

Definitive Set Semantics for LTL3

Rayhana Amjad, Rob van Glabbeek, Liam O'Connor

June 6, 2026

LTL_3 is a multi-valued variant of Linear-time Temporal Logic for runtime verification applications, originally due to Bauer et al [3]. The semantic descriptions of LTL_3 in previous work are given only in terms of the relationship to conventional LTL. In this submission, which accompanies our EXPRESS/SOS 2024 paper [1], we instead give a full model-based inductive accounting of the semantics of LTL_3 , in terms of families of *definitive prefix sets*. We show that our definitive prefix sets are isomorphic to linear-time temporal properties (sets of infinite traces), and thereby show that our semantics of LTL_3 directly correspond to the semantics of conventional LTL. In addition, we formalise the formula progression evaluation technique [2, 4], popularly used in runtime verification and testing contexts, and show its soundness and completeness up to finite traces with respect to our semantics.

Contents

1	Answer-Indexed Families	4
1.1	Example: Propositional logic	4
1.1.1	Propositional logic lemmas	5
1.1.2	Propositional logic equivalence	5
2	Traces and Definitive Prefixes	6
2.1	Traces	6
2.2	Prefix Closure	11
2.3	Definitive Prefixes	12
2.4	Definitive Sets	13
2.5	A type for definitive sets	15
2.6	Isomorphism of definitive sets and LTL properties	16
3	Linear-time Temporal Logic	24
3.1	Linear temporal logic equivalence	25
3.2	Linear temporal logic lemmas	25
4	LTL3: Semantics, Equivalence and Formula Progression	26
4.1	LTL/LTL3 equivalence	27
4.2	Equivalence to LTL3 of Bauer et al.	31
4.3	Formula Progression	32

```
theory AnswerIndexedFamilies
imports Main
begin
```

1 Answer-Indexed Families

```

typedecl 'a state
consts L :: ⟨'a state => 'a set⟩
datatype answer = T | F
type-synonym 'a AiF = ⟨answer => 'a set⟩

fun and-AiF :: ⟨'a AiF => 'a AiF => 'a AiF⟩ (infixl ⟨∧·⟩ 60) where
  ⟨(a ∧· b) T = a T ∩ b T⟩
| ⟨(a ∧· b) F = a F ∪ b F⟩

fun or-AiF :: ⟨'a AiF => 'a AiF => 'a AiF⟩ (infixl ⟨∨·⟩ 59) where
  ⟨(a ∨· b) T = a T ∪ b T⟩
| ⟨(a ∨· b) F = a F ∩ b F⟩

fun not-AiF :: ⟨'a AiF => 'a AiF⟩ (⟨¬· -⟩) where
  ⟨(¬· a) T = a F⟩
| ⟨(¬· a) F = a T⟩

fun univ-AiF :: ⟨'a AiF⟩ (⟨T·⟩) where
  ⟨T· T = UNIV⟩
| ⟨T· F = {}⟩

fun satisfying-AiF :: ⟨'a => 'a state AiF⟩ (⟨sat·⟩) where
  ⟨sat· x T = {state. x ∈ L state}⟩
| ⟨sat· x F = {state. x ∉ L state}⟩

```

1.1 Example: Propositional logic

```

datatype (atoms-plogic: 'a) plogic =
  True-plogic                               (⟨truep⟩)
| Prop-plogic ⟨'a⟩                          (⟨propp'(-)⟩)
| Not-plogic ⟨'a plogic⟩                    (⟨notp -> [85] 85)⟩
| Or-plogic ⟨'a plogic⟩ ⟨'a plogic⟩        (⟨- orp -> [82,82] 81)⟩
| And-plogic ⟨'a plogic⟩ ⟨'a plogic⟩       (⟨- andp -> [82,82] 81)⟩

fun plogic-semantics :: ⟨'a plogic => 'a state AiF⟩ (⟨[[·]]p⟩) where
  ⟨[[ truep ]]p = T·⟩
| ⟨[[ notp φ ]]p = ¬· [[φ]]p⟩
| ⟨[[ propp(a) ]]p = sat· a⟩
| ⟨[[ φ orp ψ ]]p = [[φ]]p ∨· [[ψ]]p⟩
| ⟨[[ φ andp ψ ]]p = [[φ]]p ∧· [[ψ]]p⟩

definition false-p (⟨falsep⟩) where

```

false-p-def [*simp*]: $\langle \text{false}_p = \text{not}_p \text{true}_p \rangle$

definition *implies-p* :: $\langle 'a \text{ plogic} \Rightarrow 'a \text{ plogic} \Rightarrow 'a \text{ plogic} \rangle$ ($\langle - \text{implies}_p - \rangle$ [81,81] 80) **where**
implies-p-def[*simp*]: $\langle \varphi \text{implies}_p \psi = (\text{not}_p \varphi \text{or}_p \psi) \rangle$

1.1.1 Propositional logic lemmas

lemma *AiF-cases*:

assumes $\langle A \ T = B \ T \rangle$ **and** $\langle A \ F = B \ F \rangle$

shows $\langle A = B \rangle$

proof (*rule ext*)

fix x **show** $\langle A \ x = B \ x \rangle$ **by** (*cases* $\langle x \rangle$; *simp add: assms*)

qed

lemma *or-and-negation*: $\langle \llbracket \varphi \text{or}_p \psi \rrbracket_p = \llbracket \text{not}_p ((\text{not}_p \varphi) \text{and}_p (\text{not}_p \psi)) \rrbracket_p \rangle$
by (*rule AiF-cases; simp*)

lemma *and-or-negation*: $\langle \llbracket \varphi \text{and}_p \psi \rrbracket_p = \llbracket \text{not}_p ((\text{not}_p \varphi) \text{or}_p (\text{not}_p \psi)) \rrbracket_p \rangle$
by (*rule AiF-cases; simp*)

lemma *de-morgan-1*: $\langle \llbracket \text{not}_p (\varphi \text{and}_p \psi) \rrbracket_p = \llbracket (\text{not}_p \varphi) \text{or}_p (\text{not}_p \psi) \rrbracket_p \rangle$
by (*rule AiF-cases; simp*)

lemma *de-morgan-2*: $\langle \llbracket \text{not}_p (\varphi \text{or}_p \psi) \rrbracket_p = \llbracket (\text{not}_p \varphi) \text{and}_p (\text{not}_p \psi) \rrbracket_p \rangle$
by (*rule AiF-cases; simp*)

1.1.2 Propositional logic equivalence

fun *pllogic-semantic*' :: $\langle 'a \text{ state} \Rightarrow 'a \text{ plogic} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \models_p \rangle$ 60) **where**

$\langle \Gamma \models_p \text{true}_p = \text{True} \rangle$
 $\langle \Gamma \models_p \text{not}_p \varphi = (\neg \Gamma \models_p \varphi) \rangle$
 $\langle \Gamma \models_p \text{prop}_p(a) = (a \in L \ \Gamma) \rangle$
 $\langle \Gamma \models_p \varphi \text{or}_p \psi = (\Gamma \models_p \varphi \vee \Gamma \models_p \psi) \rangle$
 $\langle \Gamma \models_p \varphi \text{and}_p \psi = (\Gamma \models_p \varphi \wedge \Gamma \models_p \psi) \rangle$

lemma *pllogic-equivalence*:

shows $\langle (\Gamma \models_p \varphi \longleftrightarrow \Gamma \in \llbracket \varphi \rrbracket_p \ T) \rangle$

and $\langle (\neg \Gamma \models_p \varphi \longleftrightarrow \Gamma \in \llbracket \varphi \rrbracket_p \ F) \rangle$

proof (*induct* $\langle \varphi \rangle$)

qed (*auto*)

end

theory *Traces*

imports *Main HOL.Lattices HOL.List*

begin

2 Traces and Definitive Prefixes

2.1 Traces

```
typedecl  $\Sigma$ 
type-synonym 'a finite-trace = ⟨'a list⟩
type-synonym 'a infinite-trace = ⟨nat  $\Rightarrow$  'a⟩
datatype 'a trace = Finite ⟨'a finite-trace⟩ | Infinite ⟨'a infinite-trace⟩
```

```
fun thead :: ⟨'a trace  $\Rightarrow$  'a⟩ where
  ⟨thead (Finite t) = t ! 0⟩
| ⟨thead (Infinite t) = t 0⟩
```

```
fun append :: ⟨'a trace  $\Rightarrow$  'a trace  $\Rightarrow$  'a trace⟩ (infixr ⟨ $\frown$ ⟩ 80) where
  ⟨(Finite t)  $\frown$  (Infinite  $\omega$ ) = Infinite ( $\lambda n$ . if  $n < \text{length } t$  then  $t!n$  else  $\omega (n - \text{length } t)$ )⟩
| ⟨(Finite t)  $\frown$  (Finite u) = Finite (t @ u)⟩
| ⟨(Infinite t)  $\frown$  u = Infinite t⟩
```

```
definition  $\varepsilon$  :: ⟨'a trace⟩ where
  ⟨ $\varepsilon$  = Finite []⟩
```

```
definition singleton :: ⟨'a  $\Rightarrow$  'a trace⟩ where
  ⟨singleton  $\sigma$  = Finite [ $\sigma$ ]⟩
```

```
interpretation trace: monoid-list ⟨( $\frown$ )⟩ ⟨ $\varepsilon$ ⟩
```

```
proof unfold-locales
```

```
  fix a :: ⟨'a trace⟩ show ⟨ $\varepsilon \frown a = a$ ⟩
  by (cases ⟨a⟩; simp add:  $\varepsilon$ -def)
```

```
next
```

```
  fix a :: ⟨'a trace⟩ show ⟨a  $\frown$   $\varepsilon = a$ ⟩
  by (cases ⟨a⟩; simp add:  $\varepsilon$ -def)
```

```
next
```

```
  fix a b c :: ⟨'a trace⟩ show ⟨(a  $\frown$  b)  $\frown$  c = a  $\frown$  (b  $\frown$  c)⟩
```

```
  apply (cases ⟨a⟩; simp)
```

```
  apply (cases ⟨b⟩; simp)
```

```
  apply (cases ⟨c⟩; simp)
```

```
  apply (rule ext; simp)
```

```
  by (smt (verit, ccfv-threshold)
```

```
      add.commute add-diff-inverse-nat add-less-cancel-left
```

```
      nth-append trans-less-add2)
```

```
qed
```

```
lemma finite-empty-suffix:
```

```
  assumes ⟨Finite xs = Finite xs  $\frown$  t⟩
```

shows $\langle t = \varepsilon \rangle$
using *assms* **by** (*cases* $\langle t \rangle$) (*simp-all* *add: ε -def*)

lemma *finite-empty-prefix*:
assumes $\langle \text{Finite } xs = t \frown \text{Finite } xs \rangle$
shows $\langle t = \varepsilon \rangle$
using *assms* **by** (*cases* $\langle t \rangle$) (*simp-all* *add: ε -def*)

lemma *finite-finite-suffix*:
assumes $\langle \text{Finite } xs = \text{Finite } ys \frown t \rangle$
obtains *zs* **where** $\langle t = \text{Finite } zs \rangle$
using *assms* **by** (*cases* $\langle t \rangle$) (*simp-all*)

lemma *finite-finite-prefix*:
assumes $\langle \text{Finite } xs = t \frown \text{Finite } ys \rangle$
obtains *zs* **where** $\langle t = \text{Finite } zs \rangle$
using *assms* **by** (*cases* $\langle t \rangle$) (*simp-all*)

lemma *append-is-empty*:
assumes $\langle t \frown u = \varepsilon \rangle$
shows $\langle t = \varepsilon \rangle$
and $\langle u = \varepsilon \rangle$
using *assms* **by** (*simp* *add: ε -def*; *cases* $\langle t \rangle$; *cases* $\langle u \rangle$; *simp*)⁺

fun *ttake* :: $\langle \text{nat} \Rightarrow 'a \text{ trace} \Rightarrow 'a \text{ finite-trace} \rangle$ **where**
 $\langle \text{ttake } k (\text{Finite } xs) = \text{take } k \text{ } xs \rangle$
 $\mid \langle \text{ttake } k (\text{Infinite } xs) = \text{map } xs [0..<k] \rangle$

definition *itdrop* :: $\langle \text{nat} \Rightarrow 'a \text{ infinite-trace} \Rightarrow 'a \text{ infinite-trace} \rangle$ **where**
 $\langle \text{itdrop } k \text{ } xs = (\lambda i. xs (i + k)) \rangle$

lemma *itdrop-itdrop[*simp*]*: $\langle \text{itdrop } i (\text{itdrop } j \text{ } x) = \text{itdrop } (i + j) \text{ } x \rangle$
by (*simp* *add: itdrop-def* *add.commute* *add.left-commute*)

lemma *itdrop-zero[*simp*]*: $\langle \text{itdrop } 0 \text{ } x = x \rangle$
by (*simp* *add: itdrop-def*)

fun *tdrop* :: $\langle \text{nat} \Rightarrow 'a \text{ trace} \Rightarrow 'a \text{ trace} \rangle$ **where**
 $\langle \text{tdrop } k (\text{Finite } xs) = \text{Finite } (\text{drop } k \text{ } xs) \rangle$
 $\mid \langle \text{tdrop } k (\text{Infinite } xs) = \text{Infinite } (\text{itdrop } k \text{ } xs) \rangle$

