

Foundations and Limits of Essentially Ordered Causal Regress Arguments

Arthur F. Ramos

David Barros Hulak

Ruy J. G. B. de Queiroz

October 28, 2026

Abstract

This entry formalizes a carrier-restricted theory of causal and grounding regress arguments in Isabelle/HOL. It distinguishes acyclicity, well-foundedness, explicit chain termination, local foundation, global foundation, and full grounding, and proves the implications that hold between them together with finite and infinite separation models. An executable finite checker returns either an exact independent-root certificate or a concrete missing-foundation witness, with kernel-checked correctness theorems. The entry also formalizes a bounded cycle analyzer with explicit finite return-path certificates, a certified finite profile linking five declarative properties to two executable tests, and premise-deletion countermodels for the Romero–Pérez and Cohoe-style packages. It formalizes two explicit Second-Way reconstructions: a Cohoe-style reconstruction in which an essentially ordered-series explication, explicit foundation, and bridge conditions yield a non-derivative first cause, and a Romero–Pérez-style well-founded reconstruction with its exact independent-member conclusion. The results are conditional on stated dependency premises; no theorem identifies a formal first cause with God, and the substantive application is deliberately focused on the Second Way.

1 Contribution and scope

The phrase “there must be a first cause” compresses several distinct claims. Acyclicity need not be well-foundedness; a selected target may have a foundation while the whole relation is non-terminating; and global foundation uses a set of independent supporters, not a singleton first-cause claim. This entry makes those distinctions explicit and mechanically checkable.

Its contribution is a compact package of four layers:

1. a carrier-restricted interface for dependence, ancestry, grounding, independence, and first causes;
2. an implication/separation atlas for acyclicity, termination, and local/global foundation;
3. executable finite certificates for independent roots, missing foundations, and closed cycle paths; and
4. source-oriented Second-Way formal explications with premise-deletion countermodels.

All conclusions are conditional on stated relation-theoretic assumptions. The entry does not assert that a physical or metaphysical domain is well-founded, or identify any formal first cause with God. It deliberately does not attempt the First or Third Ways: substantive treatments would require richer act/potency and modal semantics beyond this relation-theoretic scope.

2 Formal interface

Fix a type of entities, a carrier C , and an immediate-dependence relation D . The relation is read as “ x is an immediate essential cause, ground, or actualizer of y .” The project uses carrier-restricted predicates:

$$D_C(x, y) \equiv x \in C \wedge y \in C \wedge Dxy.$$

The predicate `dependency_ancestor` is the carrier-restricted reflexive-transitive closure of immediate dependence, while `grounded_in` records an immediate predecessor. Independence means that no such predecessor exists; `first_member_in` and `first_cause_in` add reachability and the required non-identity condition. The interface is neutral about whether D means efficient causation, grounding, or actualization.

3 Foundation principles

The entry formalizes the following predicates.

Predicate	Formal content
<code>acyclic_on</code>	no nonempty dependency cycle lies in the carrier;
<code>well_founded_on</code>	every nonempty carrier subset has a minimal member;
<code>chain_terminating_on</code>	every infinite dependency chain in the carrier is impossible;
<code>local_foundation_on</code>	every immediately grounded target in a selected set has a reachable independent member;
<code>global_foundation_on</code>	a designated set of independent members supports every grounded target in a selected set;
<code>full_foundation_on</code>	every partially grounded target in the selected set has a full ground contained in a common independent foundation set.

The local and global predicates are weaker than well-foundedness: they concern selected targets and do not quantify over every nonempty subset. In particular, `global_foundation_on` is witnessed by a set F of independent members; it is not a claim that there is one unique common first cause. Full grounding remains distinct from partial grounding.

The headline hierarchy is summarized below. The last two rows are strict separations, not additional assumptions.

Checked relation	Scope and witness
$WF \iff CT$	unrestricted equivalence of well-foundedness and chain termination;
$WF \implies AC$	unrestricted implication to acyclicity;
$WF \implies LF$; for $R \subseteq C$, $LF \iff GF$	well-foundedness implies local foundation without extra coverage; the local/global collapse requires carrier coverage;
$AC \iff WF \iff CT \implies LF$; for $R \subseteq C$, $LF \iff GF$	finite carrier; the rooted-cycle model refutes the final converse;
$AC \not\Rightarrow WF, LF$	integer predecessor model;
$LF, GF \not\Rightarrow AC$	finite rooted-cycle model.

4 Implication and separation atlas

The theorem `foundation_implication_atlas` collects the positive implications and names the bridge assumptions needed to convert full grounding into a local first-cause conclusion: target presence, target-set nonemptiness, and ancestry coverage. None is hidden in a simp rule.

4.1 Infinite separation models

The integer predecessor model is acyclic and gives every integer a predecessor, but has no independent member. The pedestal model has a foundation set while remaining non-well-founded. The finite rooted-cycle model has foundation for selected targets while retaining a separate cycle. The paired model separates partial from full grounding. These are executable Isabelle/HOL models used by the Second-Way comparison layer, not informal diagrams.

5 Executable finite checker

`Finite_Foundation_Checker.thy` represents a finite model by a `nat list` of vertices and ordered edge pairs, then derives the carrier, relation, bounded reachability, grounding, and independence predicates. The path recursion is proved sound and complete under the finite-list invariant.

The executable function `finite_foundation_result` returns one of two constructors:

Result	Meaning
<code>Foundation_Certificate xs</code>	<code>xs</code> is exactly the list of independent vertices in the checked carrier, and its set is proved to be a <code>foundation_set_for</code> ;
<code>Missing_Foundation_Certificate x</code>	<code>x</code> is a concrete carrier vertex that is grounded but has no reachable independent member.

The correctness theorems connect the result with `finite_foundation_check`, identify the positive payload with the exact independent-root set, and certify the negative witness. Regression models cover a finite chain and directed cycle; the checker is therefore a small certified assay, not an appeal to an external graph library.

5.1 Cycle certificates and the finite profile

`Finite_Foundation_Analysis.thy` adds a second executable result type. `No_Cycle_Certificate` is returned exactly when the carrier-restricted relation is acyclic. Otherwise `Cycle_Certificate x xs` returns a carrier vertex `x` together with a nonempty finite path list `xs` for which the formal transitive closure contains a return path

$$D_C^+(x, x).$$

The certificate theorem checks the closed `rtranc1_path` represented by `xs`, rather than returning only an endpoint. The internal search is bounded by the listed vertices; the emitted path is at most one longer because it prepends the first edge's endpoint. This is proved by `finite_cycle_result_cycle_length`.

The exported `finite_foundation_profile` packages five Boolean answers, computed by two executable tests:

(acyclic, well-founded, chain-terminating, local-foundation, global-foundation).

The theorem `finite_foundation_profile_correct` proves equality with the declarative tuple for every finite input model. Thus the profile exposes the finite collapse at the executable API boundary while making clear that acyclicity, well-foundedness, and chain termination share one cycle test, and local and global foundation share one foundation test.

6 Two explicit Second-Way reconstructions

The source comparison is guided by the literature on essentially ordered causal series. Cohoe’s analysis emphasizes that Aquinas’s argument concerns a series whose later members derive their causal power from earlier members [4]. Romero and Pérez present a first-order reconstruction whose formal source-level conclusion is the existence of an uncaused entity, while warning against silently adding uniqueness or a universe-cause conclusion [8].

6.1 Source-to-formal premise map

The following table separates source-level commitments from the Isabelle encoding. In particular, the bridge conditions are recorded as formal translation assumptions, not presented as quotations from the philosophical literature.

Source focus	Formal counterpart	Role in the result
Cohoe: an essentially ordered causal series [4]	<code>essentially_ordered_series</code> , with a derivative predicate and a carrier-restricted immediate relation	An explicit explication; it is not identified with well-foundedness.
Cohoe: foundation of the selected series	<code>full_foundation_on</code> plus endpoint, nonemptiness, and ancestry bridges	Supplies the binary foundation conclusion only under stated translation assumptions.
Romero–Pérez: caused items and predecessor coverage [8]	Existence, carrier, and predecessor premises in <code>romero_perez_second_way_premises</code>	Feeds the per-target first-cause theorem.
Romero–Pérez: no infinite descent	<code>well_founded_on</code> in Isabelle/HOL	The stronger global HOL explication of the source’s target-oriented no-infinite-descent premise; its deletion is witnessed by the integer predecessor model.

6.2 Cohoe-style essential ordering

The definition `essentially_ordered_series` requires, for a chosen target, an explicit derivative predicate and an immediate dependency predecessor. More importantly, on the selected ancestry it proves the equivalence

$$\text{derivative}(x) \longleftrightarrow \text{grounded_in}(C, D, x).$$

This equivalence is the development’s explicit formal explication of one reading of derivative efficacy; it is not asserted as a metaphysical identity between every sense of being caused and every sense of possessing causal power. Under that explication, the first member of a founded series is formally non-derivative rather than merely receiving a free source annotation. The predicate is still not a synonym for well-foundedness: the integer predecessor model satisfies the essential-ordering interface while remaining non-well-founded. The locale `cohoe_second_way_reconstruction` adds the formal translation conditions needed to obtain the exact conclusion: independent foundation, singleton endpoint, nonemptiness, and full-ancestry bridges. These are formal explications of the source claims, not literal quotations:

a caused target has a non-derivative first cause in the selected carrier.

The conclusion is intentionally not “God exists.” It is a first-cause statement in the typed relation-and-carrier interface.

6.3 An auxiliary bounded stress test for local causal efficacy

As an auxiliary stress test, the development introduces a deliberately explicit local support interface. It requires an immediate cause to exercise a power possessed by an earlier supporter. On the integer predecessor model, the possession and exercise predicates are instantiated by the constantly true predicates. The resulting theorem shows that this precise local support/transmission condition is insufficient to exclude an infinite regress, even alongside acyclicity and the formal essential-ordering interface. It is not a substantive semantics of derivative causal efficacy, nor a claim that every possible metaphysical theory fails to exclude such a regress; its role is to mark the exact boundary of this deliberately weak interface.

6.4 Romero–Pérez reconstruction

The locale `romero_perez_second_way_reconstruction` formalizes the following package: at least one caused entity exists; caused entities lie in the selected carrier; every caused entity has a carrier predecessor; and the carrier-restricted relation is well-founded. We represent the source’s prohibition of an infinite regress resulting in a particular item by the stronger global HOL condition `well_founded_on`, rather than by a target-relative sentence. The phrase “first-order” describes the source presentation, not this Isabelle/HOL encoding: `well_founded_on` quantifies over carrier subsets. Its exact headline conclusion is

$$\exists x \in C. \text{independent_in } C \ D \ x.$$

For a specified caused target the locale also derives a first cause. It does not add uniqueness, necessity, a cause of the universe, or a theological identity principle. Its conditions are source-oriented formal assumptions; the separate premise theory supplies the one-premise-at-a-time countermodel atlas.

6.5 Premise factorisation and deletion certificates

`Premise_Minimality.thy` factors the Romero–Pérez formal package into four conditions: existence, carrier membership, predecessor coverage, and well-foundedness. Four deletion theorems retain the others and refute one fixed combined conclusion: a global independent-support set together with a first cause for every caused item. The empty, finite, and integer models delete these conditions one at a time.

The Cohoe-style translation package uses existence, the formal essential-ordering explication, full foundation, and three bridges from full grounding to binary dependence. Isabelle proves that a separate carrier condition is redundant because the explication is carrier-restrictive. Six deletion theorems refute the formal non-derivative conclusion with small, inspectable models. These are minimality results for the encoding, not claims that the literature has only one interpretation of a source claim.

7 Related work and limitations

The formal motivation is Aquinas’s *Summa Theologiae* [1]. Cohoe and Romero–Pérez motivate the two source-oriented reconstructions, while Dixon, Pearson, Bliss, and Rabin–Rabern motivate the distinctions between grounding, termination, and vicious regress [4, 8, 5, 6, 3, 7]. Prior mechanized philosophical theology provides a precedent for separating a checked logical core from interpretation [2].

The formal contribution is comparative rather than a claim to replace these source discussions. The cited philosophical analyses do not supply the carrier-restricted Isabelle/HOL lattice proved here, its finite and infinite separation models, or executable certificates whose payloads are connected to the declarative predicates by kernel-checked theorems. Cohoe’s essentially ordered

series and Romero-Pérez’s no-infinite-descent premise are therefore used as distinct source-oriented inputs to formal explications; the additional carrier, ancestry, endpoint, and grounding bridges are exposed as translation assumptions. The result is a mechanically inspectable comparison of what follows from those choices, not a claim that the formal vocabulary exhausts the metaphysics of causation or grounding.

The entry does not formalize empirical causal truth, Aristotelian act and potency, modal semantics, or the theological identity of a first cause. It formalizes consequences of stated relation-theoretic conditions and exhibits countermodels when a condition is weakened. All definitions, theorem statements, proofs, executable examples, and explanatory material needed to understand the contribution are contained in this AFP entry.

7.1 Logical background

The development is carried out in classical Isabelle/HOL. In particular, the standard library equivalence between well-foundedness and the absence of an infinite descending chain is used in `well_founded_iff_chain_terminating`; the encoding is therefore not a first-order theory and does not claim first-order completeness. The theorem statements expose the relevant set and sequence quantification instead of concealing it behind the phrase “no infinite regress”.

8 Reproducibility

The Isabelle session is built with

```
isabelle build -D . Thomistic_Causal_Arguments.
```

For proof-only iteration, use `isabelle build -o document=false -D . Thomistic_Causal_Arguments`. The repository-level audit additionally checks the AFP layout, forbidden proof tokens, stray generated artifacts, and headline theorem counts. The Blueprint file records the dependency DAG and the proved foundation/Second-Way core.

9 Conclusion

The result is deliberately narrower than a theological proof: once the relation, carrier, and foundation conditions are explicit, Isabelle verifies the hierarchy, certificates, countermodels, and source distinctions. That combination is the contribution of the entry.

Contents

1	Contribution and scope	1
2	Formal interface	2
3	Foundation principles	2
4	Implication and separation atlas	2
4.1	Infinite separation models	3
5	Executable finite checker	3
5.1	Cycle certificates and the finite profile	3

6	Two explicit Second-Way reconstructions	4
6.1	Source-to-formal premise map	4
6.2	Cohoe-style essential ordering	4
6.3	An auxiliary bounded stress test for local causal efficacy	5
6.4	Romero-Pérez reconstruction	5
6.5	Premise factorisation and deletion certificates	5
7	Related work and limitations	5
7.1	Logical background	6
8	Reproducibility	6
9	Conclusion	6
10	Dependency Ancestry	8
11	First Members	9
12	Carrier-Restricted Dependency	10
13	Compatibility with the Existing Ancestry Predicate	11
14	Grounded and Independent Members	12
15	Competing Binary Principles	13
16	Plural Grounding	14
17	Well-Foundedness and Minimal Ancestors	15
18	Well-Foundedness and Explicit Chains	16
19	Finite Acyclicity Collapse	18
20	Global Foundation	19
21	Full-to-Binary Comparison	21
22	Integer Predecessor Model	23
23	Acyclic but not Well-Founded	23
24	Foundation without Acyclicity	25
25	A Fully Pedestalled Infinite Structure	26
26	Partial and Full Grounding Diverge	28
27	Executable Finite Models	30
28	Finite Foundation Predicate and Checker	34
29	Executable Regression Models	44
30	An Explicit Essentially Ordered-Series Interface	44
31	Cohoe-Style Formal Translation Package	46

32 Romero–Pérez Reconstruction	48
33 Comparison and Separation Summary	49
34 Exact Binary Foundation Classification	51
35 Auxiliary Local Causal-Efficacy Stress Test	53
36 Premise Factorisation	54
37 Romero–Pérez Deletion Certificates	56
38 Cohoe-Style Formal-Translation Deletion	60
39 Cycle Certificates	77
40 The Finite Foundation Profile	89
41 Summary	90
theory <i>Preliminaries</i>	
imports <i>Main</i>	
begin	

10 Dependency Ancestry

The relation $D\ x\ y$ is read as: x is an immediate essential ground, cause, or actualizer of y . The inductive predicate $ancestor\ D\ x\ y$ is the reflexive transitive closure of that dependency relation, oriented from earlier or more fundamental members toward later or dependent members.

inductive $ancestor :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
for $D :: 'a \Rightarrow 'a \Rightarrow bool$

where

refl: $ancestor\ D\ x\ x$

| *step*: $D\ x\ y \implies ancestor\ D\ y\ z \implies ancestor\ D\ x\ z$

lemma *ancestor-direct*:

assumes $D\ x\ y$

shows $ancestor\ D\ x\ y$

proof –

have $ancestor\ D\ y\ y$

by (*rule ancestor.refl*)

with *assms show ?thesis*

by (*rule ancestor.step*)

qed

lemma *ancestor-trans*:

assumes $ancestor\ D\ x\ y$ **and** $ancestor\ D\ y\ z$

shows $ancestor\ D\ x\ z$

using *assms(1) assms(2)*

proof (*induction arbitrary: z rule: ancestor.induct*)

case (*refl x*)

then show *?case* .

next

case (*step x y u*)

have $ancestor\ D\ y\ z$

by (*rule step.IH[OF step.prem]*)

with *step.hyps(1) show ?case*

by (*rule ancestor.step*)

qed

lemma *ancestor-nontrivial-has-successor*:

assumes *ancestor* $D\ x\ y$ **and** $x \neq y$

shows $\exists z. D\ x\ z$

using *assms*

proof *cases*

case *refl*

with *assms* **show** *?thesis* **by** *simp*

next

case (*step* z)

then show *?thesis*

by *blast*

qed

11 First Members

definition *ungrounded* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \Rightarrow \text{bool}$ **where**

ungrounded $D\ x \longleftrightarrow (\nexists y. D\ y\ x)$

definition *first-member-of* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ **where**

first-member-of $D\ x\ a \longleftrightarrow \text{ancestor}\ D\ x\ a \wedge \text{ungrounded}\ D\ x$

definition *first-cause* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \Rightarrow \text{bool}$ **where**

first-cause $D\ x \longleftrightarrow (\exists y. D\ x\ y) \wedge \text{ungrounded}\ D\ x$

definition *first-cause-of* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ **where**

first-cause-of $D\ x\ a \longleftrightarrow \text{first-member-of}\ D\ x\ a \wedge x \neq a$

lemma *first-member-not-target-if-target-has-predecessor*:

assumes *first-member-of* $D\ x\ a$ **and** $D\ y\ a$

shows $x \neq a$

using *assms* **by** (*auto simp: first-member-of-def ungrounded-def*)

lemma *first-cause-of-imp-first-cause*:

assumes *first-cause-of* $D\ x\ a$

shows *first-cause* $D\ x$

proof –

have *first-member-of* $D\ x\ a$ **and** $x \neq a$

using *assms* **by** (*auto simp: first-cause-of-def*)

then have *ancestor* $D\ x\ a$ **and** *ungrounded* $D\ x$

by (*auto simp: first-member-of-def*)

from *ancestor-nontrivial-has-successor*[*OF* $\langle \text{ancestor}\ D\ x\ a \rangle \langle x \neq a \rangle$]

obtain y **where** $D\ x\ y$

by *auto*

with $\langle \text{ungrounded}\ D\ x \rangle$ **show** *?thesis*

by (*auto simp: first-cause-def*)

qed

lemma *wf-has-first-member*:

assumes *wfP* D

shows $\exists m. \text{first-member-of}\ D\ m\ a$

using *assms*

proof (*induction a rule: wfp-induct-rule*)

case (*less* a)

show *?case*

proof (*cases* $\exists b. D\ b\ a$)

case *False*

```

then show ?thesis
  by (intro exI[of - a])
    (auto simp: first-member-of-def ungrounded-def intro: ancestor.refl)
next
  case True
  then obtain b where Db: D b a by blast
  from less.IH[OF Db] obtain m where first-member-of D m b
    by blast
  then have ancestor D m b and ungrounded D m
    by (auto simp: first-member-of-def)
  have ancestor D b a
    using Db by (rule ancestor-direct)
  have ancestor D m a
    using ⟨ancestor D m b⟩ ⟨ancestor D b a⟩ by (rule ancestor-trans)
  then show ?thesis
    using ⟨ungrounded D m⟩
    by (auto simp: first-member-of-def)
qed
qed

lemma wf-has-first-cause-of-if-predecessor:
  assumes wfP D and D b a
  shows  $\exists m.$  first-cause-of D m a
proof –
  from wf-has-first-member[OF assms(1), of a] obtain m
    where m: first-member-of D m a
    by blast
  moreover have m  $\neq$  a
    using first-member-not-target-if-target-has-predecessor[OF m assms(2)] .
  ultimately show ?thesis
    by (auto simp: first-cause-of-def)
qed

end
theory Dependency-Structures
  imports Preliminaries
begin

```

12 Carrier-Restricted Dependency

A dependency structure consists of a carrier and an immediate relation. The orientation is fixed throughout this development: $D\ x\ y$ means that x is prior to, grounds, or immediately supports y . The carrier restriction is explicit so that unrelated elements cannot become accidental first causes.

definition *dependency-on* ::

$'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$

where

$dependency-on\ C\ D\ x\ y \longleftrightarrow x \in C \wedge y \in C \wedge D\ x\ y$

lemma *dependency-onI*:

assumes $x \in C$ **and** $y \in C$ **and** $D\ x\ y$

shows $dependency-on\ C\ D\ x\ y$

using *assms* **by** (*simp add: dependency-on-def*)

lemma *dependency-onD*:

assumes $dependency-on\ C\ D\ x\ y$

shows $x \in C\ y \in C\ D\ x\ y$

using *assms* **by** (*auto simp: dependency-on-def*)

definition *dependency-ancestor* ::
 $'a \text{ set} \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$
where
 $\text{dependency-ancestor } C \ D \ x \ y \longleftrightarrow$
 $x \in C \wedge y \in C \wedge (\text{dependency-on } C \ D)^{**} \ x \ y$

lemma *dependency-ancestor-refl*:
assumes $x \in C$
shows $\text{dependency-ancestor } C \ D \ x \ x$
using *assms* **by** (*simp add: dependency-ancestor-def*)

lemma *dependency-ancestor-direct*:
assumes $x \in C$ **and** $y \in C$ **and** $D \ x \ y$
shows $\text{dependency-ancestor } C \ D \ x \ y$
proof –
have $\text{dependency-on } C \ D \ x \ y$
using *assms* **by** (*simp add: dependency-on-def*)
then have $(\text{dependency-on } C \ D)^{**} \ x \ y$
by (*rule r-into-rtrancpl*)
then show *?thesis*
using *assms* **by** (*simp add: dependency-ancestor-def*)
qed

lemma *dependency-ancestor-trans*:
assumes $\text{dependency-ancestor } C \ D \ x \ y$
and $\text{dependency-ancestor } C \ D \ y \ z$
shows $\text{dependency-ancestor } C \ D \ x \ z$
proof –
from *assms* **obtain**
 $x \in C \ y \in C \ (\text{dependency-on } C \ D)^{**} \ x \ y$
 $z \in C \ (\text{dependency-on } C \ D)^{**} \ y \ z$
by (*auto simp: dependency-ancestor-def*)
then show *?thesis*
by (*auto simp: dependency-ancestor-def intro: rtrancpl-trans*)
qed

13 Compatibility with the Existing Ancestry Predicate

lemma *ancestor-iff-rtrancpl*:
 $\text{ancestor } D \ x \ y \longleftrightarrow D^{**} \ x \ y$
proof
assume $\text{ancestor } D \ x \ y$
then show $D^{**} \ x \ y$
proof (*induction rule: ancestor.induct*)
case *refl*
show *?case* **by** (*rule rtrancpl.rtrancpl-refl*)
next
case (*step* $x \ y \ z$)
have $D^{**} \ x \ y$
using *step.hyps(1)* **by** (*rule r-into-rtrancpl*)
with *step.IH* **show** *?case*
by (*blast intro: rtrancpl-trans*)
qed
next
assume $D^{**} \ x \ y$
then show $\text{ancestor } D \ x \ y$
proof (*induction rule: rtrancpl-induct*)

```

case base
show ?case by (rule ancestor.refl)
next
case (step y z)
have ancestor D y z
  using step.hyps(2) by (rule ancestor-direct)
then show ?case
  using step.IH by (blast intro: ancestor-trans)
qed
qed

lemma legacy-ancestor-iff-dependency-ancestor:
  ancestor D x y  $\longleftrightarrow$  dependency-ancestor UNIV D x y
proof –
  have relation-eq: dependency-on UNIV D = D
    by (rule ext, rule ext) (simp add: dependency-on-def)
  then show ?thesis
    by (simp add: ancestor-iff-rtranclp dependency-ancestor-def)
qed

```

14 Grounded and Independent Members

definition *grounded-in* ::
'a set $\Rightarrow$ (*'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ *'a* $\Rightarrow$ *bool*
where
grounded-in C D x $\longleftrightarrow$ $x \in C \wedge (\exists y \in C. D y x)$

definition *independent-in* ::
'a set $\Rightarrow$ (*'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ *'a* $\Rightarrow$ *bool*
where
independent-in C D x $\longleftrightarrow$ $x \in C \wedge$
 $(\neg (\exists y \in C. D y x))$

lemma *grounded-inD*:
assumes *grounded-in C D x*
shows $x \in C \exists y \in C. D y x$
using *assms* **by** (*auto simp: grounded-in-def*)

lemma *independent-inD*:
assumes *independent-in C D x*
shows $x \in C \neg (\exists y \in C. D y x)$
using *assms* **by** (*auto simp: independent-in-def*)

lemma *grounded-not-independent*:
assumes *grounded-in C D x*
shows $\neg$ *independent-in C D x*
using *assms* **by** (*auto simp: grounded-in-def independent-in-def*)

definition *first-member-in* ::
'a set $\Rightarrow$ (*'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
where
first-member-in C D x a $\longleftrightarrow$
dependency-ancestor C D x a $\wedge$ *independent-in C D x*

definition *first-cause-in* ::
'a set $\Rightarrow$ (*'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
where
first-cause-in C D x a $\longleftrightarrow$

first-member-in $C\ D\ x\ a \wedge x \neq a$

lemma *first-member-inD*:
assumes *first-member-in* $C\ D\ x\ a$
shows *dependency-ancestor* $C\ D\ x\ a$ *independent-in* $C\ D\ x$
using *assms* **by** (*auto simp: first-member-in-def*)

lemma *first-cause-inD*:
assumes *first-cause-in* $C\ D\ x\ a$
shows *first-member-in* $C\ D\ x\ a\ x \neq a$
using *assms* **by** (*auto simp: first-cause-in-def*)

definition *foundation-set-for* ::
 $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow 'a\ set \Rightarrow bool$
where
foundation-set-for $C\ D\ R\ F \longleftrightarrow$
 $F \subseteq C \wedge R \subseteq C \wedge$
 $(\forall f \in F. \text{independent-in } C\ D\ f) \wedge$
 $(\forall a \in R. \text{grounded-in } C\ D\ a \longrightarrow$
 $(\exists f \in F. \text{dependency-ancestor } C\ D\ f\ a))$

lemma *foundation-set-forD*:
assumes *foundation-set-for* $C\ D\ R\ F$
shows $F \subseteq C\ R \subseteq C$
and $\forall f \in F. \text{independent-in } C\ D\ f$
and $\forall a \in R. \text{grounded-in } C\ D\ a \longrightarrow$
 $(\exists f \in F. \text{dependency-ancestor } C\ D\ f\ a)$
using *assms* **by** (*auto simp: foundation-set-for-def*)

end
theory *Foundation-Principles*
imports *Dependency-Structures*
begin

15 Competing Binary Principles

The same immediate relation supports several non-equivalent readings of a regress argument. We keep them as separate predicates. In particular, acyclicity is only a prohibition on nonempty cycles, while well-foundedness is a global condition on all predecessor chains.

definition *acyclic-on* ::
 $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
where
acyclic-on $C\ D \longleftrightarrow$
 $(\forall x \in C. \neg (\text{dependency-on } C\ D)^{++} x\ x)$

definition *well-founded-on* ::
 $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
where
well-founded-on $C\ D \longleftrightarrow \text{wfp-on } C\ (\text{dependency-on } C\ D)$

definition *chain-terminating-on* ::
 $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
where
chain-terminating-on $C\ D \longleftrightarrow$
 $(\nexists f. \forall n. \text{dependency-on } C\ D\ (f\ (\text{Suc } n))\ (f\ n))$

definition *local-foundation-on* ::

