

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	8
2.3	Definitive Prefixes	9
2.4	Definitive Sets	10
2.5	A type for definitive sets	12
2.6	Isomorphism of definitive sets and LTL properties	13
3	Linear-time Temporal Logic	17
3.1	Linear temporal logic equivalence	18
3.2	Linear temporal logic lemmas	18
4	LTL3: Semantics, Equivalence and Formula Progression	19
4.1	LTL/LTL3 equivalence	20
4.2	Equivalence to LTL3 of Bauer et al.	21
4.3	Formula Progression	21

```
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* :: $\langle 'a \text{ plogic} \Rightarrow 'a \text{ plogic} \Rightarrow 'a \text{ plogic} \rangle$ ($\langle \text{- implies}_p \text{-} \rightarrow [81,81] \ 80 \rangle$) **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>

lemma *or-and-negation*: $\langle \llbracket \varphi \text{ or}_p \psi \rrbracket_p = \llbracket \text{not}_p ((\text{not}_p \varphi) \text{ and}_p (\text{not}_p \psi)) \rrbracket_p \rangle$

<proof>

lemma *and-or-negation*: $\langle \llbracket \varphi \text{ and}_p \psi \rrbracket_p = \llbracket \text{not}_p ((\text{not}_p \varphi) \text{ or}_p (\text{not}_p \psi)) \rrbracket_p \rangle$

<proof>

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

<proof>

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

<proof>

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>

end

theory *Traces*

imports *Main HOL.Lattices HOL.List*

begin

2 Traces and Definitive Prefixes

2.1 Traces

typedecl Σ
type-synonym $'a$ *finite-trace* = $\langle 'a$ list \rangle
type-synonym $'a$ *infinite-trace* = $\langle \text{nat} \Rightarrow 'a \rangle$
datatype $'a$ *trace* = *Finite* $\langle 'a$ *finite-trace* \rangle | *Infinite* $\langle 'a$ *infinite-trace* \rangle

fun *thead* :: $\langle 'a$ *trace* $\Rightarrow 'a \rangle$ **where**
 $\langle \text{thead } (\text{Finite } t) = t ! 0 \rangle$
| $\langle \text{thead } (\text{Infinite } t) = t 0 \rangle$

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

definition ε :: $\langle 'a$ *trace* \rangle **where**
 $\langle \varepsilon = \text{Finite } [] \rangle$

definition *singleton* :: $\langle 'a \Rightarrow 'a$ *trace* \rangle **where**
 $\langle \text{singleton } \sigma = \text{Finite } [\sigma] \rangle$

interpretation *trace*: *monoid-list* $\langle (\frown) \rangle$ $\langle \varepsilon \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-empty-suffix*:
assumes $\langle \text{Finite } xs = \text{Finite } xs \frown t \rangle$
shows $\langle t = \varepsilon \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-empty-prefix*:
assumes $\langle \text{Finite } xs = t \frown \text{Finite } xs \rangle$
shows $\langle t = \varepsilon \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-finite-suffix*:
assumes $\langle \text{Finite } xs = \text{Finite } ys \frown t \rangle$
obtains zs **where** $\langle t = \text{Finite } zs \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-finite-prefix*:
assumes $\langle \text{Finite } xs = t \frown \text{Finite } ys \rangle$

obtains zs **where** $\langle t = \text{Finite } zs \rangle$
 $\langle \text{proof} \rangle$

lemma *append-is-empty*:

assumes $\langle t \frown u = \varepsilon \rangle$

shows $\langle t = \varepsilon \rangle$

and $\langle u = \varepsilon \rangle$

$\langle \text{proof} \rangle$

fun *ttake* :: $\langle \text{nat} \Rightarrow 'a \text{ trace} \Rightarrow 'a \text{ finite-trace} \rangle$ **where**

$\langle \text{ttake } k (\text{Finite } xs) = \text{take } k \ xs \rangle$

| $\langle \text{ttake } k (\text{Infinite } xs) = \text{map } xs \ [0..<k] \rangle$

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

$\langle \text{itdrop } k \ xs = (\lambda i. \ xs \ (i + k)) \rangle$

lemma *itdrop-itdrop[simp]*: $\langle \text{itdrop } i \ (\text{itdrop } j \ x) = \text{itdrop } (i + j) \ x \rangle$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