lemma *ttake-simp[*simp*]*: $\langle \text{ttake } (\text{length } xs) (\text{Finite } xs \frown t) = xs \rangle$
by (*cases* $\langle t \rangle$, *auto* *intro: list-eq-iff-nth-eq[THEN iffD2]*)

lemma *ttake-tdrop[*simp*]*: $\langle \text{Finite } (\text{ttake } k \text{ } t) \frown \text{tdrop } k \text{ } t = t \rangle$
by (*cases* $\langle t \rangle$, *auto* *simp: itdrop-def*)

definition *prefixes* :: $\langle 'a \text{ trace} \Rightarrow 'a \text{ trace set} \rangle (\langle \downarrow \rightarrow [80] 80 \rangle)$ **where**
 $\langle \downarrow t = \{ u \mid u \ v. t = u \frown v \} \rangle$

definition *extensions* :: $\langle 'a \text{ trace} \Rightarrow 'a \text{ trace set} \rangle (\langle \uparrow \rightarrow [80] 80 \rangle)$ **where**
 $\langle \uparrow t = \{ t \frown u \mid u. \text{True} \} \rangle$

lemma *prefixes-extensions*: $\langle t \in \downarrow u \longleftrightarrow u \in \uparrow t \rangle$
unfolding *prefixes-def extensions-def* **by** *simp*

interpretation *prefixes*: *order* $\langle \lambda t \ u. t \in \downarrow u \rangle \langle \lambda t \ u. t \in \downarrow u \wedge t \neq u \rangle$
proof

fix *x* :: $\langle 'a \text{ trace} \rangle$
show $\langle x \in \downarrow x \rangle$
unfolding *prefixes-def*
by (*simp, metis trace.right-neutral*)
next

fix *x y* :: $\langle 'a \text{ trace} \rangle$
show $\langle (x \in \downarrow y \wedge x \neq y) = (x \in \downarrow y \wedge \neg y \in \downarrow x) \rangle$
unfolding *prefixes-def*
by (*simp, metis append.simps(3) append-is-empty(1) finite-empty-suffix trace.assoc trace.exhaust*)
next

fix *x y* :: $\langle 'a \text{ trace} \rangle$
assume *assms*: $\langle x \in \downarrow y \rangle \langle y \in \downarrow x \rangle$
show $\langle x = y \rangle$
proof (*cases* $\langle y \rangle$)
case *Finite* **note** *yfinite = this*
show $\langle ?thesis \rangle$
proof (*cases* $\langle x \rangle$)
case *Finite*
with *assms(2)* **obtain** *z* **where** $\langle x = y \frown z \rangle$
unfolding *prefixes-def*
by *auto*
with *assms(1) yfinite* **show** $\langle ?thesis \rangle$
unfolding *prefixes-def*
by (*force simp: trace.assoc dest: finite-empty-suffix append-is-empty*)
qed (*smt (verit, del-insts) CollectD append.simps(3) assms(1) prefixes-def*)
qed (*smt (verit, del-insts) CollectD append.simps(3) assms(2) prefixes-def*)
next

fix *x y z* :: $\langle 'a \text{ trace} \rangle$
assume $\langle x \in \downarrow y \rangle \langle y \in \downarrow z \rangle$
then show $\langle x \in \downarrow z \rangle$
unfolding *prefixes-def* **by** (*force simp: trace.assoc*)
qed

lemma *prefixes-empty-least* : $\langle \varepsilon \in \downarrow t \rangle$
by (*simp add: prefixes-def*)

lemma *prefixes-infinite-greatest* : $\langle \text{Infinite } x \in \downarrow t \implies t = \text{Infinite } x \rangle$
by (*simp add: prefixes-def*)

lemma *prefixes-finite* : $\langle \text{Finite } xs \in \downarrow \text{Finite } ys \longleftrightarrow (\exists zs. ys = xs @ zs) \rangle$

proof (*rule iffI*)

show $\langle \text{Finite } xs \in \downarrow \text{Finite } ys \implies \exists zs. ys = xs @ zs \rangle$

using *finite-finite-suffix* **by** (*fastforce simp: prefixes-def*)

next

show $\langle \exists zs. ys = xs @ zs \implies \text{Finite } xs \in \downarrow \text{Finite } ys \rangle$

by (*clarsimp simp: prefixes-def*) (*metis Traces.append.simps(2)*)

qed

lemma *ttake-take* : $\langle \text{take } n (\text{ttake } m t) = \text{ttake } (\text{min } n m) t \rangle$
by (*cases <t>*) (*simp-all add: min-def take-map*)

lemma *tdrop-tdrop* : $\langle \text{tdrop } n (\text{tdrop } m t) = \text{tdrop } (n + m) t \rangle$
by (*cases <t>*) (*simp-all add: add.commute add.left-commute*)

lemma *tdrop-mono*: $\langle t \in \downarrow u \implies \text{tdrop } k t \in \downarrow \text{tdrop } k u \rangle$

proof –

{ **fix** *v* **assume** *A*: $\langle u = t \frown v \rangle$ **then have** $\langle \exists va. \text{tdrop } k (t \frown v) = \text{tdrop } k t \frown va \rangle$

proof (*cases <t>*; *cases <v>*)

fix *x1 x2* **assume** $\langle t = \text{Finite } x1 \rangle$ **and** $\langle v = \text{Finite } x2 \rangle$ **with** *A* **show** $\langle ?thesis \rangle$

by (*simp, metis Traces.append.simps(2)*)

next

fix *x1 x2* **assume** $\langle t = \text{Finite } x1 \rangle$ **and** $\langle v = \text{Infinite } x2 \rangle$ **with** *A*

have $\langle \text{tdrop } k (t \frown v) = \text{tdrop } k t \frown \text{Infinite } (\text{itdrop } (k - \text{length } x1) x2) \rangle$

apply *simp*

apply (*rule ext*)

apply *clarsimp*

apply (*rule conjI*)

apply (*simp add: add.commute itdrop-def less-diff-conv*)

by (*smt (z3) add.commute add-diff-cancel-left' add-diff-inverse-nat diff-is-0-eq' diff-right-commute itdrop-def linorder-not-less nat-less-le*)

then show $\langle \exists va. \text{tdrop } k (t \frown v) = \text{tdrop } k t \frown va \rangle$

by *auto*

qed auto } **note** *A = this*

assume $\langle t \in \downarrow u \rangle$ **with** *A* **show** $\langle ?thesis \rangle$ **unfolding** *prefixes-def* **by** *clarsimp*

qed

lemma *ttake-finite-prefixes* : $\langle \text{Finite } xs \in \downarrow t \longleftrightarrow xs = \text{ttake } (\text{length } xs) t \rangle$

proof (*rule iffI*)

show $\langle \text{Finite } xs \in \downarrow t \implies xs = \text{ttake } (\text{length } xs) t \rangle$

by (clarsimp simp: prefixes-def)
 next
 show $\langle xs = \text{ttake } (\text{length } xs) \ t \implies \text{Finite } xs \in \downarrow t \rangle$
 unfolding prefixes-def **using** ttake-tdrop
 by (metis (full-types) mem-Collect-eq)
 qed

lemma ttake-prefixes : $\langle a \leq b \implies \text{Finite } (\text{ttake } a \ t) \in \downarrow \text{Finite } (\text{ttake } b \ t) \rangle$
 by (cases $\langle t \rangle$; simp add: ttake-finite-prefixes min-def take-map)

lemma finite-directed:
assumes $\langle \text{Finite } xs \in \downarrow t \rangle \langle \text{Finite } ys \in \downarrow t \rangle$
shows $\langle \exists zs. (xs = ys @ zs) \vee (ys = xs @ zs) \rangle$
proof (cases $\langle \text{length } xs > \text{length } ys \rangle$)
 case True
 with assms **show** $\langle ?thesis \rangle$
 apply (simp add: ttake-finite-prefixes)
 using ttake-prefixes[simplified prefixes-finite]
 by (metis less-le-not-le)
 next
 case False
 from assms **this**[THEN leI] **show** $\langle ?thesis \rangle$
 apply (simp add: ttake-finite-prefixes)
 using ttake-prefixes[simplified prefixes-finite]
 by (metis)
 qed

lemma prefixes-directed: $\langle u \in \downarrow t \implies v \in \downarrow t \implies u \in \downarrow v \vee v \in \downarrow u \rangle$
proof (cases $\langle v \rangle$; cases $\langle u \rangle$)
 { **fix** $a \ b$ **assume** $\langle \text{Finite } a \in \downarrow t \rangle \langle \text{Finite } b \in \downarrow t \rangle$
 then have $\langle \text{Finite } a \in \downarrow \text{Finite } b \vee \text{Finite } b \in \downarrow \text{Finite } a \rangle$
 using finite-directed prefixes-finite **by** blast } **note** $X = \text{this}$
 fix $a \ b$ **show** $\langle u \in \downarrow t \implies v \in \downarrow t \implies v = \text{Finite } a \implies u = \text{Finite } b \implies u \in \downarrow v \vee v \in \downarrow u \rangle$
 using X **by** auto
 qed (auto simp: prefixes-def dest: prefixes-infinite-greatest)

interpretation extensions: order $\langle \lambda \ t \ u. t \in \uparrow u \rangle \langle \lambda \ t \ u. t \in \uparrow u \wedge t \neq u \rangle$
proof
 qed (auto simp: prefixes-extensions[THEN sym] dest: prefixes.leD intro: prefixes.order.trans)

lemma extensions-infinite[simp]: $\langle \uparrow \text{Infinite } xs = \{ \text{Infinite } xs \} \rangle$
 by (simp add: extensions-def)

lemma extensions-empty[simp]: $\langle \uparrow \varepsilon = \text{UNIV} \rangle$
 by (simp add: extensions-def)

lemma prefixes-empty: $\langle \downarrow \varepsilon = \{ \varepsilon \} \rangle$
apply (clarsimp simp add: set-eq-iff ε -def prefixes-def)

apply (*rule iffI*)
apply (*metis ε -def append-is-empty(1)*)
by (*metis ε -def trace.left-neutral*)

2.2 Prefix Closure

definition *prefix-closure* :: $\langle 'a \text{ trace set} \Rightarrow 'a \text{ trace set} \rangle (\downarrow_s \rightarrow [80] 80)$ **where**
 $\langle \downarrow_s X = (\bigcup t \in X. \text{prefixes } t) \rangle$

lemma *prefix-closure-subset*: $\langle X \subseteq \downarrow_s X \rangle$
unfolding *prefix-closure-def*
by *auto*

lemma *prefix-closure-infinite*: $\langle \text{Infinite } x \in \downarrow_s X \longleftrightarrow \text{Infinite } x \in X \rangle$
proof

assume $\langle \text{Infinite } x \in \downarrow_s X \rangle$ **then show** $\langle \text{Infinite } x \in X \rangle$
by (*metis UN-E prefix-closure-def prefixes-infinite-greatest*)

next

assume $\langle \text{Infinite } x \in X \rangle$ **then show** $\langle \text{Infinite } x \in \downarrow_s X \rangle$
by (*meson in-mono prefix-closure-subset*)

qed

lemma *prefix-closure-idem*: $\langle \downarrow_s \downarrow_s X = \downarrow_s X \rangle$
unfolding *prefix-closure-def*
using *prefixes.order.trans* **by** *blast*

lemma *prefix-closure-mono*: $\langle X \subseteq Y \Longrightarrow \downarrow_s X \subseteq \downarrow_s Y \rangle$
unfolding *prefix-closure-def*
by *blast*

lemma *prefix-closure-union-distrib*: $\langle \downarrow_s (X \cup Y) = \downarrow_s X \cup \downarrow_s Y \rangle$
unfolding *prefix-closure-def*
by *simp*

lemma *prefix-closure-Union-distrib*: $\langle \downarrow_s (\bigcup S) = \bigcup (\text{prefix-closure } ' S) \rangle$
unfolding *prefix-closure-def*
by *simp*

lemma *prefix-closure-Inter*: $\langle \downarrow_s (\bigcap (\text{prefix-closure } ' S)) = \bigcap (\text{prefix-closure } ' S) \rangle$
unfolding *prefix-closure-def*
using *prefixes.dual-order.trans* **by** *fastforce*

lemma *prefix-closure-inter*: $\langle \downarrow_s (\downarrow_s X \cap \downarrow_s Y) = \downarrow_s X \cap \downarrow_s Y \rangle$
by (*rule prefix-closure-Inter* [**where** $S = \langle \{X, Y\} \rangle$, *simplified*])

lemma *prefix-closure-UNIV*: $\langle \downarrow_s \text{UNIV} = \text{UNIV} \rangle$
unfolding *prefix-closure-def* **by** *blast*

lemma *prefix-closure-empty*: $\langle \downarrow_s \{\} = \{\} \rangle$

unfolding *prefix-closure-def* **by** *blast*

lemma *prefix-closure-extensions*: $\langle \downarrow_s (\uparrow t) = \uparrow t \cup \downarrow t \rangle$
by (*force intro: prefix-closure-subset dest: prefixes-directed*
simp: prefixes-extensions[THEN sym] prefix-closure-def)

lemma *prefix-closure-prefixes*: $\langle \downarrow_s (\downarrow t) = \downarrow t \rangle$
unfolding *prefix-closure-def*
by (*force intro: prefixes.dual-order.trans*)