'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ 'a set $\Rightarrow$ bool
where
 local-foundation-on C D R $\longleftrightarrow$
 ($\forall a \in R.$ grounded-in C D a $\longrightarrow$
 ($\exists f.$ first-member-in C D f a))

definition global-foundation-on ::
 'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ 'a set $\Rightarrow$ bool
where
 global-foundation-on C D R $\longleftrightarrow$
 ($\exists F.$ foundation-set-for C D R F)

lemma global-foundation-onI:
assumes foundation-set-for C D R F
shows global-foundation-on C D R
using assms **by** (auto simp: global-foundation-on-def)

lemma global-foundation-onD:
assumes global-foundation-on C D R
obtains F **where** foundation-set-for C D R F
using assms **by** (auto simp: global-foundation-on-def)

16 Plural Grounding

A full ground may be a set rather than a single member. The carrier restriction is part of the interface. This makes the later comparison with binary dependence explicit instead of silently identifying singleton and plural grounding.

definition full-grounding-on ::
 'a set $\Rightarrow$ ('a set $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ 'a set $\Rightarrow$ 'a $\Rightarrow$ bool
where
 full-grounding-on C G S x $\longleftrightarrow$
 $S \subseteq C \wedge x \in C \wedge G S x$

definition partial-grounding-on ::
 'a set $\Rightarrow$ ('a set $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ 'a $\Rightarrow$ 'a $\Rightarrow$ bool
where
 partial-grounding-on C G x y $\longleftrightarrow$
 $x \in C \wedge y \in C \wedge$
 ($\exists S.$ full-grounding-on C G S y $\wedge x \in S$)

definition full-foundation-for ::
 'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$
 ('a set $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ 'a set $\Rightarrow$ 'a set $\Rightarrow$ bool
where
 full-foundation-for C D G R F $\longleftrightarrow$
 $F \subseteq C \wedge R \subseteq C \wedge$
 ($\forall f \in F.$ independent-in C D f) $\wedge$
 ($\forall a \in R.$
 grounded-in C (partial-grounding-on C G) a $\longrightarrow$
 ($\exists S.$ full-grounding-on C G S a $\wedge S \subseteq F$))

definition full-foundation-on ::
 'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$
 ('a set $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ 'a set $\Rightarrow$ bool
where
 full-foundation-on C D G R $\longleftrightarrow$
 ($\exists F.$ full-foundation-for C D G R F)

```

lemma full-grounding-onD:
  assumes full-grounding-on C G S x
  shows  $S \subseteq C$   $x \in C$  G S x
  using assms by (auto simp: full-grounding-on-def)

lemma full-grounding-onI:
  assumes  $S \subseteq C$  and  $x \in C$  and G S x
  shows full-grounding-on C G S x
  using assms by (auto simp: full-grounding-on-def)

lemma partial-grounding-onI:
  assumes full-grounding-on C G S y and  $x \in S$ 
  shows partial-grounding-on C G x y
  using assms
  by (auto simp: partial-grounding-on-def full-grounding-on-def)

lemma partial-grounding-onD:
  assumes partial-grounding-on C G x y
  shows  $x \in C$   $y \in C$ 
    and  $\exists S. \text{full-grounding-on } C \text{ G } S \text{ y} \wedge x \in S$ 
  using assms by (auto simp: partial-grounding-on-def)

lemma full-foundation-forD:
  assumes full-foundation-for C D G R F
  shows  $F \subseteq C$   $R \subseteq C$ 
    and  $\forall f \in F. \text{independent-in } C \text{ D } f$ 
    and  $\forall a \in R. \text{grounded-in } C \text{ (partial-grounding-on } C \text{ G) } a \longrightarrow$ 
       $(\exists S. \text{full-grounding-on } C \text{ G } S \text{ a} \wedge S \subseteq F)$ 
  using assms by (auto simp: full-foundation-for-def)

lemma full-foundation-onI:
  assumes full-foundation-for C D G R F
  shows full-foundation-on C D G R
  using assms by (auto simp: full-foundation-on-def)

lemma full-foundation-onD:
  assumes full-foundation-on C D G R
  obtains F where full-foundation-for C D G R F
  using assms by (auto simp: full-foundation-on-def)

end
theory Foundation-Implications
  imports Foundation-Principles
begin

```

17 Well-Foundedness and Minimal Ancestors

```

lemma well-founded-has-minimal-ancestor:
  assumes well-founded-on C D and  $x \in C$ 
  obtains m where  $m \in C$ 
    and dependency-ancestor C D m x
    and independent-in C D m
proof -
  have wf: wfp-on C (dependency-on C D)
    using assms(1) by (simp add: well-founded-on-def)
  obtain m where mC:  $m \in C$ 

```

```

    and m-ancestor: (dependency-on C D)** m x
    and m-minimal:  $\neg (\exists z \in C. \text{dependency-on } C D z m)$ 
    using bex-rtransclp-min-element-if-wfp-on[OF wf assms(2)] by blast
  have dependency-ancestor C D m x
    using mC assms(2) m-ancestor
    by (simp add: dependency-ancestor-def)
  then have m-ancestor': dependency-ancestor C D m x .
  have m-independent: independent-in C D m
    using mC m-minimal
    by (auto simp: independent-in-def dependency-on-def)
  show thesis
    using mC m-ancestor' m-independent by (rule that)
qed

```

```

lemma well-founded-implies-local-foundation:
  assumes well-founded-on C D
  shows local-foundation-on C D R
proof -
  show ?thesis
    unfolding local-foundation-on-def
  proof (intro ballI impI)
    fix a
    assume a  $\in R$  and grounded-in C D a
    have a  $\in C$ 
      using  $\langle \text{grounded-in } C D a \rangle$  by (auto simp: grounded-in-def)
    obtain f where f  $\in C$ 
      and dependency-ancestor C D f a
      and independent-in C D f
      using well-founded-has-minimal-ancestor[OF assms  $\langle a \in C \rangle$ ] .
    show  $\exists f. \text{first-member-in } C D f a$ 
      using  $\langle \text{dependency-ancestor } C D f a \rangle \langle \text{independent-in } C D f \rangle$ 
      by (intro exI) (simp add: first-member-in-def)
  qed
qed

```

18 Well-Foundedness and Explicit Chains

```

lemma well-founded-iff-chain-terminating:
  well-founded-on C D  $\longleftrightarrow$  chain-terminating-on C D
proof -
  have relation-eq:
     $(\lambda x y. \text{dependency-on } C D x y \wedge x \in C \wedge y \in C) =$ 
     $\text{dependency-on } C D$ 
    by (rule ext, rule ext) (auto simp: dependency-on-def)
  have relation:
    well-founded-on C D  $\longleftrightarrow$ 
    wf  $\{(x, y). \text{dependency-on } C D x y\}$ 
  proof -
    have h-wfp:
      wfp-on C (dependency-on C D)  $\longleftrightarrow$ 
      wfp (dependency-on C D)
      using wfp-on-iff-wfp[of C dependency-on C D] relation-eq
      by simp
    show ?thesis
      unfolding well-founded-on-def
      using h-wfp
      by (simp add: wfp-def)
  qed
qed

```

```

have chain-relation:
  wf  $\{(x, y). \text{dependency-on } C D x y\} \longleftrightarrow$ 
    chain-terminating-on  $C D$ 
  unfolding chain-terminating-on-def
  by (simp add: wf-iff-no-infinite-down-chain)
show ?thesis
  using relation chain-relation by blast
qed

lemma well-founded-implies-chain-terminating:
  assumes well-founded-on  $C D$ 
  shows chain-terminating-on  $C D$ 
  using well-founded-iff-chain-terminating assms by blast

lemma dependency-trancl-to-set:
   $(\text{dependency-on } C D)^{++} x y \implies$ 
   $(x, y) \in \{(u, v). \text{dependency-on } C D u v\}^+$ 
proof (induction rule: tranclp-induct)
  case base
  then show ?case
  by (auto intro: trancl.r-into-trancl)
next
  case step
  show ?case
  using step.IH step.hyps(2)
  by (auto intro: trancl-into-trancl)
qed

lemma dependency-set-to-trancl:
   $(x, y) \in \{(u, v). \text{dependency-on } C D u v\}^+ \implies$ 
   $(\text{dependency-on } C D)^{++} x y$ 
proof (induction rule: trancl-induct)
  case base
  then show ?case by simp
next
  case step
  then show ?case
  by (auto intro: tranclp-into-rtranclp rtranclp-into-tranclp1)
qed

lemma well-founded-implies-acyclic:
  assumes well-founded-on  $C D$ 
  shows acyclic-on  $C D$ 
proof –
  have relation-eq:
     $(\lambda x y. \text{dependency-on } C D x y \wedge x \in C \wedge y \in C) =$ 
    dependency-on  $C D$ 
  by (rule ext, rule ext) (auto simp: dependency-on-def)
  have wfp-rel: wfp (dependency-on  $C D$ )
  using assms wfp-on-iff-wfp[of  $C$  dependency-on  $C D$ ] relation-eq
  by (simp add: well-founded-on-def)
  have acyclic-rel:
    acyclic  $\{(x, y). \text{dependency-on } C D x y\}$ 
  using wfp-rel by (rule wfp-acyclicP)
  show acyclic-on  $C D$ 
  unfolding acyclic-on-def
proof (intro ballI notI)
  fix x
  assume  $x \in C$ 

```

```

assume cycle: (dependency-on C D)++ x x
have set-cycle:
  (x, x) ∈ {(u, v). dependency-on C D u v}+
  using dependency-trancl-to-set[OF cycle] .
show False
  using acyclic-rel set-cycle by (auto simp: acyclic-def)
qed
qed

```

19 Finite Acyclicity Collapse

```

lemma finite-well-founded-iff-acyclic-on:
  assumes finite C
  shows well-founded-on C D  $\longleftrightarrow$  acyclic-on C D
proof –
  have relation-eq:
    ( $\lambda x y.$  dependency-on C D x y  $\wedge$  x  $\in$  C  $\wedge$  y  $\in$  C) =
    dependency-on C D
  by (rule ext, rule ext) (auto simp: dependency-on-def)
  have finite-relation:
    finite {(x, y). dependency-on C D x y}
proof –
  have subset: {(x, y). dependency-on C D x y}  $\subseteq$  C  $\times$  C
    by (auto simp: dependency-on-def)
  have finite-CC: finite (C  $\times$  C)
    using assms by simp
  show ?thesis
    by (rule finite-subset[OF subset finite-CC])
qed
show ?thesis
proof (rule iffI)
  assume well: well-founded-on C D
  have wfp-rel: wfp (dependency-on C D)
    using well wfp-on-iff-wfp[of C dependency-on C D] relation-eq
    by (simp add: well-founded-on-def)
  have acyclic-rel:
    acyclic {(x, y). dependency-on C D x y}
    using wfp-rel by (rule wfp-acyclicP)
  show acyclic-on C D
    unfolding acyclic-on-def
proof (intro ballI notI)
  fix x
  assume x  $\in$  C
  assume cycle: (dependency-on C D)++ x x
  have set-cycle:
    (x, x) ∈ {(u, v). dependency-on C D u v}+
    using dependency-trancl-to-set[OF cycle] .
  show False
    using acyclic-rel set-cycle by (auto simp: acyclic-def)
qed
next
  assume acyclic: acyclic-on C D
  have acyclic-rel:
    acyclic {(x, y). dependency-on C D x y}
proof (unfold acyclic-def, intro allI notI)
  fix x
  assume set-cycle:
    (x, x) ∈ {(u, v). dependency-on C D u v}+

```

```

obtain  $y$  where  $\text{edge}$ :
   $(x, y) \in \{(u, v). \text{dependency-on } C D u v\}$ 
  using  $\text{trancID}[OF \text{ set-cycle}]$  by  $\text{blast}$ 
have  $xC: x \in C$ 
  using  $\text{edge}$  by  $(\text{auto simp: dependency-on-def})$ 
have  $\text{pred-cycle}$ :
   $(\text{dependency-on } C D)^{++} x x$ 
  using  $\text{dependency-set-to-trancI}[OF \text{ set-cycle}]$  .
show  $\text{False}$ 
  using  $\text{acyclic } xC \text{ pred-cycle}$ 
  by  $(\text{auto simp: acyclic-on-def})$ 
qed
have  $\text{wf-rel}: \text{wf } \{(x, y). \text{dependency-on } C D x y\}$ 
  using  $\text{finite-acyclic-wf}[OF \text{ finite-relation acyclic-rel}]$  .
have  $\text{wfp-rel}: \text{wfp } (\text{dependency-on } C D)$ 
  using  $\text{wf-rel}$  by  $(\text{simp add: wfp-def})$ 
show  $\text{well-founded-on } C D$ 
  unfolding  $\text{well-founded-on-def}$ 
  using  $\text{wfp-on-iff-wfp}[of C \text{ dependency-on } C D]$   $\text{relation-eq wfp-rel}$ 
  by  $\text{simp}$ 
qed
qed

```

20 Global Foundation

```

lemma  $\text{global-foundation-implies-local-foundation}$ :
  assumes  $\text{global-foundation-on } C D R$ 
  shows  $\text{local-foundation-on } C D R$ 
proof –
  obtain  $F$  where  $\text{foundation: foundation-set-for } C D R F$ 
    using  $\text{global-foundation-onD}[OF \text{ assms}]$  .
  obtain  $F\text{-sub}: F \subseteq C$ 
  and  $R\text{-sub}: R \subseteq C$ 
  and  $F\text{-independent}$ :
     $\forall f \in F. \text{independent-in } C D f$ 
  and  $F\text{-supports}$ :
     $\forall a \in R. \text{grounded-in } C D a \longrightarrow$ 
     $(\exists f \in F. \text{dependency-ancestor } C D f a)$ 
  using  $\text{foundation-set-forD}[OF \text{ foundation}]$  by  $\text{blast}$ 
show  $?thesis$ 
  unfolding  $\text{local-foundation-on-def}$ 
proof  $(\text{intro ballI impI})$ 
  fix  $a$ 
  assume  $a \in R$  and  $\text{grounded-in } C D a$ 
  have  $F\text{-supports-}a$ :
     $a \in R \longrightarrow \text{grounded-in } C D a \longrightarrow$ 
     $(\exists f \in F. \text{dependency-ancestor } C D f a)$ 
  using  $F\text{-supports}$  by  $\text{blast}$ 
  obtain  $f$  where  $f \in F$  and  $\text{dependency-ancestor } C D f a$ 
    using  $F\text{-supports-}a \langle a \in R \rangle \langle \text{grounded-in } C D a \rangle$  by  $\text{blast}$ 
  have  $\text{independent-in } C D f$ 
    using  $F\text{-independent}$   $\langle f \in F \rangle$  by  $\text{blast}$ 
  show  $\exists f. \text{first-member-in } C D f a$ 
    using  $\langle \text{dependency-ancestor } C D f a \rangle \langle \text{independent-in } C D f \rangle$ 
    by  $(\text{intro exI}) (\text{simp add: first-member-in-def})$ 
qed
qed

```

```

lemma local-foundation-implies-global-foundation:
  assumes local: local-foundation-on C D R
  and R-sub:  $R \subseteq C$ 
  shows global-foundation-on C D R
proof –
  let  $?F = \{f. \text{independent-in } C \ D \ f\}$ 
  have F-sub:  $?F \subseteq C$ 
  proof (intro subsetI)
    fix  $f$ 
    assume  $f \in ?F$ 
    have independent-in C D f
    using  $\langle f \in ?F \rangle$  by simp
    then show  $f \in C$ 
    by (rule independent-inD)
  qed
  have F-independent:
     $\forall f \in ?F. \text{independent-in } C \ D \ f$ 
  proof (intro ballI)
    fix  $f$ 
    assume  $f \in ?F$ 
    then show independent-in C D f
    by simp
  qed
  have supports:
     $\forall a \in R. \text{grounded-in } C \ D \ a \longrightarrow$ 
     $(\exists f \in ?F. \text{dependency-ancestor } C \ D \ f \ a)$ 
  proof (intro ballI impI)
    fix  $a$ 
    assume aR:  $a \in R$ 
    and grounded: grounded-in C D a
    have local-expanded:
       $\forall a \in R. \text{grounded-in } C \ D \ a \longrightarrow$ 
       $(\exists f. \text{first-member-in } C \ D \ f \ a)$ 
    using local by (simp only: local-foundation-on-def)
    have first-exists:
       $\exists f. \text{first-member-in } C \ D \ f \ a$ 
    using local-expanded aR grounded by blast
    obtain  $f$  where first: first-member-in C D f a
    using first-exists by blast
    have f-ancestor: dependency-ancestor C D f a
    using first by (rule first-member-inD)
    have f-independent: independent-in C D f
    using first by (rule first-member-inD)
    show  $\exists f \in ?F. \text{dependency-ancestor } C \ D \ f \ a$ 
    proof (rule bexI[of - f])
      show  $f \in ?F$ 
      using f-independent by simp
      show dependency-ancestor C D f a
      by (rule f-ancestor)
    qed
  qed
  have foundation-unfolded:
     $?F \subseteq C \wedge R \subseteq C \wedge$ 
     $(\forall f \in ?F. \text{independent-in } C \ D \ f) \wedge$ 
     $(\forall a \in R. \text{grounded-in } C \ D \ a \longrightarrow$ 
     $(\exists f \in ?F. \text{dependency-ancestor } C \ D \ f \ a))$ 
  proof –
    show ?thesis
    proof (rule conjI)

```

```

show  $?F \subseteq C$ 
  by (rule F-sub)
show  $R \subseteq C \wedge$ 
  ( $\forall f \in ?F. \text{independent-in } C \ D \ f$ )  $\wedge$ 
  ( $\forall a \in R. \text{grounded-in } C \ D \ a \longrightarrow$ 
    ( $\exists f \in ?F. \text{dependency-ancestor } C \ D \ f \ a$ ))
proof (rule conjI)
  show  $R \subseteq C$ 
    by (rule R-sub)
  show ( $\forall f \in ?F. \text{independent-in } C \ D \ f$ )  $\wedge$ 
    ( $\forall a \in R. \text{grounded-in } C \ D \ a \longrightarrow$ 
      ( $\exists f \in ?F. \text{dependency-ancestor } C \ D \ f \ a$ ))
  proof (rule conjI)
    show  $\forall f \in ?F. \text{independent-in } C \ D \ f$ 
      by (rule F-independent)
    show  $\forall a \in R. \text{grounded-in } C \ D \ a \longrightarrow$ 
      ( $\exists f \in ?F. \text{dependency-ancestor } C \ D \ f \ a$ )
      by (rule supports)
  qed
qed
qed
qed
have foundation:
  foundation-set-for  $C \ D \ R \ ?F$ 
  unfolding foundation-set-for-def
  by (rule foundation-unfolded)
show ?thesis
  using foundation by (rule global-foundation-onI)
qed

theorem local-foundation-iff-global-foundation:
assumes  $R \subseteq C$ 
shows local-foundation-on  $C \ D \ R \longleftrightarrow$ 
  global-foundation-on  $C \ D \ R$ 
using global-foundation-implies-local-foundation
  local-foundation-implies-global-foundation[OF - assms] by blast

```

21 Full-to-Binary Comparison

A plural grounding relation does not determine a binary relation by itself. The bridge theorem therefore exposes the required endpoint, nonemptiness, and closure assumptions as hypotheses.

```

lemma full-foundation-implies-local-foundation:
assumes full: full-foundation-on  $C \ D \ G \ R$ 
and singleton-endpoint:
   $\bigwedge x \ a. x \in C \implies a \in C \implies D \ x \ a \implies$ 
    partial-grounding-on  $C \ G \ x \ a$ 
and nonempty:
   $\bigwedge S \ a. \text{full-grounding-on } C \ G \ S \ a \implies S \neq \{\}$ 
and closure:
   $\bigwedge S \ a \ x. \text{full-grounding-on } C \ G \ S \ a \implies x \in S \implies$ 
    dependency-ancestor  $C \ D \ x \ a$ 
shows local-foundation-on  $C \ D \ R$ 
proof –
obtain F where foundation: full-foundation-for  $C \ D \ G \ R \ F$ 
  using full-foundation-onD[OF full] .
obtain F-sub:  $F \subseteq C$ 
and R-sub:  $R \subseteq C$ 
and F-independent:

```

$\forall f \in F. \text{independent-in } C \ D \ f$
and $F\text{-supports}$:
 $\forall a \in R.$
 $\text{grounded-in } C \ (\text{partial-grounding-on } C \ G) \ a \longrightarrow$
 $(\exists S. \text{full-grounding-on } C \ G \ S \ a \wedge S \subseteq F)$
using $\text{full-foundation-forD}[OF \ \text{foundation}]$ **by** blast
show $?thesis$
unfolding $\text{local-foundation-on-def}$
proof (intro ballI impI)
fix a
assume $aR: a \in R$ **and** $\text{grounded}: \text{grounded-in } C \ D \ a$
obtain x **where** $xC: x \in C$ **and** $xC': a \in C$ **and** $Dxa: D \ x \ a$
using $\text{grounded-inD}[OF \ \text{grounded}]$ **by** blast
have $\text{partial}: \text{partial-grounding-on } C \ G \ x \ a$
using $\text{singleton-endpoint}[OF \ xC \ xC' \ Dxa]$.
have grounded-full :
 $\text{grounded-in } C \ (\text{partial-grounding-on } C \ G) \ a$
using $xC \ xC' \ \text{partial}$
by ($\text{auto simp: grounded-in-def}$)
have $F\text{-supports-}a$:
 $a \in R \longrightarrow$
 $\text{grounded-in } C \ (\text{partial-grounding-on } C \ G) \ a \longrightarrow$
 $(\exists S. \text{full-grounding-on } C \ G \ S \ a \wedge S \subseteq F)$
using $F\text{-supports}$ **by** blast
obtain S **where** $\text{full-}S: \text{full-grounding-on } C \ G \ S \ a$ **and** $S \subseteq F$
using $F\text{-supports-}a \ aR \ \text{grounded-full}$ **by** blast
obtain f **where** $f \in S$
using $\text{nonempty}[OF \ \text{full-}S]$ **by** blast
have $f \in F$
using $\langle S \subseteq F \rangle \langle f \in S \rangle$ **by** blast
have $\text{independent-in } C \ D \ f$
using $F\text{-independent } \langle f \in F \rangle$ **by** blast
have $\text{dependency-ancestor } C \ D \ f \ a$
using $\text{closure}[OF \ \text{full-}S \ \langle f \in S \rangle]$.
show $\exists f. \text{first-member-in } C \ D \ f \ a$
using $\langle \text{dependency-ancestor } C \ D \ f \ a \rangle \langle \text{independent-in } C \ D \ f \rangle$
by (intro exI) ($\text{simp add: first-member-in-def}$)
qed
qed

theorem $\text{foundation-implication-atlas}$:

assumes $\text{singleton-endpoint}$:

$\bigwedge x \ a. x \in C \implies a \in C \implies D \ x \ a \implies$

$\text{partial-grounding-on } C \ G \ x \ a$

and nonempty :

$\bigwedge S \ a. \text{full-grounding-on } C \ G \ S \ a \implies S \neq \{\}$

and closure :

$\bigwedge S \ a \ x. \text{full-grounding-on } C \ G \ S \ a \implies x \in S \implies$

$\text{dependency-ancestor } C \ D \ x \ a$

shows

$(\text{well-founded-on } C \ D \longrightarrow \text{local-foundation-on } C \ D \ R) \wedge$

$(\text{well-founded-on } C \ D \longrightarrow \text{acyclic-on } C \ D) \wedge$

$(\text{well-founded-on } C \ D \longrightarrow \text{chain-terminating-on } C \ D) \wedge$

$(\text{global-foundation-on } C \ D \ R \longrightarrow \text{local-foundation-on } C \ D \ R) \wedge$

$(\text{full-foundation-on } C \ D \ G \ R \longrightarrow \text{local-foundation-on } C \ D \ R)$

proof –

have wf-local :

$\text{well-founded-on } C \ D \longrightarrow \text{local-foundation-on } C \ D \ R$

by ($\text{blast intro: well-founded-implies-local-foundation}$)

```

have wf-chain:
  well-founded-on C D  $\longrightarrow$  chain-terminating-on C D
  by (blast intro: well-founded-implies-chain-terminating)
have wf-acyclic:
  well-founded-on C D  $\longrightarrow$  acyclic-on C D
  by (blast intro: well-founded-implies-acyclic)
have global-local:
  global-foundation-on C D R  $\longrightarrow$  local-foundation-on C D R
  by (blast intro: global-foundation-implies-local-foundation)
have full-local:
  full-foundation-on C D G R  $\longrightarrow$  local-foundation-on C D R
proof (intro impI)
  assume full: full-foundation-on C D G R
  then show local-foundation-on C D R
    using full-foundation-implies-local-foundation[
      where C = C and D = D and G = G and R = R]
      singleton-endpoint nonempty closure by blast
qed
show ?thesis
  using wf-local wf-acyclic wf-chain global-local full-local by blast
qed

end
theory Foundation-Separations
  imports Foundation-Implications
begin

```

22 Integer Predecessor Model

The integer predecessor relation is the canonical infinite separation model: every integer has an immediate predecessor, but no integer is independent. It is reused by the acyclicity, well-foundedness, essential-ordering, and premise-deletion results below.

definition *integer-predecessor-cause* :: *int* $\Rightarrow$ *int* $\Rightarrow$ *bool* **where**
integer-predecessor-cause *x y* $\longleftrightarrow$ *y* = *x* + 1

lemma *integer-predecessor-has-incoming*:
 $\exists y. \text{integer-predecessor-cause } y \ x$
unfolding *integer-predecessor-cause-def*
by (rule exI[of - *x* - 1]) *simp*

lemma *integer-predecessor-not-well-founded*:
 $\neg \text{wfP integer-predecessor-cause}$
proof
assume wfP *integer-predecessor-cause*
from wf-has-first-member[OF this, of 0] **obtain** *m*
where first-member-of *integer-predecessor-cause* *m* 0
by blast
then show False
using *integer-predecessor-has-incoming*[of *m*]
by (auto *simp*: first-member-of-def ungrounded-def)
qed

23 Acyclic but not Well-Founded

lemma *integer-predecessor-dependency-trancl-less*:
(*dependency-on UNIV integer-predecessor-cause*)⁺⁺ *x y* $\implies$ *x* < *y*
proof (*induction rule: tranclp-induct*)

```

case base
then show ?case
  by (auto simp: dependency-on-def integer-predecessor-cause-def)
next
  case (step y z)
  have  $x < y$ 
    using step.IH .
  moreover have  $y < z$ 
    using step.hyps(2)
    by (auto simp: dependency-on-def integer-predecessor-cause-def)
  ultimately show ?case
    by (rule less-trans)
qed

```

```

lemma integer-predecessor-acyclic-on:
  acyclic-on UNIV integer-predecessor-cause
  unfolding acyclic-on-def
proof (intro ballI notI)
  fix  $x$ 
  assume cycle:
    (dependency-on UNIV integer-predecessor-cause)++  $x x$ 
  have  $x < x$ 
    using integer-predecessor-dependency-transl-less[OF cycle] .
  then show False
    by simp
qed

```

```

lemma integer-predecessor-not-well-founded-on:
   $\neg$  well-founded-on UNIV integer-predecessor-cause
proof
  have relation-eq:
    dependency-on UNIV integer-predecessor-cause =
      integer-predecessor-cause
    by (rule ext, rule ext) (simp add: dependency-on-def)
  assume well-founded-on UNIV integer-predecessor-cause
  have wf-rel:
    wfp-on UNIV (dependency-on UNIV integer-predecessor-cause)
    using  $\langle$ well-founded-on UNIV integer-predecessor-cause $\rangle$ 
    by (simp only: well-founded-on-def)
  have wfp-rel:
    wfp (dependency-on UNIV integer-predecessor-cause)
    using wfp-on-iff-wfp[of UNIV
      dependency-on UNIV integer-predecessor-cause]
    wf-rel by simp
  have wfP integer-predecessor-cause
    using wfp-rel relation-eq by (simp only: relation-eq)
  then show False
    using integer-predecessor-not-well-founded by blast
qed