fun *tdrop* :: $\langle \text{nat} \Rightarrow 'a \text{ trace} \Rightarrow 'a \text{ trace} \rangle$ **where**

$\langle \text{tdrop } k (\text{Finite } xs) = \text{Finite } (\text{drop } k \ xs) \rangle$

| $\langle \text{tdrop } k (\text{Infinite } xs) = \text{Infinite } (\text{itdrop } k \ xs) \rangle$

lemma *ttake-simp[simp]*: $\langle \text{ttake } (\text{length } xs) (\text{Finite } xs \frown t) = xs \rangle$

$\langle \text{proof} \rangle$

lemma *ttake-tdrop[simp]*: $\langle \text{Finite } (\text{ttake } k \ t) \frown \text{tdrop } k \ t = t \rangle$

$\langle \text{proof} \rangle$

definition *prefixes* :: $\langle 'a \text{ trace} \Rightarrow 'a \text{ trace set} \rangle$ ($\downarrow \rightarrow [80] \ 80$) **where**

$\langle \downarrow t = \{ u \mid u \ v. \ t = u \frown v \} \rangle$

definition *extensions* :: $\langle 'a \text{ trace} \Rightarrow 'a \text{ trace set} \rangle$ ($\uparrow \rightarrow [80] \ 80$) **where**

$\langle \uparrow t = \{ t \frown u \mid u. \ \text{True} \} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *prefixes-empty-least* : $\langle \varepsilon \in \downarrow t \rangle$

$\langle \text{proof} \rangle$

lemma *prefixes-infinite-greatest* : $\langle \text{Infinite } x \in \downarrow t \implies t = \text{Infinite } x \rangle$
 $\langle \text{proof} \rangle$

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

lemma *ttake-take* : $\langle \text{take } n (\text{ttake } m t) = \text{ttake } (\text{min } n m) t \rangle$
 $\langle \text{proof} \rangle$

lemma *tdrop-tdrop* : $\langle \text{tdrop } n (\text{tdrop } m t) = \text{tdrop } (n + m) t \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *ttake-prefixes* : $\langle a \leq b \implies \text{Finite } (\text{ttake } a t) \in \downarrow \text{Finite } (\text{ttake } b t) \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-directed*:

assumes $\langle \text{Finite } xs \in \downarrow t \rangle \langle \text{Finite } ys \in \downarrow t \rangle$

shows $\langle \exists zs. (xs = ys @ zs) \vee (ys = xs @ zs) \rangle$

$\langle \text{proof} \rangle$

lemma *prefixes-directed*: $\langle u \in \downarrow t \implies v \in \downarrow t \implies u \in \downarrow v \vee v \in \downarrow u \rangle$
 $\langle \text{proof} \rangle$

interpretation *extensions: order* $\langle \lambda t u. t \in \uparrow u \rangle \langle \lambda t u. t \in \uparrow u \wedge t \neq u \rangle$
 $\langle \text{proof} \rangle$

lemma *extensions-infinite[simp]*: $\langle \uparrow \text{Infinite } xs = \{ \text{Infinite } xs \} \rangle$
 $\langle \text{proof} \rangle$

lemma *extensions-empty[simp]*: $\langle \uparrow \varepsilon = \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *prefixes-empty*: $\langle \downarrow \varepsilon = \{ \varepsilon \} \rangle$
 $\langle \text{proof} \rangle$

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*: $\langle X \subseteq \downarrow_s X \rangle$
 $\langle \text{proof} \rangle$

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

lemma *prefix-closure-idem*: $\langle \downarrow_s \downarrow_s X = \downarrow_s X \rangle$
 $\langle \text{proof} \rangle$

lemma *prefix-closure-mono*: $\langle X \subseteq Y \implies \downarrow_s X \subseteq \downarrow_s Y \rangle$
 $\langle \text{proof} \rangle$

lemma *prefix-closure-union-distrib*: $\langle \downarrow_s (X \cup Y) = \downarrow_s X \cup \downarrow_s Y \rangle$
 $\langle \text{proof} \rangle$

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

lemma *prefix-closure-Inter*: $\langle \downarrow_s (\bigcap (\text{prefix-closure } 'S)) = \bigcap (\text{prefix-closure } 'S) \rangle$
 $\langle \text{proof} \rangle$

lemma *prefix-closure-inter*: $\langle \downarrow_s (\downarrow_s X \cap \downarrow_s Y) = \downarrow_s X \cap \downarrow_s Y \rangle$
 $\langle \text{proof} \rangle$

lemma *prefix-closure-UNIV*: $\langle \downarrow_s \text{UNIV} = \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