2.3 Definitive Prefixes

definition *dprefixes* :: $\langle 'a \text{ trace set} \Rightarrow 'a \text{ trace set} \rangle$ ($\langle \downarrow_d \rightarrow [80] 80 \rangle$) **where**
 $\langle \downarrow_d X = \{ t \mid t. \uparrow t \subseteq \downarrow_s X \} \rangle$

lemma *dprefixes-are-prefixes* : $\langle \downarrow_d X \subseteq \downarrow_s X \rangle$
unfolding *dprefixes-def*
using *extensions.order.refl* **by** *blast*

lemma *prefix-closure-dprefixes* : $\langle \downarrow_s (\downarrow_d X) \subseteq \downarrow_s X \rangle$
using *dprefixes-are-prefixes prefix-closure-idem prefix-closure-mono*
by *blast*

lemma *dprefixes-idem*: $\langle \downarrow_d \downarrow_d X = \downarrow_d X \rangle$

proof

show $\langle \downarrow_d \downarrow_d X \subseteq \downarrow_d X \rangle$
using *prefix-closure-dprefixes*
by (*force simp: dprefixes-def*)

next

show $\langle \downarrow_d X \subseteq \downarrow_d \downarrow_d X \rangle$
using *extensions.order.trans prefix-closure-subset*
by (*force simp: dprefixes-def*)

qed

lemma *dprefixes-contains-extensions*: $\langle t \in \downarrow_d X \Longrightarrow \uparrow t \subseteq \downarrow_d X \rangle$
unfolding *dprefixes-def*
using *extensions.dual-order.trans* **by** *auto*

lemma *dprefixes-infinite*: $\langle \text{Infinite } x \in \downarrow_d X \longleftrightarrow \text{Infinite } x \in X \rangle$

proof

show $\langle \text{Infinite } x \in X \Longrightarrow \text{Infinite } x \in \downarrow_d X \rangle$
unfolding *dprefixes-def*
using *prefix-closure-subset* **by** *fastforce*

next

show $\langle \text{Infinite } x \in \downarrow_d X \Longrightarrow \text{Infinite } x \in X \rangle$
unfolding *dprefixes-def*
by (*clarsimp simp: prefix-closure-infinite*)

qed

```

lemma dprefixes-UNIV:  $\langle \downarrow_d UNIV = UNIV \rangle$ 
  unfolding dprefixes-def
  using prefix-closure-UNIV by force

lemma dprefixes-empty:  $\langle \downarrow_d \{\} = \{\} \rangle$ 
  unfolding dprefixes-def
  using prefix-closure-empty by blast

lemma dprefixes-Inter-distrib:  $\langle \downarrow_d (\bigcap S) \subseteq \bigcap (dprefixes \text{ ' } S) \rangle$ 
  unfolding dprefixes-def prefix-closure-def
  by auto

lemma dprefixes-Inter:  $\langle \downarrow_d (\bigcap (dprefixes \text{ ' } S)) = \bigcap (dprefixes \text{ ' } S) \rangle$ 
proof
  show  $\langle \bigcap (dprefixes \text{ ' } S) \subseteq \downarrow_d \bigcap (dprefixes \text{ ' } S) \rangle$ 
    unfolding dprefixes-def prefix-closure-def
    using prefixes.order.refl extensions.dual-order.trans
    by force
  next
  show  $\langle \downarrow_d \bigcap (dprefixes \text{ ' } S) \subseteq \bigcap (dprefixes \text{ ' } S) \rangle$ 
    using dprefixes-idem dprefixes-Inter-distrib
    by blast
qed

lemma dprefixes-mono:
  assumes  $\langle X \subseteq Y \rangle$ 
  shows  $\langle \downarrow_d X \subseteq \downarrow_d Y \rangle$ 
  using assms
  apply (simp add: dprefixes-def)
  apply (simp add: prefix-closure-def)
  apply (rule subsetI)
  using prefixes-extensions by blast

lemma dprefixes-inter:  $\langle \downarrow_d (\downarrow_d X \cap \downarrow_d Y) = (\downarrow_d X \cap \downarrow_d Y) \rangle$ 
  by (rule dprefixes-Inter[where  $S = \langle \{X, Y\} \rangle$ , simplified])

lemma dprefixes-inter-distrib:  $\langle \downarrow_d (X \cap Y) \subseteq \downarrow_d X \cap \downarrow_d Y \rangle$ 
  using dprefixes-Inter-distrib[where  $S = \langle \{X, Y\} \rangle$ ] by auto

```

2.4 Definitive Sets

```

definition definitive::  $\langle 'a \text{ trace set} \Rightarrow \text{bool} \rangle$  where
   $\langle \text{definitive } X \longleftrightarrow \downarrow_d X = X \rangle$ 

```

```

lemma definitive-image:  $\langle \forall X \in S. \text{definitive } X \Longrightarrow dprefixes \text{ ' } S = S \rangle$ 
  unfolding definitive-def by auto

```

lemma *definitive-dprefixes*: $\langle \text{definitive } (\downarrow_d X) \rangle$
unfolding *definitive-def* **by** (*rule dprefixes-idem*)

lemma *definitive-contains-extensions*: $\langle \text{definitive } X \implies t \in X \implies \uparrow t \subseteq X \rangle$
unfolding *definitive-def* **using** *dprefixes-contains-extensions* **by** *blast*

lemma *definitive-UNIV*: $\langle \text{definitive } UNIV \rangle$
unfolding *definitive-def* **by** (*rule dprefixes-UNIV*)

lemma *definitive-empty*: $\langle \text{definitive } \{\} \rangle$
unfolding *definitive-def* **by** (*rule dprefixes-empty*)

lemma *definitive-Inter*: $\langle \forall X \in S. \text{definitive } X \implies \text{definitive } (\bigcap S) \rangle$
unfolding *definitive-def* **using** *dprefixes-Inter* *definitive-image[simplified definitive-def]*
by *metis*

lemma *definitive-inter*: $\langle \text{definitive } X \implies \text{definitive } Y \implies \text{definitive } (X \cap Y) \rangle$
using *definitive-Inter[where S = \{X, Y\}, simplified]* **by** *blast*

lemma *definitive-infinite-extension*:
assumes $\langle \text{definitive } X \rangle$ **and** $\langle t \in X \rangle$
shows $\langle \exists f. \text{Infinite } f \in X \wedge t \in \downarrow \text{Infinite } f \rangle$
using *assms* **proof** (*cases \langle t \rangle*)
case (*Finite xs*) **then show** $\langle ?thesis \rangle$
apply (*intro exI[where x = \lambda n. if n < length xs then xs!n else undefined]*)
by (*force simp: prefixes-extensions[THEN sym] prefixes-def*
intro!: definitive-contains-extensions[THEN subsetD, OF assms]
intro: exI[where x = \langle Infinite (\lambda-. undefined) \rangle])

qed *auto*

lemma *definitive-elemI*:
assumes $\langle \text{definitive } X \rangle$ $\langle \uparrow t \subseteq \downarrow_s X \rangle$
shows $\langle t \in X \rangle$
using *assms*
by (*auto simp add: definitive-def dprefixes-def*)

definition *dUnion* :: $\langle 'a \text{ trace set set} \Rightarrow 'a \text{ trace set} \rangle$ ($\langle \bigcup_d \rangle$) **where**
 $\langle \bigcup_d X = \downarrow_d \bigcup X \rangle$

abbreviation *dunion* :: $\langle 'a \text{ trace set} \Rightarrow 'a \text{ trace set} \Rightarrow 'a \text{ trace set} \rangle$ (**infixl** $\langle \bigcup_d \rangle$ 65) **where**
 $\langle X \cup_d Y \equiv \bigcup_d \{X, Y\} \rangle$

lemma *dprefixes-dUnion*: $\langle \downarrow_d \bigcup_d S = \bigcup_d S \rangle$
by (*simp add: dUnion-def dprefixes-idem*)

lemma *definitive-dUnion*: $\langle \text{definitive } (\bigcup_d S) \rangle$
by (*simp add: dprefixes-dUnion definitive-def*)

lemma *dUnion-contains-dprefixes*: $\langle t \in S \implies \downarrow_d t \subseteq \bigcup_d S \rangle$
by (*auto simp: dUnion-def dprefixes-def prefix-closure-def*)

lemma *dUnion-contains-definitive*: $\langle X \in S \implies \text{definitive } X \implies X \subseteq \bigcup_d S \rangle$
unfolding *definitive-def*
using *dUnion-contains-dprefixes* **by** *blast*

lemma *dUnion-empty[simp]*: $\langle \bigcup_d \{ \} = \{ \} \rangle$
unfolding *dUnion-def*
by (*simp add: dprefixes-empty*)

lemma *dUnion-least-dprefixes*: $\langle (\bigwedge X. X \in S \implies X \subseteq \downarrow_d Z) \implies \downarrow_d (\bigcup (\text{dprefixes } S)) \subseteq \downarrow_d Z \rangle$
unfolding *dprefixes-def prefix-closure-def*
by (*simp add: subset-iff, meson extensions.order-refl prefixes.order.trans*)

lemma *dUnion-least-definitive*:
assumes *all-defn*: $\langle \forall X \in S. \text{definitive } X \rangle$
shows $\langle (\bigwedge X. X \in S \implies X \subseteq Z) \implies \text{definitive } Z \implies \downarrow_d \bigcup S \subseteq Z \rangle$
using *definitive-image[OF all-defn, THEN sym]* *dUnion-least-dprefixes* *definitive-def*
by *metis*

2.5 A type for definitive sets

typedef *'a dset* = $\langle \{ p :: 'a \text{ trace set. definitive } p \} \rangle$
using *definitive-UNIV* **by** *blast*

setup-lifting *type-definition-dset*

lift-definition *Inter-dset* :: $\langle 'a \text{ dset set} \Rightarrow 'a \text{ dset} \rangle$ ($\langle \bigcap \rangle$) **is** $\langle \lambda ss. \bigcap ss \rangle$
by (*simp add: definitive-Inter*)

abbreviation *inter-dset* :: $\langle 'a \text{ dset} \Rightarrow 'a \text{ dset} \Rightarrow 'a \text{ dset} \rangle$ (**infixl** $\langle \bigcap \rangle$ 66) **where**
 $\langle X \bigcap Y \equiv \bigcap \{ X, Y \} \rangle$

lift-definition *Union-cset* :: $\langle 'a \text{ dset set} \Rightarrow 'a \text{ dset} \rangle$ ($\langle \bigcup \rangle$) **is** $\langle \lambda ss. \bigcup_d ss \rangle$
by (*rule definitive-dUnion*)

abbreviation *union-dset* :: $\langle 'a \text{ dset} \Rightarrow 'a \text{ dset} \Rightarrow 'a \text{ dset} \rangle$ (**infixl** $\langle \bigcup \rangle$ 65) **where**
 $\langle X \bigcup Y \equiv \bigcup \{ X, Y \} \rangle$

lift-definition *empty-dset* :: $\langle 'a \text{ dset} \rangle$ ($\langle \emptyset \rangle$) **is** $\langle \{ \} \rangle$
by (*rule definitive-empty*)

lift-definition *univ-dset* :: $\langle 'a \text{ dset} \rangle$ ($\langle \Sigma \infty \rangle$) **is** $\langle UNIV \rangle$
by (*rule definitive-UNIV*)

lift-definition *subset-dset* :: $\langle 'a \text{ dset} \Rightarrow 'a \text{ dset} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \subseteq \rangle$ 50) **is** $\langle (\subseteq) \rangle$
done

lift-definition *strict-subset-cset* :: $\langle 'a \text{ dset} \Rightarrow 'a \text{ dset} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \sqsubset \rangle$ 50) **is** $\langle (\subset) \rangle$
done

lift-definition *in-dset* :: $\langle 'a \text{ trace} \Rightarrow 'a \text{ dset} \Rightarrow \text{bool} \rangle$ **is** $\langle (\in) \rangle$
done

lift-definition *notin-dset* :: $\langle 'a \text{ trace} \Rightarrow 'a \text{ dset} \Rightarrow \text{bool} \rangle$ **is** $\langle (\notin) \rangle$
done

lemma *in-dset-ε*: $\langle \text{in-dset } \varepsilon \ A \Longrightarrow A = \Sigma\infty \rangle$
apply (*transfer*)
using *definitive-contains-extensions extensions-empty* **by** *blast*

lemma *in-dset-UNIV*: $\langle \text{in-dset } x \ \Sigma\infty \rangle$
by (*transfer, simp*)

lemma *in-dset-subset*: $\langle A \sqsubseteq B \Longrightarrow \text{in-dset } x \ A \Longrightarrow \text{in-dset } x \ B \rangle$
by (*transfer, auto*)

lemma *in-dset-inter*: $\langle \text{in-dset } x \ A \Longrightarrow \text{in-dset } x \ B \Longrightarrow \text{in-dset } x \ (A \sqcap B) \rangle$
by (*transfer, simp*)

interpretation *dset*: *complete-lattice* $\langle \sqcap \rangle \langle \sqcup \rangle \langle (\cap) \rangle \langle (\sqsubseteq) \rangle \langle (\subset) \rangle \langle (\sqcup) \rangle \langle \emptyset \rangle \langle \Sigma\infty \rangle$

proof (*unfold-locales;transfer*)

fix $X \ Y \ Z$:: $\langle 'a \text{ trace set} \rangle$ **assume** $\langle \text{definitive } X \rangle \langle \text{definitive } Y \rangle \langle \text{definitive } Z \rangle$

then show $\langle Y \subseteq X \Longrightarrow Z \subseteq X \Longrightarrow (Y \cup_d Z) \subseteq X \rangle$

by (*metis dUnion-def dUnion-least-definitive insert-iff singletonD*)

next

fix A :: $\langle 'a \text{ trace set set} \rangle$ **and** Z :: $\langle 'a \text{ trace set} \rangle$

assume $\langle \forall X \in A. \text{definitive } X \rangle \langle \text{definitive } Z \rangle \langle (\bigwedge X. \text{definitive } X \Longrightarrow X \in A \Longrightarrow X \subseteq Z) \rangle$

then show $\langle \bigcup_d A \subseteq Z \rangle$

by (*simp add: dUnion-def dUnion-least-definitive*)

qed (*auto simp: dUnion-contains-definitive*)