```

```

theorem acyclic-not-well-founded-integer-model:
  acyclic-on UNIV integer-predecessor-cause  $\wedge$ 
   $\neg$  well-founded-on UNIV integer-predecessor-cause
  using integer-predecessor-acyclic-on
    integer-predecessor-not-well-founded-on by blast

```

24 Foundation without Acyclicity

A local foundation can coexist with an unrelated cycle. The root 0 supports the cyclic component below, but the cycle is deliberately retained in the immediate relation. This separates foundation from acyclicity on arbitrary (in particular, non-well-founded) structures.

definition *rooted-cycle-dependency* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$

where

```
rooted-cycle-dependency x y  $\longleftrightarrow$ 
  (x = 0  $\wedge$  y = 1)  $\vee$ 
  (x = 1  $\wedge$  y = 2)  $\vee$ 
  (x = 2  $\wedge$  y = 1)
```

lemma *rooted-cycle-zero-independent*:

independent-in {0, 1, 2} *rooted-cycle-dependency* 0

by (auto simp: *independent-in-def* *rooted-cycle-dependency-def*)

lemma *rooted-cycle-zero-ancestor*:

assumes $a \in \{0, 1, 2\}$

shows *dependency-ancestor* {0, 1, 2} *rooted-cycle-dependency* 0 a

proof –

have *cases*: $a = 0 \vee a = 1 \vee a = 2$

using *assms* **by** auto

then show ?thesis

proof (elim *disjE*)

assume $a = 0$

then show ?thesis

by (simp add: *dependency-ancestor-refl*)

next

assume $a = 1$

then show ?thesis

by (auto intro: *dependency-ancestor-direct*
simp: *rooted-cycle-dependency-def*)

next

assume $a = 2$

have *first*:

dependency-ancestor {0, 1, 2} *rooted-cycle-dependency* 0 1

by (auto intro: *dependency-ancestor-direct*
simp: *rooted-cycle-dependency-def*)

have *second*:

dependency-ancestor {0, 1, 2} *rooted-cycle-dependency* 1 2

by (auto intro: *dependency-ancestor-direct*
simp: *rooted-cycle-dependency-def*)

have *combined*:

dependency-ancestor {0, 1, 2} *rooted-cycle-dependency* 0 2

using *first second* **by** (rule *dependency-ancestor-trans*)

show ?thesis

using *combined a2* **by** simp

qed

qed

lemma *rooted-cycle-local-foundation*:

local-foundation-on {0, 1, 2} *rooted-cycle-dependency* {0, 1, 2}

unfolding *local-foundation-on-def*

proof (intro *ballI impI*)

fix $a :: \text{nat}$

assume aC : $a \in \{0, 1, 2\}$

assume *grounded-in* {0, 1, 2} *rooted-cycle-dependency* a

have *ancestor*:

```

    dependency-ancestor  $\{0, 1, 2\}$  rooted-cycle-dependency 0 a
  using rooted-cycle-zero-ancestor[OF aC] .
have first:
  first-member-in  $\{0, 1, 2\}$  rooted-cycle-dependency 0 a
  using ancestor rooted-cycle-zero-independent
  by (simp add: first-member-in-def)
show  $\exists f.$  first-member-in  $\{0, 1, 2\}$ 
  rooted-cycle-dependency f a
  using first by blast
qed

```

```

lemma rooted-cycle-not-acyclic:
   $\neg$  acyclic-on  $\{0, 1, 2\}$  rooted-cycle-dependency
proof
  assume acyclic: acyclic-on  $\{0, 1, 2\}$  rooted-cycle-dependency
  have edge12:
    dependency-on  $\{0, 1, 2\}$  rooted-cycle-dependency 1 2
  by (auto simp: dependency-on-def rooted-cycle-dependency-def)
  have edge21:
    dependency-on  $\{0, 1, 2\}$  rooted-cycle-dependency 2 1
  by (auto simp: dependency-on-def rooted-cycle-dependency-def)
  have back-path:
    (dependency-on  $\{0, 1, 2\}$  rooted-cycle-dependency)** 2 1
  by (rule r-into-rtrancpl) (rule edge21)
  have cycle:
    (dependency-on  $\{0, 1, 2\}$  rooted-cycle-dependency)++ 1 1
  using rtrancpl-into-trancpl2[OF edge12 back-path] .
show False
  using acyclic cycle by (auto simp: acyclic-on-def)
qed

```

```

theorem local-foundation-not-acyclic-rooted-cycle:
  local-foundation-on  $\{0, 1, 2\}$  rooted-cycle-dependency  $\{0, 1, 2\} \wedge$ 
  global-foundation-on  $\{0, 1, 2\}$  rooted-cycle-dependency  $\{0, 1, 2\} \wedge$ 
   $\neg$  acyclic-on  $\{0, 1, 2\}$  rooted-cycle-dependency
proof -
  have local:
    local-foundation-on  $\{0, 1, 2\}$  rooted-cycle-dependency  $\{0, 1, 2\}$ 
  using rooted-cycle-local-foundation .
  have global:
    global-foundation-on  $\{0, 1, 2\}$  rooted-cycle-dependency  $\{0, 1, 2\}$ 
  using local
    local-foundation-implies-global-foundation[of
       $\{0, 1, 2\}$  rooted-cycle-dependency  $\{0, 1, 2\}$ ]
  by (auto simp: subset-iff)
show ?thesis
  using local global rooted-cycle-not-acyclic by blast
qed

```

25 A Fully Pedestalled Infinite Structure

The pedestal relation has a distinguished independent member 0. It supports every nonzero member, while the negative tail still contains an infinite predecessor chain. Thus a local or global foundation witness need not make the whole immediate relation well-founded.

```

definition pedestal-dependency :: int  $\Rightarrow$  int  $\Rightarrow$  bool
where
  pedestal-dependency x y  $\longleftrightarrow$ 

```

$$(y = x + 1 \wedge y < 0) \vee (x = 0 \wedge y \neq 0)$$

definition *pedestal-chain* :: nat $\Rightarrow$ int

where

pedestal-chain n = - of-nat n - 1

lemma *pedestal-chain-negative*:

pedestal-chain n < 0

by (simp add: pedestal-chain-def)

lemma *pedestal-chain-successor*:

pedestal-chain n = *pedestal-chain* (Suc n) + 1

by (simp add: pedestal-chain-def)

lemma *pedestal-chain-edge*:

pedestal-dependency (*pedestal-chain* (Suc n)) (*pedestal-chain* n)

using *pedestal-chain-negative*[of n] *pedestal-chain-successor*[of n]

by (auto simp: pedestal-dependency-def)

lemma *pedestal-zero-independent*:

independent-in UNIV *pedestal-dependency* 0

by (auto simp: independent-in-def pedestal-dependency-def)

lemma *pedestal-zero-ancestor*:

dependency-ancestor UNIV *pedestal-dependency* 0 x

proof (cases x = 0)

case True

then show ?thesis

by (simp add: dependency-ancestor-def)

next

case False

have *pedestal-dependency* 0 x

using False **by** (auto simp: pedestal-dependency-def)

then show ?thesis

by (auto intro: dependency-ancestor-direct)

qed

lemma *pedestal-local-foundation*:

local-foundation-on UNIV *pedestal-dependency* UNIV

unfolding *local-foundation-on-def*

proof (intro ballI impI)

fix a

have *first-member-in* UNIV *pedestal-dependency* 0 a

using *pedestal-zero-ancestor*[of a] *pedestal-zero-independent*

by (simp add: first-member-in-def)

then show $\exists f. \text{first-member-in UNIV pedestal-dependency } f \text{ a}$

by blast

qed

lemma *pedestal-global-foundation*:

global-foundation-on UNIV *pedestal-dependency* UNIV

unfolding *global-foundation-on-def*

by (intro exI[*of* - {0}])

(auto simp: foundation-set-for-def pedestal-zero-independent
pedestal-zero-ancestor)

lemma *pedestal-chain-exists*:

$\exists f. \forall n.$

dependency-on UNIV *pedestal-dependency* (f (Suc n)) (f n)

by (*intro* *exI*[*of* - *pedestal-chain*] *allI*)
 (*simp add*: *dependency-on-def pedestal-chain-edge*)

lemma *pedestal-not-chain-terminating*:
 $\neg$ *chain-terminating-on UNIV pedestal-dependency*
unfolding *chain-terminating-on-def*
using *pedestal-chain-exists* **by** *blast*

lemma *pedestal-not-well-founded*:
 $\neg$ *well-founded-on UNIV pedestal-dependency*
using *well-founded-iff-chain-terminating*[*of UNIV pedestal-dependency*]
pedestal-not-chain-terminating **by** *blast*

theorem *foundation-without-well-foundedness-pedestal*:
local-foundation-on UNIV pedestal-dependency UNIV $\wedge$
global-foundation-on UNIV pedestal-dependency UNIV $\wedge$
 $\neg$ *well-founded-on UNIV pedestal-dependency*
using *pedestal-local-foundation pedestal-global-foundation*
pedestal-not-well-founded **by** *blast*

26 Partial and Full Grounding Diverge

definition *paired-dependency* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where

paired-dependency *x y* $\longleftrightarrow$
 (*x* = 0 $\wedge$ *y* = 1) $\vee$
 (*x* = 0 $\wedge$ *y* = 2) $\vee$
 (*x* = 1 $\wedge$ *y* = 2)

definition *paired-full-grounding* :: *nat set* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where
paired-full-grounding *S x* $\longleftrightarrow$ *S* = {0, 1} $\wedge$ *x* = 2

lemma *paired-full-grounding-example*:
full-grounding-on {0, 1, 2} *paired-full-grounding* {0, 1} 2
by (*auto simp*: *full-grounding-on-def paired-full-grounding-def*)

lemma *paired-partial-grounding-example*:
partial-grounding-on {0, 1, 2} *paired-full-grounding* 0 2
using *paired-full-grounding-example*
by (*rule partial-grounding-onI*) *simp*

lemma *paired-zero-independent*:
independent-in {0, 1, 2} *paired-dependency* 0
by (*auto simp*: *independent-in-def paired-dependency-def*)

lemma *paired-zero-ancestor*:
dependency-ancestor {0, 1, 2} *paired-dependency* 0 2
by (*rule dependency-ancestor-direct*)
 (*auto simp*: *paired-dependency-def*)

lemma *paired-partial-local-foundation*:
local-foundation-on {0, 1, 2} *paired-dependency* {2}
unfolding *local-foundation-on-def*

proof (*intro ballI impI*)
fix *a* :: *nat*
assume *a* $\in$ {2}
then have *a* = 2

```

  by simp
have first-two:
  first-member-in {0, 1, 2} paired-dependency 0 2
  using paired-zero-ancestor paired-zero-independent
  by (simp add: first-member-in-def)
show  $\exists f$ . first-member-in {0, 1, 2} paired-dependency f a
  using first-two  $\langle a = 2 \rangle$ 
  by (intro exI[of - 0]) (simp add: first-member-in-def)
qed

lemma paired-full-foundation-fails:
 $\neg$  full-foundation-on {0, 1, 2} paired-dependency
  paired-full-grounding {2}
proof
  assume full:
    full-foundation-on {0, 1, 2} paired-dependency
    paired-full-grounding {2}
  obtain F where foundation:
    full-foundation-for {0, 1, 2} paired-dependency
    paired-full-grounding {2} F
  using full-foundation-onD[OF full] .
  obtain F-sub:  $F \subseteq \{0, 1, 2\}$ 
  and R-sub:  $\{2\} \subseteq \{0, 1, 2\}$ 
  and F-independent:
     $\forall f \in F$ . independent-in {0, 1, 2} paired-dependency f
  and F-supports:
     $\forall a \in \{2\}$ .
    grounded-in {0, 1, 2}
    (partial-grounding-on {0, 1, 2} paired-full-grounding) a
     $\longrightarrow$ 
    ( $\exists S$ . full-grounding-on {0, 1, 2}
    paired-full-grounding S a  $\wedge S \subseteq F$ )
  using full-foundation-forD[OF foundation] by blast
  have grounded:
    grounded-in {0, 1, 2}
    (partial-grounding-on {0, 1, 2} paired-full-grounding) 2
  using paired-partial-grounding-example
  by (auto simp: grounded-in-def)
  have support:
    grounded-in {0, 1, 2}
    (partial-grounding-on {0, 1, 2} paired-full-grounding) 2
     $\longrightarrow$ 
    ( $\exists S$ . full-grounding-on {0, 1, 2}
    paired-full-grounding S 2  $\wedge S \subseteq F$ )
  using F-supports by blast
  obtain S where full-S:
    full-grounding-on {0, 1, 2} paired-full-grounding S 2
  and S-sub:  $S \subseteq F$ 
  using support grounded by blast
  have S-eq:  $S = \{0, 1\}$ 
  using full-S
  by (auto simp: full-grounding-on-def paired-full-grounding-def)
  have 1  $\in F$ 
  using S-eq S-sub by auto
  have independent-in {0, 1, 2} paired-dependency 1
  using F-independent  $\langle 1 \in F \rangle$  by blast
  then show False
  by (auto simp: independent-in-def paired-dependency-def)
qed

```

```

theorem partial-full-grounding-diverge:
  local-foundation-on {0, 1, 2} paired-dependency {2}  $\wedge$ 
  partial-grounding-on {0, 1, 2} paired-full-grounding 0 2  $\wedge$ 
   $\neg$  full-foundation-on {0, 1, 2} paired-dependency
    paired-full-grounding {2}
using paired-partial-local-foundation paired-partial-grounding-example
  paired-full-foundation-fails by blast

end
theory Finite-Foundation-Checker
  imports Foundation-Separations HOL-Library.Transitive-Closure-Table
begin

```

27 Executable Finite Models

A finite model is represented by a vertex list and an edge list. The declarative abstraction uses the set of listed vertices and carrier-restricts the edge relation. Duplicated vertices do not change the abstraction, while well-formed clients may additionally require *distinct* (*fst* *M*).

```

type-synonym finite-dependency-model = nat list  $\times$  (nat  $\times$  nat) list

```

```

definition finite-vertices :: finite-dependency-model  $\Rightarrow$  nat list
where
  finite-vertices M = fst M

```

```

definition finite-edges :: finite-dependency-model  $\Rightarrow$  (nat  $\times$  nat) list
where
  finite-edges M = snd M

```

```

definition finite-model-immediate ::
  finite-dependency-model  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  bool
where
  finite-model-immediate M x y  $\longleftrightarrow$ 
    (x, y)  $\in$  set (finite-edges M)

```

```

definition finite-model-relation ::
  finite-dependency-model  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  bool
where
  finite-model-relation M x y  $\longleftrightarrow$ 
    dependency-on (set (finite-vertices M))
      (finite-model-immediate M) x y

```

```

definition finite-model-carrier ::
  finite-dependency-model  $\Rightarrow$  nat set
where
  finite-model-carrier M = set (finite-vertices M)

```

```

fun finite-list-all :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a list  $\Rightarrow$  bool
where
  finite-list-all P [] = True
| finite-list-all P (x # xs) = (P x  $\wedge$  finite-list-all P xs)

```

```

fun finite-list-ex :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a list  $\Rightarrow$  bool
where
  finite-list-ex P [] = False
| finite-list-ex P (x # xs) = (P x  $\vee$  finite-list-ex P xs)

```

```

fun finite-first-failure ::
  ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a list  $\Rightarrow$  'a option
where
  finite-first-failure P [] = None
| finite-first-failure P (x # xs) =
  (if P x then Some x else finite-first-failure P xs)

lemma finite-list-all-iff:
  finite-list-all P xs  $\longleftrightarrow$  ( $\forall x \in \text{set } xs. P x$ )
by (induction xs) simp-all

lemma finite-list-ex-iff:
  finite-list-ex P xs  $\longleftrightarrow$  ( $\exists x \in \text{set } xs. P x$ )
by (induction xs) simp-all

lemma finite-list-exI:
  assumes  $x \in \text{set } xs$  and P x
  shows finite-list-ex P xs
  using assms
  by (induction xs) auto

fun finite-reach ::
  nat  $\Rightarrow$  'a list  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$ 
  'a  $\Rightarrow$  'a  $\Rightarrow$  bool
where
  finite-reach 0 V R x y = (x = y)
| finite-reach (Suc n) V R x y =
  (x = y  $\vee$ 
   finite-list-ex
    ( $\lambda z. R x z \wedge \text{finite-reach } n V R z y$ ) V)

lemma finite-reach-imp-rtranclp:
  finite-reach n V R x y  $\Longrightarrow$   $R^{**} x y$ 
proof (induction n arbitrary: x y)
  case 0
  then show ?case
    by simp
next
  case (Suc n)
  then show ?case
    by (auto simp: finite-list-ex-iff
      intro: r-into-rtranclp converse-rtranclp-into-rtranclp)
qed

lemma finite-reach-mono:
  assumes  $n \leq m$  and finite-reach n V R x y
  shows finite-reach m V R x y
using assms
proof (induction m arbitrary: n x y)
  case 0
  then show ?case
    by simp
next
  case (Suc m)
  show ?case
  proof (cases n)
    case 0
    then show ?thesis
      using Suc.prem1(2) by simp

```

```

next
  case (Suc n)
  have  $n \leq m$ 
    using Suc.premis(1) Suc by simp
  have  $x = y \vee$ 
    finite-list-ex
     $(\lambda z. R\ x\ z \wedge \text{finite-reach}\ n\ V\ R\ z\ y)\ V$ 
    using Suc.premis(2) Suc by simp
  then show ?thesis
proof
  assume  $x = y$ 
  then show ?thesis by simp
next
  assume ex:
    finite-list-ex
     $(\lambda z. R\ x\ z \wedge \text{finite-reach}\ n\ V\ R\ z\ y)\ V$ 
  obtain z where zV:  $z \in \text{set } V$ 
  and R-xz:  $R\ x\ z$ 
  and reach:  $\text{finite-reach}\ n\ V\ R\ z\ y$ 
  using ex by (auto simp: finite-list-ex-iff)
  have reach-m:  $\text{finite-reach}\ m\ V\ R\ z\ y$ 
  using Suc.IH[OF  $\langle n \leq m \rangle$  reach] .
  have ex-m:
    finite-list-ex
     $(\lambda w. R\ x\ w \wedge \text{finite-reach}\ m\ V\ R\ w\ y)\ V$ 
  using finite-list-exI[of z V
     $\lambda w. R\ x\ w \wedge \text{finite-reach}\ m\ V\ R\ w\ y$ 
     $zV\ R\ xz\ \text{reach-m}$ ]
  by blast
  show ?thesis
  using ex-m by (simp add: finite-reach.simps)
qed
qed
qed

lemma rtrancl-path-vertices:
  assumes path: rtrancl-path R x xs y
  and xV:  $x \in \text{set } V$ 
  and closed:  $\bigwedge u\ v. R\ u\ v \implies u \in \text{set } V \wedge v \in \text{set } V$ 
  shows  $\text{set } xs \subseteq \text{set } V \wedge y \in \text{set } V$ 
  using path xV
proof (induction rule: rtrancl-path.induct)
  case base
  then show ?case by simp
next
  case (step x z ys y)
  have zV:  $z \in \text{set } V$ 
  using closed step.hyps(1) by blast
  have tail:  $\text{set } ys \subseteq \text{set } V \wedge y \in \text{set } V$ 
  using step.IH zV by blast
  then show ?case
  using zV by simp
qed

lemma finite-reach-path:
  assumes path: rtrancl-path R x xs y
  and xV:  $x \in \text{set } V$ 
  and closed:  $\bigwedge u\ v. R\ u\ v \implies u \in \text{set } V \wedge v \in \text{set } V$ 
  shows  $\text{finite-reach}\ (\text{length } xs)\ V\ R\ x\ y$ 

```

```

    using path xV
  proof (induction rule: rtrancl-path.induct)
    case base
    then show ?case by simp
  next
    case (step x z ys y)
    have zV: z ∈ set V
      using closed step.hyps(1) by blast
    have tail: finite-reach (length ys) V R z y
      using step.IH zV by blast
    have ex:
      finite-list-ex
      (λw. R x w ∧ finite-reach (length ys) V R w y) V
    using finite-list-exI[of z V
      λw. R x w ∧ finite-reach (length ys) V R w y]
      zV step.hyps(1) tail
      by blast
    show ?case
      using ex by (simp add: finite-reach.simps)
  qed

```

```

lemma rtrancl-imp-finite-reach:
  assumes reach:  $R^{**} x y$ 
  and xV:  $x \in \text{set } V$ 
  and closed:  $\bigwedge u v. R u v \implies u \in \text{set } V \wedge v \in \text{set } V$ 
  shows finite-reach (length V) V R x y
proof -
  have path-exists:  $\exists xs. \text{rtrancl-path } R x xs y$ 
    using reach by (simp add: rtrancl-eq-rtrancl-path)
  obtain xs where path: rtrancl-path R x xs y
    using path-exists by blast
  obtain xs' where path': rtrancl-path R x xs' y
    and distinct: distinct (x # xs')
    and subset:  $\text{set } xs' \subseteq \text{set } xs$ 
    using rtrancl-path-distinct[OF path] by blast
  have path-V:  $\text{set } xs \subseteq \text{set } V \wedge y \in \text{set } V$ 
    using rtrancl-path-vertices[OF path xV closed] .
  have xs-V:  $\text{set } xs \subseteq \text{set } V$ 
    using path-V by blast
  have xs'-V:  $\text{set } xs' \subseteq \text{set } V$ 
    using subset xs-V by blast
  have len-xs':  $\text{length } xs' \leq \text{length } V$ 
proof -
  have card-xs':  $\text{card } (\text{set } xs') = \text{length } xs'$ 
    using distinct by (simp add: distinct-card)
  have card (set xs') ≤ card (set V)
    using card-mono xs'-V by blast
  moreover have card (set V) ≤ length V
    by (rule card-length)
  ultimately show ?thesis
    using card-xs' by linarith
  qed
  have finite-path: finite-reach (length xs') V R x y
    using finite-reach-path[OF path' xV closed] .
  have monotone:
    finite-reach (length V) V R x y
    using finite-reach-mono[
      where n = length xs' and m = length V
      and V = V and R = R and x = x and y = y]

```

```

    len-xs' finite-path .
  show finite-reach (length V) V R x y
    using monotone by blast
qed

lemma finite-first-failure-sound:
  assumes finite-first-failure P xs = Some x
  shows x ∈ set xs P x
  using assms
  by (induction xs) (auto split: if-splits)

lemma finite-first-failure-complete:
  assumes ∃ x ∈ set xs. P x
  obtains x where finite-first-failure P xs = Some x
    and x ∈ set xs and P x
  proof -
    have exists:
      ∃ x. finite-first-failure P xs = Some x ∧
        x ∈ set xs ∧ P x
    using assms
    by (induction xs) (auto split: if-splits)
  then obtain x where finite-first-failure P xs = Some x
    and x ∈ set xs and P x
  by blast
  then show thesis
    by (rule that)
qed

```

28 Finite Foundation Predicate and Checker

definition *finite-model-independent* ::
finite-dependency-model $\Rightarrow$ nat $\Rightarrow$ bool
where
finite-model-independent M x $\longleftrightarrow$
 x ∈ *finite-model-carrier* M ∧
 $\neg$ *finite-list-ex*
 (λy. *dependency-on* (*finite-model-carrier* M)
 (*finite-model-immediate* M) y x) (*finite-vertices* M)

definition *finite-model-grounded* ::
finite-dependency-model $\Rightarrow$ nat $\Rightarrow$ bool
where
finite-model-grounded M x $\longleftrightarrow$
finite-list-ex
 (λy. *dependency-on* (*finite-model-carrier* M)
 (*finite-model-immediate* M) y x) (*finite-vertices* M)

definition *finite-model-supports* ::
finite-dependency-model $\Rightarrow$ nat $\Rightarrow$ bool
where
finite-model-supports M x $\longleftrightarrow$
finite-list-ex
 (λf. *finite-model-independent* M f ∧
finite-reach (length (*finite-vertices* M))
 (*finite-vertices* M)
 (*dependency-on* (*finite-model-carrier* M)
 (*finite-model-immediate* M)) f x)
 (*finite-vertices* M)

definition *finite-model-bad* ::

finite-dependency-model $\Rightarrow$ nat $\Rightarrow$ bool

where

finite-model-bad *M* *x* $\longleftrightarrow$

finite-model-grounded *M* *x* $\wedge \neg$ *finite-model-supports* *M* *x*

definition *finite-foundation-check* ::

finite-dependency-model $\Rightarrow$ bool

where

finite-foundation-check *M* $\longleftrightarrow$

finite-list-all ($\lambda x. \neg$ *finite-model-bad* *M* *x*)
(*finite-vertices* *M*)

definition *finite-failure-witness* ::

finite-dependency-model $\Rightarrow$ nat

where

finite-failure-witness *M* =

(*case* *finite-first-failure* (*finite-model-bad* *M*) (*finite-vertices* *M*) of
None $\Rightarrow$ 0
| *Some* *x* $\Rightarrow$ *x*)

The total executable helper has a harmless default for the impossible empty-failure case. The specification theorem below proves that whenever the checker takes its failure branch, the returned natural number is a genuine carrier vertex and a genuine bad vertex.

datatype *finite-foundation-result* =

Foundation-Certificate nat list

| *Missing-Foundation-Certificate* nat

definition *finite-foundation-result* ::

finite-dependency-model $\Rightarrow$ *finite-foundation-result*

where

finite-foundation-result *M* =

(*if* *finite-foundation-check* *M* then
 Foundation-Certificate
 (*filter* (*finite-model-independent* *M*) (*finite-vertices* *M*))
else
 Missing-Foundation-Certificate (*finite-failure-witness* *M*))

lemma *finite-model-relation-eq*:

finite-model-relation *M* =

dependency-on (*finite-model-carrier* *M*) (*finite-model-immediate* *M*)

by (*rule ext*, *rule ext*)

(*simp add*: *finite-model-relation-def* *finite-model-carrier-def*)

lemma *finite-model-immediate-eq*:

finite-model-immediate *M* =

($\lambda x y. (x, y) \in \text{set } (\text{finite-edges } M)$)

by (*rule ext*, *rule ext*)

