:
Fun f ss : τ *in* $\mathcal{T}(F,V) \longleftrightarrow (\exists \sigma s.\ f : \sigma s \to \tau$ *in* $F \land ss :_l \sigma s$ *in* $\mathcal{T}(F,V))$
⟨*proof*⟩

lemma *Fun-in-dom-imp-arg-in-dom*: *Fun f ss* $\in$ *dom* $\mathcal{T}(F,V) \Longrightarrow s \in$ *set ss* $\Longrightarrow s \in$ *dom* $\mathcal{T}(F,V)$
⟨*proof*⟩

lemma *Fun-hastypeI*: $f : \sigma s \to \tau$ *in* $F \Longrightarrow ss :_l \sigma s$ *in* $\mathcal{T}(F,V) \Longrightarrow$ *Fun f ss* : τ *in* $\mathcal{T}(F,V)$
⟨*proof*⟩

lemma *hastype-in-Term-induct*[*case-names Var Fun*, *induct pred*]:
assumes *s*: $s : \sigma$ *in* $\mathcal{T}(F,V)$
and *V*: $\bigwedge v\ \sigma.\ v : \sigma$ *in* $V \Longrightarrow P$ (*Var v*) σ
and *F*: $\bigwedge f\ ss\ \sigma s\ \tau.$
$f : \sigma s \to \tau$ *in* $F \Longrightarrow ss :_l \sigma s$ *in* $\mathcal{T}(F,V) \Longrightarrow$ *list-all2 P ss* $\sigma s \Longrightarrow P$ (*Fun f ss*) τ
shows $P\ s\ \sigma$
⟨*proof*⟩

lemma *in-dom-Term-induct*[*case-names Var Fun*, *induct pred*]:
assumes *s*: $s \in$ *dom* $\mathcal{T}(F,V)$
assumes *V*: $\bigwedge v\ \sigma.\ v : \sigma$ *in* $V \Longrightarrow P$ (*Var v*)
assumes *F*: $\bigwedge f\ ss\ \sigma s\ \tau.$
$f : \sigma s \to \tau$ *in* $F \Longrightarrow ss :_l \sigma s$ *in* $\mathcal{T}(F,V) \Longrightarrow \forall s \in$ *set ss*. $P\ s \Longrightarrow P$ (*Fun f ss*)
shows $P\ s$
⟨*proof*⟩

lemma *Term-mono-left*: **assumes** *FF*: $F \subseteq_m F'$ **shows** $\mathcal{T}(F,V) \subseteq_m \mathcal{T}(F',V)$
⟨*proof*⟩

lemmas *hastype-in-Term-mono-left* = *Term-mono-left*[*THEN subssetD*]

lemmas *dom-Term-mono-left* = *Term-mono-left*[*THEN map-le-implies-dom-le*]

lemma *Term-mono-right*: **assumes** *VV*: $V \subseteq_m V'$ **shows** $\mathcal{T}(F,V) \subseteq_m \mathcal{T}(F,V')$
⟨*proof*⟩

lemmas *hastype-in-Term-mono-right* = *Term-mono-right*[*THEN subssetD*]

lemmas *dom-Term-mono-right* = *Term-mono-right*[*THEN map-le-implies-dom-le*]

lemmas *Term-mono* = *map-le-trans*[*OF Term-mono-left Term-mono-right*]

lemmas *hastype-in-Term-mono* = *Term-mono*[*THEN subssetD*]

lemmas *dom-Term-mono* = *Term-mono*[*THEN map-le-implies-dom-le*]

lemma *hastype-in-Term-restrict-vars*: $s : \sigma$ *in* $\mathcal{T}(F,\ V\ |\text{`}\ vars\ s) \longleftrightarrow s : \sigma$ *in* $\mathcal{T}(F,V)$
(**is** *?l s* ⟷ *?r s*)
⟨*proof*⟩

lemma *hastype-in-Term-imp-vars*: $s : \sigma$ *in* $\mathcal{T}(F,V) \Longrightarrow v \in vars\ s \Longrightarrow v \in dom\ V$
⟨*proof*⟩

lemma *in-dom-Term-imp-vars*: $s \in dom\ \mathcal{T}(F,V) \Longrightarrow v \in vars\ s \Longrightarrow v \in dom\ V$
⟨*proof*⟩

lemma *hastype-in-Term-imp-vars-subset*: $t : s$ *in* $\mathcal{T}(F,V) \Longrightarrow vars\ t \subseteq dom\ V$
⟨*proof*⟩

interpretation *Var*: *sorted-map Var V* $\mathcal{T}(F,V)$ **for** *F V* ⟨*proof*⟩