2.6 Isomorphism of definitive sets and LTL properties

definition *infinites* :: $\langle 'a \text{ trace set} \Rightarrow 'a \text{ infinite-trace set} \rangle$ **where**

$\langle \text{infinites } X = (\bigcup x \in X. \text{case } x \text{ of } \text{Finite } xs \Rightarrow \{ \} \mid \text{Infinite } xs \Rightarrow \{xs\}) \rangle$

lemma *infinites-alt*: $\langle \text{Infinite } ' \text{infinites } A = A \cap \text{range } \text{Infinite} \rangle$

unfolding *set-eq-iff* **proof**

fix x { **assume** $\langle x \in \text{Infinite } ' \text{infinites } A \rangle$ **hence** $\langle x \in A \cap \text{range } \text{Infinite} \rangle$ }

by (*clarsimp simp: infinities-def split!: trace.split-asm*)

} **moreover** { **assume** $\langle x \in A \cap \text{range } \text{Infinite} \rangle$ **hence** $\langle x \in \text{Infinite } ' \text{infinites } A \rangle$ }

by (*force simp: infinities-def split!: trace.split intro!: imageI*)

} ultimately show $\langle x \in \text{Infinite } ' \text{infinites } A \rangle = \langle x \in A \cap \text{range } \text{Infinite} \rangle$
by *blast*
qed

lemma *infinites-append-right*: $\langle t \frown \text{Infinite } \omega \in \text{range } \text{Infinite} \rangle$
by (*cases* $\langle t \rangle$; *auto*)

lemma *infinites-prefix-closure*:
assumes $\langle \text{definitive } X \rangle$
shows $\langle \downarrow_s \text{Infinite } ' \text{infinites } X = \downarrow_s X \rangle$
unfolding *prefix-closure-def* *infinites-def*
using *definitive-infinite-extension*[*OF* *assms*] *prefixes.order.trans*
by (*force* *split*: *trace.split-asm*)

lemma *infinites-UNIV*[*simp*]: $\langle \text{infinites } \text{UNIV} = \text{UNIV} \rangle$
by (*auto* *simp*: *infinites-def* *split*: *trace.split*)

lemma *infinites-empty*[*simp*]: $\langle \text{infinites } \{\} = \{\} \rangle$
by (*auto* *simp*: *infinites-def*)

lemma *infinites-Inter*: $\langle \text{infinites } (\bigcap S) = \bigcap (\text{infinites } ' S) \rangle$
unfolding *infinites-def*
apply (*rule* *set-eqI*; *rule* *iffI*)
apply (*force*)
apply (*simp* *split*: *trace.split* *trace.split-asm*)
by (*metis* *InterI* *trace.distinct*(1) *trace.exhaust* *trace.inject*(2))

lemma *infinites-Union*: $\langle \text{infinites } (\bigcup S) = \bigcup (\text{infinites } ' S) \rangle$
unfolding *infinites-def*
by *auto*

lemma *infinites-dprefixes*: $\langle \text{infinites } (\downarrow_d X) = \text{infinites } X \rangle$
unfolding *infinites-def*
by (*force* *simp*: *dprefixes-infinite* *split*: *trace.split* *trace.split-asm*)

lemma *infinites-dprefixes-Infinite*: $\langle \text{infinites } (\downarrow_d \text{Infinite } ' X) = X \rangle$
proof

show $\langle \text{infinites } (\downarrow_d \text{Infinite } ' X) \subseteq X \rangle$
unfolding *infinites-def*
using *prefixes-infinite-greatest*
by (*force* *split*: *trace.split-asm* *simp*: *dprefixes-def* *prefix-closure-def*)

next

show $\langle X \subseteq \text{infinites } (\downarrow_d \text{Infinite } ' X) \rangle$
by (*force* *simp*: *infinites-def* *dprefixes-def* *prefix-closure-def* *split*: *trace.split*)

qed

lift-definition *property* :: $\langle 'a \text{ dset} \Rightarrow 'a \text{ infinite-trace set} \rangle$ **is** $\langle \text{infinites} \rangle$
done

lift-definition *definitives* :: $\langle 'a \text{ infinite-trace set} \Rightarrow 'a \text{ dset} \rangle$ **is** $\langle \lambda x. \downarrow_d (\text{Infinite } 'x) \rangle$
by (*rule definitive-dprefixes*)

lemma *property-inverse*: $\langle \text{property } (\text{definitives } X) = X \rangle$
by (*transfer, simp add: infinites-dprefixes-Infinite*)

lemma *definitives-inverse*: $\langle \text{definitives } (\text{property } X) = X \rangle$

proof (*rule dset.order-antisym*)

show $\langle \text{definitives } (\text{property } X) \sqsubseteq X \rangle$

by (*transfer, force simp: dprefixes-def infinites-prefix-closure*
intro: definitive-elemI)

next

show $\langle X \sqsubseteq \text{definitives } (\text{property } X) \rangle$

apply *transfer*

using *definitive-contains-extensions definitive-infinite-extension*

by (*force simp: dprefixes-def prefix-closure-def infinites-def*)

qed

lemma *definitives-mono*: $\langle A \subseteq B \Longrightarrow \text{definitives } A \sqsubseteq \text{definitives } B \rangle$

by (*transfer,metis dprefixes-inter-distrib image-mono inf.order-iff le-infE*)

lemma *property-mono*: $\langle A \sqsubseteq B \Longrightarrow \text{property } A \subseteq \text{property } B \rangle$

by (*transfer, auto simp: infinites-def*)

lemma *definitives-reflecting*: $\langle \text{definitives } A \sqsubseteq \text{definitives } B \Longrightarrow A \subseteq B \rangle$

using *property-inverse property-mono* **by** *metis*

lemma *completions-reflecting*: $\langle \text{property } A \subseteq \text{property } B \Longrightarrow A \sqsubseteq B \rangle$

using *definitives-inverse definitives-mono* **by** *metis*

lemma *property-Inter*: $\langle \text{property } (\bigcap S) = \bigcap (\text{property } 'S) \rangle$

by (*transfer, simp add: infinites-Inter*)

lemma *property-Union*: $\langle \text{property } (\bigcup S) = \bigcup (\text{property } 'S) \rangle$

by (*transfer, simp add: dUnion-def infinites-dprefixes infinites-Union*)

interpretation *dset*: *complete-distrib-lattice* $\langle \bigcap \rangle \langle \bigcup \rangle \langle (\bigcap) \rangle \langle (\sqsubseteq) \rangle \langle (\sqsupset) \rangle \langle (\sqcup) \rangle \langle \emptyset \rangle \langle \Sigma \infty \rangle$

by (*unfold-locales*)

(*auto intro: completions-reflecting simp add: property-Inter property-Union INF-SUP-set*)

definition *iprepend* :: $\langle 'a \text{ infinite-trace set} \Rightarrow 'a \text{ infinite-trace set} \rangle$ **where**

$\langle \text{iprepend } X = \{t. \text{itdrop } 1 t \in X\} \rangle$

lemma *iprepend-itdrop*: $\langle \text{itdrop } k x \in \text{iprepend } B \longleftrightarrow \text{itdrop } (\text{Suc } k) x \in B \rangle$

by (*simp add: iprepend-def*)

lemmas *iprepend-itdrop-0*[*simp*] = *iprepend-itdrop*[**where** $k = \langle 0 \rangle$, *simplified*]

definition *prepend'* :: $\langle 'a \text{ trace set} \Rightarrow 'a \text{ trace set} \rangle$ **where**
 $\langle \text{prepend}' X = \{t. \text{tdrop } 1 t \in X\} \rangle$

lemma *trace-uncons-cases* [*case-names Cons Nil*]:

assumes $\langle \bigwedge \sigma t. x = \text{singleton } \sigma \frown t \Longrightarrow P \rangle$

and $\langle x = \varepsilon \Longrightarrow P \rangle$

shows $\langle P \rangle$

proof (*cases* $\langle x \rangle$)

case (*Finite xs*)

then show $\langle ?thesis \rangle$

by (*cases* $\langle xs \rangle$;

force simp: assms(2)[simplified ε -def]

intro: assms(1)[where $t = \langle \text{Finite } ts \rangle$ for ts ,

simplified singleton-def append.simps List.append.simps])

next

case (*Infinite f*) **note** $A = \text{this}$

have $\langle f = (\lambda n. \text{if } n = 0 \text{ then } [f\ 0] ! n \text{ else } (f \circ \text{Suc}) (n - \text{length } [f\ 0])) \rangle$

by (*rule ext, simp*)

with A **show** $\langle ?thesis \rangle$

using *assms(1)[where $\sigma = \langle f\ 0 \rangle$ and $t = \langle \text{Infinite } (f \circ \text{Suc}) \rangle$,*

simplified singleton-def append.simps, simplified]

by *simp*

qed

lemma *append-prefixes-left*: $\langle a \in \downarrow b \Longrightarrow c \frown a \in \downarrow c \frown b \rangle$

by (*simp add: prefixes-def*) (*metis trace.assoc*)

lemma *tdrop-singleton-append*[*simp*]: $\langle \text{tdrop } (\text{Suc } n) (\text{singleton } \sigma \frown t) = \text{tdrop } n t \rangle$

by (*cases* $\langle t \rangle$, *simp-all add: singleton-def itdrop-def*)

lemma *tdrop-zero*[*simp*]: $\langle \text{tdrop } 0 t = t \rangle$

by (*cases* $\langle t \rangle$; *simp*)

lemma *tdrop- ε* [*simp*]: $\langle \text{tdrop } k \varepsilon = \varepsilon \rangle$

by (*simp add: ε -def*)

lemma *prepend'-prefix-closure*: $\langle \downarrow_s (\text{prepend}' X) \subseteq \text{prepend}' (\downarrow_s X) \rangle$

proof (*rule subsetI*)

fix x

assume $A: \langle x \in \downarrow_s \text{prepend}' X \rangle$

show $\langle x \in \text{prepend}' (\downarrow_s X) \rangle$

proof (*cases* $\langle x \rangle$ *rule: trace-uncons-cases*)

case (*Cons σt*)

with A **show** $\langle ?thesis \rangle$

unfolding *prefix-closure-def prepend'-def prefixes-def*

by (*fastforce simp: trace.assoc*)

next

case *Nil*

with A **show** $\langle ?thesis \rangle$

```

      unfolding prefix-closure-def prepend'-def
      by (force simp: prefixes-empty-least)
    qed
  qed

lemma prepend'-dprefixes :
  assumes ‹definitive X›
  shows ‹ $\downarrow_d$  prepend' X = prepend' X›
  proof
    show ‹ $\downarrow_d$  prepend' X  $\subseteq$  prepend' X›
    proof (rule subsetI)
      fix x assume A: ‹x  $\in$   $\downarrow_d$  prepend' X› show ‹x  $\in$  prepend' X›
      proof (cases ‹x› rule: trace-uncons-cases)
        case (Cons  $\sigma$  t)
          with A show ‹?thesis›
            unfolding dprefixes-def
            apply (subst assms[simplified definitive-def, THEN sym])
            apply (clarsimp dest!: subset-trans[OF - prepend'-prefix-closure])
            using append-prefixes-left
            by (force simp: dprefixes-def prepend'-def prefix-closure-def subset-iff
                prefixes-extensions[THEN sym])
      next
        case Nil
          with A show ‹?thesis›
            apply (subst assms[simplified definitive-def, THEN sym])
            apply (clarsimp simp: prefixes-empty-least prefixes-def dprefixes-def
                prepend'-def prefix-closure-def subset-iff
                prefixes-extensions[THEN sym])
            by (metis tdrop-singleton-append tdrop-zero trace.assoc)
      qed
    qed
  next
    show ‹prepend' X  $\subseteq$   $\downarrow_d$  prepend' X›
    proof (rule subsetI)
      fix x assume A: ‹x  $\in$  prepend' X› show ‹x  $\in$   $\downarrow_d$  prepend' X›
      proof (cases ‹x› rule: trace-uncons-cases)
        case (Cons  $\sigma$  t)
          with A show ‹?thesis›
            by (clarsimp simp: dprefixes-def prefixes-def prepend'-def
                prefix-closure-def prefixes-extensions[THEN sym])
            (metis (mono-tags, lifting) assms definitive-contains-extensions
                mem-Collect-eq prefixes-def prefixes-extensions subset-eq
                tdrop-singleton-append tdrop-zero trace.assoc)
      next
        case Nil
          with A show ‹?thesis›
            using assms definitive-contains-extensions
            by (force simp: dprefixes-def prepend'-def prefix-closure-def)
      qed
    qed
  next
    case Nil
    with A show ‹?thesis›
      using assms definitive-contains-extensions
      by (force simp: dprefixes-def prepend'-def prefix-closure-def)
  qed