(*simp add*: *finite-model-immediate-def*)

lemma *finite-model-grounded-iff*:

assumes *x* $\in$ *finite-model-carrier* *M*

shows *finite-model-grounded* *M* *x* $\longleftrightarrow$

grounded-in (*finite-model-carrier* *M*) (*finite-model-immediate* *M*) *x*

using *assms*

unfolding *finite-model-grounded-def*

by (*auto simp*: *finite-list-ex-iff*

finite-model-relation-def)

finite-model-carrier-def *finite-vertices-def*
finite-model-immediate-eq *finite-model-immediate-def*
finite-edges-def *grounded-in-def*
intro: *dependency-onI* *dest*: *dependency-onD*)

lemma *finite-model-independent-iff*:
assumes $x \in \text{finite-model-carrier } M$
shows $\text{finite-model-independent } M \ x \longleftrightarrow$
 $\text{independent-in } (\text{finite-model-carrier } M) (\text{finite-model-immediate } M) \ x$
using *assms*
unfolding *finite-model-independent-def*
by (*auto simp: finite-list-ex-iff*
finite-model-relation-def
finite-model-carrier-def *finite-vertices-def*
finite-model-immediate-eq *finite-model-immediate-def*
finite-edges-def *independent-in-def*
intro: *dependency-onI* *dest*: *dependency-onD*)

lemma *finite-model-supports-iff*:
assumes $x \in \text{finite-model-carrier } M$
shows $\text{finite-model-supports } M \ x \longleftrightarrow$
 $(\exists f. \text{first-member-in } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ f \ x)$
proof –
have *closed*:
 $\bigwedge u \ v. \text{dependency-on } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ u \ v \implies$
 $u \in \text{finite-model-carrier } M \wedge$
 $v \in \text{finite-model-carrier } M$
by (*auto simp: dependency-on-def*)
have *closedV*:
 $\bigwedge u \ v. \text{dependency-on } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ u \ v \implies$
 $u \in \text{set } (\text{finite-vertices } M) \wedge$
 $v \in \text{set } (\text{finite-vertices } M)$
using *closed* **by** (*auto simp: finite-model-carrier-def*)
have *reach-iff*:
 $\bigwedge f. f \in \text{finite-model-carrier } M \implies$
 $\text{finite-reach } (\text{length } (\text{finite-vertices } M))$
 $(\text{finite-vertices } M)$
 $(\text{dependency-on } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M)) \ f \ x$
 $\longleftrightarrow$
 $\text{dependency-ancestor } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ f \ x$

proof –
fix *f*
assume *fC*: $f \in \text{finite-model-carrier } M$
show $\text{finite-reach } (\text{length } (\text{finite-vertices } M))$
 $(\text{finite-vertices } M)$
 $(\text{dependency-on } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M)) \ f \ x$
 $\longleftrightarrow$
 $\text{dependency-ancestor } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ f \ x$
proof (*rule iffI*)
assume *bounded*:
 $\text{finite-reach } (\text{length } (\text{finite-vertices } M))$
 $(\text{finite-vertices } M)$

```

      (dependency-on (finite-model-carrier M)
        (finite-model-immediate M)) f x
    have rtranc1:
      (dependency-on (finite-model-carrier M)
        (finite-model-immediate M))** f x
    using finite-reach-imp-rtranc1[OF bounded] .
  show dependency-ancestor (finite-model-carrier M)
    (finite-model-immediate M) f x
    using fC assms rtranc1
  by (simp add: dependency-ancestor-def)
next
  assume ancestor:
    dependency-ancestor (finite-model-carrier M)
      (finite-model-immediate M) f x
  obtain rtranc1:
    (dependency-on (finite-model-carrier M)
      (finite-model-immediate M))** f x
  using ancestor by (auto simp: dependency-ancestor-def)
  show finite-reach (length (finite-vertices M))
    (finite-vertices M)
    (dependency-on (finite-model-carrier M)
      (finite-model-immediate M)) f x
  proof -
    have fV: f ∈ set (finite-vertices M)
      using fC by (simp add: finite-model-carrier-def)
    show finite-reach (length (finite-vertices M))
      (finite-vertices M)
      (dependency-on (finite-model-carrier M)
        (finite-model-immediate M)) f x
    using rtranc1-imp-finite-reach[
      where R = dependency-on (finite-model-carrier M)
        (finite-model-immediate M)
      and x = f and y = x and V = finite-vertices M]
      rtranc1 fV closedV .
  qed
qed
qed
show ?thesis
  unfolding finite-model-supports-def
proof (rule iffI)
  assume support:
    finite-list-ex
    (λf. finite-model-independent M f ∧
      finite-reach (length (finite-vertices M))
        (finite-vertices M)
        (dependency-on (finite-model-carrier M)
          (finite-model-immediate M)) f x)
    (finite-vertices M)
  obtain f where fV: f ∈ set (finite-vertices M)
    and f-ind: finite-model-independent M f
    and bounded:
      finite-reach (length (finite-vertices M))
        (finite-vertices M)
        (dependency-on (finite-model-carrier M)
          (finite-model-immediate M)) f x
  using support by (auto simp: finite-list-ex-iff)
  have fC: f ∈ finite-model-carrier M
    using fV by (simp add: finite-model-carrier-def)
  have independent:

```

```

    independent-in (finite-model-carrier M)
    (finite-model-immediate M) f
  using finite-model-independent-iff[OF fC] f-ind by blast
have ancestor:
  dependency-ancestor (finite-model-carrier M)
  (finite-model-immediate M) f x
  using reach-iff[OF fC] bounded by blast
show  $\exists f$ . first-member-in (finite-model-carrier M)
  (finite-model-immediate M) f x
  using ancestor independent
  by (intro exI) (simp add: first-member-in-def)
next
assume first:
   $\exists f$ . first-member-in (finite-model-carrier M)
  (finite-model-immediate M) f x
then obtain f where first-f:
  first-member-in (finite-model-carrier M)
  (finite-model-immediate M) f x
  by blast
have ancestor:
  dependency-ancestor (finite-model-carrier M)
  (finite-model-immediate M) f x
  using first-f by (rule first-member-inD)
have independent:
  independent-in (finite-model-carrier M)
  (finite-model-immediate M) f
  using first-f by (rule first-member-inD)
have fC:  $f \in \text{finite-model-carrier } M$ 
  using independent by (rule independent-inD)
have f-ind: finite-model-independent M f
  using finite-model-independent-iff[OF fC] independent by blast
have bounded:
  finite-reach (length (finite-vertices M))
  (finite-vertices M)
  (dependency-on (finite-model-carrier M)
  (finite-model-immediate M)) f x
  using reach-iff[OF fC] ancestor by blast
have fV:  $f \in \text{set } (\text{finite-vertices } M)$ 
  using fC by (simp add: finite-model-carrier-def)
have Pf:
  finite-model-independent M f  $\wedge$ 
  finite-reach (length (finite-vertices M))
  (finite-vertices M)
  (dependency-on (finite-model-carrier M)
  (finite-model-immediate M)) f x
  using f-ind bounded by blast
show finite-list-ex
  ( $\lambda f$ . finite-model-independent M f  $\wedge$ 
  finite-reach (length (finite-vertices M))
  (finite-vertices M)
  (dependency-on (finite-model-carrier M)
  (finite-model-immediate M)) f x)
  (finite-vertices M)
  using fV Pf
  by (rule finite-list-exI)
qed
qed

lemma finite-model-bad-iff:

```

assumes $x \in \text{finite-model-carrier } M$
shows $\text{finite-model-bad } M \ x \longleftrightarrow$
 $\text{grounded-in } (\text{finite-model-carrier } M) (\text{finite-model-immediate } M) \ x \wedge$
 $\neg (\exists f. \text{first-member-in } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ f \ x)$
using $\text{assms } \text{finite-model-grounded-iff} [OF \ \text{assms}]$
 $\text{finite-model-supports-iff} [OF \ \text{assms}]$
by $(\text{auto simp: finite-model-bad-def})$

theorem $\text{finite-foundation-check-iff}$:

$\text{finite-foundation-check } M \longleftrightarrow$
 $\text{local-foundation-on } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) (\text{finite-model-carrier } M)$

proof –

show $?thesis$

unfolding $\text{finite-foundation-check-def local-foundation-on-def}$

proof (rule iffI)

assume $check$:

$\text{finite-list-all } (\lambda x. \neg \text{finite-model-bad } M \ x)$
 $(\text{finite-vertices } M)$

show $\forall a \in \text{finite-model-carrier } M.$

$\text{grounded-in } (\text{finite-model-carrier } M) (\text{finite-model-immediate } M) \ a$
 $\longrightarrow (\exists f. \text{first-member-in } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ f \ a)$

proof $(\text{intro ballI impI})$

fix a

assume aC : $a \in \text{finite-model-carrier } M$

and grounded :

$\text{grounded-in } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ a$

have not-bad : $\neg \text{finite-model-bad } M \ a$

using $check \ aC$

by $(\text{simp add: finite-list-all-iff finite-model-carrier-def})$

have bad-iff : $\text{finite-model-bad } M \ a \longleftrightarrow$

$\text{grounded-in } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ a \wedge$
 $\neg (\exists f. \text{first-member-in } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ f \ a)$

by $(\text{rule finite-model-bad-iff} [OF \ aC])$

show $\exists f. \text{first-member-in } (\text{finite-model-carrier } M)$

$(\text{finite-model-immediate } M) \ f \ a$

using $\text{not-bad grounded bad-iff}$ **by** blast

qed

next

assume foundation :

$\forall a \in \text{finite-model-carrier } M.$

$\text{grounded-in } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ a$
 $\longrightarrow (\exists f. \text{first-member-in } (\text{finite-model-carrier } M)$
 $(\text{finite-model-immediate } M) \ f \ a)$

have all-bad :

$\forall a \in \text{set } (\text{finite-vertices } M). \neg \text{finite-model-bad } M \ a$

proof (intro ballI)

fix a

assume $a\text{-list}$: $a \in \text{set } (\text{finite-vertices } M)$

have aC : $a \in \text{finite-model-carrier } M$

using $a\text{-list}$ **by** $(\text{simp add: finite-model-carrier-def})$

have bad-iff : $\text{finite-model-bad } M \ a \longleftrightarrow$

$\text{grounded-in } (\text{finite-model-carrier } M)$

```

      (finite-model-immediate M) a ∧
    ¬ (∃ f. first-member-in (finite-model-carrier M)
      (finite-model-immediate M) f a)
  by (rule finite-model-bad-iff[OF aC])
  show ¬ finite-model-bad M a
    using foundation aC bad-iff by blast
qed
show finite-list-all (λx. ¬ finite-model-bad M x)
  (finite-vertices M)
  using all-bad by (simp add: finite-list-all-iff)
qed
qed

```

```

lemma finite-failure-witness-spec:
  assumes ¬ finite-foundation-check M
  shows finite-failure-witness M ∈ finite-model-carrier M
    and finite-model-bad M (finite-failure-witness M)
proof -
  have bad-exists:
    ∃ x ∈ set (finite-vertices M). finite-model-bad M x
  using assms
  by (simp add: finite-foundation-check-def finite-list-all-iff)
  obtain x where failure:
    finite-first-failure (finite-model-bad M) (finite-vertices M) =
      Some x
  and x-list: x ∈ set (finite-vertices M)
  and bad: finite-model-bad M x
  using finite-first-failure-complete[OF bad-exists] .
  have witness-eq: finite-failure-witness M = x
    using failure by (simp add: finite-failure-witness-def)
  have xC: x ∈ finite-model-carrier M
    using x-list by (simp add: finite-model-carrier-def)
  show finite-failure-witness M ∈ finite-model-carrier M
    using witness-eq xC by simp
  show finite-model-bad M (finite-failure-witness M)
    using witness-eq bad by simp
qed

```

```

lemma finite-foundation-result-success:
  assumes finite-foundation-result M =
    Foundation-Certificate roots
  shows local-foundation-on (finite-model-carrier M)
    (finite-model-immediate M) (finite-model-carrier M)
proof -
  have finite-foundation-check M
  using assms
  by (simp add: finite-foundation-result-def split: if-splits)
  then show ?thesis
    using finite-foundation-check-iff by blast
qed

```

```

theorem finite-foundation-result-roots:
  assumes finite-foundation-result M =
    Foundation-Certificate roots
  shows set roots =
    {x ∈ finite-model-carrier M.
      independent-in (finite-model-carrier M)
        (finite-model-immediate M) x}
proof -

```

```

have check: finite-foundation-check M
  using assms
  by (simp add: finite-foundation-result-def split: if-splits)
have roots-eq:
  roots = filter (finite-model-independent M) (finite-vertices M)
  using assms check
  by (simp add: finite-foundation-result-def)
have model-roots:
  set roots =
    {x ∈ finite-model-carrier M. finite-model-independent M x}
  using roots-eq
  by (simp add: finite-model-carrier-def)
have root-independent:
  {x ∈ finite-model-carrier M. finite-model-independent M x} =
    {x ∈ finite-model-carrier M.
      independent-in (finite-model-carrier M)
        (finite-model-immediate M) x}
proof (rule set-eqI)
  fix x
  show x ∈ {x ∈ finite-model-carrier M.
    finite-model-independent M x}  $\longleftrightarrow$ 
    x ∈ {x ∈ finite-model-carrier M.
      independent-in (finite-model-carrier M)
        (finite-model-immediate M) x}
proof (rule iffI)
  assume left:
    x ∈ {x ∈ finite-model-carrier M.
      finite-model-independent M x}
  have xC: x ∈ finite-model-carrier M
  using left by simp
  have finite-independent: finite-model-independent M x
  using left by simp
  have independent:
    independent-in (finite-model-carrier M)
      (finite-model-immediate M) x
  using finite-model-independent-iff[OF xC] finite-independent
  by blast
  show x ∈ {x ∈ finite-model-carrier M.
    independent-in (finite-model-carrier M)
      (finite-model-immediate M) x}
  using xC independent by simp
next
  assume right:
    x ∈ {x ∈ finite-model-carrier M.
      independent-in (finite-model-carrier M)
        (finite-model-immediate M) x}
  have xC: x ∈ finite-model-carrier M
  using right by simp
  have independent:
    independent-in (finite-model-carrier M)
      (finite-model-immediate M) x
  using right by simp
  have finite-independent: finite-model-independent M x
  using finite-model-independent-iff[OF xC] independent
  by blast
  show x ∈ {x ∈ finite-model-carrier M.
    finite-model-independent M x}
  using xC finite-independent by simp
qed

```

```

qed
then show ?thesis
  using model-roots by simp
qed

theorem finite-foundation-result-certificate:
  assumes finite-foundation-result  $M =$ 
    Foundation-Certificate roots
  shows foundation-set-for (finite-model-carrier  $M$ )
    (finite-model-immediate  $M$ ) (finite-model-carrier  $M$ ) (set roots)
proof -
  have root-set:
    set roots =
      { $x \in$  finite-model-carrier  $M$ .
        independent-in (finite-model-carrier  $M$ )
          (finite-model-immediate  $M$ )  $x$ }
    using finite-foundation-result-roots[OF assms] .
  have roots-carrier:
    set roots  $\subseteq$  finite-model-carrier  $M$ 
  using root-set by auto
  have roots-independent:
     $\forall f \in$  set roots.
      independent-in (finite-model-carrier  $M$ )
        (finite-model-immediate  $M$ )  $f$ 
  using root-set by auto
  have local:
     $\forall a \in$  finite-model-carrier  $M$ .
      grounded-in (finite-model-carrier  $M$ )
        (finite-model-immediate  $M$ )  $a \longrightarrow$ 
        ( $\exists f$ . first-member-in (finite-model-carrier  $M$ )
          (finite-model-immediate  $M$ )  $f$   $a$ )
    using finite-foundation-result-success[OF assms]
    by (simp add: local-foundation-on-def)
  have supports:
     $\forall a \in$  finite-model-carrier  $M$ .
      grounded-in (finite-model-carrier  $M$ )
        (finite-model-immediate  $M$ )  $a \longrightarrow$ 
        ( $\exists f \in$  set roots.
          dependency-ancestor (finite-model-carrier  $M$ )
            (finite-model-immediate  $M$ )  $f$   $a$ )
  proof (intro ballI impI)
    fix  $a$ 
    assume  $aC$ :  $a \in$  finite-model-carrier  $M$ 
    and grounded:
      grounded-in (finite-model-carrier  $M$ )
        (finite-model-immediate  $M$ )  $a$ 
    obtain  $f$  where first: first-member-in (finite-model-carrier  $M$ )
      (finite-model-immediate  $M$ )  $f$   $a$ 
    using local  $aC$  grounded by blast
    have  $f$ -independent:
      independent-in (finite-model-carrier  $M$ )
        (finite-model-immediate  $M$ )  $f$ 
    using first by (rule first-member-inD)
    have  $f$ -ancestor:
      dependency-ancestor (finite-model-carrier  $M$ )
        (finite-model-immediate  $M$ )  $f$   $a$ 
    using first by (rule first-member-inD)
    have  $fC$ :  $f \in$  finite-model-carrier  $M$ 
    using  $f$ -independent by (rule independent-inD)
  end
end

```

```

have f-root:  $f \in \text{set roots}$ 
  using root-set fC f-independent by auto
then show  $\exists f \in \text{set roots.}$ 
  dependency-ancestor (finite-model-carrier M)
    (finite-model-immediate M) f a
  using f-ancestor by blast
qed
show ?thesis
  unfolding foundation-set-for-def
  using roots-carrier roots-independent supports by blast
qed

lemma finite-foundation-result-failure:
  assumes finite-foundation-result M =
    Missing-Foundation-Certificate x
  shows  $x \in \text{finite-model-carrier M}$  and finite-model-bad M x
proof –
  have not-check:  $\neg \text{finite-foundation-check M}$ 
    using assms
  by (simp add: finite-foundation-result-def split: if-splits)
  have witness:
    finite-failure-witness M  $\in \text{finite-model-carrier M}$ 
    finite-model-bad M (finite-failure-witness M)
    using finite-failure-witness-spec[OF not-check] .
  have x-eq:  $x = \text{finite-failure-witness M}$ 
    using assms not-check
  by (simp add: finite-foundation-result-def)
  show  $x \in \text{finite-model-carrier M}$ 
    using x-eq witness(1) by simp
  show finite-model-bad M x
    using x-eq witness(2) by simp
qed

theorem finite-foundation-result-complete:
  assumes  $\neg \text{local-foundation-on (finite-model-carrier M)}$ 
    (finite-model-immediate M) (finite-model-carrier M)
  obtains x where
    finite-foundation-result M =
      Missing-Foundation-Certificate x
    and  $x \in \text{finite-model-carrier M}$ 
    and finite-model-bad M x
proof –
  have not-check:  $\neg \text{finite-foundation-check M}$ 
    using assms finite-foundation-check-iff by blast
  have witness:
    finite-failure-witness M  $\in \text{finite-model-carrier M}$ 
    finite-model-bad M (finite-failure-witness M)
    using finite-failure-witness-spec[OF not-check] .
  have result:
    finite-foundation-result M =
      Missing-Foundation-Certificate (finite-failure-witness M)
    using not-check
  by (simp add: finite-foundation-result-def)
  show thesis
    using result witness by (rule that)
qed

theorem finite-foundation-result-correct:
  ( $\exists \text{roots. finite-foundation-result M =$ 

```

```

    Foundation-Certificate roots)  $\longleftrightarrow$ 
    local-foundation-on (finite-model-carrier M)
    (finite-model-immediate M) (finite-model-carrier M)
proof
  assume  $\exists$  roots. finite-foundation-result M =
    Foundation-Certificate roots
  then obtain roots where result:
    finite-foundation-result M = Foundation-Certificate roots
  by blast
  then show local-foundation-on (finite-model-carrier M)
    (finite-model-immediate M) (finite-model-carrier M)
  by (rule finite-foundation-result-success)
next
  assume foundation:
    local-foundation-on (finite-model-carrier M)
    (finite-model-immediate M) (finite-model-carrier M)
  have finite-foundation-check M
  using foundation finite-foundation-check-iff by blast
  then show  $\exists$  roots. finite-foundation-result M =
    Foundation-Certificate roots
  by (auto simp: finite-foundation-result-def)
qed

```

29 Executable Regression Models

definition *finite-chain-model* :: *finite-dependency-model*
where
finite-chain-model = ([0, 1, 2], [(0, 1), (1, 2)])

definition *finite-cycle-model* :: *finite-dependency-model*
where
finite-cycle-model = ([0, 1], [(0, 1), (1, 0)])

value *finite-foundation-result finite-chain-model*
value *finite-foundation-result finite-cycle-model*

end
theory *Second-Way-Reconstructions*
imports *Foundation-Separations*
begin

30 An Explicit Essentially Ordered-Series Interface

The phrase essentially ordered is not identified with well-foundedness. We expose one additional, source-facing predicate *derivative* for one explicit formal explication of the claim that members of a selected causal series have derivative causal efficacy. In this explication, on the selected ancestry, derivative status is tied to the dependency relation itself: a member is derivative exactly when it has an immediate predecessor in the carrier. Thus an independent member is a genuine non-derivative endpoint of the series, rather than merely receiving an arbitrary annotation. This is a modeling choice, not a definition of derivative causal efficacy in every metaphysical respect. The separate possession and exercise interface in the research layer deliberately tests a different local reading. Well-foundedness remains a separate hypothesis, so this interface can describe an infinite derivative series as well as a founded one.

definition *essentially-ordered-series* ::
 'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$
 ('a $\Rightarrow$ bool) $\Rightarrow$ 'a $\Rightarrow$ bool

where

essentially-ordered-series $C\ D$ *derivative* $a \longleftrightarrow$
 $a \in C \wedge \text{derivative } a \wedge \text{grounded-in } C\ D\ a \wedge$
 $(\forall x. \text{dependency-ancestor } C\ D\ x\ a \longrightarrow$
 $(\text{derivative } x \longleftrightarrow \text{grounded-in } C\ D\ x))$

lemma *essentially-ordered-seriesD*:

assumes *essentially-ordered-series* $C\ D$ *derivative* a
shows $a \in C$ *derivative* a *grounded-in* $C\ D\ a$
and $\forall x. \text{dependency-ancestor } C\ D\ x\ a \longrightarrow$
 $(\text{derivative } x \longleftrightarrow \text{grounded-in } C\ D\ x)$
using *assms* **by** (*auto simp: essentially-ordered-series-def*)

lemma *essentially-ordered-series-derivative-iff*:

assumes *essentially-ordered-series* $C\ D$ *derivative* a
and *dependency-ancestor* $C\ D\ x\ a$
shows *derivative* $x \longleftrightarrow \text{grounded-in } C\ D\ x$
using *assms* **by** (*auto simp: essentially-ordered-series-def*)

lemma *essentially-ordered-series-has-non-derivative-endpoint*:

assumes *essentially-ordered-series* $C\ D$ *derivative* a
and *first-member-in* $C\ D\ f\ a$
shows $\neg \text{derivative } f$

proof –

have *ancestor*: *dependency-ancestor* $C\ D\ f\ a$
using *first-member-inD*[*OF* *assms*(2)] **by** *blast*
have *independent*: *independent-in* $C\ D\ f$
using *first-member-inD*[*OF* *assms*(2)] **by** *blast*
have *not-grounded*: $\neg \text{grounded-in } C\ D\ f$
using *independent* **by** (*auto simp: grounded-in-def independent-in-def*)
have *derivative-iff*: *derivative* $f \longleftrightarrow \text{grounded-in } C\ D\ f$
using *essentially-ordered-series-derivative-iff*[*OF* *assms*(1) *ancestor*] .
then show *?thesis*
using *not-grounded* **by** *blast*

qed

lemma *integer-predecessor-is-essentially-ordered*:

essentially-ordered-series *UNIV* *integer-predecessor-cause*
 $(\lambda-. \text{True})\ (0 :: \text{int})$

proof –

have *incoming0*:
 $\exists y. \text{integer-predecessor-cause } y\ (0 :: \text{int})$
using *integer-predecessor-has-incoming*[*of* $0 :: \text{int}$] .
have *incoming*:
 $\forall x. \exists y. \text{integer-predecessor-cause } y\ x$
using *integer-predecessor-has-incoming* **by** *blast*
show *?thesis*
unfolding *essentially-ordered-series-def*
using *incoming0 incoming*
by (*auto simp: grounded-in-def integer-predecessor-cause-def*)

qed

theorem *essentially-ordered-does-not-mean-well-founded*:

essentially-ordered-series *UNIV* *integer-predecessor-cause*
 $(\lambda-. \text{True})\ (0 :: \text{int}) \wedge$
 $\neg \text{well-founded-on } \text{UNIV } \text{integer-predecessor-cause}$
using *integer-predecessor-is-essentially-ordered*
integer-predecessor-not-well-founded-on **by** *blast*

31 Cohoe-Style Formal Translation Package

The formal Cohoe-style translation keeps three interpretive commitments separate: there is a caused target, every caused target belongs to an essentially ordered derivative series, and a full-grounding relation supplies a foundation witness. These are formal explications of source-level claims, not literal quotations. The series predicate itself already states that its target lies in the carrier, so carrier membership is not an independent premise. Because it also states that the target is grounded, no duplicate predecessor premise is needed. The endpoint, nonemptiness, and ancestry-closure bridges from plural grounding to binary dependence are explicit locale assumptions.

definition *cohoe-second-way-premises* ::

$'a \text{ set} \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow$
 $('a \text{ set} \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow$
 $('a \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$

where

cohoe-second-way-premises $C D G$ *derivative caused* $\longleftrightarrow$
 $(\exists a. \text{caused } a) \wedge$
 $(\forall a. \text{caused } a \longrightarrow$
 $\text{essentially-ordered-series } C D \text{ derivative } a) \wedge$
 $\text{full-foundation-on } C D G C$

definition *cohoe-second-way-conclusion* ::

$'a \text{ set} \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow$
 $('a \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$

where

cohoe-second-way-conclusion $C D$ *derivative caused* $\longleftrightarrow$
 $(\forall a. \text{caused } a \longrightarrow$
 $(\exists x. \text{first-cause-in } C D x a \wedge \neg \text{derivative } x))$

locale *cohoe-second-way-reconstruction* =

fixes $C :: 'a \text{ set}$
and $D :: 'a \Rightarrow 'a \Rightarrow \text{bool}$
and $G :: 'a \text{ set} \Rightarrow 'a \Rightarrow \text{bool}$
and *derivative* $:: 'a \Rightarrow \text{bool}$
and *caused* $:: 'a \Rightarrow \text{bool}$
assumes *cohoe-assumptions*:
cohoe-second-way-premises $C D G$ *derivative caused*
and *singleton-endpoint*:
 $\bigwedge x a. x \in C \Longrightarrow a \in C \Longrightarrow D x a \Longrightarrow$
partial-grounding-on $C G x a$
and *full-nonempty*:
 $\bigwedge S a. \text{full-grounding-on } C G S a \Longrightarrow S \neq \{\}$
and *full-ancestry*:
 $\bigwedge S a x. \text{full-grounding-on } C G S a \Longrightarrow x \in S \Longrightarrow$
dependency-ancestor $C D x a$

begin

lemma *cohoe-has-local-foundation*:

local-foundation-on $C D C$

proof –

obtain *full-foundation-on* $C D G C$
using *cohoe-assumptions* **by** (*auto simp: cohoe-second-way-premises-def*)
then show *?thesis*
using *full-foundation-implies-local-foundation* [
where $C = C$ **and** $D = D$ **and** $G = G$ **and** $R = C$]
singleton-endpoint full-nonempty full-ancestry **by** *blast*

qed

```

theorem cohoe-first-cause-of:
  assumes caused a
  shows  $\exists x. \text{first-cause-in } C \ D \ x \ a$ 
proof –
  have premises-expanded:
     $(\forall a. \text{caused } a \longrightarrow$ 
      essentially-ordered-series C D derivative a)
    using cohoe-assumptions by (auto simp: cohoe-second-way-premises-def)
  obtain series: essentially-ordered-series C D derivative a
    using premises-expanded assms by blast
  have aC:  $a \in C$ 
    using series by (rule essentially-ordered-seriesD)
  have grounded: grounded-in C D a
    using series by (rule essentially-ordered-seriesD)
  have local:
     $\forall u \in C. \text{grounded-in } C \ D \ u \longrightarrow$ 
       $(\exists f. \text{first-member-in } C \ D \ f \ u)$ 
    using cohoe-has-local-foundation
    by (simp add: local-foundation-on-def)
  have first-exists:
     $\exists f. \text{first-member-in } C \ D \ f \ a$ 
    using local aC grounded by blast
  then obtain f where first-member: first-member-in C D f a
    by blast
  have f-neg:  $f \neq a$ 
proof
  assume  $f = a$ 
  have f-independent: independent-in C D f
    using first-member-inD[OF first-member] by blast
  obtain b where  $b \in C$  and  $D \ b \ a$ 
    using grounded by (auto simp: grounded-in-def)
  then show False
    using f-independent  $\langle f = a \rangle$ 
    by (auto simp: independent-in-def)
qed
show ?thesis
  using first-member f-neg
  by (auto simp: first-cause-in-def)
qed

```

```

theorem cohoe-non-derivative-first-cause-of:
  assumes caused a
  shows  $\exists x. \text{first-cause-in } C \ D \ x \ a \wedge \neg \text{derivative } x$ 
proof –
  have premises-expanded:
     $(\forall a. \text{caused } a \longrightarrow$ 
      essentially-ordered-series C D derivative a)
    using cohoe-assumptions by (auto simp: cohoe-second-way-premises-def)
  obtain series: essentially-ordered-series C D derivative a
    using premises-expanded assms by blast
  obtain x where first-cause: first-cause-in C D x a
    using cohoe-first-cause-of[OF assms] by blast
  have first-member: first-member-in C D x a
    using first-cause-inD[OF first-cause] by blast
  have not-derivative:  $\neg \text{derivative } x$ 
    using essentially-ordered-series-has-non-derivative-endpoint
      [OF series first-member] .
  then show ?thesis