4.3 Sorted Algebras

locale *sorted-algebra-syntax* =
fixes F :: ('*f*,'*s*) *ssig* **and** A :: '*a* ⇀ '*s* **and** I :: '*f* ⇒ '*a list* ⇒ '*a*

locale *sorted-algebra* = *sorted-algebra-syntax* +
assumes *sort-matches*: $f : \sigma s \to \tau$ *in* $F \Longrightarrow as :_l \sigma s$ *in* $A \Longrightarrow I\ f\ as : \tau$ *in* A
begin

context
fixes α V
assumes α: $\alpha :_s V \to A$
begin

lemma *eval-hastype*:
assumes *s*: $s : \sigma$ *in* $\mathcal{T}(F,V)$ **shows** $I[\![s]\!]\alpha : \sigma$ *in* A

⟨*proof*⟩

interpretation *eval*: *sorted-map* $\lambda s.\ I[\![s]\!]\alpha\ \mathcal{T}(F,V)\ A$
⟨*proof*⟩

lemmas *eval-sorted-map* = *eval.sorted-map-axioms*
lemmas *eval-dom* = *eval.in-dom*
lemmas *map-eval-hastype* = *eval.sorted-map-list*
lemmas *eval-has-same-type* = *eval.target-has-same-type*
lemmas *eval-dom-iff-hastype* = *eval.target-dom-iff-hastype*
lemmas *dom-iff-hastype* = *eval.source-dom-iff-hastype*

end

lemmas *eval-hastype-vars* =
eval-hastype[*OF* - *hastype-in-Term-restrict-vars*[*THEN iffD2*]]

lemmas *eval-has-same-type-vars* =
eval-has-same-type[*OF* - *hastype-in-Term-restrict-vars*[*THEN iffD2*]]

end

lemma *sorted-algebra-cong*:
assumes $F = F'$ **and** $A = A'$
and $\bigwedge f\ \sigma s\ \tau\ as.\ f : \sigma s \to \tau\ in\ F' \Longrightarrow as :_l \sigma s\ in\ A' \Longrightarrow I\ f\ as = I'\ f\ as$
shows *sorted-algebra* $F\ A\ I$ = *sorted-algebra* $F'\ A'\ I'$
⟨*proof*⟩

4.3.1 Term Algebras

The sorted set of terms constitutes a sorted algebra, in which evaluation is substitution.

interpretation *term*: *sorted-algebra* $F\ \mathcal{T}(F,V)$ *Fun* **for** $F\ V$
⟨*proof*⟩

Sorted substitution preserves type:

lemma *subst-hastype*: $\vartheta :_s X \to \mathcal{T}(F,V) \Longrightarrow s : \sigma\ in\ \mathcal{T}(F,X) \Longrightarrow s{\cdot}\vartheta : \sigma\ in\ \mathcal{T}(F,V)$
⟨*proof*⟩

lemmas *subst-hastype-imp-dom-iff* = *term.dom-iff-hastype*
lemmas *subst-hastype-vars* = *term.eval-hastype-vars*
lemmas *subst-has-same-type* = *term.eval-has-same-type*
lemmas *subst-same-vars* = *eval-same-vars*[*of* - - - *Fun*]
lemmas *subst-map-vars* = *eval-map-vars*[*of Fun*]
lemmas *subst-o* = *eval-o*[*of Fun*]
lemmas *subst-sorted-map* = *term.eval-sorted-map*
lemmas *map-subst-hastype* = *term.map-eval-hastype*

lemma *subst-compose-sorted-map*:
assumes $\vartheta :_s X \rightarrow \mathcal{T}(F,Y)$ **and** $\varrho :_s Y \rightarrow \mathcal{T}(F,Z)$
shows $\vartheta \circ_s \varrho :_s X \rightarrow \mathcal{T}(F,Z)$
⟨*proof*⟩

lemma *subst-hastype-iff-vars*:
assumes $\forall x \in vars\ s.\ \forall \sigma.\ \vartheta\ x : \sigma\ in\ \mathcal{T}(F,W) \longleftrightarrow x : \sigma\ in\ V$
shows $s \cdot \vartheta : \sigma\ in\ \mathcal{T}(F,W) \longleftrightarrow s : \sigma\ in\ \mathcal{T}(F,V)$
⟨*proof*⟩

lemma *subst-in-dom-imp-var-in-dom*:
assumes $s{\cdot}\vartheta \in dom\ \mathcal{T}(F,V)$ **and** $x \in vars\ s$ **shows** $\vartheta\ x \in dom\ \mathcal{T}(F,V)$
⟨*proof*⟩

lemma *subst-sorted-map-restrict-vars*:
assumes ϑ: $\vartheta :_s X \rightarrow \mathcal{T}(F,V)$ **and** WV: $W \subseteq_m V$ **and** $s\vartheta$: $s{\cdot}\vartheta \in dom\ \mathcal{T}(F,W)$
shows $\vartheta :_s X \mid`\ vars\ s \rightarrow \mathcal{T}(F,W)$
⟨*proof*⟩