```

qed
qed

lemma *prepend'-definitive* :
 assumes $\langle \text{definitive } X \rangle$
 shows $\langle \text{definitive } (\text{prepend}' X) \rangle$
 unfolding *definitive-def* **using** *assms*
 by (*rule prepend'-dprefixes*)

lift-definition *prepend* :: $\langle 'a \text{ dset} \Rightarrow 'a \text{ dset} \rangle$ **is** $\langle \text{prepend}' \rangle$
 by (*rule prepend'-definitive*)

lemma *prepend-Inter*: $\langle \bigcap (\text{prepend}' S) = \text{prepend} (\bigcap S) \rangle$
 apply *transfer*
 by (*auto simp add: prepend'-def*)

lemma *in-dset-prependD*: $\langle \text{in-dset } (\text{Finite } [a] \frown x) (\text{prepend } A) \implies \text{in-dset } x A \rangle$
 by (*transfer, metis One-nat-def Traces.singleton-def mem-Collect-eq prepend'-def*
 tdrop-singleton-append tdrop-zero)

lemma *in-dset-prependI*: $\langle \text{in-dset } x A \implies \text{in-dset } (\text{Finite } [a] \frown x) (\text{prepend } A) \rangle$
 by (*transfer, metis One-nat-def Traces.singleton-def mem-Collect-eq prepend'-def*
 tdrop-singleton-append tdrop-zero)

lemma *prepend'-mono*:
 assumes $\langle A \subseteq B \rangle$
 shows $\langle \text{prepend}' A \subseteq \text{prepend}' B \rangle$
 using *assms* **unfolding** *prepend'-def*
 by *blast*

lemma *property-prepend*: $\langle \text{property } (\text{prepend } X) = \text{iprepend } (\text{property } X) \rangle$
 apply *transfer*
 by (*clarsimp simp: definitive-def infinites-def prepend'-def*
 split!: trace.split-asm trace.split intro!: set-eqI;
 blast)

lemma *iprepend-Union*: $\langle \bigcup (\text{iprepend}' S) = \text{iprepend} (\bigcup S) \rangle$
 by *fastforce*

lemma *definitives-inverse-eqI*: $\langle \text{definitives } (\text{property } X) = \text{definitives } (\text{property } Y) \implies X = Y \rangle$
 by (*simp add: definitives-inverse*)

lemma *prepend-Union*: $\langle \bigcup (\text{prepend}' S) = \text{prepend} (\bigcup S) \rangle$
 apply (*rule definitives-inverse-eqI*)
 apply (*simp add: property-Union property-prepend*)
 by (*metis UN-extend-simps(10) iprepend-Union*)

lemma *non-empty-trace*: $\langle x \neq \varepsilon \iff (\exists \sigma x'. x = \text{Finite } [\sigma] \frown x') \rangle$
 apply (*cases* $\langle x \rangle$ *rule: trace-uncons-cases;clarsimp*)

apply (*metis* *Traces.singleton-def* *ε -def* *append-is-empty(1)* *not-Cons-self2* *trace.inject(1)*)
by (*metis* *ε -def* *append-is-empty(1)* *list.discI* *trace.inject(1)*)

lemma *thead-append*: $\langle x \neq \varepsilon \implies \text{thead } (x \frown y) = \text{thead } x \rangle$
by (*cases* $\langle x \rangle$; *cases* $\langle y \rangle$; *simp* *add*: *ε -def* *nth-append*)

lemma *thead-prefix*: $\langle x \in \downarrow y \implies x \neq \varepsilon \implies \text{thead } x = \text{thead } y \rangle$
apply (*simp* *add*: *prefixes-def* *non-empty-trace*)
using *thead-append* [**where** $x = \langle \text{Finite } [-] \rangle$, *simplified* *ε -def*, *simplified*]
by (*metis* *append-is-empty(1)* *thead-append*)

lemma *compr'-inter-thead*:

$$\langle \downarrow_d \{x. x \neq \varepsilon \wedge P(\text{thead } x)\} \cap \downarrow_d \{x. x \neq \varepsilon \wedge Q(\text{thead } x)\} \\ = \downarrow_d \{x. x \neq \varepsilon \wedge P(\text{thead } x) \wedge Q(\text{thead } x)\} \rangle$$

proof (*rule antisym*)

{ **fix** $x t$

assume $\langle \forall t. x \in \downarrow t \longrightarrow (\exists x. x \neq \varepsilon \wedge P(\text{thead } x) \wedge t \in \downarrow x) \rangle$

and $\langle \forall t. x \in \downarrow t \longrightarrow (\exists x. x \neq \varepsilon \wedge Q(\text{thead } x) \wedge t \in \downarrow x) \rangle$

and $\langle x \in \downarrow t \rangle$

then have $\langle \exists x. x \neq \varepsilon \wedge P(\text{thead } x) \wedge Q(\text{thead } x) \wedge t \in \downarrow x \rangle$

by (*cases* $\langle t = \varepsilon \rangle$; *fastforce* *dest*: *thead-prefix* *simp*: *prefixes-empty* *prefixes-empty-least*)

} **then show** $\langle \downarrow_d \{x. x \neq \varepsilon \wedge P(\text{thead } x)\} \cap \downarrow_d \{x. x \neq \varepsilon \wedge Q(\text{thead } x)\} \subseteq \downarrow_d \{x. x \neq \varepsilon \wedge P(\text{thead } x) \wedge Q(\text{thead } x)\} \rangle$

by (*clarsimp* *simp*: *set-eq-iff* *subset-iff* *dprefixes-def* *prefix-closure-def* *prefixes-extensions*[*THEN sym*])

next

{ **fix** x

assume $\langle \forall t. x \in \downarrow t \longrightarrow (\exists x. x \neq \varepsilon \wedge P(\text{thead } x) \wedge Q(\text{thead } x) \wedge t \in \downarrow x) \rangle$

then have $\langle \forall t. x \in \downarrow t \longrightarrow (\exists x. x \neq \varepsilon \wedge P(\text{thead } x) \wedge t \in \downarrow x) \wedge$

$$(\forall t. x \in \downarrow t \longrightarrow (\exists x. x \neq \varepsilon \wedge Q(\text{thead } x) \wedge t \in \downarrow x)) \rangle$$

by *fastforce* }

then show $\langle \downarrow_d \{x. x \neq \varepsilon \wedge P(\text{thead } x)\} \cap \downarrow_d \{x. x \neq \varepsilon \wedge Q(\text{thead } x)\} \supseteq \downarrow_d \{x. x \neq \varepsilon \wedge P(\text{thead } x) \wedge Q(\text{thead } x)\} \rangle$

by (*clarsimp* *simp*: *set-eq-iff* *subset-iff* *dprefixes-def* *prefix-closure-def* *prefixes-extensions*[*THEN sym*])

qed

lift-definition *compr* :: $\langle 'a \text{ trace} \implies \text{bool} \rangle \Rightarrow 'a \text{ dset}$ **is** $\langle \lambda p. \downarrow_d \{x. p \ x\} \rangle$

by (*rule* *definitive-dprefixes*)

lift-definition *complement* :: $\langle 'a \text{ dset} \implies 'a \text{ dset} \rangle$ **is** $\langle \lambda p. \downarrow_d (\text{range } \text{Infinite} - p) \rangle$

by (*rule* *definitive-dprefixes*)

lemma *property-complement*[*simp*]: $\langle \text{property } (\text{complement } X) = \text{UNIV} - \text{property } X \rangle$

by (*transfer*, *force* *simp*: *infinites-dprefixes*[*simplified* *infinites-def*] *infinites-def* *split*: *trace.split-asm* *trace.split*)

```
end  
theory LinearTemporalLogic  
imports Traces AnswerIndexedFamilies Main  
begin
```

3 Linear-time Temporal Logic

datatype (*atoms-ltl*: 'a) *ltl* =

<i>True-ltl</i>	$\langle \text{true}_l \rangle$
<i>Not-ltl</i> 'a <i>ltl</i>	$\langle \text{not}_l \rightarrow [85] 85 \rangle$
<i>Prop-ltl</i> 'a	$\langle \text{prop}_l'(-) \rangle$
<i>Or-ltl</i> 'a <i>ltl</i> 'a <i>ltl</i>	$\langle \text{- or}_l \rightarrow [82,82] 81 \rangle$
<i>And-ltl</i> 'a <i>ltl</i> 'a <i>ltl</i>	$\langle \text{- and}_l - \rightarrow [82,82] 81 \rangle$
<i>Next-ltl</i> 'a <i>ltl</i>	$\langle X_l \rightarrow [88] 87 \rangle$
<i>Until-ltl</i> 'a <i>ltl</i> 'a <i>ltl</i>	$\langle \text{- } U_l \text{ -} \rightarrow [84,84] 83 \rangle$

fun *lsatisfying-AiF* :: 'a \Rightarrow 'a *state infinite-trace AiF* \Rightarrow 'a *ltsat* **where**
 $\langle \text{lsat} \cdot x T = \{t. x \in L (t 0)\} \rangle$ |
 $\langle \text{lsat} \cdot x F = \{t. x \notin L (t 0)\} \rangle$

fun *x-operator* :: 'a *infinite-trace AiF* \Rightarrow 'a *infinite-trace AiF* \Rightarrow 'a *X* **where**
 $\langle X \cdot t T = \{x \mid x. \text{itdrop } 1 x \in (t T)\} \rangle$ |
 $\langle X \cdot t F = \{x \mid x. \text{itdrop } 1 x \in (t F)\} \rangle$

fun *u-operator* :: 'a *infinite-trace AiF* \Rightarrow 'a *infinite-trace AiF* \Rightarrow 'a *infinite-trace AiF* **(infix**
 $\langle U \cdot \rangle$ 61) **where**
 $\langle (a U \cdot b) T = \{x \mid x. \exists k. (\forall i < k. \text{itdrop } i x \in (a T)) \wedge \text{itdrop } k x \in (b T)\} \rangle$ |
 $\langle (a U \cdot b) F = \{x \mid x. \forall k. (\exists i < k. \text{itdrop } i x \in (a F)) \vee \text{itdrop } k x \in (b F)\} \rangle$

fun *ltl-semantic* :: 'a *ltl* \Rightarrow 'a *state infinite-trace AiF* \Rightarrow 'a $\llbracket - \rrbracket_l$ **where**
 $\langle \llbracket \text{true}_l \rrbracket_l = T \cdot \rangle$ |
 $\langle \llbracket \text{not}_l \varphi \rrbracket_l = \neg \cdot \llbracket \varphi \rrbracket_l \rangle$ |
 $\langle \llbracket \text{prop}_l(a) \rrbracket_l = \text{lsat} \cdot a \rangle$ |
 $\langle \llbracket \varphi \text{ or}_l \psi \rrbracket_l = \llbracket \varphi \rrbracket_l \vee \cdot \llbracket \psi \rrbracket_l \rangle$ |
 $\langle \llbracket \varphi \text{ and}_l \psi \rrbracket_l = \llbracket \varphi \rrbracket_l \wedge \cdot \llbracket \psi \rrbracket_l \rangle$ |
 $\langle \llbracket X_l \varphi \rrbracket_l = X \cdot \llbracket \varphi \rrbracket_l \rangle$ |
 $\langle \llbracket \varphi U_l \psi \rrbracket_l = \llbracket \varphi \rrbracket_l U \cdot \llbracket \psi \rrbracket_l \rangle$

lemma *excluded-middle-ltl'* :
shows $\langle (\Gamma \notin \llbracket \varphi \rrbracket_l T) \longleftrightarrow (\Gamma \in \llbracket \varphi \rrbracket_l F) \rangle$
and $\langle (\Gamma \notin \llbracket \varphi \rrbracket_l F) \longleftrightarrow (\Gamma \in \llbracket \varphi \rrbracket_l T) \rangle$
proof (*induct* $\langle \varphi \rangle$ *arbitrary*: $\langle \Gamma \rangle$)
qed *auto*

lemma *excluded-middle-ltl*: $\langle \Gamma \in \llbracket \varphi \rrbracket_l T \vee \Gamma \in \llbracket \varphi \rrbracket_l F \rangle$
using *excluded-middle-ltl'* **by** *blast*

definition *false-ltl* ($\langle \text{false}_l \rangle$) **where**
false-ltl-def[*simp*]: $\langle \text{false}_l = \text{not}_l \text{true}_l \rangle$

definition *implies-ctl* :: $\langle 'a \text{ ctl} \Rightarrow 'a \text{ ltl} \Rightarrow 'a \text{ ltl} \rangle$ (**infix** $\langle \text{implies}_l \rangle$ 80) **where**
implies-ctl-def[simp]: $\langle \varphi \text{ implies}_l \psi = (\text{not}_l \varphi \text{ or}_l \psi) \rangle$

definition *final-ctl* :: $\langle 'a \text{ ltl} \Rightarrow 'a \text{ ltl} \rangle$ ($\langle F_l \rightarrow \rangle$) **where**
final-ctl-def[simp]: $\langle F_l \varphi = (\text{true}_l U_l \varphi) \rangle$

definition *global-ctl* :: $\langle 'a \text{ ltl} \Rightarrow 'a \text{ ltl} \rangle$ ($\langle G_l \rightarrow \rangle$) **where**
global-ctl-def[simp]: $\langle G_l \varphi = (\text{not}_l F_l (\text{not}_l \varphi)) \rangle$