```

```

    using first-cause by blast
qed

theorem cohoe-exact-conclusion:
  cohoe-second-way-conclusion C D derivative caused
  unfolding cohoe-second-way-conclusion-def
  using cohoe-non-derivative-first-cause-of by blast

theorem cohoe-global-first-cause:
  assumes  $\exists a. \text{caused } a$ 
  shows  $\exists x. \text{independent-in } C D x$ 
proof -
  obtain a where caused a
    using assms by blast
  obtain x where first-cause: first-cause-in C D x a
    using cohoe-first-cause-of[OF <caused a>] by blast
  have first-member: first-member-in C D x a
    using first-cause-inD[OF first-cause] by blast
  have independent: independent-in C D x
    using first-member-inD[OF first-member] by blast
  then show ?thesis by blast
qed

end

```

32 Romero–Pérez Reconstruction

Romero and Pérez present a first-order reconstruction in the source discussion. The Isabelle/HOL encoding below is not itself a first-order sentence: *well-founded-on* is a higher-order predicate quantifying over carrier subsets. The section therefore separates the source-level presentation from the formal encoding. Its conclusion is the existence of at least one member of the carrier with no immediate dependency predecessor. It does not assert uniqueness, necessity, agency, or that the first member is the cause of the universe. The equivalence between well-foundedness and absence of an infinite descending sequence uses Isabelle/HOL’s classical well-foundedness library, including its standard choice-based background.

```

definition romero-perez-second-way-premises ::
  'a set  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$ 
  ('a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  romero-perez-second-way-premises C D caused  $\longleftrightarrow$ 
    ( $\exists a \in C. \text{caused } a$ )  $\wedge$ 
    ( $\forall a. \text{caused } a \longrightarrow$ 
       $a \in C \wedge (\exists b \in C. D b a)$ )  $\wedge$ 
    well-founded-on C D

```

```

definition romero-perez-second-way-conclusion ::
  'a set  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  romero-perez-second-way-conclusion C D  $\longleftrightarrow$ 
    ( $\exists x \in C. \text{independent-in } C D x$ )

```

```

locale romero-perez-second-way-reconstruction =
  fixes C :: 'a set
  and D :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool
  and caused :: 'a  $\Rightarrow$  bool
  assumes rp-assumptions:
    romero-perez-second-way-premises C D caused

```

begin

lemma *romero-perez-well-founded*:

well-founded-on C D

using *rp-assumptions* **by** (*auto simp: romero-perez-second-way-premises-def*)

theorem *romero-perez-exact-conclusion*:

romero-perez-second-way-conclusion C D

proof –

obtain *a* **where** $a \in C$ **and** *caused a*

using *rp-assumptions* **by** (*auto simp: romero-perez-second-way-premises-def*)

obtain *x* **where** $x \in C$

and *dependency-ancestor C D x a*

and *independent-in C D x*

using *well-founded-has-minimal-ancestor* [
 OF romero-perez-well-founded $\langle a \in C \rangle$]

then show *?thesis*

by (*auto simp: romero-perez-second-way-conclusion-def*)

qed

theorem *romero-perez-first-cause-of*:

assumes *caused a*

shows $\exists x. \text{first-cause-in } C \ D \ x \ a$

proof –

have *aC*: $a \in C$

using *rp-assumptions* *assms*

by (*auto simp: romero-perez-second-way-premises-def*)

have *predecessor*: $\exists b \in C. \ D \ b \ a$

using *rp-assumptions* *assms*

by (*auto simp: romero-perez-second-way-premises-def*)

have *grounded*: *grounded-in C D a*

using *aC predecessor* **by** (*auto simp: grounded-in-def*)

obtain *f* **where** *first-member: first-member-in C D f a*

using *well-founded-implies-local-foundation* [
 OF romero-perez-well-founded, of C]

unfolding *local-foundation-on-def*

using *aC grounded* **by** *blast*

have *f-neq*: $f \neq a$

proof

assume $f = a$

have *f-independent: independent-in C D f*

using *first-member-inD* [*OF first-member*] **by** *blast*

obtain *b* **where** $b \in C$ **and** $D \ b \ a$

using *predecessor* **by** *blast*

then show *False*

using *f-independent* $\langle f = a \rangle$

by (*auto simp: independent-in-def*)

qed

show *?thesis*

using *first-member f-neq*

by (*auto simp: first-cause-in-def*)

qed

end

33 Comparison and Separation Summary

theorem *cohoe-conclusion-implies-global-independence*:

assumes *cohoe-second-way-conclusion* $C\ D$ derivative caused
and $\exists a. \text{caused } a$
shows $\exists x. \text{independent-in } C\ D\ x$
using *assms*
unfolding *cohoe-second-way-conclusion-def*
by (*meson first-cause-inD first-member-inD*)

theorem *romero-perez-premises-imply-cohoe-style-conclusion*:
assumes *romero-perez-second-way-premises* $C\ D$ caused
shows $\forall a. \text{caused } a \longrightarrow$
 $(\exists x. \text{first-cause-in } C\ D\ x\ a)$
proof (*intro allI impI*)
fix a
assume $\text{caused } a$
interpret R : *romero-perez-second-way-reconstruction* $C\ D$ caused
by (*rule romero-perez-second-way-reconstruction.intro[OF assms]*)
show $\exists x. \text{first-cause-in } C\ D\ x\ a$
using $R.\text{romero-perez-first-cause-of}[OF \langle \text{caused } a \rangle]$.
qed

theorem *local-foundation-does-not-imply-well-founded*:
local-foundation-on UNIV pedestal-dependency UNIV $\wedge$
 $\neg \text{well-founded-on UNIV pedestal-dependency}$
using *pedestal-local-foundation pedestal-not-well-founded* **by** *blast*

theorem *global-foundation-does-not-imply-full-foundation*:
local-foundation-on $\{0, 1, 2\}$ *paired-dependency* $\{2\}$ $\wedge$
partial-grounding-on $\{0, 1, 2\}$ *paired-full-grounding* $0\ 2$ $\wedge$
 $\neg \text{full-foundation-on } \{0, 1, 2\} \text{ paired-dependency}$
 $\text{paired-full-grounding } \{2\}$
using *partial-full-grounding-diverge* **by** *blast*

theorem *second-way-reconstruction-comparison*:
 $(\forall a. \text{caused } a \longrightarrow$
 $(\exists x. \text{first-cause-in } C\ D\ x\ a)) \longrightarrow$
 $((\exists a. \text{caused } a) \longrightarrow (\exists x. \text{independent-in } C\ D\ x))$
by (*auto simp: first-cause-in-def first-member-in-def*
independent-in-def)

theorem *second-way-separation-summary*:
 $(\exists a. \text{essentially-ordered-series UNIV}$
 $\text{integer-predecessor-cause } (\lambda-. \text{True}) (a :: \text{int})) \wedge$
 $\neg \text{well-founded-on UNIV integer-predecessor-cause}$ $\wedge$
local-foundation-on UNIV pedestal-dependency UNIV $\wedge$
global-foundation-on UNIV pedestal-dependency UNIV $\wedge$
 $\neg \text{well-founded-on UNIV pedestal-dependency}$ $\wedge$
partial-grounding-on $\{0, 1, 2\}$ *paired-full-grounding* $0\ 2$ $\wedge$
 $\neg \text{full-foundation-on } \{0, 1, 2\} \text{ paired-dependency}$
 $\text{paired-full-grounding } \{2\}$
using *integer-predecessor-is-essentially-ordered*
integer-predecessor-not-well-founded-on
foundation-without-well-foundedness-pedestal
partial-full-grounding-diverge **by** *blast*

theorem *second-way-foundation-headline*:
assumes $(\forall a. \text{caused } a \longrightarrow$
 $(\exists x. \text{first-cause-in } C\ D\ x\ a))$
shows $(\exists a. \text{caused } a) \longrightarrow (\exists x. \text{independent-in } C\ D\ x)$
using *second-way-reconstruction-comparison assms* **by** *blast*

```

end
theory Research-Results
  imports Second-Way-Reconstructions
begin

```

34 Exact Binary Foundation Classification

The central binary vocabulary admits an exact classification theorem. Under carrier coverage, meaning that the selected target set is a subset of C , the two apparently different foundation notions coincide because the set of all independent members is itself a foundation set whenever local foundation holds. This is stronger than merely recording the implication from a designated global witness to local first members.

```

lemma integer-predecessor-not-local-foundation:
  ¬ local-foundation-on UNIV integer-predecessor-cause UNIV
proof
  assume local:
    local-foundation-on UNIV integer-predecessor-cause UNIV
  have grounded-zero:
    grounded-in UNIV integer-predecessor-cause (0 :: int)
    using integer-predecessor-has-incoming[of 0 :: int]
    by (auto simp: grounded-in-def)
  obtain f where first:
    first-member-in UNIV integer-predecessor-cause f (0 :: int)
    using local grounded-zero
    by (auto simp: local-foundation-on-def)
  have independent:
    independent-in UNIV integer-predecessor-cause f
    using first by (rule first-member-inD)
  obtain y where integer-predecessor-cause y f
    using integer-predecessor-has-incoming[of f] by blast
  then show False
    using independent by (auto simp: independent-in-def)
qed

```

```

theorem binary-foundation-lattice:
  assumes  $R \subseteq C$ 
  shows (well-founded-on  $C \ D \longleftrightarrow$  chain-terminating-on  $C \ D$ )  $\wedge$ 
    (well-founded-on  $C \ D \longrightarrow$  acyclic-on  $C \ D$ )  $\wedge$ 
    (local-foundation-on  $C \ D \ R \longleftrightarrow$ 
      global-foundation-on  $C \ D \ R$ )  $\wedge$ 
    (well-founded-on  $C \ D \longrightarrow$  local-foundation-on  $C \ D \ R$ )
  using well-founded-iff-chain-terminating
    well-founded-implies-acyclic
    local-foundation-iff-global-foundation[OF assms]
    well-founded-implies-local-foundation by blast

```

```

theorem finite-binary-foundation-lattice:
  assumes finite  $C$  and  $R \subseteq C$ 
  shows
    (well-founded-on  $C \ D \longleftrightarrow$  acyclic-on  $C \ D$ )  $\wedge$ 
    (well-founded-on  $C \ D \longleftrightarrow$  chain-terminating-on  $C \ D$ )  $\wedge$ 
    (local-foundation-on  $C \ D \ R \longleftrightarrow$ 
      global-foundation-on  $C \ D \ R$ )
  using finite-well-founded-iff-acyclic-on[OF assms(1)]
    well-founded-iff-chain-terminating
    local-foundation-iff-global-foundation[OF assms(2)] by blast

```

theorem *finite-acyclic-implies-foundation*:
assumes *finite C and $R \subseteq C$ and acyclic-on C D*
shows *local-foundation-on C D R $\wedge$*
global-foundation-on C D R
proof –
have *well-founded*:
well-founded-on C D
using *finite-well-founded-iff-acyclic-on[OF assms(1)] assms(3)*
by *blast*
have *local*:
local-foundation-on C D R
using *well-founded-implies-local-foundation[OF well-founded]* .
have *global*:
global-foundation-on C D R
using *local-foundation-implies-global-foundation[OF local assms(2)]* .
show *?thesis*
using *local global by blast*
qed

theorem *finite-foundation-not-acyclic-rooted-cycle*:
finite ($\{0, 1, 2\} :: \text{nat set}$) $\wedge$
local-foundation-on $\{0, 1, 2\}$ rooted-cycle-dependency $\{0, 1, 2\} \wedge$
global-foundation-on $\{0, 1, 2\}$ rooted-cycle-dependency $\{0, 1, 2\} \wedge$
 $\neg$ *acyclic-on $\{0, 1, 2\}$ rooted-cycle-dependency*
using *local-foundation-not-acyclic-rooted-cycle by simp*

theorem *binary-foundation-separation-matrix*:
acyclic-on UNIV integer-predecessor-cause $\wedge$
 $\neg$ *local-foundation-on UNIV integer-predecessor-cause UNIV $\wedge$*
local-foundation-on UNIV pedestal-dependency UNIV $\wedge$
global-foundation-on UNIV pedestal-dependency UNIV $\wedge$
 $\neg$ *well-founded-on UNIV pedestal-dependency $\wedge$*
local-foundation-on $\{0, 1, 2\}$ rooted-cycle-dependency $\{0, 1, 2\} \wedge$
global-foundation-on $\{0, 1, 2\}$ rooted-cycle-dependency $\{0, 1, 2\} \wedge$
 $\neg$ *acyclic-on $\{0, 1, 2\}$ rooted-cycle-dependency $\wedge$*
local-foundation-on $\{0, 1, 2\}$ paired-dependency $\{2\} \wedge$
 $\neg$ *full-foundation-on $\{0, 1, 2\}$ paired-dependency*
paired-full-grounding $\{2\}$
using *integer-predecessor-acyclic-on*
integer-predecessor-not-local-foundation
foundation-without-well-foundedness-pedestal
local-foundation-not-acyclic-rooted-cycle
partial-full-grounding-diverge by blast

lemma *integer-predecessor-is-essentially-ordered-grounded*:
essentially-ordered-series UNIV integer-predecessor-cause
 $(\lambda x. \text{grounded-in UNIV integer-predecessor-cause } x) (0 :: \text{int})$

proof –
have *incoming*:
 $\forall x. \text{grounded-in UNIV integer-predecessor-cause } x$
using *integer-predecessor-has-incoming*
by *(auto simp: grounded-in-def)*
show *?thesis*
unfolding *essentially-ordered-series-def*
using *incoming by auto*
qed

35 Auxiliary Local Causal-Efficacy Stress Test

As an auxiliary stress test rather than a full semantics of derivative causal efficacy, we introduce a small structural efficacy interface. $H\ x\ p$ says that entity x possesses causal power p ; $E\ x\ p\ a$ says that it exercises that power toward effect a . A locally borrowed exercise requires an immediate cause x of a , and requires the exercised power to be possessed by an earlier supporter of x . This is deliberately local: it expresses transmission at each step without postulating a global well-foundedness or foundation principle.

definition *locally-borrowed-power-on* ::

```
'a set ⇒ ('a ⇒ 'a ⇒ bool) ⇒
('a ⇒ 'p ⇒ bool) ⇒
('a ⇒ 'p ⇒ 'a ⇒ bool) ⇒
'a set ⇒ bool
```

where

```
locally-borrowed-power-on C D H E R ←→
(∀ a ∈ R. grounded-in C D a →
(∃ x p y. x ∈ C ∧ D x a ∧ H x p ∧ E x p a ∧
y ∈ C ∧ D y x ∧ H y p))
```

lemma *integer-regress-has-local-borrowed-power*:

```
locally-borrowed-power-on UNIV integer-predecessor-cause
(λ(- :: int) (- :: unit). True)
(λ(- :: int) (- :: unit) (- :: int). True) UNIV
```

proof –

```
show ?thesis
unfolding locally-borrowed-power-on-def
proof (intro ballI impI)
fix a :: int
assume a ∈ (UNIV :: int set)
and grounded-in UNIV integer-predecessor-cause a
show ∃ x p y. x ∈ (UNIV :: int set) ∧
integer-predecessor-cause x a ∧
(λ(- :: int) (- :: unit). True) x p ∧
(λ(- :: int) (- :: unit) (- :: int). True) x p a ∧
y ∈ (UNIV :: int set) ∧
integer-predecessor-cause y x ∧
(λ(- :: int) (- :: unit). True) y p
by (intro exI[of - a - 1] exI[of - ()] exI[of - a - 2])
(simp add: integer-predecessor-cause-def)
```

qed

qed

theorem *essential-ordering-local-efficacy-countermodel*:

```
acyclic-on UNIV integer-predecessor-cause ∧
locally-borrowed-power-on UNIV integer-predecessor-cause
(λ(- :: int) (- :: unit). True)
(λ(- :: int) (- :: unit) (- :: int). True) UNIV ∧
essentially-ordered-series UNIV integer-predecessor-cause
(λx. grounded-in UNIV integer-predecessor-cause x) (0 :: int) ∧
¬ local-foundation-on UNIV integer-predecessor-cause UNIV
using integer-predecessor-acyclic-on
integer-regress-has-local-borrowed-power
integer-predecessor-is-essentially-ordered-grounded
integer-predecessor-not-local-foundation by blast
```

Thus the formalized local support/transmission condition leaves an infinite descending regress possible. Any theorem excluding this model must add a genuinely global principle such as well-foundedness, local/global foundation, or an equivalent termination condition. This is a result

about the precise interface formalized here, not a claim about every possible metaphysical theory of causal efficacy.

```

end
theory Premise-Minimality
  imports Research-Results
begin

```

36 Premise Factorisation

The Romero–Pérez package contains four logically distinct ingredients: existence of a caused item, carrier membership of caused items, an immediate predecessor for each caused item, and well-foundedness. The published package writes the first two ingredients together in one bounded existential. We factor them here so that premise deletion is a literal theorem rather than an informal checklist.

```

definition rp-existence-premise ::
  ('a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  rp-existence-premise caused  $\longleftrightarrow (\exists a. \text{caused } a)$ 

```

```

definition rp-carrier-premise ::
  'a set  $\Rightarrow$  ('a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  rp-carrier-premise C caused  $\longleftrightarrow$ 
    ( $\forall a. \text{caused } a \longrightarrow a \in C$ )

```

```

definition rp-predecessor-premise ::
  'a set  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$ 
    ('a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  rp-predecessor-premise C D caused  $\longleftrightarrow$ 
    ( $\forall a. \text{caused } a \longrightarrow (\exists b \in C. D \ b \ a)$ )

```

```

definition rp-well-founded-premise ::
  'a set  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  rp-well-founded-premise C D  $\longleftrightarrow$  well-founded-on C D

```

```

definition rp-global-root-conclusion ::
  'a set  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  rp-global-root-conclusion C D  $\longleftrightarrow$ 
    ( $\exists x \in C. \text{independent-in } C \ D \ x$ )

```

```

definition rp-local-first-cause-conclusion ::
  'a set  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$ 
    ('a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  rp-local-first-cause-conclusion C D caused  $\longleftrightarrow$ 
    ( $\forall a. \text{caused } a \longrightarrow$ 
      ( $\exists x. \text{first-cause-in } C \ D \ x \ a$ ))

```

```

definition rp-combined-conclusion ::
  'a set  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$ 
    ('a  $\Rightarrow$  bool)  $\Rightarrow$  bool
where
  rp-combined-conclusion C D caused  $\longleftrightarrow$ 
    rp-global-root-conclusion C D  $\wedge$ 

```

rp-local-first-cause-conclusion C D caused

lemma *romero-perez-premises-factor:*

romero-perez-second-way-premises C D caused $\longleftrightarrow$

rp-existence-premise caused $\wedge$

rp-carrier-premise C caused $\wedge$

rp-predecessor-premise C D caused $\wedge$

rp-well-founded-premise C D

unfolding *romero-perez-second-way-premises-def*

rp-existence-premise-def rp-carrier-premise-def

rp-predecessor-premise-def rp-well-founded-premise-def

by *blast*

lemma *rp-premises-yield-combined-conclusion:*

assumes *rp-existence-premise caused*

and *rp-carrier-premise C caused*

and *rp-predecessor-premise C D caused*

and *rp-well-founded-premise C D*

shows *rp-combined-conclusion C D caused*

proof —

have *package:*

romero-perez-second-way-premises C D caused

using *assms* **by** (*simp add: romero-perez-premises-factor*)

interpret *reconstruction:*

romero-perez-second-way-reconstruction C D caused

by (*rule romero-perez-second-way-reconstruction.intro[OF package]*)

have *global-raw:*

romero-perez-second-way-conclusion C D

using *reconstruction.romero-perez-exact-conclusion* .

have *global:*

rp-global-root-conclusion C D

using *global-raw*

by (*simp add: romero-perez-second-way-conclusion-def*

rp-global-root-conclusion-def)

have *local:*

rp-local-first-cause-conclusion C D caused

unfolding *rp-local-first-cause-conclusion-def*

proof (*intro allI impI*)

fix *a*

assume *caused a*

then show $\exists x. \text{first-cause-in } C D x a$

by (*rule reconstruction.romero-perez-first-cause-of*)

qed

show *?thesis*

unfolding *rp-combined-conclusion-def*

using *global local* **by** *blast*

qed

lemma *rp-local-first-cause-implies-global-root:*

assumes *rp-existence-premise caused*

and *rp-local-first-cause-conclusion C D caused*

shows *rp-global-root-conclusion C D*

proof —

obtain *a* **where** *caused a*

using *assms(1)* **by** (*auto simp: rp-existence-premise-def*)

have *local-expanded:*

$\forall a. \text{caused } a \longrightarrow (\exists x. \text{first-cause-in } C D x a)$

using *assms(2)*

by (*simp only: rp-local-first-cause-conclusion-def*)

```

have first-exists:
   $\exists x. \text{first-cause-in } C \ D \ x \ a$ 
  using local-expanded  $\langle \text{caused } a \rangle$  by blast
obtain x where first-cause-in  $C \ D \ x \ a$ 
  using first-exists by blast
have first-member: first-member-in  $C \ D \ x \ a$ 
  using first-cause-inD[OF  $\langle \text{first-cause-in } C \ D \ x \ a \rangle$ ] by blast
have independent: independent-in  $C \ D \ x$ 
  using first-member-inD[OF first-member] by blast
have xC:  $x \in C$ 
  using independent by (rule independent-inD)
show ?thesis
  unfolding rp-global-root-conclusion-def
  using xC independent by blast
qed

```

37 Romero–Pérez Deletion Certificates

Each theorem below retains all but the named premise and refutes one fixed combined conclusion: a global independent root together with a first cause for every caused item. This makes the deletion results premise-minimal relative to a single target, rather than switching conclusions between countermodels.

lemma *no-tranclp-if-no-edge*:

```

assumes  $\bigwedge u \ v. \neg R \ u \ v$ 
shows  $\neg R^{++} \ x \ y$ 
proof
  assume path:  $R^{++} \ x \ y$ 
  from path show False
proof (induction rule: tranclp-induct)
  case base
  then show False
    using assms by blast
next
  case (step y z)
  then show False
    using assms by blast
qed
qed

```

lemma *rp-empty-well-founded*:

```

well-founded-on ( $\{\}$  :: nat set) ( $\lambda(- :: nat) \neg. \text{False}$ )
proof –
  have finite: finite ( $\{\}$  :: nat set)
    by simp
  have acyclic:
    acyclic-on ( $\{\}$  :: nat set) ( $\lambda(- :: nat) \neg. \text{False}$ )
    unfolding acyclic-on-def
    by simp
  then show ?thesis
    using finite-well-founded-iff-acyclic-on[OF finite] by blast
qed

```

theorem *romero-perez-delete-existence*:

```

rp-carrier-premise ( $\{\}$  :: nat set) ( $\lambda-. \text{False}$ )  $\wedge$ 
rp-predecessor-premise  $\{\}$  ( $\lambda(- :: nat) \neg. \text{False}$ ) ( $\lambda-. \text{False}$ )  $\wedge$ 
rp-well-founded-premise  $\{\}$  ( $\lambda(- :: nat) \neg. \text{False}$ )  $\wedge$ 
 $\neg$  rp-combined-conclusion  $\{\}$  ( $\lambda(- :: nat) \neg. \text{False}$ )

```

```

    (λ-. False)
using rp-empty-well-founded
by (simp add: rp-carrier-premise-def rp-predecessor-premise-def
    rp-well-founded-premise-def rp-combined-conclusion-def
    rp-global-root-conclusion-def rp-local-first-cause-conclusion-def
    )

definition rp-no-carrier-dependency :: nat ⇒ nat ⇒ bool
where
    rp-no-carrier-dependency x y ⟷ x = 0 ∧ y = 1

definition rp-no-carrier-caused :: nat ⇒ bool
where
    rp-no-carrier-caused a ⟷ a = 1

lemma rp-no-carrier-well-founded:
    well-founded-on {0 :: nat} rp-no-carrier-dependency
proof –
    have finite: finite ({0 :: nat})
    by simp
    have acyclic:
        acyclic-on {0 :: nat} rp-no-carrier-dependency
    unfolding acyclic-on-def
    proof (intro ballI impI)
    fix x
    assume x ∈ ({0 :: nat})
    have no-edge:
        ∧ u v. ¬ dependency-on {0 :: nat}
            rp-no-carrier-dependency u v
    by (auto simp: dependency-on-def rp-no-carrier-dependency-def)
    show ¬ (dependency-on {0 :: nat}
        rp-no-carrier-dependency)++ x x
    by (rule no-tranclp-if-no-edge[OF no-edge])
    qed
    then show ?thesis
    using finite-well-founded-iff-acyclic-on[OF finite] by blast
qed

theorem romero-perez-delete-carrier:
    rp-existence-premise rp-no-carrier-caused ∧
    rp-predecessor-premise {0 :: nat} rp-no-carrier-dependency
    rp-no-carrier-caused ∧
    rp-well-founded-premise {0 :: nat} rp-no-carrier-dependency ∧
    ¬ rp-carrier-premise {0 :: nat} rp-no-carrier-caused ∧
    ¬ rp-combined-conclusion {0 :: nat}
    rp-no-carrier-dependency rp-no-carrier-caused
using rp-no-carrier-well-founded
unfolding rp-existence-premise-def rp-predecessor-premise-def
    rp-well-founded-premise-def rp-carrier-premise-def
    rp-combined-conclusion-def rp-local-first-cause-conclusion-def
    rp-global-root-conclusion-def first-cause-in-def
    first-member-in-def dependency-ancestor-def dependency-on-def
    rp-no-carrier-dependency-def rp-no-carrier-caused-def
by simp

definition rp-no-predecessor-dependency :: nat ⇒ nat ⇒ bool
where
    rp-no-predecessor-dependency - - ⟷ False

```

definition *rp-no-predecessor-caused* :: *nat* $\Rightarrow$ *bool*
where
rp-no-predecessor-caused *a* $\longleftrightarrow$ *a* = 0

lemma *rp-no-predecessor-well-founded*:
well-founded-on {*0* :: *nat*} *rp-no-predecessor-dependency*
proof –
have *finite*: *finite* ({*0* :: *nat*})
by *simp*
have *acyclic*:
acyclic-on {*0* :: *nat*} *rp-no-predecessor-dependency*
unfolding *acyclic-on-def*
proof (*intro ballI impI*)
fix *x*
assume *x* $\in$ ({*0* :: *nat*})
have *no-edge*:
 $\bigwedge u v. \neg \text{dependency-on } \{0 :: \text{nat}\}$
 $\text{rp-no-predecessor-dependency } u v$
by (*simp add: dependency-on-def rp-no-predecessor-dependency-def*)
show $\neg (\text{dependency-on } \{0 :: \text{nat}\}$
 $\text{rp-no-predecessor-dependency})^{++} x x$
by (*rule no-tranclp-if-no-edge[OF no-edge]*)
qed
then show *?thesis*
using *finite-well-founded-iff-acyclic-on[OF finite]* **by** *blast*
qed

theorem *romero-perez-delete-predecessor*:
rp-existence-premise *rp-no-predecessor-caused* $\wedge$
rp-carrier-premise {*0* :: *nat*} *rp-no-predecessor-caused* $\wedge$
rp-well-founded-premise {*0* :: *nat*} *rp-no-predecessor-dependency* $\wedge$
 $\neg \text{rp-predecessor-premise } \{0 :: \text{nat}\} \text{rp-no-predecessor-dependency}$
 $\text{rp-no-predecessor-caused} \wedge$
 $\neg \text{rp-combined-conclusion } \{0 :: \text{nat}\}$
 $\text{rp-no-predecessor-dependency } \text{rp-no-predecessor-caused}$
using *rp-no-predecessor-well-founded*
unfolding *rp-existence-premise-def* *rp-carrier-premise-def*
rp-well-founded-premise-def *rp-predecessor-premise-def*
rp-combined-conclusion-def *rp-local-first-cause-conclusion-def*
rp-global-root-conclusion-def *first-cause-in-def*
first-member-in-def *dependency-ancestor-def* *dependency-on-def*
rp-no-predecessor-dependency-def *rp-no-predecessor-caused-def*
by *simp*

theorem *romero-perez-delete-well-foundedness*:
rp-existence-premise ($\lambda(- :: \text{int}). \text{True}$) $\wedge$
rp-carrier-premise *UNIV* ($\lambda-. \text{True}$) $\wedge$
rp-predecessor-premise *UNIV* *integer-predecessor-cause*
($\lambda-. \text{True}$) $\wedge$
 $\neg \text{rp-well-founded-premise } \text{UNIV } \text{integer-predecessor-cause} \wedge$
 $\neg \text{rp-combined-conclusion } \text{UNIV } \text{integer-predecessor-cause}$
($\lambda-. \text{True}$)
proof –
have *existence*:
rp-existence-premise ($\lambda(- :: \text{int}). \text{True}$)
by (*simp add: rp-existence-premise-def*)
have *carrier*:
rp-carrier-premise *UNIV* ($\lambda-. \text{True}$)
by (*simp add: rp-carrier-premise-def*)

```

have predecessor:
  rp-predecessor-premise UNIV integer-predecessor-cause
    (λ-. True)
  unfolding rp-predecessor-premise-def
proof (intro allI impI)
  fix a :: int
  assume (λ-. True) a
  obtain b where integer-predecessor-cause b a
  using integer-predecessor-has-incoming[of a] by blast
  show ∃ b ∈ (UNIV :: int set).
    integer-predecessor-cause b a
  using ⟨integer-predecessor-cause b a⟩ by blast
qed
have not-well-founded:
  ¬ rp-well-founded-premise UNIV integer-predecessor-cause
  using integer-predecessor-not-well-founded-on
  by (simp add: rp-well-founded-premise-def)
have no-first-cause:
  ∧x a. ¬ first-cause-in UNIV integer-predecessor-cause x a
proof -
  fix x a :: int
  show ¬ first-cause-in UNIV integer-predecessor-cause x a
  using integer-predecessor-has-incoming[of x]
  by (auto simp: first-cause-in-def first-member-in-def
    independent-in-def integer-predecessor-cause-def)
qed
have not-local:
  ¬ rp-local-first-cause-conclusion UNIV
    integer-predecessor-cause (λ-. True)
  unfolding rp-local-first-cause-conclusion-def
  using no-first-cause
  by simp
have not-combined:
  ¬ rp-combined-conclusion UNIV integer-predecessor-cause
    (λ-. True)
  unfolding rp-combined-conclusion-def
  using not-local by blast
show ?thesis
proof (rule conjI)
  show rp-existence-premise (λ(- :: int). True)
  using existence .
  show rp-carrier-premise UNIV (λ-. True) ∧
    rp-predecessor-premise UNIV integer-predecessor-cause
      (λ-. True) ∧
    ¬ rp-well-founded-premise UNIV integer-predecessor-cause ∧
    ¬ rp-combined-conclusion UNIV
      integer-predecessor-cause (λ-. True)
proof (rule conjI)
  show rp-carrier-premise UNIV (λ-. True)
  using carrier .
  show rp-predecessor-premise UNIV integer-predecessor-cause
    (λ-. True) ∧
    ¬ rp-well-founded-premise UNIV integer-predecessor-cause ∧
    ¬ rp-combined-conclusion UNIV
      integer-predecessor-cause (λ-. True)
proof (rule conjI)
  show rp-predecessor-premise UNIV integer-predecessor-cause
    (λ-. True)
  using predecessor .