4.3.2 Homomorphisms

locale *sorted-distributive* =
sort-preserving φ A + *source*: *sorted-algebra* $F\ A\ I$ **for** $F\ \varphi\ A\ I\ J$ +
assumes *distrib*: $f : \sigma s \rightarrow \tau\ in\ F \Longrightarrow as :_l \sigma s\ in\ A \Longrightarrow \varphi\ (I\ f\ as) = J\ f\ (map\ \varphi\ as)$
begin

lemma *distrib-eval*:
assumes α: $\alpha :_s V \rightarrow A$ **and** s: $s : \sigma\ in\ \mathcal{T}(F,V)$
shows $\varphi\ (I[\![s]\!]\alpha) = J[\![s]\!](\varphi \circ \alpha)$
⟨*proof*⟩

The image of a distributive map forms a sorted algebra.

sublocale *image*: *sorted-algebra* $F\ \varphi\ `^s\ A\ J$
⟨*proof*⟩

end

lemma *sorted-distributive-cong*:
fixes $A\ A' :: 'a \rightharpoonup 's$ **and** $\varphi :: 'a \Rightarrow 'b$ **and** $I :: 'f \Rightarrow 'a\ list \Rightarrow 'a$
assumes φ: $\bigwedge a\ \sigma.\ a : \sigma\ in\ A \Longrightarrow \varphi\ a = \varphi'\ a$
and A: $A = A'$
and I: $\bigwedge f\ \sigma s\ \tau\ as.\ f : \sigma s \rightarrow \tau\ in\ F \Longrightarrow as :_l \sigma s\ in\ A \Longrightarrow I\ f\ as = I'\ f\ as$
and J: $\bigwedge f\ \sigma s\ \tau\ as.\ f : \sigma s \rightarrow \tau\ in\ F \Longrightarrow as :_l \sigma s\ in\ A \Longrightarrow J\ f\ (map\ \varphi\ as) = J'\ f\ (map\ \varphi\ as)$
shows *sorted-distributive* $F\ \varphi\ A\ I\ J$ = *sorted-distributive* $F\ \varphi'\ A'\ I'\ J'$
⟨*proof*⟩

lemma *sorted-distributive-o*:
assumes *sorted-distributive* $F\ \varphi\ A\ I\ J$ **and** *sorted-distributive* $F\ \psi\ (\varphi\ `^s\ A)\ J\ K$
shows *sorted-distributive* $F\ (\psi \circ \varphi)\ A\ I\ K$
⟨*proof*⟩

locale *sorted-homomorphism* = *sorted-distributive* $F\ \varphi\ A\ I\ J$ + *sorted-map* $\varphi\ A\ B$ +
target: *sorted-algebra* $F\ B\ J$ **for** $F\ \varphi\ A\ I\ B\ J$
begin
end

lemma *sorted-homomorphism-o*:
assumes *sorted-homomorphism* $F\ \varphi\ A\ I\ B\ J$ **and** *sorted-homomorphism* $F\ \psi\ B\ J\ C\ K$
shows *sorted-homomorphism* $F\ (\psi \circ \varphi)\ A\ I\ C\ K$
⟨*proof*⟩

context *sorted-algebra* **begin**

context fixes $\alpha\ V$ **assumes** *sorted*: $\alpha :_s V \to A$
begin

The term algebra is free in all F-algebras; that is, every assignment $\alpha :_s V \to A$ is extended to a homomorphism $\lambda s.\ I[\![s]\!]\alpha$.