3.1 Linear temporal logic equivalence

fun *ctl-semantics'* :: $\langle 'a \text{ state infinite-trace} \Rightarrow 'a \text{ ltl} \Rightarrow \text{bool} \rangle$ (**infix** $\langle \models_l \rangle$ 60) **where**
 $\langle \Gamma \models_l \text{true}_l = \text{True} \rangle$
 $\langle \Gamma \models_l \text{not}_l \varphi = (\neg \Gamma \models_l \varphi) \rangle$
 $\langle \Gamma \models_l \text{prop}_l(a) = (a \in L (\Gamma \theta)) \rangle$
 $\langle \Gamma \models_l \varphi \text{ or}_l \psi = (\Gamma \models_l \varphi \vee \Gamma \models_l \psi) \rangle$
 $\langle \Gamma \models_l \varphi \text{ and}_l \psi = (\Gamma \models_l \varphi \wedge \Gamma \models_l \psi) \rangle$
 $\langle \Gamma \models_l (X_l \varphi) = \text{itdrop } 1 \Gamma \models_l \varphi \rangle$
 $\langle \Gamma \models_l (\varphi U_l \psi) = (\exists k. (\forall i < k. \text{itdrop } i \Gamma \models_l \varphi) \wedge \text{itdrop } k \Gamma \models_l \psi) \rangle$

3.2 Linear temporal logic lemmas

lemma $\langle \llbracket F_l (F_l \varphi) \rrbracket_l = \llbracket F_l \varphi \rrbracket_l \rangle$

proof (*rule AiF-cases*)

show $\langle \llbracket F_l F_l \varphi \rrbracket_l T = \llbracket F_l \varphi \rrbracket_l T \rangle$
apply (*clarsimp intro!: set-eqI, rule*)
apply *auto[1]*
by (*clarsimp, metis add-0*)

next

show $\langle \llbracket F_l F_l \varphi \rrbracket_l F = \llbracket F_l \varphi \rrbracket_l F \rangle$
apply (*clarsimp intro!: set-eqI, rule*)
apply (*clarsimp, metis add-0*)
by *simp*

qed

lemma *ctl-equivalence*:

shows $\langle \Gamma \models_l \varphi = (\Gamma \in \llbracket \varphi \rrbracket_l T) \rangle$
and $\langle (\neg \Gamma \models_l \varphi) = (\Gamma \in \llbracket \varphi \rrbracket_l F) \rangle$

proof (*induct* $\langle \varphi \rangle$ *arbitrary*: $\langle \Gamma \rangle$)

qed *auto*

end

theory *LTL3*

imports *Main Traces AnswerIndexedFamilies LinearTemporalLogic*

begin

4 LTL3: Semantics, Equivalence and Formula Progression

type-synonym $'a \text{ AiF}_3 = \langle \text{answer} \Rightarrow 'a \text{ state dset} \rangle$

primrec *and-AiF₃* :: $\langle 'a \text{ AiF}_3 \Rightarrow 'a \text{ AiF}_3 \Rightarrow 'a \text{ AiF}_3 \rangle$ (**infixl** $\langle \wedge_3 \cdot \rangle$ 60) **where**

$\langle (a \wedge_3 \cdot b) T = a T \sqcap b T \rangle$
 $\mid \langle (a \wedge_3 \cdot b) F = a F \sqcup b F \rangle$

primrec *or-AiF₃* :: $\langle 'a \text{ AiF}_3 \Rightarrow 'a \text{ AiF}_3 \Rightarrow 'a \text{ AiF}_3 \rangle$ (**infixl** $\langle \vee_3 \cdot \rangle$ 59) **where**

$\langle (a \vee_3 \cdot b) T = a T \sqcup b T \rangle$
 $\mid \langle (a \vee_3 \cdot b) F = a F \sqcap b F \rangle$

fun *not-AiF₃* :: $\langle 'a \text{ AiF}_3 \Rightarrow 'a \text{ AiF}_3 \rangle$ ($\langle \neg_3 \cdot \neg \rangle$) **where**

$\langle (\neg_3 \cdot a) T = a F \rangle$
 $\mid \langle (\neg_3 \cdot a) F = a T \rangle$

fun *univ-AiF₃* :: $\langle 'a \text{ AiF}_3 \rangle$ ($\langle T_3 \cdot \rangle$) **where**

$\langle T_3 \cdot T = \Sigma \infty \rangle$
 $\mid \langle T_3 \cdot F = \emptyset \rangle$

fun *lsatisfying-AiF₃* :: $\langle 'a \Rightarrow 'a \text{ AiF}_3 \rangle$ ($\langle \text{lsat}_3 \cdot \rangle$) **where**

$\langle \text{lsat}_3 \cdot x T = \text{compr } (\lambda t. t \neq \varepsilon \wedge x \in L (\text{thead } t)) \rangle$
 $\mid \langle \text{lsat}_3 \cdot x F = \text{compr } (\lambda t. t \neq \varepsilon \wedge x \notin L (\text{thead } t)) \rangle$

fun *x₃-operator* :: $\langle 'a \text{ AiF}_3 \Rightarrow 'a \text{ AiF}_3 \rangle$ ($\langle X_3 \cdot \rangle$) **where**

$\langle X_3 \cdot t T = \text{prepend } (t T) \rangle$
 $\mid \langle X_3 \cdot t F = \text{prepend } (t F) \rangle$

fun *iterate* :: $\langle ('a \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow 'a) \rangle$ **where**

$\langle \text{iterate } f \ 0 \ x = x \rangle$
 $\langle \text{iterate } f \ (\text{Suc } n) \ x = f (\text{iterate } f \ n \ x) \rangle$

primrec *u₃-operator* :: $\langle 'a \text{ AiF}_3 \Rightarrow 'a \text{ AiF}_3 \Rightarrow 'a \text{ AiF}_3 \rangle$ (**infix** $\langle U_3 \cdot \rangle$ 61) **where**

$\langle (a U_3 \cdot b) T = \bigsqcup (\text{range } (\lambda i. \text{iterate } (\lambda x. \text{prepend } x \sqcap a T) \ i \ (b T))) \rangle$
 $\mid \langle (a U_3 \cdot b) F = \bigsqcap (\text{range } (\lambda i. \text{iterate } (\lambda x. \text{prepend } x \sqcup a F) \ i \ (b F))) \rangle$

fun *triv-true* :: $\langle 'a \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{triv-true } x = (\forall s. x \in L \ s) \rangle$

fun *triv-false* :: $\langle 'a \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{triv-false } x = (\forall s. x \notin L \ s) \rangle$

```

fun nontrivial :: ‹'a ⇒ bool› where
  ‹nontrivial x = ((∃ s. x ∈ L s) ∧ (∃ t. x ∉ L t)›

fun zero-length :: ‹'a trace ⇒ bool› where
  ‹zero-length (Finite t) = (length t = 0)›
  | ‹zero-length (Infinite t) = False›

fun ltl-semantic3 :: ‹'a ltl ⇒ 'a AiF3› (‹[-]3›) where
  ‹[[ truel ]3 = T3›
  | ‹[[ notl φ ]3 = ¬3• [[φ]3›
  | ‹[[ propl(a) ]3 = lsat3• a›
  | ‹[[ φ orl ψ ]3 = [[φ]3 ∨3• [[ψ]3›
  | ‹[[ φ andl ψ ]3 = [[φ]3 ∧3• [[ψ]3›
  | ‹[[ Xl φ ]3 = X3• [[φ]3›
  | ‹[[ φ Ul ψ ]3 = [[φ]3 U3• [[ψ]3›

```

4.1 LTL/LTL3 equivalence

```

declare dset.Inf-insert[simp del]
declare dset.Sup-insert[simp del]

```

```

lemma itdrop-all-split:
  assumes ‹x ∈ A› and ‹∀ i < k. itdrop (Suc i) x ∈ A›
  shows ‹i < Suc k ⇒ itdrop i x ∈ A›
using assms proof (cases ‹i›)
qed (auto simp: itdrop-def)

```

```

lemma itdrop-exists-split[simp]:
  shows ‹(∃ i < Suc k. itdrop i x ∈ A) ⟷ (∃ i < k. itdrop (Suc i) x ∈ A) ∨ x ∈ A›
proof (rule iffI)
  { fix i
    assume ‹i < Suc k› ‹itdrop i x ∈ A› ‹x ∉ A›
    then have ‹∃ i < k. itdrop (Suc i) x ∈ A›
    proof (cases ‹i›)
      qed auto
  } then show ‹∃ i < Suc k. itdrop i x ∈ A ⇒ (∃ i < k. itdrop (Suc i) x ∈ A) ∨ x ∈ A› by auto
next
  assume ‹(∃ i < k. itdrop (Suc i) x ∈ A) ∨ x ∈ A›
  then show ‹∃ i < Suc k. itdrop i x ∈ A›
    by auto
qed

```

```

lemma until-iterate :
  ‹{x. ∃ k. (∀ i < k. itdrop i x ∈ A) ∧ itdrop k x ∈ B} = ⋃ (range (λk. iterate (λx. iprepend x ∩ A) k B))›
proof (rule set-eqI; rule iffI)
  fix x
  { fix k
    assume ‹∀ i < k. itdrop i x ∈ A› and ‹itdrop k x ∈ B›

```

```

then have  $\langle x \in \text{iterate } (\lambda x. \text{iprepend } x \cap A) k B \rangle$ 
proof (induct  $\langle k \rangle$  arbitrary:  $\langle x \rangle$ )
  case 0
  then show  $\langle ?case \rangle$  by simp
next
  case (Suc k)
  from this(2,3) show  $\langle ?case \rangle$ 
  by (auto intro!: Suc.hyps[where  $x = \langle \text{itdrop } 1 x \rangle$ , simplified])
qed }
then show  $\langle x \in \{x. \exists k. (\forall i < k. \text{itdrop } i x \in A) \wedge \text{itdrop } k x \in B\} \rangle$ 
   $\implies x \in (\bigcup k. \text{iterate } (\lambda x. \text{iprepend } x \cap A) k B) \rangle$ 
  by blast
next
fix x
{ fix k
  assume  $\langle x \in \text{iterate } (\lambda x. \text{iprepend } x \cap A) k B \rangle$ 
  then have  $\langle (\forall i < k. \text{itdrop } i x \in A) \wedge \text{itdrop } k x \in B \rangle$ 
  proof (induct  $\langle k \rangle$  arbitrary:  $\langle x \rangle$ )
    case 0
    then show  $\langle ?case \rangle$  by auto
  next
    case (Suc k)
    from this(2) show  $\langle ?case \rangle$ 
    by (auto dest: Suc.hyps[where  $x = \langle \text{itdrop } 1 x \rangle$ , simplified]
      intro: itdrop-all-split)
  qed }
  then show  $\langle x \in (\bigcup k. \text{iterate } (\lambda x. \text{iprepend } x \cap A) k B) \implies x \in \{x. \exists k. (\forall i < k. \text{itdrop } i x \in A) \wedge \text{itdrop } k x \in B\} \rangle$ 
  by blast
}
qed

```

lemma *release-iterate*:

$\langle \{u. \forall k. (\exists i < k. \text{itdrop } i u \in A) \vee \text{itdrop } k u \in B\} = \bigcap (\text{range } (\lambda i. \text{iterate } (\lambda x. \text{iprepend } x \cup A) i B)) \rangle$

proof (*rule set-eqI*; *rule iffI*)

```

fix x
{ fix i assume  $\langle \forall k. (\exists i < k. \text{itdrop } i x \in A) \vee \text{itdrop } k x \in B \rangle$ 
  then have  $\langle x \in \text{iterate } (\lambda x. \text{iprepend } x \cup A) i B \rangle$ 
  proof (induct  $\langle i \rangle$  arbitrary:  $\langle x \rangle$ )
    case 0
    then show  $\langle ?case \rangle$  by auto
  next
    case (Suc i)
    show  $\langle ?case \rangle$ 
    apply (clarsimp)
    apply (rule Suc.hyps[where  $x = \langle \text{itdrop } 1 x \rangle$ , simplified])
    using Suc(2)[THEN spec, where  $x = \langle \text{Suc } - \rangle$ , simplified]
    by auto
  qed }