```

```

show  $\neg$  rp-well-founded-premise UNIV integer-predecessor-cause  $\wedge$ 
 $\neg$  rp-combined-conclusion UNIV
integer-predecessor-cause ( $\lambda$ -. True)
proof (rule conjI)
show  $\neg$  rp-well-founded-premise UNIV integer-predecessor-cause
using not-well-founded .
show  $\neg$  rp-combined-conclusion UNIV
integer-predecessor-cause ( $\lambda$ -. True)
using not-combined .
qed
qed
qed
qed
qed

```

38 Cohoe-Style Formal-Translation Deletion

definition *cohoe-existence-premise* ::
 $('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$

where
cohoe-existence-premise caused $\longleftrightarrow (\exists a. \text{caused } a)$

definition *cohoe-carrier-premise* ::
 $'a \text{ set} \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$

where
cohoe-carrier-premise C caused $\longleftrightarrow$
 $(\forall a. \text{caused } a \longrightarrow a \in C)$

definition *cohoe-essential-ordering-premise* ::
 $'a \text{ set} \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow$
 $('a \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$

where
cohoe-essential-ordering-premise C D derivative caused $\longleftrightarrow$
 $(\forall a. \text{caused } a \longrightarrow$
essentially-ordered-series C D derivative a)

lemma *cohoe-essential-ordering-implies-carrier*:

assumes *essential*:
cohoe-essential-ordering-premise C D derivative caused
shows *cohoe-carrier-premise C caused*

proof –

have *essential-all*:
 $\forall a. \text{caused } a \longrightarrow$
essentially-ordered-series C D derivative a
using *essential*
by (*simp only: cohoe-essential-ordering-premise-def*)

show *?thesis*
unfolding *cohoe-carrier-premise-def*

proof (*intro allI impI*)
fix *a*
assume *caused-a: caused a*
obtain *series: essentially-ordered-series C D derivative a*
using *essential-all caused-a* **by** *blast*
show $a \in C$
using *series* **by** (*rule essentially-ordered-seriesD*)

qed
qed

definition *cohoe-full-foundation-premise* ::

'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$
('a set $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ bool

where

cohoe-full-foundation-premise C D G $\longleftrightarrow$
full-foundation-on C D G C

definition *cohoe-singleton-endpoint-premise* ::

'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$
('a set $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ bool

where

cohoe-singleton-endpoint-premise C D G $\longleftrightarrow$
($\forall x a. x \in C \longrightarrow a \in C \longrightarrow D x a \longrightarrow$
partial-grounding-on C G x a)

definition *cohoe-full-nonempty-premise* ::

'a set $\Rightarrow$ ('a set $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ bool

where

cohoe-full-nonempty-premise C G $\longleftrightarrow$
($\forall S a. \text{full-grounding-on } C G S a \longrightarrow S \neq \{\}$)

definition *cohoe-full-ancestry-premise* ::

'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$
('a set $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$ bool

where

cohoe-full-ancestry-premise C D G $\longleftrightarrow$
($\forall S a x. \text{full-grounding-on } C G S a \longrightarrow x \in S \longrightarrow$
dependency-ancestor C D x a)

definition *cohoe-global-non-derivative-conclusion* ::

'a set $\Rightarrow$ ('a $\Rightarrow$ 'a $\Rightarrow$ bool) $\Rightarrow$
('a $\Rightarrow$ bool) $\Rightarrow$ bool

where

cohoe-global-non-derivative-conclusion C D derivative $\longleftrightarrow$
($\exists x. \text{independent-in } C D x \wedge \neg \text{derivative } x$)

lemma *cohoe-premises-factor*:

cohoe-second-way-premises C D G derivative caused $\longleftrightarrow$
cohoe-existence-premise caused $\wedge$
cohoe-essential-ordering-premise C D derivative caused $\wedge$
cohoe-full-foundation-premise C D G

unfolding *cohoe-second-way-premises-def*

cohoe-existence-premise-def
cohoe-essential-ordering-premise-def
cohoe-full-foundation-premise-def

by blast

lemma *cohoe-premises-derive-global-non-derivative*:

assumes *cohoe-existence-premise* caused
and *cohoe-essential-ordering-premise* C D derivative caused
and *cohoe-full-foundation-premise* C D G
and *cohoe-singleton-endpoint-premise* C D G
and *cohoe-full-nonempty-premise* C G
and *cohoe-full-ancestry-premise* C D G

shows *cohoe-global-non-derivative-conclusion* C D derivative

proof –

have package:

cohoe-second-way-premises C D G derivative caused
using *assms*(1–3) **by** (*simp add: cohoe-premises-factor*)

```

have endpoint-all:
   $\forall x a. x \in C \longrightarrow a \in C \longrightarrow D x a \longrightarrow$ 
  partial-grounding-on  $C G x a$ 
  using assms(4)
  by (simp only: cohoe-singleton-endpoint-premise-def)
have nonempty-all:
   $\forall S a. \text{full-grounding-on } C G S a \longrightarrow S \neq \{\}$ 
  using assms(5)
  by (simp only: cohoe-full-nonempty-premise-def)
have ancestry-all:
   $\forall S a x. \text{full-grounding-on } C G S a \longrightarrow x \in S \longrightarrow$ 
  dependency-ancestor  $C D x a$ 
  using assms(6)
  by (simp only: cohoe-full-ancestry-premise-def)
interpret reconstruction:
  cohoe-second-way-reconstruction  $C D G$  derivative caused
proof (rule cohoe-second-way-reconstruction.intro[OF package])
  show  $\bigwedge x a. x \in C \implies a \in C \implies D x a \implies$ 
  partial-grounding-on  $C G x a$ 
  using endpoint-all by blast
  show  $\bigwedge S a. \text{full-grounding-on } C G S a \implies S \neq \{\}$ 
  using nonempty-all by blast
  show  $\bigwedge S a x. \text{full-grounding-on } C G S a \implies x \in S \implies$ 
  dependency-ancestor  $C D x a$ 
  using ancestry-all by blast
qed
have exists-caused:  $\exists a. \text{caused } a$ 
  using assms(1) by (simp only: cohoe-existence-premise-def)
obtain a where caused a
  using exists-caused by blast
obtain x where first-cause:
  first-cause-in  $C D x a \wedge \neg \text{derivative } x$ 
  using reconstruction.cohoe-non-derivative-first-cause-of[
    OF  $\langle \text{caused } a \rangle$ ] by blast
have first-cause-only: first-cause-in  $C D x a$ 
  using first-cause by (rule conjunct1)
have first-member: first-member-in  $C D x a$ 
  using first-cause-inD[OF first-cause-only] by blast
have not-derivative:  $\neg \text{derivative } x$ 
  using first-cause by (rule conjunct2)
have independent: independent-in  $C D x$ 
  using first-member by (rule first-member-inD)
show ?thesis
  unfolding cohoe-global-non-derivative-conclusion-def
  using independent not-derivative by blast
qed

definition cohoe-chain-dependency ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$ 
where
  cohoe-chain-dependency  $x y \longleftrightarrow x = 0 \wedge y = 1$ 

definition cohoe-chain-grounding ::  $\text{nat set} \Rightarrow \text{nat} \Rightarrow \text{bool}$ 
where
  cohoe-chain-grounding  $S a \longleftrightarrow S = \{0\} \wedge a = 1$ 

definition cohoe-integer-singleton-grounding ::
   $\text{int set} \Rightarrow \text{int} \Rightarrow \text{bool}$ 
where
  cohoe-integer-singleton-grounding  $S a \longleftrightarrow$ 

```

$(\exists x. S = \{x\} \wedge \text{integer-predecessor-cause } x \ a)$

definition *cohoe-integer-empty-or-singleton-grounding* ::
 $\text{int set} \Rightarrow \text{int} \Rightarrow \text{bool}$

where

cohoe-integer-empty-or-singleton-grounding $S \ a \longleftrightarrow$
 $S = \{\}$ $\vee$
 $(\exists x. S = \{x\} \wedge \text{integer-predecessor-cause } x \ a)$

lemma *cohoe-chain-full-foundation*:

full-foundation-on $\{0, 1\}$ *cohoe-chain-dependency*
cohoe-chain-grounding $\{0, 1\}$

proof –

have *foundation*:

full-foundation-for $\{0, 1\}$ *cohoe-chain-dependency*
cohoe-chain-grounding $\{0, 1\}$ $\{0\}$

unfolding *full-foundation-for-def*

by (*simp add: full-grounding-on-def partial-grounding-on-def*
grounded-in-def dependency-on-def cohoe-chain-dependency-def
cohoe-chain-grounding-def independent-in-def)

then show *?thesis*

by (*rule full-foundation-onI*)

qed

lemma *cohoe-chain-endpoint*:

cohoe-singleton-endpoint-premise $\{0, 1\}$ *cohoe-chain-dependency*
cohoe-chain-grounding

unfolding *cohoe-singleton-endpoint-premise-def*

proof (*intro allI impI*)

fix $x \ a :: \text{nat}$

assume $x \in \{0, 1\}$ $a \in \{0, 1\}$

and Dxa : *cohoe-chain-dependency* $x \ a$

have $x0$: $x = 0$

using Dxa **by** (*simp add: cohoe-chain-dependency-def*)

have $a1$: $a = 1$

using Dxa **by** (*simp add: cohoe-chain-dependency-def*)

have *full*:

full-grounding-on $\{0, 1\}$ *cohoe-chain-grounding* $\{0\}$ 1

by (*simp add: full-grounding-on-def cohoe-chain-grounding-def*)

have *partial01*:

partial-grounding-on $\{0, 1\}$ *cohoe-chain-grounding* $0 \ 1$

using *full* **by** (*rule partial-grounding-onI*) *simp*

show *partial-grounding-on* $\{0, 1\}$ *cohoe-chain-grounding* $x \ a$

using *partial01* $x0 \ a1$ **by** *simp*

qed

lemma *cohoe-chain-nonempty*:

cohoe-full-nonempty-premise $\{0, 1\}$ *cohoe-chain-grounding*

unfolding *cohoe-full-nonempty-premise-def*

by (*auto simp: full-grounding-on-def cohoe-chain-grounding-def*)

lemma *cohoe-chain-ancestry*:

cohoe-full-ancestry-premise $\{0, 1\}$ *cohoe-chain-dependency*
cohoe-chain-grounding

unfolding *cohoe-full-ancestry-premise-def*

proof (*intro allI impI impI*)

fix $S :: \text{nat set}$ **and** $a \ x :: \text{nat}$

assume *full*: *full-grounding-on* $\{0, 1\}$ *cohoe-chain-grounding* $S \ a$

and $x \in S$

have $S0: S = \{0\}$ **and** $a1: a = 1$
using full by (*auto simp: full-grounding-on-def cohoe-chain-grounding-def*)
have $x0: x = 0$
using $S0 \langle x \in S \rangle$ **by** *simp*
show *dependency-ancestor* $\{0, 1\}$ *cohoe-chain-dependency* $x a$
using $x0 a1$ **by** (*simp add: dependency-ancestor-direct cohoe-chain-dependency-def*)
qed

theorem *cohoe-delete-existence:*

cohoe-essential-ordering-premise $\{\}$ $(\lambda(- :: \text{nat}) -. \text{False})$
 $(\lambda-. \text{False}) (\lambda-. \text{False}) \wedge$
cohoe-full-foundation-premise $\{\}$ $(\lambda(- :: \text{nat}) -. \text{False})$
 $(\lambda- -. \text{False}) \wedge$
cohoe-singleton-endpoint-premise $\{\}$ $(\lambda(- :: \text{nat}) -. \text{False})$
 $(\lambda- -. \text{False}) \wedge$
cohoe-full-nonempty-premise $\{\}$ $(\lambda- -. \text{False}) \wedge$
cohoe-full-ancestry-premise $\{\}$ $(\lambda(- :: \text{nat}) -. \text{False})$
 $(\lambda- -. \text{False}) \wedge$
 $\neg$ *cohoe-global-non-derivative-conclusion* $\{\}$
 $(\lambda(- :: \text{nat}) -. \text{False}) (\lambda-. \text{False})$
by (*simp add: cohoe-essential-ordering-premise-def cohoe-full-foundation-premise-def cohoe-singleton-endpoint-premise-def cohoe-full-nonempty-premise-def cohoe-full-ancestry-premise-def cohoe-global-non-derivative-conclusion-def full-foundation-on-def full-foundation-for-def full-grounding-on-def partial-grounding-on-def independent-in-def*)

theorem *cohoe-delete-essential-ordering:*

cohoe-existence-premise $(\lambda(a :: \text{nat}). a = 1) \wedge$
cohoe-full-foundation-premise $\{0, 1\}$ *cohoe-chain-dependency*
cohoe-chain-grounding $\wedge$
cohoe-singleton-endpoint-premise $\{0, 1\}$ *cohoe-chain-dependency*
cohoe-chain-grounding $\wedge$
cohoe-full-nonempty-premise $\{0, 1\}$ *cohoe-chain-grounding* $\wedge$
cohoe-full-ancestry-premise $\{0, 1\}$ *cohoe-chain-dependency*
cohoe-chain-grounding $\wedge$
 $\neg$ *cohoe-global-non-derivative-conclusion* $\{0, 1\}$
cohoe-chain-dependency $(\lambda-. \text{True})$
using *cohoe-chain-full-foundation cohoe-chain-endpoint cohoe-chain-nonempty cohoe-chain-ancestry*
by (*simp add: cohoe-existence-premise-def cohoe-full-foundation-premise-def cohoe-singleton-endpoint-premise-def cohoe-full-nonempty-premise-def cohoe-full-ancestry-premise-def cohoe-global-non-derivative-conclusion-def*)

lemma *cohoe-integer-singleton-endpoint:*

cohoe-singleton-endpoint-premise *UNIV integer-predecessor-cause*
cohoe-integer-singleton-grounding
unfolding *cohoe-singleton-endpoint-premise-def*

proof (*intro allI impI*)

fix $x a :: \text{int}$
assume $x \in (\text{UNIV} :: \text{int set})$ $a \in (\text{UNIV} :: \text{int set})$
and $Dxa: \text{integer-predecessor-cause } x a$
have *full*:

```

    full-grounding-on (UNIV :: int set)
    cohoe-integer-singleton-grounding {x} a
  by (simp add: full-grounding-on-def
    cohoe-integer-singleton-grounding-def Dxa)
show partial-grounding-on (UNIV :: int set)
    cohoe-integer-singleton-grounding x a
  using partial-grounding-onI[OF full] by simp
qed

```

```

lemma cohoe-integer-singleton-nonempty:
  cohoe-full-nonempty-premise (UNIV :: int set)
  cohoe-integer-singleton-grounding
  unfolding cohoe-full-nonempty-premise-def
proof (intro allI impI)
  fix S :: int set and a :: int
  assume full:
    full-grounding-on (UNIV :: int set)
    cohoe-integer-singleton-grounding S a
  obtain x where S = {x}
  using full by (auto simp: full-grounding-on-def
    cohoe-integer-singleton-grounding-def)
  show S ≠ {}
  using ⟨S = {x}⟩ by simp
qed

```

```

lemma cohoe-integer-singleton-ancestry:
  cohoe-full-ancestry-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-integer-singleton-grounding
  unfolding cohoe-full-ancestry-premise-def
proof (intro allI impI impI)
  fix S :: int set and a x :: int
  assume full:
    full-grounding-on (UNIV :: int set)
    cohoe-integer-singleton-grounding S a
    and x ∈ S
  obtain y where S = {y} and Dya:
    integer-predecessor-cause y a
  using full by (auto simp: full-grounding-on-def
    cohoe-integer-singleton-grounding-def)
  have xy: x = y
  using ⟨S = {y}⟩ ⟨x ∈ S⟩ by simp
  show dependency-ancestor (UNIV :: int set)
    integer-predecessor-cause x a
  using dependency-ancestor-direct[
    where C = UNIV :: int set
    and D = integer-predecessor-cause and x = y and y = a]
    Dya xy by simp
qed

```

```

definition cohoe-empty-grounding :: int set ⇒ int ⇒ bool
where
  cohoe-empty-grounding S a ⟷ False

```

```

lemma cohoe-empty-full-foundation:
  full-foundation-on (UNIV :: int set) integer-predecessor-cause
  cohoe-empty-grounding UNIV
proof –
  have foundation:
    full-foundation-for (UNIV :: int set) integer-predecessor-cause

```

```

    cohoe-empty-grounding UNIV {}
  unfolding full-foundation-for-def
  by (simp add: cohoe-empty-grounding-def grounded-in-def
    partial-grounding-on-def full-grounding-on-def independent-in-def)
  then show ?thesis
    by (rule full-foundation-onI)
qed

```

```

lemma cohoe-empty-nonempty:
  cohoe-full-nonempty-premise (UNIV :: int set)
  cohoe-empty-grounding
  unfolding cohoe-full-nonempty-premise-def
  by (simp add: full-grounding-on-def cohoe-empty-grounding-def)

```

```

lemma cohoe-empty-ancestry:
  cohoe-full-ancestry-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-empty-grounding
  unfolding cohoe-full-ancestry-premise-def
  by (simp add: full-grounding-on-def cohoe-empty-grounding-def)

```

```

lemma cohoe-empty-endpoint-fails:
  ¬ cohoe-singleton-endpoint-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-empty-grounding

```

```

proof
  assume endpoint:
    cohoe-singleton-endpoint-premise (UNIV :: int set)
    integer-predecessor-cause cohoe-empty-grounding
  have endpoint-all:
     $\bigwedge x a. x \in (UNIV :: int set) \implies$ 
     $a \in (UNIV :: int set) \implies$ 
     $integer-predecessor-cause\ x\ a \implies$ 
     $partial-grounding-on\ (UNIV :: int set)\ cohoe-empty-grounding\ x\ a$ 
    using endpoint
    by (simp only: cohoe-singleton-endpoint-premise-def)
  have partial:
    partial-grounding-on (UNIV :: int set) cohoe-empty-grounding (-1) 0
  proof (rule endpoint-all)
    show  $(-1 :: int) \in (UNIV :: int set)$  by simp
    show  $(0 :: int) \in (UNIV :: int set)$  by simp
    show integer-predecessor-cause  $(-1 :: int)\ 0$ 
      by (simp add: integer-predecessor-cause-def)
  qed
  show False
    using partial
    by (simp add: partial-grounding-on-def full-grounding-on-def
      cohoe-empty-grounding-def)
qed

```

```

theorem cohoe-delete-singleton-endpoint:
  cohoe-existence-premise  $(\lambda(a :: int). a = 0) \wedge$ 
  cohoe-essential-ordering-premise (UNIV :: int set)
  integer-predecessor-cause  $(\lambda::int. True)\ (\lambda a::int. a = 0) \wedge$ 
  cohoe-full-foundation-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-empty-grounding  $\wedge$ 
  ¬ cohoe-singleton-endpoint-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-empty-grounding  $\wedge$ 
  cohoe-full-nonempty-premise (UNIV :: int set) cohoe-empty-grounding  $\wedge$ 
  cohoe-full-ancestry-premise (UNIV :: int set) integer-predecessor-cause
  cohoe-empty-grounding  $\wedge$ 

```

```

  ¬ cohoe-global-non-derivative-conclusion (UNIV :: int set)
  integer-predecessor-cause (λ::int. True)
proof -
  have existence:
    cohoe-existence-premise (λ(a :: int). a = 0)
  by (simp add: cohoe-existence-premise-def)
  have essential:
    cohoe-essential-ordering-premise (UNIV :: int set)
    integer-predecessor-cause (λ::int. True) (λa::int. a = 0)
  unfolding cohoe-essential-ordering-premise-def
proof (intro allI impI)
  fix a :: int
  assume a = 0
  then show essentially-ordered-series UNIV integer-predecessor-cause
    (λ-. True) a
  using integer-predecessor-is-essentially-ordered by simp
qed
have full:
  cohoe-full-foundation-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-empty-grounding
  unfolding cohoe-full-foundation-premise-def
  by (rule cohoe-empty-full-foundation)
have not-endpoint:
  ¬ cohoe-singleton-endpoint-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-empty-grounding
  by (rule cohoe-empty-endpoint-fails)
have nonempty:
  cohoe-full-nonempty-premise (UNIV :: int set) cohoe-empty-grounding
  using cohoe-empty-nonempty by blast
have ancestry:
  cohoe-full-ancestry-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-empty-grounding
  by (rule cohoe-empty-ancestry)
have not-global:
  ¬ cohoe-global-non-derivative-conclusion (UNIV :: int set)
  integer-predecessor-cause (λ::int. True)
  by (simp add: cohoe-global-non-derivative-conclusion-def)
show ?thesis
  using existence essential full not-endpoint nonempty ancestry
  not-global by blast
qed

```

lemma *cohoe-empty-or-singleton-full-foundation:*
full-foundation-on (UNIV :: int set) integer-predecessor-cause
cohoe-integer-empty-or-singleton-grounding UNIV

```

proof -
  have foundation:
    full-foundation-for (UNIV :: int set) integer-predecessor-cause
    cohoe-integer-empty-or-singleton-grounding UNIV {}
  unfolding full-foundation-for-def
  by (simp add: full-grounding-on-def
    cohoe-integer-empty-or-singleton-grounding-def)
  then show ?thesis
  by (rule full-foundation-onI)
qed

```

lemma *cohoe-empty-or-singleton-endpoint:*
cohoe-singleton-endpoint-premise (UNIV :: int set)
integer-predecessor-cause cohoe-integer-empty-or-singleton-grounding

```

unfolding cohoe-singleton-endpoint-premise-def
proof (intro allI impI)
  fix x a :: int
  assume x ∈ (UNIV :: int set) a ∈ (UNIV :: int set)
  and Dxa: integer-predecessor-cause x a
  have full:
    full-grounding-on (UNIV :: int set)
    cohoe-integer-empty-or-singleton-grounding {x} a
  by (simp add: full-grounding-on-def
    cohoe-integer-empty-or-singleton-grounding-def Dxa)
  show partial-grounding-on (UNIV :: int set)
    cohoe-integer-empty-or-singleton-grounding x a
  using partial-grounding-onI[OF full] by simp
qed

```

```

lemma cohoe-empty-or-singleton-nonempty-fails:
  ¬ cohoe-full-nonempty-premise (UNIV :: int set)
  cohoe-integer-empty-or-singleton-grounding
proof
  assume nonempty:
    cohoe-full-nonempty-premise (UNIV :: int set)
    cohoe-integer-empty-or-singleton-grounding
  have empty-full:
    full-grounding-on (UNIV :: int set)
    cohoe-integer-empty-or-singleton-grounding {} 0
  by (simp add: full-grounding-on-def
    cohoe-integer-empty-or-singleton-grounding-def)
  have nonempty-all:
    ∀ S a. (full-grounding-on (UNIV :: int set)
      cohoe-integer-empty-or-singleton-grounding S a ⟶
      S ≠ {})
  using nonempty
  by (simp only: cohoe-full-nonempty-premise-def)
  have false-nonempty: ({} :: int set) ≠ {}
  using nonempty-all empty-full by blast
  show False
  using false-nonempty by simp
qed

```

```

lemma cohoe-empty-or-singleton-ancestry:
  cohoe-full-ancestry-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-integer-empty-or-singleton-grounding
unfolding cohoe-full-ancestry-premise-def
proof (intro allI impI impI)
  fix S :: int set and a x :: int
  assume full:
    full-grounding-on (UNIV :: int set)
    cohoe-integer-empty-or-singleton-grounding S a
  and x ∈ S
  have G:
    S = {} ∨
    (∃ y. S = {y} ∧ integer-predecessor-cause y a)
  using full
  by (simp only: full-grounding-on-def
    cohoe-integer-empty-or-singleton-grounding-def)
  show dependency-ancestor (UNIV :: int set)
    integer-predecessor-cause x a
  proof (rule disjE[OF G])
    assume S-empty: S = {}