interpretation *sorted-map* $\alpha\ V\ A$ ⟨*proof*⟩

interpretation *eval*: *sorted-map* ‹$\lambda s.\ I[\![s]\!]\alpha$› ‹$\mathcal{T}(F,V)$› A ⟨*proof*⟩

interpretation *eval*: *sorted-homomorphism* F ‹$\lambda s.\ I[\![s]\!]\alpha$› ‹$\mathcal{T}(F,V)$› *Fun* $A\ I$
⟨*proof*⟩

lemmas *eval-sorted-homomorphism* = *eval.sorted-homomorphism-axioms*

end

end

lemma *sorted-homomorphism-cong*:
fixes $A\ A' :: 'a \rightharpoonup 's$ **and** $\varphi :: 'a \Rightarrow 'b$ **and** $I :: 'f \Rightarrow 'a\ list \Rightarrow 'a$
assumes φ: $\bigwedge a\ \sigma.\ a : \sigma\ in\ A \Longrightarrow \varphi\ a = \varphi'\ a$
and A: $A = A'$
and I: $\bigwedge f\ \sigma s\ \tau\ as.\ f : \sigma s \to \tau\ in\ F \Longrightarrow as :_l \sigma s\ in\ A \Longrightarrow I\ f\ as = I'\ f\ as$
and B: $B = B'$
and J: $\bigwedge f\ \sigma s\ \tau\ bs.\ f : \sigma s \to \tau\ in\ F \Longrightarrow bs :_l \sigma s\ in\ B \Longrightarrow J\ f\ bs = J'\ f\ bs$
shows *sorted-homomorphism* $F\ \varphi\ A\ I\ B\ J$ = *sorted-homomorphism* $F\ \varphi'\ A'\ I'\ B'\ J'$ (**is** $?l \longleftrightarrow ?r$)
⟨*proof*⟩

context *sort-preserving* **begin**

lemma *sort-preserving-map-vars*: *sort-preserving* (*map-vars f*) $\mathcal{T}(F,A)$
⟨*proof*⟩

lemma *map-vars-image-Term*: *map-vars f* '^{s} $\mathcal{T}(F,A) = \mathcal{T}(F,f$ '^{s} $A)$ (**is** *?L* = *?R*)
⟨*proof*⟩

end

context *sorted-map* **begin**

lemma *sorted-map-map-vars*: *map-vars f* $:_s$ $\mathcal{T}(F,A) \rightarrow \mathcal{T}(F,B)$
⟨*proof*⟩

end

4.4 Lifting Sorts

By 'uni-sorted' we mean the situation where there is only one sort (). This situation is isomorphic to sets.

definition *unisorted A a* ≡ *if a* ∈ *A then Some* () *else None*

lemma *unisorted-eq-Some*[*simp*]: *unisorted A a* = *Some* $\sigma \longleftrightarrow a \in A$
  and *unisorted-eq-None*[*simp*]: *unisorted A a* = *None* $\longleftrightarrow a \notin A$
  and *hastype-in-unisorted*[*simp*]: $a : \sigma$ *in unisorted A* $\longleftrightarrow a \in A$
  ⟨*proof*⟩

lemma *hastype-list-in-unisorted*[*simp*]: *as* $:_l$ σs *in unisorted A* ⟷ *length as* = *length* σs ∧ *set as* ⊆ *A*
  ⟨*proof*⟩

lemma *dom-unisorted*[*simp*]: *dom* (*unisorted A*) = *A*
  ⟨*proof*⟩

lemma *unisorted-map*[*simp*]:
  $f :_s$ *unisorted A* $\rightarrow \tau \longleftrightarrow f : A \rightarrow$ *dom* τ
  $f :_s \sigma \rightarrow$ *unisorted B* $\longleftrightarrow f :$ *dom* $\sigma \rightarrow B$
  ⟨*proof*⟩

lemma *image-unisorted*[*simp*]: *f* '^{s} *unisorted A* = *unisorted* (*f* ' *A*)
  ⟨*proof*⟩

definition *unisorted-sig* :: ('*f*×*nat*) *set* ⇒ ('*f*,*unit*) *ssig*
  where *unisorted-sig F* ≡ λ(*f*,σs). *if* (*f*, *length* σs) ∈ *F then Some* () *else None*

lemma *in-unisorted-sig*[*simp*]: $f : \sigma s \rightarrow \tau$ *in unisorted-sig F* ⟷ (*f*,*length* σs) ∈ *F*
  ⟨*proof*⟩

inductive-set *uTerm* ($\mathfrak{T}'$(-,-') [*1*,*1*]*1000*) **for** *F V* **where**
Var v $\in \mathfrak{T}(F, V)$ **if** $v \in V$
| $\forall$ *s* $\in$ *set ss*. *s* $\in \mathfrak{T}(F, V) \Longrightarrow$ *Fun f ss* $\in \mathfrak{T}(F, V)$ **if** (*f*,*length ss*) $\in$ *F*

lemma *Var-in-Term*[*simp*]: *Var x* $\in \mathfrak{T}(F, V) \longleftrightarrow x \in V$
⟨*proof*⟩