```

```

then show  $\langle x \in \{u. \forall k. (\exists i < k. \text{itdrop } i \ u \in A) \vee \text{itdrop } k \ u \in B\} \implies x \in (\bigcap i. \text{iterate } (\lambda x. \text{iprepend } x \cup A) \ i \ B)\rangle$ 
  by auto
next
  fix  $x$ 
  { fix  $k$ 
    assume  $\langle \forall i. x \in \text{iterate } (\lambda x. \text{iprepend } x \cup A) \ i \ B \rangle$ 
    then have  $P$ :  $\langle \forall i. x \in \text{iterate } (\lambda x. \text{iprepend } x \cup A) \ i \ B \rangle$ 
      by blast
    assume  $\langle \text{itdrop } k \ x \notin B \rangle$  with  $P$  have  $\langle \exists i < k. \text{itdrop } i \ x \in A \rangle$ 
    proof (induct  $\langle k \rangle$  arbitrary:  $\langle x \rangle$ )
      case  $0$ 
        then show  $\langle ?case \rangle$  by (simp, metis iterate.simps(1))
      next
        case (Suc  $k$ )
          from this(3) show  $\langle ?case \rangle$ 
            apply clarsimp
            apply (rule Suc.hyps[where  $x = \langle \text{itdrop } 1 \ x \rangle$ , simplified])
            using Suc(2)[THEN spec, where  $x = \langle \text{Suc } - \rangle$ ]
            by auto
          qed }
    then show  $\langle x \in (\bigcap i. \text{iterate } (\lambda x. \text{iprepend } x \cup A) \ i \ B) \implies x \in \{u. \forall k. (\exists i < k. \text{itdrop } i \ u \in A) \vee \text{itdrop } k \ u \in B\}\rangle$ 
      by auto
    qed
  }

```

lemma *property-until-iterate*:

```

 $\langle \text{property } (\text{iterate } (\lambda x. \text{prepend } x \sqcap A) \ k \ B) = \text{iterate } (\lambda x. \text{iprepend } x \cap \text{property } A) \ k \ (\text{property } B) \rangle$ 
by (induct  $\langle k \rangle$ , auto simp: property-Inter property-prepend)

```

lemma *property-release-iterate*:

```

 $\langle \text{property } (\text{iterate } (\lambda x. \text{prepend } x \sqcup A) \ k \ B) = \text{iterate } (\lambda x. \text{iprepend } x \cup \text{property } A) \ k \ (\text{property } B) \rangle$ 
by (induct  $\langle k \rangle$ , auto simp: property-Union property-prepend)

```

lemma *ltl3-equiv-ltl*:

```

shows  $\langle \text{property } (\llbracket \varphi \rrbracket_3 T) = \llbracket \varphi \rrbracket_l T \rangle$ 
and  $\langle \text{property } (\llbracket \varphi \rrbracket_3 F) = \llbracket \varphi \rrbracket_l F \rangle$ 
proof (induct  $\langle \varphi \rangle$ )
  case True-ltl
  {
    case  $1$ 
    then show  $\langle ?case \rangle$  by (simp, transfer, simp)
  }
  next
  case  $2$ 
  then show  $\langle ?case \rangle$  by (simp, transfer, simp)
  }
next

```

```

case (Not-ltl  $\varphi$ )
{
  case 1
  then show  $\langle ?case \rangle$  using Not-ltl by simp
next
  case 2
  then show  $\langle ?case \rangle$  using Not-ltl by simp
}
next
case (Prop-ltl  $x$ )
{
  case 1
  then show  $\langle ?case \rangle$ 
  apply simp
  apply transfer
  apply (simp add: infinites-dprefixes)
  apply (clarsimp simp add: infinites-def split: trace.split-asm trace.split)
  apply (rule set-eqI, rule iffI)
  apply (clarsimp split: trace.split-asm trace.split)
  apply (metis zero-length.cases)
  apply (clarsimp split: trace.split-asm trace.split)
  by (metis Traces.append.simps(3) append-is-empty(2) trace.distinct(1) trace.inject(2))
next
  case 2
  then show  $\langle ?case \rangle$ 
  apply simp
  apply transfer
  apply (simp add: infinites-dprefixes)
  apply (clarsimp simp add: infinites-def split: trace.split-asm trace.split)
  apply (rule set-eqI, rule iffI)
  apply (clarsimp split: trace.split-asm trace.split)
  apply (metis zero-length.cases)
  apply (clarsimp split: trace.split-asm trace.split)
  by (metis Traces.append.simps(3) append-is-empty(2) trace.distinct(1) trace.inject(2))
}
next
case (Or-ltl  $\varphi_1 \varphi_2$ )
{
  case 1
  then show  $\langle ?case \rangle$  by (simp add: property-Union Or-ltl)
next
  case 2
  then show  $\langle ?case \rangle$  by (simp add: property-Inter Or-ltl)
}
next
case (And-ltl  $\varphi_1 \varphi_2$ )
{
  case 1
  then show  $\langle ?case \rangle$  by (simp add: property-Inter And-ltl)
}

```

```

next
  case 2
  then show ⟨?case⟩ by (simp add: property-Union And-ltl)
}
next
case (Next-ltl φ)
{
  case 1
  then show ⟨?case⟩ by (simp add: property-prepend Next-ltl iprepend-def)
  next
  case 2
  then show ⟨?case⟩ by (simp add: property-prepend Next-ltl iprepend-def)
}
next
case (Until-ltl φ1 φ2)
{
  case 1
  then show ⟨?case⟩
  apply (simp add: Until-ltl[THEN sym] property-Union image-Collect property-until-iterate)
  using until-iterate[simplified] by blast
  next
  case 2
  then show ⟨?case⟩
  apply (simp add: Until-ltl[THEN sym] property-Inter image-Collect property-release-iterate)
  using release-iterate[simplified] by metis
}
qed

```

4.2 Equivalence to LTL3 of Bauer et al.

lemma *extension-lemma*: $\langle \text{in-dset } t \ A = (\forall \omega. t \frown \text{Infinite } \omega \in \text{Infinite } \text{'property } A) \rangle$

proof *transfer*

fix t and $A :: \langle 'a \text{ trace set} \rangle$

assume $D :: \langle \text{definitive } A \rangle$

show $\langle t \in A = (\forall \omega. t \frown \text{Infinite } \omega \in \text{Infinite } \text{'infinites } A) \rangle$

proof (rule *iffI*)

assume $\langle t \in A \rangle$

with D have $D' :: \langle \uparrow t \subseteq A \rangle$ by (rule *definitive-contains-extensions*)

{ fix ω have $\langle t \frown \text{Infinite } \omega \in A \rangle$

by (rule *subsetD[OF D']*, force simp add: *extensions-def*)

} then have $\langle \forall \omega. t \frown \text{Infinite } \omega \in A \rangle$ by *auto*

thus $\langle \forall \omega. t \frown \text{Infinite } \omega \in \text{Infinite } \text{'infinites } A \rangle$

by (simp add: *infinites-append-right infinities-alt*)

next

assume $\langle \forall \omega. t \frown \text{Infinite } \omega \in \text{Infinite } \text{'infinites } A \rangle$ then

have *inA*: $\langle \forall \omega. t \frown \text{Infinite } \omega \in A \rangle$

by (simp add: *infinities-alt infinities-append-right*)

have $\langle \uparrow t \subseteq \downarrow_s A \rangle$

proof –

```

{ fix u
  obtain  $\omega :: \langle 'a \text{ infinite-trace} \rangle$  where  $\langle \text{Infinite } \omega = u \frown \text{Infinite undefined} \rangle$ 
  by (cases  $\langle u \rangle$ ; simp)
  then have  $\langle \exists v. (t \frown u) \frown v \in A \rangle$ 
  using  $\text{inA}[THEN \text{spec}, \text{where } x = \langle \omega \rangle]$  by (metis trace.assoc)
} thus  $\langle ?thesis \rangle$  unfolding extensions-def prefix-closure-def prefixes-def by auto
qed
with  $D$  show  $\langle t \in A \rangle$  by (rule definitive-elemI)
qed
qed

```

lemma extension:

```

shows  $\langle \text{in-dset } t \text{ (ltl-semantic}_3 \varphi T) = (\forall \omega. (t \frown \text{Infinite } \omega) \in \text{Infinite } \langle \text{ltl-semantic } \varphi T \rangle) \rangle$ 
and  $\langle \text{in-dset } t \text{ (ltl-semantic}_3 \varphi F) = (\forall \omega. (t \frown \text{Infinite } \omega) \in \text{Infinite } \langle \text{ltl-semantic } \varphi F \rangle) \rangle$ 
by (simp-all add: ltl3-equiv-ltl[THEN sym] extension-lemma)

```

4.3 Formula Progression

```

fun progress ::  $\langle 'a \text{ ltl} \Rightarrow 'a \text{ state} \Rightarrow 'a \text{ ltl} \rangle$  where
   $\langle \text{progress true}_l \sigma = \text{true}_l \rangle$ 
|  $\langle \text{progress (not}_l \varphi) \sigma = \text{not}_l (\text{progress } \varphi) \sigma \rangle$ 
|  $\langle \text{progress (prop}_l(a)) \sigma = (\text{if } a \in L \sigma \text{ then true}_l \text{ else not}_l \text{true}_l) \rangle$ 
|  $\langle \text{progress } (\varphi \text{ or}_l \psi) \sigma = (\text{progress } \varphi \sigma) \text{ or}_l (\text{progress } \psi \sigma) \rangle$ 
|  $\langle \text{progress } (\varphi \text{ and}_l \psi) \sigma = (\text{progress } \varphi \sigma) \text{ and}_l (\text{progress } \psi \sigma) \rangle$ 
|  $\langle \text{progress } (X_l \varphi) \sigma = \varphi \rangle$ 
|  $\langle \text{progress } (\varphi U_l \psi) \sigma = (\text{progress } \psi \sigma) \text{ or}_l ((\text{progress } \varphi \sigma) \text{ and}_l (\varphi U_l \psi)) \rangle$ 

```

```

lemma unroll-Union:  $\langle \bigsqcup (\text{range } P) = P \ 0 \sqcup (\bigsqcup (\text{range } (P \circ \text{Suc}))) \rangle$ 
apply (rule definitives-inverse-eqI)
apply (simp add: property-Union)
apply (rule dset.order-antisym)
apply (clarsimp intro!: definitives-mono; metis not0-implies-Suc)
by (force intro: definitives-mono)

```

```

lemma unroll-Inter:  $\langle \bigsqcap (\text{range } P) = P \ 0 \sqcap (\bigsqcap (\text{range } (P \circ \text{Suc}))) \rangle$ 
apply (rule definitives-inverse-eqI)
apply (simp add: property-Inter)
apply (rule dset.order-antisym)
apply (force intro: definitives-mono)
by (clarsimp intro!: definitives-mono; metis not0-implies-Suc)

```

```

lemma iterates-nonempty:  $\langle \text{range } (\lambda i. \text{iterate } f \ i \ X) \neq \{\} \rangle$ 
by blast

```

```

lemma until-cont:  $\langle A \neq \{\} \implies \text{prepend } (\bigsqcup A) \sqcap X = \bigsqcup ((\lambda x. \text{prepend } x \sqcap X) \langle A \rangle) \rangle$ 
by (simp add: prepend-Union[THEN sym] dset.SUP-inf)

```

```

lemma release-cont:  $\langle A \neq \{\} \implies \text{prepend } (\bigsqcap A) \sqcup X = \bigsqcap ((\lambda x. \text{prepend } x \sqcup X) \langle A \rangle) \rangle$ 

```

by (simp add: prepend-Inter[THEN sym] dset.INF-sup)

lemma *iterate-unroll-Inter*:

assumes $\langle \bigwedge A. A \neq \{\} \implies f (\bigcap A) = \bigcap (f ` A) \rangle$

shows $\langle \bigcap (\text{range } (\lambda i. \text{iterate } f \ i \ X)) = X \cap f (\bigcap (\text{range } (\lambda i. \text{iterate } f \ i \ X))) \rangle$

apply (subst unroll-Inter)

by (force simp: assms[OF iterates-nonempty] property-Inter intro: definitives-inverse-eqI)

lemma *iterate-unroll-Union*:

assumes $\langle \bigwedge A. A \neq \{\} \implies f (\bigcup A) = \bigcup (f ` A) \rangle$

shows $\langle \bigcup (\text{range } (\lambda i. \text{iterate } f \ i \ X)) = X \cup f (\bigcup (\text{range } (\lambda i. \text{iterate } f \ i \ X))) \rangle$

apply (subst unroll-Union)

by (force simp: assms[OF iterates-nonempty] property-Union intro: definitives-inverse-eqI)

lemma *inf-inf*: $\langle x \cap (y \cap z) = (x \cap y) \cap (x \cap z) \rangle$

by (simp add: dset.inf-assoc dset.inf-left-commute)

theorem *progression-tf* :

$\langle \text{prepend } (\llbracket \text{progress } \varphi \ \sigma \rrbracket_3 T) \cap \text{compr } (\lambda t. t \neq \varepsilon \wedge \text{thead } t = \sigma) \sqsubseteq \llbracket \varphi \rrbracket_3 T \rangle$

$\langle \text{prepend } (\llbracket \text{progress } \varphi \ \sigma \rrbracket_3 F) \cap \text{compr } (\lambda t. t \neq \varepsilon \wedge \text{thead } t = \sigma) \sqsubseteq \llbracket \varphi \rrbracket_3 F \rangle$

proof (induct $\langle \varphi \rangle$)

case *True-ltl*

{

case 1

then show $\langle ?case \rangle$ **by** *simp*

next

case 2

then show $\langle ?case \rangle$ **by** (*simp, transfer, simp add: prepend'-def*)

}

next

case (*Not-ltl* φ)