```

```

then show ?thesis
  using  $\langle x \in S \rangle$  by simp
next
  assume singleton:
     $\exists y. S = \{y\} \wedge \text{integer-predecessor-cause } y \ a$ 
  obtain  $y$  where  $S = \{y\}$  and  $Dya$ :
     $\text{integer-predecessor-cause } y \ a$ 
    using singleton by blast
  have  $xy: x = y$ 
    using  $\langle S = \{y\} \rangle \langle x \in S \rangle$  by simp
  show ?thesis
    using dependency-ancestor-direct[
      where  $C = UNIV :: \text{int set}$ 
      and  $D = \text{integer-predecessor-cause}$  and  $x = y$  and  $y = a$ ]
       $Dya \ xy$  by simp
qed
qed

theorem cohoe-delete-full-nonempty:
  cohoe-existence-premise  $(\lambda(a :: \text{int}). a = 0) \wedge$ 
  cohoe-essential-ordering-premise  $(UNIV :: \text{int set})$ 
  integer-predecessor-cause  $(\lambda::\text{int}. \text{True}) (\lambda a::\text{int}. a = 0) \wedge$ 
  cohoe-full-foundation-premise  $(UNIV :: \text{int set})$ 
  integer-predecessor-cause cohoe-integer-empty-or-singleton-grounding  $\wedge$ 
  cohoe-singleton-endpoint-premise  $(UNIV :: \text{int set})$ 
  integer-predecessor-cause cohoe-integer-empty-or-singleton-grounding  $\wedge$ 
   $\neg$  cohoe-full-nonempty-premise  $(UNIV :: \text{int set})$ 
  cohoe-integer-empty-or-singleton-grounding  $\wedge$ 
  cohoe-full-ancestry-premise  $(UNIV :: \text{int set})$  integer-predecessor-cause
  cohoe-integer-empty-or-singleton-grounding  $\wedge$ 
   $\neg$  cohoe-global-non-derivative-conclusion  $(UNIV :: \text{int set})$ 
  integer-predecessor-cause  $(\lambda::\text{int}. \text{True})$ 
proof –
  have existence:
    cohoe-existence-premise  $(\lambda(a :: \text{int}). a = 0)$ 
    by (simp add: cohoe-existence-premise-def)
  have essential:
    cohoe-essential-ordering-premise  $(UNIV :: \text{int set})$ 
    integer-predecessor-cause  $(\lambda::\text{int}. \text{True}) (\lambda a::\text{int}. a = 0)$ 
    unfolding cohoe-essential-ordering-premise-def
  proof (intro allI impI)
    fix  $a :: \text{int}$ 
    assume  $a = 0$ 
    then show essentially-ordered-series  $UNIV$  integer-predecessor-cause
       $(\lambda-. \text{True}) \ a$ 
      using integer-predecessor-is-essentially-ordered by simp
    qed
  have full:
    cohoe-full-foundation-premise  $(UNIV :: \text{int set})$ 
    integer-predecessor-cause cohoe-integer-empty-or-singleton-grounding
    unfolding cohoe-full-foundation-premise-def
    by (rule cohoe-empty-or-singleton-full-foundation)
  have endpoint:
    cohoe-singleton-endpoint-premise  $(UNIV :: \text{int set})$ 
    integer-predecessor-cause cohoe-integer-empty-or-singleton-grounding
    by (rule cohoe-empty-or-singleton-endpoint)
  have not-nonempty:
     $\neg$  cohoe-full-nonempty-premise  $(UNIV :: \text{int set})$ 
    cohoe-integer-empty-or-singleton-grounding

```

```

  by (rule cohoe-empty-or-singleton-nonempty-fails)
have ancestry:
  cohoe-full-ancestry-premise (UNIV :: int set) integer-predecessor-cause
  cohoe-integer-empty-or-singleton-grounding
  by (rule cohoe-empty-or-singleton-ancestry)
have not-global:
  ¬ cohoe-global-non-derivative-conclusion (UNIV :: int set)
  integer-predecessor-cause (λ::int. True)
  by (simp add: cohoe-global-non-derivative-conclusion-def)
show ?thesis
  using existence essential full endpoint not-nonempty ancestry
  not-global by blast
qed

definition cohoe-cycle-dependency :: nat ⇒ nat ⇒ bool
where
  cohoe-cycle-dependency x y ⟷
    (x = 0 ∧ y = 1) ∨ (x = 1 ∧ y = 0)

definition cohoe-cycle-bogus-grounding :: nat set ⇒ nat ⇒ bool
where
  cohoe-cycle-bogus-grounding S a ⟷
    (S = {2} ∧ a ∈ {0, 1}) ∨
    (∃ x. S = {x} ∧ cohoe-cycle-dependency x a)

lemma cohoe-cycle-ancestor-source:
  assumes dependency-ancestor {0, 1, 2} cohoe-cycle-dependency x (1 :: nat)
  shows x ∈ {0, 1}
proof –
  have path:
    (dependency-on {0, 1, 2} cohoe-cycle-dependency)** x (1 :: nat)
  using assms by (simp add: dependency-ancestor-def)
  have edge-source:
    ∧ u v. dependency-on {0, 1, 2} cohoe-cycle-dependency u v
    ⇒ u ∈ {0, 1}
  by (auto simp: dependency-on-def cohoe-cycle-dependency-def)
show ?thesis
using path
proof (induction rule: converse-rtranclp-induct)
  case base
  show ?case by simp
next
  case step
  then show ?case
    using edge-source[OF step.hyps(1)] by simp
qed
qed

lemma cohoe-cycle-two-not-ancestor:
  ¬ dependency-ancestor {0, 1, 2} cohoe-cycle-dependency
    (2 :: nat) 1
proof
  assume ancestor:
    dependency-ancestor {0, 1, 2} cohoe-cycle-dependency (2 :: nat) 1
  have path:
    (dependency-on {0, 1, 2} cohoe-cycle-dependency)** (2 :: nat) 1
  using ancestor by (simp add: dependency-ancestor-def)
show False
proof (rule converse-rtranclpE[OF path])

```

```

assume 2 = (1 :: nat)
then show False by simp
next
  fix y
  assume edge:
    dependency-on {0, 1, 2} cohoe-cycle-dependency (2 :: nat) y
  show False
    using edge by (auto simp: dependency-on-def
      cohoe-cycle-dependency-def)
qed
qed

lemma cohoe-cycle-full-foundation:
  full-foundation-on {0, 1, 2} cohoe-cycle-dependency
  cohoe-cycle-bogus-grounding {0, 1, 2}
proof –
  have support:
     $\forall a \in (\{0, 1, 2\} :: \text{nat set}).$ 
    grounded-in {0, 1, 2}
    (partial-grounding-on {0, 1, 2}
      cohoe-cycle-bogus-grounding) a  $\longrightarrow$ 
    ( $\exists S. \text{full-grounding-on } \{0, 1, 2\}$ 
      cohoe-cycle-bogus-grounding S a  $\wedge S \subseteq \{2\}$ )
proof (intro ballI impI)
  fix a :: nat
  assume a  $\in (\{0, 1, 2\} :: \text{nat set})$ 
  and grounded:
    grounded-in {0, 1, 2}
    (partial-grounding-on {0, 1, 2}
      cohoe-cycle-bogus-grounding) a
  obtain y where y  $\in (\{0, 1, 2\} :: \text{nat set})$ 
  and partial:
    partial-grounding-on {0, 1, 2}
    cohoe-cycle-bogus-grounding y a
  using grounded by (auto simp: grounded-in-def)
  obtain S where full-S:
    full-grounding-on {0, 1, 2}
    cohoe-cycle-bogus-grounding S a
  and y  $\in S$ 
  using partial by (auto simp: partial-grounding-on-def)
  have G:
    (S = {2}  $\wedge$  a  $\in$  {0, 1})  $\vee$ 
    ( $\exists x. S = \{x\} \wedge \text{cohoe-cycle-dependency } x \ a$ )
  using full-S
  by (simp only: full-grounding-on-def
    cohoe-cycle-bogus-grounding-def)
  have a01: a  $\in (\{0, 1\} :: \text{nat set})$ 
  proof (rule disjE[OF G])
    assume first: S = {2}  $\wedge$  a  $\in$  {0, 1}
    then show ?thesis by simp
  next
    assume second:
       $\exists x. S = \{x\} \wedge \text{cohoe-cycle-dependency } x \ a$ 
    obtain x where S = {x}
    and Dxa: cohoe-cycle-dependency x a
    using second by blast
    have cycle-values:
      (x = 0  $\wedge$  a = 1)  $\vee$  (x = 1  $\wedge$  a = 0)
    using Dxa by (simp add: cohoe-cycle-dependency-def)

```

```

    show ?thesis
    using cycle-values by blast
qed
have aC:  $a \in (\{0, 1, 2\} :: \text{nat set})$ 
    using a01 by auto
have full2:
    full-grounding-on  $\{0, 1, 2\}$ 
    cohoe-cycle-bogus-grounding  $\{2\}$  a
    unfolding full-grounding-on-def
    using aC a01 by (auto simp: cohoe-cycle-bogus-grounding-def)
show  $\exists S. \text{full-grounding-on } \{0, 1, 2\}$ 
    cohoe-cycle-bogus-grounding  $S$  a  $\wedge S \subseteq \{2\}$ 
    using full2 by blast
qed
have foundation:
    full-foundation-for  $\{0, 1, 2\}$  cohoe-cycle-dependency
    cohoe-cycle-bogus-grounding  $\{0, 1, 2\}$   $\{2\}$ 
proof -
    have base:
         $\{2\} \subseteq (\{0, 1, 2\} :: \text{nat set}) \wedge$ 
         $\{0, 1, 2\} \subseteq (\{0, 1, 2\} :: \text{nat set}) \wedge$ 
         $(\forall f \in \{2\}. \text{independent-in } \{0, 1, 2\} \text{ cohoe-cycle-dependency } f)$ 
    by (simp add: independent-in-def cohoe-cycle-dependency-def)
    show ?thesis
    unfolding full-foundation-for-def
    using base support by blast
qed
then show ?thesis
    by (rule full-foundation-onI)
qed

lemma cohoe-cycle-endpoint:
    cohoe-singleton-endpoint-premise  $\{0, 1, 2\}$  cohoe-cycle-dependency
    cohoe-cycle-bogus-grounding
    unfolding cohoe-singleton-endpoint-premise-def
proof (intro allI impI)
    fix x a :: nat
    assume  $x \in \{0, 1, 2\}$   $a \in \{0, 1, 2\}$ 
    and Dxa: cohoe-cycle-dependency x a
    have cycle-values:
         $(x = 0 \wedge a = 1) \vee (x = 1 \wedge a = 0)$ 
    using Dxa by (simp add: cohoe-cycle-dependency-def)
    have full:
        full-grounding-on  $\{0, 1, 2\}$  cohoe-cycle-bogus-grounding  $\{x\}$  a
    unfolding full-grounding-on-def
    using cycle-values by (auto simp: cohoe-cycle-bogus-grounding-def
        cohoe-cycle-dependency-def)
    show partial-grounding-on  $\{0, 1, 2\}$  cohoe-cycle-bogus-grounding x a
    using partial-grounding-onI[OF full] by simp
qed

lemma cohoe-cycle-nonempty:
    cohoe-full-nonempty-premise  $\{0, 1, 2\}$  cohoe-cycle-bogus-grounding
    unfolding cohoe-full-nonempty-premise-def
proof (intro allI impI)
    fix S :: nat set and a :: nat
    assume full:
        full-grounding-on  $\{0, 1, 2\}$  cohoe-cycle-bogus-grounding S a

```

```

have G:
  (S = {2} ∧ a ∈ {0, 1}) ∨
  (∃ x. S = {x} ∧ cohoe-cycle-dependency x a)
using full
by (simp only: full-grounding-on-def cohoe-cycle-bogus-grounding-def)
show S ≠ {}
proof (rule disjE[OF G])
  assume first: S = {2} ∧ a ∈ {0, 1}
  then show ?thesis by simp
next
  assume second:
    ∃ x. S = {x} ∧ cohoe-cycle-dependency x a
  then obtain x where S = {x} by blast
  then show ?thesis by simp
qed
qed

```

lemma *cohoe-cycle-ancestry-fails:*

```

¬ cohoe-full-ancestry-premise {0, 1, 2} cohoe-cycle-dependency
  cohoe-cycle-bogus-grounding
proof
  assume ancestry:
    cohoe-full-ancestry-premise {0, 1, 2} cohoe-cycle-dependency
    cohoe-cycle-bogus-grounding
  have full:
    full-grounding-on {0, 1, 2} cohoe-cycle-bogus-grounding {2} 1
  by (simp add: full-grounding-on-def
    cohoe-cycle-bogus-grounding-def)
  have ancestor:
    dependency-ancestor {0, 1, 2} cohoe-cycle-dependency (2 :: nat) 1
proof -
  have ancestry-all:
    ∀ S a x. full-grounding-on {0, 1, 2}
      cohoe-cycle-bogus-grounding S a ⟶ x ∈ S ⟶
      dependency-ancestor {0, 1, 2} cohoe-cycle-dependency x a
  using ancestry
  by (simp only: cohoe-full-ancestry-premise-def)
  show ?thesis
  using ancestry-all full by blast
qed
show False
using cohoe-cycle-two-not-ancestor ancestor by blast
qed

```

lemma *cohoe-cycle-essential-ordering:*

```

cohoe-essential-ordering-premise {0, 1, 2} cohoe-cycle-dependency
  (λ::nat. True) (λa::nat. a = 1)
unfolding cohoe-essential-ordering-premise-def
proof (intro allI impI)
  fix a :: nat
  assume a = 1
  have grounded1:
    grounded-in {0, 1, 2} cohoe-cycle-dependency (1 :: nat)
  by (simp add: grounded-in-def cohoe-cycle-dependency-def)
  have ancestors-grounded:
    ∀ x. dependency-ancestor {0, 1, 2}
      cohoe-cycle-dependency x (1 :: nat) ⟶
      grounded-in {0, 1, 2} cohoe-cycle-dependency x
proof (intro allI impI)

```

```

fix x
assume ancestor:
  dependency-ancestor {0, 1, 2} cohoe-cycle-dependency x (1 :: nat)
have x01: x ∈ {0, 1}
  by (rule cohoe-cycle-ancestor-source[OF ancestor])
show grounded-in {0, 1, 2} cohoe-cycle-dependency x
  using x01 by (auto simp: grounded-in-def cohoe-cycle-dependency-def)
qed
show essentially-ordered-series {0, 1, 2} cohoe-cycle-dependency
  (λ::nat. True) a
  using ⟨a = 1⟩ grounded1 ancestors-grounded
  by (auto simp: essentially-ordered-series-def)
qed

```

theorem cohoe-delete-full-ancestry:

```

cohoe-existence-premise (λ(a :: nat). a = 1) ∧
cohoe-essential-ordering-premise {0, 1, 2} cohoe-cycle-dependency
  (λ::nat. True) (λa::nat. a = 1) ∧
cohoe-full-foundation-premise {0, 1, 2} cohoe-cycle-dependency
  cohoe-cycle-bogus-grounding ∧
cohoe-singleton-endpoint-premise {0, 1, 2} cohoe-cycle-dependency
  cohoe-cycle-bogus-grounding ∧
cohoe-full-nonempty-premise {0, 1, 2} cohoe-cycle-bogus-grounding ∧
¬ cohoe-full-ancestry-premise {0, 1, 2} cohoe-cycle-dependency
  cohoe-cycle-bogus-grounding ∧
¬ cohoe-global-non-derivative-conclusion {0, 1, 2}
  cohoe-cycle-dependency (λ::nat. True)
proof –
  have existence:
    cohoe-existence-premise (λ(a :: nat). a = 1)
  by (simp add: cohoe-existence-premise-def)
  have essential:
    cohoe-essential-ordering-premise {0, 1, 2}
    cohoe-cycle-dependency (λ::nat. True) (λa::nat. a = 1)
  by (rule cohoe-cycle-essential-ordering)
  have full:
    cohoe-full-foundation-premise {0, 1, 2} cohoe-cycle-dependency
    cohoe-cycle-bogus-grounding
  unfolding cohoe-full-foundation-premise-def
  by (rule cohoe-cycle-full-foundation)
  have endpoint:
    cohoe-singleton-endpoint-premise {0, 1, 2} cohoe-cycle-dependency
    cohoe-cycle-bogus-grounding
  by (rule cohoe-cycle-endpoint)
  have nonempty:
    cohoe-full-nonempty-premise {0, 1, 2}
    cohoe-cycle-bogus-grounding
  by (rule cohoe-cycle-nonempty)
  have not-ancestry:
    ¬ cohoe-full-ancestry-premise {0, 1, 2} cohoe-cycle-dependency
    cohoe-cycle-bogus-grounding
  by (rule cohoe-cycle-ancestry-fails)
  have not-global:
    ¬ cohoe-global-non-derivative-conclusion {0, 1, 2}
    cohoe-cycle-dependency (λ::nat. True)
  by (simp add: cohoe-global-non-derivative-conclusion-def)
show ?thesis
  using existence essential full endpoint nonempty not-ancestry
  not-global by blast

```

qed

theorem *cohoe-delete-full-foundation:*

cohoe-existence-premise $(\lambda(a :: \text{int}). a = 0) \wedge$
cohoe-essential-ordering-premise $(UNIV :: \text{int set})$
integer-predecessor-cause $(\lambda :: \text{int}. \text{True}) (\lambda a :: \text{int}. a = 0) \wedge$
 $\neg$ *cohoe-full-foundation-premise* $(UNIV :: \text{int set})$
integer-predecessor-cause
cohoe-integer-singleton-grounding $\wedge$
cohoe-singleton-endpoint-premise $(UNIV :: \text{int set})$ *integer-predecessor-cause*
cohoe-integer-singleton-grounding $\wedge$
cohoe-full-nonempty-premise $(UNIV :: \text{int set})$ *cohoe-integer-singleton-grounding* $\wedge$
cohoe-full-ancestry-premise $(UNIV :: \text{int set})$ *integer-predecessor-cause*
cohoe-integer-singleton-grounding $\wedge$
 $\neg$ *cohoe-global-non-derivative-conclusion* $(UNIV :: \text{int set})$
integer-predecessor-cause $(\lambda :: \text{int}. \text{True})$

proof –

have *existence:*

cohoe-existence-premise $(\lambda(a :: \text{int}). a = 0)$

by (*simp add: cohoe-existence-premise-def*)

have *essential:*

cohoe-essential-ordering-premise $(UNIV :: \text{int set})$

integer-predecessor-cause $(\lambda -. \text{True}) (\lambda a. a = 0)$

unfolding *cohoe-essential-ordering-premise-def*

proof (*intro allI impI*)

fix $a :: \text{int}$

assume $a = 0$

then show *essentially-ordered-series UNIV integer-predecessor-cause*
 $(\lambda -. \text{True}) a$

using *integer-predecessor-is-essentially-ordered* **by** *simp*

qed

have *endpoint:*

cohoe-singleton-endpoint-premise $(UNIV :: \text{int set})$

integer-predecessor-cause cohoe-integer-singleton-grounding

by (*rule cohoe-integer-singleton-endpoint*)

have *nonempty:*

cohoe-full-nonempty-premise $(UNIV :: \text{int set})$

cohoe-integer-singleton-grounding

by (*rule cohoe-integer-singleton-nonempty*)

have *ancestry:*

cohoe-full-ancestry-premise $(UNIV :: \text{int set})$

integer-predecessor-cause cohoe-integer-singleton-grounding

by (*rule cohoe-integer-singleton-ancestry*)

have *not-full:*

$\neg$ *cohoe-full-foundation-premise* $(UNIV :: \text{int set})$

integer-predecessor-cause cohoe-integer-singleton-grounding

proof

assume *full-premise:*

cohoe-full-foundation-premise $(UNIV :: \text{int set})$

integer-predecessor-cause cohoe-integer-singleton-grounding

have *full:*

full-foundation-on $(UNIV :: \text{int set})$ *integer-predecessor-cause*

cohoe-integer-singleton-grounding UNIV

using *full-premise* **by** (*simp only: cohoe-full-foundation-premise-def*)

have *endpoint-all:*

$\bigwedge x a. x \in (UNIV :: \text{int set}) \implies$

$a \in (UNIV :: \text{int set}) \implies$

integer-predecessor-cause $x a \implies$

partial-grounding-on $(UNIV :: \text{int set})$

```

    cohoe-integer-singleton-grounding  $x$   $a$ 
using endpoint
by (simp only: cohoe-singleton-endpoint-premise-def)
have nonempty-all:
   $\forall S$   $a$ . (full-grounding-on ( $UNIV :: int$  set)
    cohoe-integer-singleton-grounding  $S$   $a \longrightarrow S \neq \{\}$ )
using nonempty
by (simp only: cohoe-full-nonempty-premise-def)
have ancestry-all:
   $\forall S$   $a$   $x$ . (full-grounding-on ( $UNIV :: int$  set)
    cohoe-integer-singleton-grounding  $S$   $a \longrightarrow x \in S \longrightarrow$ 
    dependency-ancestor ( $UNIV :: int$  set)
    integer-predecessor-cause  $x$   $a$ )
using ancestry
by (simp only: cohoe-full-ancestry-premise-def)
have local:
  local-foundation-on ( $UNIV :: int$  set)
  integer-predecessor-cause  $UNIV$ 
using full-foundation-implies-local-foundation[OF full]
  endpoint-all nonempty-all ancestry-all by blast
show False
using integer-predecessor-not-local-foundation local by blast
qed
have not-global:
   $\neg$  cohoe-global-non-derivative-conclusion ( $UNIV :: int$  set)
  integer-predecessor-cause ( $\lambda$ -. True)
by (simp add: cohoe-global-non-derivative-conclusion-def)
have h-ancestry:
  cohoe-full-ancestry-premise ( $UNIV :: int$  set)
  integer-predecessor-cause cohoe-integer-singleton-grounding  $\wedge$ 
   $\neg$  cohoe-global-non-derivative-conclusion ( $UNIV :: int$  set)
  integer-predecessor-cause ( $\lambda$ -. True)
using ancestry not-global by (rule conjI)
have h-nonempty:
  cohoe-full-nonempty-premise ( $UNIV :: int$  set)
  cohoe-integer-singleton-grounding  $\wedge$ 
  cohoe-full-ancestry-premise ( $UNIV :: int$  set)
  integer-predecessor-cause cohoe-integer-singleton-grounding  $\wedge$ 
   $\neg$  cohoe-global-non-derivative-conclusion ( $UNIV :: int$  set)
  integer-predecessor-cause ( $\lambda$ -. True)
using nonempty h-ancestry by (rule conjI)
have h-endpoint:
  cohoe-singleton-endpoint-premise ( $UNIV :: int$  set)
  integer-predecessor-cause cohoe-integer-singleton-grounding  $\wedge$ 
  cohoe-full-nonempty-premise ( $UNIV :: int$  set)
  cohoe-integer-singleton-grounding  $\wedge$ 
  cohoe-full-ancestry-premise ( $UNIV :: int$  set)
  integer-predecessor-cause cohoe-integer-singleton-grounding  $\wedge$ 
   $\neg$  cohoe-global-non-derivative-conclusion ( $UNIV :: int$  set)
  integer-predecessor-cause ( $\lambda$ -. True)
using endpoint h-nonempty by (rule conjI)
have h-full:
   $\neg$  cohoe-full-foundation-premise ( $UNIV :: int$  set)
  integer-predecessor-cause cohoe-integer-singleton-grounding  $\wedge$ 
  cohoe-singleton-endpoint-premise ( $UNIV :: int$  set)
  integer-predecessor-cause cohoe-integer-singleton-grounding  $\wedge$ 
  cohoe-full-nonempty-premise ( $UNIV :: int$  set)
  cohoe-integer-singleton-grounding  $\wedge$ 
  cohoe-full-ancestry-premise ( $UNIV :: int$  set)

```

```

    integer-predecessor-cause cohoe-integer-singleton-grounding ∧
    ¬ cohoe-global-non-derivative-conclusion (UNIV :: int set)
    integer-predecessor-cause (λ-. True)
using not-full h-endpoint by (rule conjI)
have h-essential:
  cohoe-essential-ordering-premise (UNIV :: int set)
  integer-predecessor-cause (λ-. True) (λa. a = 0) ∧
  ¬ cohoe-full-foundation-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-integer-singleton-grounding ∧
  cohoe-singleton-endpoint-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-integer-singleton-grounding ∧
  cohoe-full-nonempty-premise (UNIV :: int set)
  cohoe-integer-singleton-grounding ∧
  cohoe-full-ancestry-premise (UNIV :: int set)
  integer-predecessor-cause cohoe-integer-singleton-grounding ∧
  ¬ cohoe-global-non-derivative-conclusion (UNIV :: int set)
  integer-predecessor-cause (λ-. True)
using essential h-full by (rule conjI)
show ?thesis
proof (rule conjI)
  show cohoe-existence-premise (λ(a :: int). a = 0)
  using existence .
  show cohoe-essential-ordering-premise (UNIV :: int set)
    integer-predecessor-cause
    (λ-::int. True) (λa::int. a = 0) ∧
  ¬ cohoe-full-foundation-premise (UNIV :: int set)
    integer-predecessor-cause
    cohoe-integer-singleton-grounding ∧
  cohoe-singleton-endpoint-premise (UNIV :: int set)
    integer-predecessor-cause
    cohoe-integer-singleton-grounding ∧
  cohoe-full-nonempty-premise (UNIV :: int set)
    cohoe-integer-singleton-grounding ∧
  cohoe-full-ancestry-premise (UNIV :: int set)
    integer-predecessor-cause
    cohoe-integer-singleton-grounding ∧
  ¬ cohoe-global-non-derivative-conclusion (UNIV :: int set)
    integer-predecessor-cause (λ-::int. True)
  using h-essential .
qed
qed

end
theory Finite-Foundation-Analysis
imports Finite-Foundation-Checker Research-Results
begin

```

39 Cycle Certificates

The existing finite checker certifies the local foundation predicate. This cycle analyzer exposes the other obstruction that matters for the binary lattice: a concrete carrier vertex together with a bounded return path. The internal return-segment bound is the length of the supplied vertex list; the emitted closed certificate prepends one endpoint and is therefore bounded by one greater. The path lemmas below prove completeness for carrier-closed finite relations and expose the resulting payload bound.

```

fun finite-path-to ::
  nat ⇒ 'a list ⇒ ('a ⇒ 'a ⇒ bool) ⇒

```

```

'a ⇒ 'a ⇒ 'a list option
where
  finite-path-to 0 V R x y =
    (if x = y then Some [] else None)
| finite-path-to (Suc n) V R x y =
    (if x = y then Some []
     else
      (case List.find
        (λz. R x z ∧ finite-reach n V R z y) V of
        None ⇒ None
        | Some z ⇒
          (case finite-path-to n V R z y of
            None ⇒ None
            | Some ys ⇒ Some (z # ys))))))

lemma finite-path-to-sound:
  finite-path-to n V R x y = Some ys ⇒
    rtrancl-path R x ys y
proof (induction n arbitrary: x y ys)
case 0
have base: x = y ∧ ys = []
  using ⟨finite-path-to 0 V R x y = Some ys⟩
  by (simp add: finite-path-to.simps split: if_splits)
have xy: x = y
  using base by simp
have ys0: ys = []
  using base by simp
show ?case
  unfolding xy ys0
proof (rule rtrancl-path.base)
qed
next
case (Suc n)
then show ?case
proof (cases x = y)
case True
have ys0: ys = []
  using Suc.prem1 True by (simp add: finite-path-to.simps)
show ?thesis
  unfolding True ys0
proof (rule rtrancl-path.base)
qed
next
case False
obtain z where find:
  List.find
    (λz. R x z ∧ finite-reach n V R z y) V = Some z
and path: finite-path-to n V R z y = Some (tl ys)
  using Suc.prem1 False
  by (auto split: option.splits)
have pred:
  R x z ∧ finite-reach n V R z y
  using find by (auto simp add: List.find-Some-iff)
have path': rtrancl-path R z (tl ys) y
  using Suc.IH[OF path] .
have z-head: ys = z # tl ys
  using Suc.prem1 False find path by (auto split: option.splits)
obtain ys' where ys-eq: ys = z # ys'
  using z-head by (cases ys) auto