lemma *Fun-in-Term*[*simp*]: *Fun f ss* $\in \mathfrak{T}(F, V) \longleftrightarrow$ (*f*,*length ss*) $\in F \wedge$ *set ss* $\subseteq$ $\mathfrak{T}(F, V)$
⟨*proof*⟩

lemma *hastype-in-unisorted-Term*[*simp*]:
s : σ *in* $\mathcal{T}$(*unisorted-sig F*, *unisorted V*) $\longleftrightarrow$ *s* $\in \mathfrak{T}(F, V)$
⟨*proof*⟩

lemma *unisorted-Term*: $\mathcal{T}$(*unisorted-sig F*, *unisorted V*) = *unisorted* $\mathfrak{T}(F, V)$
⟨*proof*⟩

locale *algebra* =
fixes *F* :: (*'f* $\times$ *nat*) *set* **and** *A* :: *'a set* **and** *I*
assumes *closed*: (*f*, *length as*) $\in F \Longrightarrow$ *set as* $\subseteq A \Longrightarrow$ *I f as* $\in A$
begin
end

lemma *unisorted-algebra*: *sorted-algebra* (*unisorted-sig F*) (*unisorted A*) *I* $\longleftrightarrow$ *algebra F A I*
(**is** *?l* $\longleftrightarrow$ *?r*)
⟨*proof*⟩

context *algebra* **begin**

interpretation *unisorted*: *sorted-algebra* ‹*unisorted-sig F*› ‹*unisorted A*› *I*
⟨*proof*⟩

lemma *eval-closed*: α : $V \to A \Longrightarrow s \in \mathfrak{T}(F, V) \Longrightarrow I[\![s]\!]\alpha \in A$
⟨*proof*⟩

end

locale *distributive* =
source: *algebra F A I* **for** *F* φ *A I J* +
assumes *distrib*: (*f*, *length as*) $\in F \Longrightarrow$ *set as* $\subseteq A \Longrightarrow \varphi$ (*I f as*) = *J f* (*map* φ *as*)

lemma *unisorted-distributive*:
sorted-distributive (*unisorted-sig F*) φ (*unisorted A*) *I J* $\longleftrightarrow$
distributive F φ *A I J* (**is** *?l* $\longleftrightarrow$ *?r*)
⟨*proof*⟩

locale *homomorphism* =
distributive $F\ \varphi\ A\ I\ J$ + *target*: *algebra* $F\ B\ J$ **for** $F\ \varphi\ A\ I\ B\ J$ +
assumes *funcset*: $\varphi : A \to B$

lemma *unisorted-homomorphism*:
sorted-homomorphism (*unisorted-sig* F) φ (*unisorted* A) I (*unisorted* B) $J \longleftrightarrow$
homomorphism $F\ \varphi\ A\ I\ B\ J$ (**is** $?l \longleftrightarrow ?r$)
⟨*proof*⟩

lemma *homomorphism-cong*:
assumes φ: $\bigwedge a.\ a \in A \Longrightarrow \varphi\ a = \varphi'\ a$
and A: $A = A'$
and I: $\bigwedge f\ as.\ (f,\ length\ as) \in F \Longrightarrow I\ f\ as = I'\ f\ as$
and B: $B = B'$
and J: $\bigwedge f\ bs.\ (f,\ length\ bs) \in F \Longrightarrow J\ f\ bs = J'\ f\ bs$
shows *homomorphism* $F\ \varphi\ A\ I\ B\ J$ = *homomorphism* $F\ \varphi'\ A'\ I'\ B'\ J'$
⟨*proof*⟩

context *algebra* **begin**

interpretation *unisorted*: *sorted-algebra* ‹*unisorted-sig* F› ‹*unisorted* A› I
⟨*proof*⟩

lemma *eval-homomorphism*: $\alpha : V \to A \Longrightarrow$ *homomorphism* F $(\lambda s.\ I[\![s]\!]\alpha)$ $\mathfrak{T}(F,V)$ *Fun* $A\ I$
⟨*proof*⟩

end

context *homomorphism* **begin**

interpretation *unisorted*: *sorted-homomorphism* ‹*unisorted-sig* F› φ ‹*unisorted* A› I ‹*unisorted* B› J
⟨*proof*⟩

lemma *distrib-eval*: $\alpha : V \to A \Longrightarrow s \in \mathfrak{T}(F,V) \Longrightarrow \varphi\ (I[\![s]\!]\alpha) = J[\![s]\!](\varphi \circ \alpha)$
⟨*proof*⟩

end

By 'unsorted' we mean the situation where any element has the unique type ().