{

case 1

then show $\langle ?case \rangle$ **using** *Not-ltl* **by** *simp*

next

case 2

then show $\langle ?case \rangle$ **using** *Not-ltl* **by** *simp*

}

next

case (*Prop-ltl* x)

{

case 1

then show $\langle ?case \rangle$

by (*simp, transfer, auto simp: prepend'-def intro: dprefixes-mono[THEN subsetD, rotated]*)

next

case 2

```

    then show ⟨?case⟩
      by (simp, transfer, auto simp: prepend'-def intro: dprefixes-mono[THEN subsetD, rotated])
  }
next
case (Or-ctl  $\varphi_1 \varphi_2$ )
{
  case 1
  then show ⟨?case⟩
    apply (simp add: prepend-Union[THEN sym])
    using Or-ctl(1, 3)
    by (metis (no-types, lifting) dset.inf-sup-distrib2 dset.sup-mono)
next
  case 2
  then show ⟨?case⟩
    apply (simp add: prepend-Inter[THEN sym])
    using Or-ctl(2,4)
    by (meson dset.dual-order.refl dset.dual-order.trans dset.inf.coboundedI2
        dset.inf-le1 dset.inf-mono)
}
next
case (And-ctl  $\varphi_1 \varphi_2$ )
{
  case 1
  then show ⟨?case⟩
    apply (simp add: prepend-Inter[THEN sym])
    using And-ctl(1,3)
    by (meson dset.dual-order.trans dset.inf-le1 dset.inf-le2 dset.le-infI)
next
  case 2
  then show ⟨?case⟩
    apply (simp add: prepend-Union[THEN sym])
    using And-ctl(2, 4)
    by (metis (no-types, lifting) dset.inf-sup-distrib2 dset.sup-mono)
}
next
case (Next-ctl  $\varphi$ )
{
  case 1
  then show ⟨?case⟩ by simp
next
  case 2
  then show ⟨?case⟩ by simp
}
next
case (Until-ctl  $\varphi_1 \varphi_2$ )
{
  case 1
  then show ⟨?case⟩
    apply (simp only: progress.simps)

```

```

apply (simp add: prepend-Union[THEN sym] prepend-Inter[THEN sym])
  apply (subst dset.inf-commute)
apply (subst dset.distrib(3))
apply (rule dset.order-trans)
  apply (rule dset.sup-mono[OF - dset.order-refl])
apply (subst dset.inf-commute)
  apply (rule Until-ltl(3))
apply (subst dset.inf-assoc[THEN sym])
apply (rule dset.order-trans)
  apply (rule dset.sup-mono[OF dset.order-refl])
  apply (rule dset.inf-mono[OF - dset.order-refl])
  apply (subst dset.inf-commute)
  apply (rule Until-ltl(1))
apply (subst iterate-unroll-Union[OF until-cont], simp)
by (simp add: dset.inf-commute prepend-Union)
next
case 2
then show ⟨?case⟩
  apply simp
  apply (subst prepend-Inter[THEN sym] prepend-Union[THEN sym], simp)
  apply (subst dset.inf-commute)
  apply (subst inf-inf)
  apply (rule dset.order-trans)
  apply (rule dset.inf-mono)
  apply (subst dset.inf-commute)
  apply (rule Until-ltl(4))
apply (simp add: prepend-Union[THEN sym])
apply (subst dset.distrib(3))
  apply (rule dset.sup-mono)
apply (subst dset.inf-commute)
  apply (rule Until-ltl(2))
  apply (rule dset.le-infI2, rule dset.order-refl)
apply (subst iterate-unroll-Inter[OF release-cont,simplified]; simp)
by (metis dset.inf-le2 dset.sup-commute)
}
qed

```

```

theorem progression-tf' :
  ⟨ $\llbracket \varphi \rrbracket_3 T \sqcap \text{compr } (\lambda t. t \neq \varepsilon \wedge \text{thead } t = \sigma) \sqsubseteq \text{prepend } (\llbracket \text{progress } \varphi \sigma \rrbracket_3 T) \rangle$ 
  ⟨ $\llbracket \varphi \rrbracket_3 F \sqcap \text{compr } (\lambda t. t \neq \varepsilon \wedge \text{thead } t = \sigma) \sqsubseteq \text{prepend } (\llbracket \text{progress } \varphi \sigma \rrbracket_3 F) \rangle$ 
proof (induct ⟨ $\varphi$ ⟩)
  case True-ltl
  {
  case 1
  then show ⟨?case⟩ by (simp, transfer, simp add: prepend'-def)
  next
  case 2
  then show ⟨?case⟩ by simp
  }

```

```

next
  case (Not-ltl  $\varphi$ )
  {
    case 1
    then show  $\langle ?case \rangle$  using Not-ltl by simp
  next
    case 2
    then show  $\langle ?case \rangle$  using Not-ltl by simp
  }
next
  case (Prop-ltl  $x$ )
  {
    case 1
    then show  $\langle ?case \rangle$  apply simp
      apply transfer
      apply clarsimp
      apply (clarsimp simp: prepend'-def)
      apply (subst compr'-inter-thead)
      by (metis (mono-tags, lifting) Collect-empty-eq dprefixes-empty)
  next
    case 2
    then show  $\langle ?case \rangle$ 
      apply simp
      apply transfer
      apply (clarsimp simp: prepend'-def)
      apply (subst compr'-inter-thead)
      by (metis (mono-tags, lifting) Collect-empty-eq dprefixes-empty)
  }
next
  case (Or-ltl  $\varphi_1 \varphi_2$ )
  {
    case 1
    then show  $\langle ?case \rangle$ 
      apply (simp add: prepend-Union[THEN sym])
      using Or-ltl(1,3)
      by (metis (no-types, lifting) dset.inf-sup-distrib2 dset.sup-mono)
  next
    case 2
    then show  $\langle ?case \rangle$ 
      apply (simp add: prepend-Inter[THEN sym])
      using Or-ltl(2,4)
      by (meson dset.dual-order.refl dset.dual-order.trans dset.inf.coboundedI2 dset.inf-le1 dset.inf-mono)
  }
next
  case (And-ltl  $\varphi_1 \varphi_2$ )
  {
    case 1
    then show  $\langle ?case \rangle$ 
      apply (simp add: prepend-Inter[THEN sym])
  }

```

```

    using And-ltl(1,3)
    by (meson dset.dual-order.refl dset.dual-order.trans dset.le-inf-iff)
next
case 2
then show <?case>
  apply (simp add: prepend-Union[THEN sym])
  using And-ltl(2,4)
  by (metis (no-types,lifting) dset.inf-sup-distrib2 dset.sup-mono)
}
next
case (Next-ltl  $\varphi$ )
{
  case 1
  then show <?case> using Next-ltl by simp
next
  case 2
  then show <?case> using Next-ltl by simp
}
next
case (Until-ltl  $\varphi_1 \varphi_2$ )
{
  case 1
  then show <?case>
  unfolding ltl-semantics3.simps u3-operator.simps
    ltl-semantics.simps progress.simps u-operator.simps or-AiF3.simps and-AiF3.simps
  apply (simp add: full-SetCompr-eq prepend-Union[THEN sym])
  apply (rule dset.order-trans[rotated])
  apply (rule dset.sup-mono [OF - dset.order-refl], rule Until-ltl(3))
  apply (simp add: prepend-Inter[THEN sym])
  apply (rule dset.order-trans[rotated])
  apply (rule dset.sup-mono [OF dset.order-refl])
  apply (rule dset.inf-mono [OF - dset.order-refl])
  apply (rule Until-ltl(1))
  apply (subst iterate-unroll-Union[OF until-cont], simp)
  apply (subst dset.inf-commute)
  apply (subst dset.inf-sup-distrib1)
  apply (simp, rule conjI)
  apply (subst dset.inf-commute)
  apply auto[1]
  by (meson dset.eq-refl dset.inf.boundedI dset.le-infE dset.le-supI2)
next
  case 2
  then show <?case>
  unfolding ltl-semantics3.simps u3-operator.simps
    ltl-semantics.simps progress.simps u-operator.simps or-AiF3.simps and-AiF3.simps
  apply (simp add: full-SetCompr-eq prepend-Inter[THEN sym])
  apply (rule conjI, rule dset.order-trans[rotated])
  apply (rule Until-ltl(4))
  apply (rule dset.inf-mono; simp?)
}
}

```

```

  apply (metis iterate.simps(1) dset.Inf-lower rangeI)
  apply (simp add: prepend-Union[THEN sym])
  apply (rule dset.order-trans[rotated])
  apply (rule dset.sup-mono)
  apply (rule Until-lll(2))
  apply (rule dset.order-refl)
  apply (subst iterate-unroll-Inter[OF release-cont], simp)
  apply (simp add: prepend-Inter[THEN sym] image-image)
  apply (subst dset.inf-assoc)
  apply (subst dset.sup-inf-distrib2)
  apply (rule dset.le-infI2)
  by (simp add: dset.inf.coboundedI1 insert-commute)
}
qed

```

theorem *progression-tf'-u:*

```

  shows  $\langle \llbracket \varphi \rrbracket_3 A \sqcap \text{compr } (\lambda t. t \neq \varepsilon \wedge \text{thead } t = \sigma) \sqsubseteq \text{prepend } (\llbracket \text{progress } \varphi \sigma \rrbracket_3 A) \rangle$ 
  by (cases  $\langle A \rangle$ ; simp add: progression-tf')

```

theorem *progression-tf-u:*

```

  shows  $\langle \text{prepend } (\llbracket \text{progress } \varphi \sigma \rrbracket_3 A) \sqcap \text{compr } (\lambda t. t \neq \varepsilon \wedge \text{thead } t = \sigma) \sqsubseteq \llbracket \varphi \rrbracket_3 A \rangle$ 
  by (cases  $\langle A \rangle$ ; simp add: progression-tf)

```

lemma *fp-compr-helper:* $\langle \text{in-dset } (\text{Finite } (a \# t)) (\text{compr } (\lambda x. x \neq \varepsilon \wedge \text{thead } x = a)) \rangle$

```

  apply transfer
  apply (simp add: dprefixes-def subset-iff extensions-def prefix-closure-def prefixes-def)
  by (metis  $\varepsilon$ -def list.distinct(1) nth-Cons-0 thead.simps(1) thead-append trace.inject(1)
      trace.left-neutral trace.right-neutral)

```

theorem *fp:*

```

  shows  $\langle \text{in-dset } (\text{Finite } t) (\llbracket \varphi \rrbracket_3 A) \longleftrightarrow \llbracket \text{foldl } \text{progress } \varphi t \rrbracket_3 A = \Sigma_\infty \rangle$ 

```

proof (induct $\langle t \rangle$ arbitrary: $\langle \varphi \rangle$)

case *Nil*

then show $\langle ?\text{case} \rangle$

```

  by (rule iffI; simp add: in-dset- $\varepsilon$ [simplified  $\varepsilon$ -def] in-dset-UNIV)

```

next

case (*Cons a t*)

show $\langle ?\text{case} \rangle$

proof (simp add: Cons[**where** $\varphi = \langle \text{progress } \varphi a \rangle$, *THEN sym*], rule)

assume $\langle \text{in-dset } (\text{Finite } (a \# t)) (\llbracket \varphi \rrbracket_3 A) \rangle$

then show $\langle \text{in-dset } (\text{Finite } t) (\llbracket \text{progress } \varphi a \rrbracket_3 A) \rangle$

```

  by (force intro: in-dset-prependD in-dset-subset[OF progression-tf'-u]
      in-dset-inter fp-compr-helper)

```

next

assume $\langle \text{in-dset } (\text{Finite } t) (\llbracket \text{progress } \varphi a \rrbracket_3 A) \rangle$

then show $\langle \text{in-dset } (\text{Finite } (a \# t)) (\llbracket \varphi \rrbracket_3 A) \rangle$

```

  by (force intro: in-dset-subset[OF progression-tf-u] in-dset-inter fp-compr-helper
      in-dset-prependI[where  $x = \langle \text{Finite } u \rangle$  for  $u$ , simplified])

```

qed
qed

lemma *em-ltl*: $\langle \llbracket \varphi \rrbracket_l T = UNIV - (\llbracket \varphi \rrbracket_l F) \rangle$
by (*rule set-eqI; clarsimp simp add: subset-iff ltl-equivalence[THEN sym]*)

theorem *em*:
shows $\langle \llbracket \varphi \rrbracket_3 T = \text{complement } (\llbracket \varphi \rrbracket_3 F) \rangle$
by (*force intro: definitives-inverse-eqI simp: ltl3-equiv-ltl em-ltl*)

end

Bibliography

- [1] R. Amjad, R. van Glabbeek, and L. O'Connor. Semantics for linear-time temporal logic with finite observations. In *Proceedings of the Combined 31st International Workshop on Expressiveness in Concurrency and 21st Workshop on Structural Operational Semantics, EXPRESS/SOS 2024*, EPTCS, 2024. To appear.
- [2] F. Bacchus and F. Kabanza. *Using Temporal Logic to Control Search in a Forward Chaining Planner*, page 141153. IOS Press, 1996.
- [3] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering Methodology*, 20(4), sep 2011.
- [4] F. Kabanza and S. Thiébaux. Search control in planning for temporally extended goals. In *International Conference on Automated Planning and Scheduling*, pages 130–139. AAAI, 2005.