```

```

have tail-eq: tl ys = ys'
  using ys-eq by simp
have path-tail: rtrancl-path R z ys' y
  using path' tail-eq by simp
show ?thesis
  unfolding ys-eq
proof (rule rtrancl-path.step)
  show R x z
    using pred by (rule conjunct1)
  show rtrancl-path R z ys' y
    using path-tail .
qed
qed
qed

lemma finite-path-to-complete:
  finite-reach n V R x y  $\implies$ 
  ( $\exists$  ys. finite-path-to n V R x y = Some ys)
proof (induction n arbitrary: x y)
  case 0
  then show ?case
    by (auto intro: exI[of - []])
next
  case (Suc n)
  show ?case
  proof (cases x = y)
    case True
    have eq: finite-path-to (Suc n) V R x y = Some []
      using True by (simp add: finite-path-to.simps)
    show ?thesis
      using eq by (rule exI)
  next
    case False
    obtain z where zV: z  $\in$  set V and edge: R x z
      and reach: finite-reach n V R z y
      using Suc.premys False
      by (auto simp: finite-reach.simps finite-list-ex-iff)
    have find-ne:
      List.find
      ( $\lambda w. R x w \wedge$  finite-reach n V R w y) V  $\neq$  None
    proof
      assume none:
        List.find
        ( $\lambda w. R x w \wedge$  finite-reach n V R w y) V = None
      have no-candidate:
         $\neg (\exists w \in \text{set } V. R x w \wedge \text{finite-reach } n \text{ V R } w y)$ 
      using List.find-None-iff[of  $\lambda w. R x w \wedge$ 
        finite-reach n V R w y V] none by blast
      show False
        using no-candidate zV edge reach by blast
    qed
    obtain w where find:
      List.find
      ( $\lambda w. R x w \wedge$  finite-reach n V R w y) V = Some w
    using find-ne by (cases List.find
      ( $\lambda w. R x w \wedge$  finite-reach n V R w y) V) auto
    have pred: R x w  $\wedge$  finite-reach n V R w y
      using find by (auto simp add: List.find-Some-iff)

```

```

have reach-w: finite-reach n V R w y
  using pred by (rule conjunct2)
obtain ys where path: finite-path-to n V R w y = Some ys
  using Suc.IH[OF reach-w] by blast
have result:
  finite-path-to (Suc n) V R x y = Some (w # ys)
  using False find path by (simp add: finite-path-to.simps)
show ?thesis
  using result by (rule exI)
qed
qed

lemma finite-path-to-obtain:
  assumes finite-reach n V R x y
  obtains ys where finite-path-to n V R x y = Some ys
  using finite-path-to-complete[OF assms] by blast

lemma finite-path-to-length-le:
  finite-path-to n V R x y = Some ys  $\implies$  length ys  $\leq$  n
proof (induction n arbitrary: x y ys)
  case 0
  then show ?case
    by (simp add: finite-path-to.simps split: if-splits)
next
  case (Suc n)
  show ?case
  proof (cases x = y)
    case True
    then show ?thesis
      using Suc.prem by (simp add: finite-path-to.simps)
  next
    case False
    obtain z where find:
      List.find
        ( $\lambda z. R x z \wedge$  finite-reach n V R z y) V = Some z
      and path: finite-path-to n V R z y = Some (tl ys)
      using Suc.prem False
      by (auto split: option.splits)
    have z-head: ys = z # tl ys
      using Suc.prem False find path by (auto split: option.splits)
    have tail-bound: length (tl ys)  $\leq$  n
      using Suc.IH[OF path] .
    have nonempty: ys  $\neq$  []
      using z-head by (cases ys) auto
    have length ys = Suc (length (tl ys))
      using nonempty by simp
    then show ?thesis
      using tail-bound by simp
  qed
qed
qed

definition finite-cycle-at ::
  finite-dependency-model  $\Rightarrow$  nat  $\Rightarrow$  bool
where
  finite-cycle-at M x  $\longleftrightarrow$ 
    x  $\in$  finite-model-carrier M  $\wedge$ 
    finite-list-ex
      ( $\lambda y. \text{finite-model-relation } M x y \wedge$ 
        finite-reach (length (finite-vertices M))

```

(finite-vertices M) (finite-model-relation M) y x)
(finite-vertices M)

definition *finite-cycle-path-at* ::

finite-dependency-model $\Rightarrow$ *nat* $\Rightarrow$ *nat list option*

where

finite-cycle-path-at M x =
(if x $\in$ *finite-model-carrier* M then
(case List.find
(λ y. *finite-model-relation* M x y $\wedge$
finite-reach (length (finite-vertices M))
(finite-vertices M) (finite-model-relation M) y x)
(finite-vertices M) of
None $\Rightarrow$ None
| Some y $\Rightarrow$
(case *finite-path-to* (length (finite-vertices M))
(finite-vertices M) (finite-model-relation M) y x of
None $\Rightarrow$ None
| Some ys $\Rightarrow$ Some (y # ys)))
else None)

lemma *finite-cycle-path-at-sound*:

assumes *finite-cycle-path-at* M x = Some xs

shows x $\in$ *finite-model-carrier* M

and xs $\neq$ []

and *rtrancl-path* (*finite-model-relation* M) x xs x

and (*finite-model-relation* M)⁺⁺ x x

proof –

have xC: x $\in$ *finite-model-carrier* M

using *assms* **by** (auto simp: *finite-cycle-path-at-def*
split: *if-splits option.splits*)

obtain y **where** *find*:

List.find
(λ y. *finite-model-relation* M x y $\wedge$
finite-reach (length (finite-vertices M))
(finite-vertices M) (finite-model-relation M) y x)
(finite-vertices M) = Some y

and *path*:

finite-path-to (length (finite-vertices M))
(finite-vertices M) (finite-model-relation M) y x = Some (tl xs)

and *xs-head*: xs = y # tl xs

using *assms* xC

by (auto simp: *finite-cycle-path-at-def*
split: *if-splits option.splits*)

have *pred*:

finite-model-relation M x y $\wedge$
finite-reach (length (finite-vertices M))
(finite-vertices M) (finite-model-relation M) y x
using *find* **by** (auto simp add: List.find-Some-iff)

have *edge*: *finite-model-relation* M x y

using *pred* **by** (rule *conjunct1*)

have *bounded*:

finite-reach (length (finite-vertices M))
(finite-vertices M) (finite-model-relation M) y x
using *pred* **by** (rule *conjunct2*)

have *path-tail*:

rtrancl-path (*finite-model-relation* M) y (tl xs) x
using *finite-path-to-sound*[OF *path*] .

obtain ys **where** *xs-eq*: xs = y # ys

```

    using xs-head by (cases xs) auto
have tail-eq: tl xs = ys
    using xs-eq by simp
have path-tail':
    rtrancl-path (finite-model-relation M) y ys x
    using path-tail tail-eq by simp
have path-full':
    rtrancl-path (finite-model-relation M) x (y # ys) x
proof (rule rtrancl-path.step)
    show finite-model-relation M x y
        using edge .
    show rtrancl-path (finite-model-relation M) y ys x
        using path-tail' .
qed
have path-full:
    rtrancl-path (finite-model-relation M) x xs x
    using path-full' xs-eq by simp
have rtrancl:
    (finite-model-relation M)** y x
    using finite-reach-imp-rtranclp[OF bounded] .
have cycle:
    (finite-model-relation M)++ x x
    using rtranclp-into-tranclp2[OF edge rtrancl] .
show x ∈ finite-model-carrier M
    using xC .
show xs ≠ []
    using xs-head by (cases xs) auto
show rtrancl-path (finite-model-relation M) x xs x
    using path-full .
show (finite-model-relation M)++ x x
    using cycle .
qed

lemma finite-cycle-path-at-complete:
  assumes finite-cycle-at M x
  obtains xs where finite-cycle-path-at M x = Some xs
    and xs ≠ []
    and rtrancl-path (finite-model-relation M) x xs x
proof -
  obtain y where yV: y ∈ set (finite-vertices M)
    and pred:
      finite-model-relation M x y ∧
      finite-reach (length (finite-vertices M))
        (finite-vertices M) (finite-model-relation M) y x
    using assms
  by (auto simp: finite-cycle-at-def finite-list-ex-iff)
have find-ne:
    List.find
      (λz. finite-model-relation M x z ∧
        finite-reach (length (finite-vertices M))
          (finite-vertices M) (finite-model-relation M) z x)
      (finite-vertices M) ≠ None
proof
  assume none:
    List.find
      (λz. finite-model-relation M x z ∧
        finite-reach (length (finite-vertices M))
          (finite-vertices M) (finite-model-relation M) z x)
      (finite-vertices M) = None

```

```

have no-candidate:
  ¬ (∃ z ∈ set (finite-vertices M).
    finite-model-relation M x z ∧
    finite-reach (length (finite-vertices M))
      (finite-vertices M) (finite-model-relation M) z x)
using List.find-None-iff[of λz. finite-model-relation M x z ∧
  finite-reach (length (finite-vertices M))
    (finite-vertices M) (finite-model-relation M) z x
  finite-vertices M] none by blast
show False
  using no-candidate yV pred by blast
qed
obtain y' where find:
  List.find
    (λz. finite-model-relation M x z ∧
      finite-reach (length (finite-vertices M))
        (finite-vertices M) (finite-model-relation M) z x)
    (finite-vertices M) = Some y'
using find-ne by (cases List.find
  (λz. finite-model-relation M x z ∧
    finite-reach (length (finite-vertices M))
      (finite-vertices M) (finite-model-relation M) z x)
  (finite-vertices M)) auto
have pred':
  finite-model-relation M x y' ∧
  finite-reach (length (finite-vertices M))
    (finite-vertices M) (finite-model-relation M) y' x
using find by (auto simp add: List.find-Some-iff)
have reach':
  finite-reach (length (finite-vertices M))
    (finite-vertices M) (finite-model-relation M) y' x
using pred' by blast
have at-spec: finite-cycle-at M x
  using assms .
have xC: x ∈ finite-model-carrier M
  using at-spec by (simp add: finite-cycle-at-def)
obtain ys where path:
  finite-path-to (length (finite-vertices M))
    (finite-vertices M) (finite-model-relation M) y' x = Some ys
using finite-path-to-obtain[OF reach'] by blast
have result: finite-cycle-path-at M x = Some (y' # ys)
  using xC find path
  by (simp add: finite-cycle-path-at-def)
have sound:
  rtranc-path (finite-model-relation M) x (y' # ys) x
  using finite-cycle-path-at-sound[OF result] by blast
have exists-conclusion:
  ∃ xs. finite-cycle-path-at M x = Some xs ∧
  xs ≠ [] ∧
  rtranc-path (finite-model-relation M) x xs x
proof –
  have witness:
    finite-cycle-path-at M x = Some (y' # ys) ∧
    (y' # ys) ≠ [] ∧
    rtranc-path (finite-model-relation M) x (y' # ys) x
  using result sound by blast
  show ?thesis
  using witness by blast
qed

```

```

then obtain xs where
  finite-cycle-path-at M x = Some xs
and xs ≠ []
and rtranc-path (finite-model-relation M) x xs x
by blast
then show thesis
  by (rule that)
qed

lemma finite-cycle-path-at-length-le:
  assumes finite-cycle-path-at M x = Some xs
  shows length xs ≤ Suc (length (finite-vertices M))
proof –
  have xC: x ∈ finite-model-carrier M
    using assms by (auto simp: finite-cycle-path-at-def
      split: if-splits option.splits)
  obtain y where find:
    List.find
      ( $\lambda y. \text{finite-model-relation } M \ x \ y \wedge$ 
        finite-reach (length (finite-vertices M))
          (finite-vertices M) (finite-model-relation M) y x)
        (finite-vertices M) = Some y
  and path:
    finite-path-to (length (finite-vertices M))
      (finite-vertices M) (finite-model-relation M) y x = Some (tl xs)
  and xs-head: xs = y # tl xs
  using assms xC
  by (auto simp: finite-cycle-path-at-def
    split: if-splits option.splits)
  have tail-bound:
    length (tl xs) ≤ length (finite-vertices M)
    using finite-path-to-length-le[OF path] .
  have nonempty: xs ≠ []
    using xs-head by (cases xs) auto
  have length xs = Suc (length (tl xs))
    using nonempty by simp
  then show ?thesis
    using tail-bound by simp
qed

```

```

lemma finite-model-relation-closed:
  fixes M :: finite-dependency-model and u v :: nat
  assumes finite-model-relation M u v
  shows u ∈ set (finite-vertices M) ∧
    v ∈ set (finite-vertices M)
  using assms
  by (auto simp: finite-model-relation-def finite-model-carrier-def
    dependency-on-def)

```

```

definition finite-cycle-witness ::
  finite-dependency-model ⇒ nat option
where
  finite-cycle-witness M =
    finite-first-failure (finite-cycle-at M) (finite-vertices M)

```

```

definition finite-cycle-check ::
  finite-dependency-model ⇒ bool
where
  finite-cycle-check M ⇔

```

finite-cycle-witness $M = \text{None}$

definition *finite-cycle-failure-witness* ::
finite-dependency-model $\Rightarrow$ nat

where

finite-cycle-failure-witness $M =$
 (case *finite-cycle-witness* M of
 None $\Rightarrow 0$
 | Some $x \Rightarrow x$)

definition *finite-cycle-failure-path* ::
finite-dependency-model $\Rightarrow$ nat list

where

finite-cycle-failure-path $M =$
 (case *finite-cycle-witness* M of
 None $\Rightarrow []$
 | Some $x \Rightarrow$
 (case *finite-cycle-path-at* M x of
 None $\Rightarrow []$
 | Some $xs \Rightarrow xs$))

datatype *finite-cycle-result* =
 No-Cycle-Certificate
 | Cycle-Certificate nat nat list

definition *finite-cycle-result* ::
finite-dependency-model $\Rightarrow$ *finite-cycle-result*

where

finite-cycle-result $M =$
 (if *finite-cycle-check* M then
 No-Cycle-Certificate
 else
 Cycle-Certificate (*finite-cycle-failure-witness* M)
 (*finite-cycle-failure-path* M))

lemma *finite-first-failure-none-iff*:

finite-first-failure P $xs = \text{None} \longleftrightarrow$
 ($\forall x \in \text{set } xs. \neg P\ x$)
by (*induction* xs) (*auto split: if-splits*)

lemma *finite-cycle-witness-some*:

assumes *finite-cycle-witness* $M = \text{Some } x$
shows $x \in \text{finite-model-carrier } M$ *finite-cycle-at* M x

proof –

have *failure*:
 finite-first-failure (*finite-cycle-at* M) (*finite-vertices* M) =
 Some x
using *assms* **by** (*simp add: finite-cycle-witness-def*)
have $x \in \text{set } (\text{finite-vertices } M)$
and *finite-cycle-at* M x
using *finite-first-failure-sound*[*OF failure*] .
show $x \in \text{finite-model-carrier } M$
using $\langle x \in \text{set } (\text{finite-vertices } M) \rangle$
by (*simp add: finite-model-carrier-def*)
show *finite-cycle-at* M x
using $\langle \text{finite-cycle-at } M\ x \rangle$.

qed

lemma *finite-cycle-at-iff*:

```

finite-cycle-at  $M$   $x \longleftrightarrow$ 
 $x \in \text{finite-model-carrier } M \wedge$ 
 $(\text{dependency-on } (\text{finite-model-carrier } M)$ 
 $(\text{finite-model-immediate } M))^{++} x x$ 
proof
  assume at: finite-cycle-at  $M$   $x$ 
  have  $xC$ :  $x \in \text{finite-model-carrier } M$ 
    using at by (simp add: finite-cycle-at-def)
  obtain  $yV$  where  $yV$ :  $y \in \text{set } (\text{finite-vertices } M)$ 
    and edge: finite-model-relation  $M$   $x$   $y$ 
    and reach:
      finite-reach (length (finite-vertices  $M$ ))
      (finite-vertices  $M$ ) (finite-model-relation  $M$ )  $y$   $x$ 
    using at
    by (auto simp: finite-cycle-at-def finite-list-ex-iff)
  have rtrancl: (finite-model-relation  $M$ )**  $y$   $x$ 
    using finite-reach-imp-rtranclp[OF reach] .
  have cycle: (finite-model-relation  $M$ )++  $x$   $x$ 
    using rtranclp-into-tranclp2[OF edge rtrancl] .
  show  $x \in \text{finite-model-carrier } M \wedge$ 
     $(\text{dependency-on } (\text{finite-model-carrier } M)$ 
     $(\text{finite-model-immediate } M))^{++} x x$ 
    using  $xC$  cycle finite-model-relation-eq by simp
next
  assume cycle:
     $x \in \text{finite-model-carrier } M \wedge$ 
     $(\text{dependency-on } (\text{finite-model-carrier } M)$ 
     $(\text{finite-model-immediate } M))^{++} x x$ 
  have  $xC$ :  $x \in \text{finite-model-carrier } M$ 
    using cycle by blast
  have relation-eq:
    finite-model-relation  $M =$ 
     $\text{dependency-on } (\text{finite-model-carrier } M)$ 
     $(\text{finite-model-immediate } M)$ 
    by (rule finite-model-relation-eq)
  have cycle-relation:
    (finite-model-relation  $M$ )++  $x$   $x$ 
    using cycle relation-eq by simp
  obtain  $y$  where edge: finite-model-relation  $M$   $x$   $y$ 
    and tail: (finite-model-relation  $M$ )**  $y$   $x$ 
    using tranclpD[OF cycle-relation] by blast
  have  $yV$ :  $y \in \text{set } (\text{finite-vertices } M)$ 
    using edge finite-model-relation-closed by blast
  have relation-closed:
     $\bigwedge u v. \text{finite-model-relation } M u v \implies$ 
     $u \in \text{set } (\text{finite-vertices } M) \wedge$ 
     $v \in \text{set } (\text{finite-vertices } M)$ 
    using finite-model-relation-closed by blast
  have bounded:
    finite-reach (length (finite-vertices  $M$ ))
    (finite-vertices  $M$ ) (finite-model-relation  $M$ )  $y$   $x$ 
    using rtranclp-imp-finite-reach[
      where  $R = \text{finite-model-relation } M$ 
      and  $V = \text{finite-vertices } M$  and  $x = y$  and  $y = x$ ]
    tail  $yV$  relation-closed .
  have path:
    finite-list-ex
     $(\lambda z. \text{finite-model-relation } M x z \wedge$ 
     $\text{finite-reach } (\text{length } (\text{finite-vertices } M))$ 

```

```

      (finite-vertices M) (finite-model-relation M) z x)
    (finite-vertices M)
  using finite-list-exI[of y
    finite-vertices M
    λz. finite-model-relation M x z ∧
      finite-reach (length (finite-vertices M))
        (finite-vertices M) (finite-model-relation M) z x]
    yV edge bounded by blast
  show finite-cycle-at M x
    unfolding finite-cycle-at-def
    using xC path by blast
qed

theorem finite-cycle-check-iff:
  finite-cycle-check M ⟷
    acyclic-on (finite-model-carrier M) (finite-model-immediate M)
proof -
  have no-failure:
    finite-cycle-witness M = None ⟷
      (∀ x ∈ finite-model-carrier M. ¬ finite-cycle-at M x)
    unfolding finite-cycle-witness-def
    by (simp add: finite-first-failure-none-iff finite-model-carrier-def)
  show ?thesis
    unfolding finite-cycle-check-def acyclic-on-def
    using no-failure finite-cycle-at-iff by simp
qed

theorem finite-cycle-result-no-cycle:
  finite-cycle-result M = No-Cycle-Certificate ⟷
    acyclic-on (finite-model-carrier M) (finite-model-immediate M)
  unfolding finite-cycle-result-def
  using finite-cycle-check-iff by simp

theorem finite-cycle-result-cycle:
  assumes finite-cycle-result M = Cycle-Certificate x xs
  shows x ∈ finite-model-carrier M
    and xs ≠ []
    and rtrancl-path (finite-model-relation M) x xs x
    and (dependency-on (finite-model-carrier M)
      (finite-model-immediate M))++ x x
proof -
  have not-check: ¬ finite-cycle-check M
    using assms by (simp add: finite-cycle-result-def split: if-splits)
  have witness-ne: finite-cycle-witness M ≠ None
    using not-check by (auto simp: finite-cycle-check-def)
  obtain y where witness-y:
    finite-cycle-witness M = Some y
    using witness-ne by (cases finite-cycle-witness M) auto
  have at: finite-cycle-at M y
    using finite-cycle-witness-some[OF witness-y] by blast
  obtain ys where path-at:
    finite-cycle-path-at M y = Some ys
    and ys-nonempty: ys ≠ []
    and path:
      rtrancl-path (finite-model-relation M) y ys y
    using finite-cycle-path-at-complete[OF at] .
  have result-expanded:
    finite-cycle-result M = Cycle-Certificate y ys
    using not-check witness-y path-at

```

```

  by (simp add: finite-cycle-result-def
    finite-cycle-failure-witness-def finite-cycle-failure-path-def)
have certificate-eq:
  Cycle-Certificate x xs = Cycle-Certificate y ys
  using assms result-expanded by simp
have x-eq: x = y
  using certificate-eq by simp
have xs-eq: xs = ys
  using certificate-eq by simp
have sound-spec:
  y ∈ finite-model-carrier M ∧
  ys ≠ [] ∧
  rtrancl-path (finite-model-relation M) y ys y ∧
  (finite-model-relation M)++ y y
  using finite-cycle-path-at-sound[OF path-at] by blast
show x ∈ finite-model-carrier M
  using sound-spec x-eq by blast
show xs ≠ []
  using ys-nonempty xs-eq by simp
show rtrancl-path (finite-model-relation M) x xs x
  using path x-eq xs-eq by simp
have cycle:
  (finite-model-relation M)++ x x
  using sound-spec x-eq by blast
show (dependency-on (finite-model-carrier M)
  (finite-model-immediate M))++ x x
  using cycle finite-model-relation-eq by simp
qed

lemma finite-cycle-result-cycle-spec:
  assumes finite-cycle-result M = Cycle-Certificate x xs
  shows x ∈ finite-model-carrier M ∧
  xs ≠ [] ∧
  rtrancl-path (finite-model-relation M) x xs x ∧
  (dependency-on (finite-model-carrier M)
    (finite-model-immediate M))++ x x
  using finite-cycle-result-cycle[OF assms] by blast

lemma finite-cycle-result-cycle-length:
  assumes finite-cycle-result M = Cycle-Certificate x xs
  shows length xs ≤ Suc (length (finite-vertices M))
proof -
  have not-check: ¬ finite-cycle-check M
    using assms by (simp add: finite-cycle-result-def split: if-splits)
  have witness-ne: finite-cycle-witness M ≠ None
    using not-check by (auto simp: finite-cycle-check-def)
  obtain y where witness-y:
    finite-cycle-witness M = Some y
    using witness-ne by (cases finite-cycle-witness M) auto
  have at: finite-cycle-at M y
    using finite-cycle-witness-some[OF witness-y] by blast
  obtain ys where path-at:
    finite-cycle-path-at M y = Some ys
    using finite-cycle-path-at-complete[OF at] by blast
  have result-expanded:
    finite-cycle-result M = Cycle-Certificate y ys
    using not-check witness-y path-at
  by (simp add: finite-cycle-result-def
    finite-cycle-failure-witness-def finite-cycle-failure-path-def)

```

```

have certificate-eq:
  Cycle-Certificate x xs = Cycle-Certificate y ys
  using assms result-expanded by simp
have x-eq: x = y
  using certificate-eq by simp
have xs-eq: xs = ys
  using certificate-eq by simp
have path-at-xs: finite-cycle-path-at M x = Some xs
  using path-at x-eq xs-eq by simp
show ?thesis
  using finite-cycle-path-at-length-le[OF path-at-xs] .
qed

```

40 The Finite Foundation Profile

type-synonym *finite-foundation-profile* =
 $\text{bool} \times \text{bool} \times \text{bool} \times \text{bool} \times \text{bool}$

definition *finite-foundation-profile* ::
 $\text{finite-dependency-model} \Rightarrow \text{finite-foundation-profile}$

where

```

finite-foundation-profile M =
  (finite-cycle-check M,
   finite-cycle-check M,
   finite-cycle-check M,
   finite-foundation-check M,
   finite-foundation-check M)

```

theorem *finite-foundation-profile-correct*:

```

finite-foundation-profile M =
  (acyclic-on (finite-model-carrier M) (finite-model-immediate M),
   well-founded-on (finite-model-carrier M) (finite-model-immediate M),
   chain-terminating-on (finite-model-carrier M)
    (finite-model-immediate M),
   local-foundation-on (finite-model-carrier M)
    (finite-model-immediate M) (finite-model-carrier M),
   global-foundation-on (finite-model-carrier M)
    (finite-model-immediate M) (finite-model-carrier M))

```

proof –

```

have finite-carrier:
  finite (finite-model-carrier M)
  by (simp add: finite-model-carrier-def)
have cycle:
  finite-cycle-check M =
    acyclic-on (finite-model-carrier M) (finite-model-immediate M)
  using finite-cycle-check-iff by blast
have wf-acyclic:
  well-founded-on (finite-model-carrier M)
    (finite-model-immediate M) =
    acyclic-on (finite-model-carrier M)
      (finite-model-immediate M)
  using finite-well-founded-iff-acyclic-on[OF finite-carrier]
  by blast
have wf-chain:
  well-founded-on (finite-model-carrier M)
    (finite-model-immediate M) =
    chain-terminating-on (finite-model-carrier M)
      (finite-model-immediate M)

```

```

    using well-founded-iff-chain-terminating by blast
  have local-global:
    local-foundation-on (finite-model-carrier M)
      (finite-model-immediate M) (finite-model-carrier M) =
      global-foundation-on (finite-model-carrier M)
        (finite-model-immediate M) (finite-model-carrier M)
    using local-foundation-iff-global-foundation[OF subset-refl] by blast
  have local-check:
    finite-foundation-check M =
      local-foundation-on (finite-model-carrier M)
        (finite-model-immediate M) (finite-model-carrier M)
    using finite-foundation-check-iff by blast
  show ?thesis
    unfolding finite-foundation-profile-def
    using cycle wf-acyclic wf-chain local-global local-check
    by simp
qed

```

```

value finite-cycle-result finite-chain-model
value finite-cycle-result finite-cycle-model
value finite-foundation-profile finite-chain-model
value finite-foundation-profile finite-cycle-model

```

```

end
theory Thomistic-Causal-Arguments
imports
  Dependency-Structures
  Foundation-Principles
  Foundation-Implications
  Foundation-Separations
  Finite-Foundation-Checker
  Second-Way-Reconstructions
  Research-Results
  Premise-Minimality
  Finite-Foundation-Analysis
begin

```

41 Summary

This session formalizes a carrier-restricted theory of causal and grounding regress arguments. It distinguishes acyclicity, well-foundedness, chain termination, local foundation, global foundation, and full grounding; proves the valid implications and finite equivalences; and supplies infinite and finite separation models, including a rooted-cycle witness showing that foundation and acyclicity are incomparable on arbitrary carriers. It also contains an executable finite checker with exact certificate correctness, explicit finite cycle paths, premise-deletion countermodels, and two explicit Second-Way reconstruction packages. The Cohoe package additionally records the proved redundancy of a separate carrier premise. The entry deliberately does not formalize the First or Third Ways: substantive treatments would require richer act/potency and modal semantics. All positive conclusions remain conditional on the dependency structures and bridge premises stated in the theories, and no formal first cause is named as God.

```

end

```

References

- [1] T. Aquinas. *Summa Theologiae*. Benziger Bros., 1947. Translated by Fathers of the English Dominican Province.
- [2] C. Benz Müller and B. Woltzenlogel Paleo. Gödel’s god in isabelle/hol. *Archive of Formal Proofs*, 2013. <https://www.isa-afp.org/entries/GoedelGod.html>.
- [3] R. Bliss. Viciousness and circles of ground. *Metaphilosophy*, 45(2):245–256, 2014. DOI: <https://doi.org/10.1111/meta.12072>.
- [4] C. Cohoe. There must be a first: Why thomas aquinas rejects infinite, essentially ordered, causal series. *British Journal for the History of Philosophy*, 21(5):838–856, 2013. DOI: <https://doi.org/10.1080/09608788.2013.816934>.
- [5] T. S. Dixon. What is the well-foundedness of grounding? *Mind*, 125(498):439–468, 2016. DOI: <https://doi.org/10.1093/mind/fzv112>.
- [6] O. Pearson. Grounding, well-foundedness, and terminating chains. *Philosophia*, 51:1539–1554, 2023. DOI: <https://doi.org/10.1007/s11406-022-00593-x>.
- [7] G. O. Rabin and B. Rabern. Well founding grounding grounding. *Journal of Philosophical Logic*, 45(4):349–379, 2016. DOI: <https://doi.org/10.1007/s10992-015-9376-4>.
- [8] G. E. Romero and D. Pérez. New remarks on the cosmological argument. *International Journal for Philosophy of Religion*, 72(2):103–113, 2012. DOI: <https://doi.org/10.1007/s11153-012-9337-6>.