lemma *Term-UNIV*[*simp*]: $\mathfrak{T}(UNIV,UNIV) = UNIV$
⟨*proof*⟩

When the carrier is unsorted, any interpretation forms an algebra.

interpretation *unsorted*: *algebra UNIV UNIV I*
rewrites $\bigwedge a.\ a \in UNIV \longleftrightarrow True$
and $\bigwedge P0.\ (True \Longrightarrow P0) \equiv Trueprop\ P0$

and $\bigwedge P0.\ (True \Longrightarrow PROP\ P0) \equiv PROP\ P0$
and $\bigwedge P0\ P1.\ (True \Longrightarrow PROP\ P1 \Longrightarrow P0) \equiv (PROP\ P1 \Longrightarrow P0)$
for $F\ I$
⟨*proof*⟩

interpretation *unsorted.eval*: *homomorphism UNIV* $\lambda s.\ I[\![s]\!]\alpha$ *UNIV Fun UNIV I*
rewrites $\bigwedge a.\ a \in UNIV \longleftrightarrow True$
and $\bigwedge X.\ X \subseteq UNIV \longleftrightarrow True$
and $\bigwedge P0.\ (True \Longrightarrow P0) \equiv Trueprop\ P0$
and $\bigwedge P0.\ (True \Longrightarrow PROP\ P0) \equiv PROP\ P0$
and $\bigwedge P0\ P1.\ (True \Longrightarrow PROP\ P1 \Longrightarrow P0) \equiv (PROP\ P1 \Longrightarrow P0)$
for I
⟨*proof*⟩

Evaluation distributes over evaluations in the term algebra, i.e., substitutions.

lemma *subst-eval*: $I[\![s{\cdot}\vartheta]\!]\alpha = I[\![s]\!](\lambda x.\ I[\![\vartheta\ x]\!]\alpha)$
⟨*proof*⟩

lemmas *subst-subst* = *subst-eval*[*of Fun*]

4.4.1 Collecting Variables via Evaluation

definition *var-list-term* $t \equiv (\lambda f.\ concat)[\![t]\!](\lambda v.\ [v])$

lemma *var-list-Fun*[*simp*]: *var-list-term* (*Fun f ss*) = *concat* (*map var-list-term ss*)
and *var-list-Var*[*simp*]: *var-list-term* (*Var x*) = [*x*]
⟨*proof*⟩

lemma *set-var-list*[*simp*]: *set* (*var-list-term s*) = *vars s*
⟨*proof*⟩

lemma *eval-subset-Un-vars*:
assumes $\forall f\ as.\ foo\ (I\ f\ as) \subseteq \bigcup(foo\ `\ set\ as)$
shows $foo\ (I[\![s]\!]\alpha) \subseteq (\bigcup x{\in}vars\text{-}term\ s.\ foo\ (\alpha\ x))$
⟨*proof*⟩

4.4.2 Ground terms

lemma *hastype-in-Term-empty-imp-vars*: $s : \sigma\ in\ \mathcal{T}(F,\emptyset) \Longrightarrow vars\ s = \{\}$
⟨*proof*⟩

lemma *hastype-in-Term-empty-imp-vars-subst*: $s : \sigma\ in\ \mathcal{T}(F,\emptyset) \Longrightarrow vars\ (s{\cdot}\vartheta) = \{\}$
⟨*proof*⟩

lemma *ground-Term-iff*: $s : \sigma\ in\ \mathcal{T}(F,V) \wedge ground\ s \longleftrightarrow s : \sigma\ in\ \mathcal{T}(F,\emptyset)$
⟨*proof*⟩

lemma *hastype-in-Term-empty-imp-subst*:
  $s : \sigma \ in \ \mathcal{T}(F,\emptyset) \Longrightarrow s{\cdot}\vartheta : \sigma \ in \ \mathcal{T}(F,V)$
  ⟨*proof*⟩

locale *subsignature* = **fixes** $F\ G$ :: $('f,'s)$ *ssig* **assumes** *subssig*: $F \subseteq_m G$
begin

lemmas *Term-subsset* = *Term-mono-left*[*OF subssig*]
lemmas *hastype-in-Term-sub* = *Term-subsset*[*THEN subssetD*]

lemma *subsignature*: $f : \sigma s \to \tau \ in \ F \Longrightarrow f : \sigma s \to \tau \ in \ G$
  ⟨*proof*⟩

end

locale *subsignature-algebra* = *subsignature* + *super*: *sorted-algebra G*
begin

sublocale *sorted-algebra F A I*
  ⟨*proof*⟩

end

locale *subalgebra* = *sorted-algebra F A I* + *super*: *sorted-algebra G B J* +
  subsignature F G
  for F :: $('f,'s)$ *ssig* **and** A :: $'a \rightharpoonup 's$ **and** I
  and G :: $('f,'s)$ *ssig* **and** B :: $'a \rightharpoonup 's$ **and** J +
  assumes *subcar*: $A \subseteq_m B$
  assumes *subintp*: $f : \sigma s \to \tau \ in \ F \Longrightarrow as :_l \sigma s \ in \ A \Longrightarrow I\ f\ as = J\ f\ as$
begin

lemma *subcarrier*: $a : \sigma \ in \ A \Longrightarrow a : \sigma \ in \ B$
  ⟨*proof*⟩

lemma *subeval*:
  assumes *s*: $s : \sigma \ in \ \mathcal{T}(F,V)$ **and** α: $\alpha :_s V \to A$ **shows** $J[\![s]\!]\alpha = I[\![s]\!]\alpha$
⟨*proof*⟩

end

lemma *term-subalgebra*:
  assumes *FG*: $F \subseteq_m G$ **and** *VW*: $V \subseteq_m W$
  shows *subalgebra* $F\ \mathcal{T}(F,V)$ *Fun* $G\ \mathcal{T}(G,W)$ *Fun*
  ⟨*proof*⟩

An algebra where every element has a representation:

locale *sorted-algebra-constant* = *sorted-algebra-syntax* +
  fixes *const*

assumes *vars-const*[*simp*]: $\bigwedge d.\ vars\ (const\ d) = \{\}$
assumes *eval-const*[*simp*]: $\bigwedge d\ \alpha.\ I[\![const\ d]\!]\alpha = d$
begin

lemma *eval-subst-const*[*simp*]: $I[\![e{\cdot}(const{\circ}\alpha)]\!]\beta = I[\![e]\!]\alpha$
⟨*proof*⟩

lemma *eval-upd-as-subst*: $I[\![e]\!]\alpha(x{:=}a) = I[\![e\ \cdot\ Var(x{:=}const\ a)]\!]\alpha$
⟨*proof*⟩

end

context *sorted-algebra-syntax* **begin**

definition *constant-at* $f\ \sigma s\ i \equiv$
$\forall\, as\ b.\ as :_l \sigma s\ in\ A \longrightarrow A\ b = A\ (as!i) \longrightarrow I\, f\ (as[i{:=}b]) = I\, f\ as$

lemma *constant-atI*[*intro*]:
assumes $\bigwedge as\ b.\ as :_l \sigma s\ in\ A \Longrightarrow A\ b = A\ (as!i) \Longrightarrow I\, f\ (as[i{:=}b]) = I\, f\ as$
shows *constant-at* $f\ \sigma s\ i$ ⟨*proof*⟩

lemma *constant-atD*:
constant-at $f\ \sigma s\ i \Longrightarrow as :_l \sigma s\ in\ A \Longrightarrow A\ b = A\ (as!i) \Longrightarrow I\, f\ (as[i{:=}b]) = I\, f$
as
⟨*proof*⟩

lemma *constant-atE*[*elim*]:
assumes *constant-at* $f\ \sigma s\ i$
and $(\bigwedge as\ b.\ as :_l \sigma s\ in\ A \Longrightarrow A\ b = A\ (as!i) \Longrightarrow I\, f\ (as[i{:=}b]) = I\, f\ as) \Longrightarrow$
thesis
shows *thesis* ⟨*proof*⟩

definition *constant-term-on* $s\ x \equiv \forall\, \alpha\ a.\ I[\![s]\!]\alpha(x{:=}a) = I[\![s]\!]\alpha$

lemma *constant-term-onI*:
assumes $\bigwedge\alpha\ a.\ I[\![s]\!]\alpha(x{:=}a) = I[\![s]\!]\alpha$ **shows** *constant-term-on* $s\ x$
⟨*proof*⟩

lemma *constant-term-onD*:
assumes *constant-term-on* $s\ x$ **shows** $I[\![s]\!]\alpha(x{:=}a) = I[\![s]\!]\alpha$
⟨*proof*⟩

lemma *constant-term-onE*:
assumes *constant-term-on* $s\ x$ **and** $(\bigwedge\alpha\ a.\ I[\![s]\!]\alpha(x{:=}a) = I[\![s]\!]\alpha) \Longrightarrow thesis$
shows *thesis* ⟨*proof*⟩

lemma *constant-term-on-extra-var*: $x \notin vars\ s \Longrightarrow$ *constant-term-on* $s\ x$
⟨*proof*⟩

lemma *constant-term-on-eq*:
assumes *st*: $I[\![s]\!] = I[\![t]\!]$ **and** *s*: *constant-term-on s x* **shows** *constant-term-on t x*
⟨*proof*⟩

definition *constant-term s* $\equiv \forall x.$ *constant-term-on s x*

lemma *constant-termI*: **assumes** $\bigwedge x.$ *constant-term-on s x* **shows** *constant-term s*
⟨*proof*⟩

lemma *ground-imp-constant*: *vars s* = {} $\Longrightarrow$ *constant-term s*
⟨*proof*⟩

end

end

5 Sorted Contexts

theory *Sorted-Contexts*
imports
First-Order-Terms.Subterm-and-Context
Sorted-Terms
begin

lemma *subt-in-dom*:
assumes *s*: $s \in$ *dom* $\mathcal{T}(F,V)$ **and** *st*: $s \trianglerighteq t$ **shows** $t \in$ *dom* $\mathcal{T}(F,V)$
⟨*proof*⟩

inductive *has-type-context* :: $('f,'s)$ *ssig* $\Rightarrow$ $('v \rightharpoonup 's) \Rightarrow 's \Rightarrow ('f,'v)$*ctxt* $\Rightarrow 's \Rightarrow$ *bool*
for $F\ V\ \sigma$ **where**
Hole: *has-type-context* $F\ V\ \sigma$ *Hole* σ
| *More*: $f : \sigma b$ @ ϱ # $\sigma a \to \tau$ *in* $F \Longrightarrow$
bef $:_l \sigma b$ *in* $\mathcal{T}(F,V) \Longrightarrow$ *has-type-context* $F\ V\ \sigma\ C\ \varrho \Longrightarrow$ *aft* $:_l \sigma a$ *in* $\mathcal{T}(F,V)$ $\Longrightarrow$
has-type-context $F\ V\ \sigma$ (*More f bef C aft*) τ

hide-fact (**open**) *Hole More*

abbreviation *has-type-context'* (((-) $:_c$/ (-) $\to$ (-) *in*/ $\mathcal{T}'$(-,-')) [*50*,*61*,*51*,*51*,*51*]*50*)
where $C :_c \sigma \to \tau$ *in* $\mathcal{T}(F,V) \equiv$ *has-type-context* $F\ V\ \sigma\ C\ \tau$

lemma *hastype-context-apply*:
assumes $C :_c \sigma \to \tau$ *in* $\mathcal{T}(F,V)$ **and** $t : \sigma$ *in* $\mathcal{T}(F,V)$

shows $C\langle t\rangle : \tau\ in\ \mathcal{T}(F,V)$
$\langle proof\rangle$

lemma *hastype-context-decompose*:
assumes $C\langle t\rangle : \tau\ in\ \mathcal{T}(F,V)$
shows $\exists\ \sigma.\ C :_c \sigma \to \tau\ in\ \mathcal{T}(F,V) \wedge t : \sigma\ in\ \mathcal{T}(F,V)$
$\langle proof\rangle$

lemma *apply-ctxt-in-dom-imp-in-dom*:
assumes $C\langle t\rangle \in dom\ \mathcal{T}(F,V)$
shows $t \in dom\ \mathcal{T}(F,V)$
$\langle proof\rangle$

lemma *apply-ctxt-hastype-imp-hastype-context*:
assumes C: $C\langle t\rangle : \tau\ in\ \mathcal{T}(F,V)$ **and** t: $t : \sigma\ in\ \mathcal{T}(F,V)$
shows $C :_c \sigma \to \tau\ in\ \mathcal{T}(F,V)$
$\langle proof\rangle$

lemma *subst-apply-ctxt-sorted*:
assumes $C :_c \sigma \to \tau\ in\ \mathcal{T}(F,X)$ **and** $\vartheta :_s X \to \mathcal{T}(F,V)$
shows $C \cdot_c \vartheta :_c \sigma \to \tau\ in\ \mathcal{T}(F,V)$
$\langle proof\rangle$

end

References

[1] C. Sternagel and R. Thiemann. First-order terms. *Archive of Formal Proofs*, February 2018. https://isa-afp.org/entries/First_Order_Terms.html, Formal proof development.

[2] R. Thiemann and A. Yamada. A verified algorithm for deciding pattern completeness. In J. Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction, FSCD 2024, July 10-13, 2024, Tallinn, Estonia*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. To appear.