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

lemma *prefix-closure-extensions*: $\langle \downarrow_s (\uparrow t) = \uparrow t \cup \downarrow t \rangle$
 $\langle \text{proof} \rangle$

lemma *prefix-closure-prefixes*: $\langle \downarrow_s (\downarrow t) = \downarrow t \rangle$
 $\langle \text{proof} \rangle$

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* : $\langle \downarrow_d X \subseteq \downarrow_s X \rangle$
 $\langle \text{proof} \rangle$

lemma *prefix-closure-dprefixes* : $\langle \downarrow_s (\downarrow_d X) \subseteq \downarrow_s X \rangle$
 $\langle \text{proof} \rangle$

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

<proof>

lemma *dprefixes-contains-extensions*: $\langle t \in \downarrow_d X \implies \uparrow t \subseteq \downarrow_d X \rangle$
<proof>

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

lemma *dprefixes-UNIV*: $\langle \downarrow_d \text{UNIV} = \text{UNIV} \rangle$
<proof>

lemma *dprefixes-empty*: $\langle \downarrow_d \{\} = \{\} \rangle$
<proof>

lemma *dprefixes-Inter-distrib*: $\langle \downarrow_d (\bigcap S) \subseteq \bigcap (\text{dprefixes } 'S) \rangle$
<proof>

lemma *dprefixes-Inter*: $\langle \downarrow_d (\bigcap (\text{dprefixes } 'S)) = \bigcap (\text{dprefixes } 'S) \rangle$
<proof>

lemma *dprefixes-mono*:
assumes $\langle X \subseteq Y \rangle$
shows $\langle \downarrow_d X \subseteq \downarrow_d Y \rangle$
<proof>

lemma *dprefixes-inter*: $\langle \downarrow_d (\downarrow_d X \cap \downarrow_d Y) = (\downarrow_d X \cap \downarrow_d Y) \rangle$
<proof>

lemma *dprefixes-inter-distrib*: $\langle \downarrow_d (X \cap Y) \subseteq \downarrow_d X \cap \downarrow_d Y \rangle$
<proof>

2.4 Definitive Sets

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

lemma *definitive-image*: $\langle \forall X \in S. \text{definitive } X \implies \text{dprefixes } 'S = S \rangle$
<proof>

lemma *definitive-dprefixes*: $\langle \text{definitive } (\downarrow_d X) \rangle$
<proof>

lemma *definitive-contains-extensions*: $\langle \text{definitive } X \implies t \in X \implies \uparrow t \subseteq X \rangle$
<proof>

lemma *definitive-UNIV*: $\langle \text{definitive UNIV} \rangle$
<proof>

lemma *definitive-empty*: $\langle \text{definitive } \{\} \rangle$
 $\langle \text{proof} \rangle$

lemma *definitive-Inter*: $\langle \forall X \in S. \text{definitive } X \implies \text{definitive } (\bigcap S) \rangle$
 $\langle \text{proof} \rangle$

lemma *definitive-inter*: $\langle \text{definitive } X \implies \text{definitive } Y \implies \text{definitive } (X \cap Y) \rangle$
 $\langle \text{proof} \rangle$

lemma *definitive-infinite-extension*:
assumes $\langle \text{definitive } X \rangle$ **and** $\langle t \in X \rangle$
shows $\langle \exists f. \text{Infinite } f \in X \wedge t \in \downarrow \text{Infinite } f \rangle$
 $\langle \text{proof} \rangle$

lemma *definitive-elemI*:
assumes $\langle \text{definitive } X \rangle$ $\langle \uparrow t \subseteq \downarrow_s X \rangle$
shows $\langle t \in X \rangle$
 $\langle \text{proof} \rangle$

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**
 $\langle X \bigcup_d Y \equiv \bigcup_d \{X, Y\} \rangle$

lemma *dprefixes-dUnion*: $\langle \downarrow_d \bigcup_d S = \bigcup_d S \rangle$
 $\langle \text{proof} \rangle$

lemma *definitive-dUnion*: $\langle \text{definitive } (\bigcup_d S) \rangle$
 $\langle \text{proof} \rangle$

lemma *dUnion-contains-dprefixes*: $\langle t \in S \implies \downarrow_d t \subseteq \bigcup_d S \rangle$
 $\langle \text{proof} \rangle$

lemma *dUnion-contains-definitive*: $\langle X \in S \implies \text{definitive } X \implies X \subseteq \bigcup_d S \rangle$
 $\langle \text{proof} \rangle$

lemma *dUnion-empty[simp]*: $\langle \bigcup_d \{\} = \{\} \rangle$
 $\langle \text{proof} \rangle$

lemma *dUnion-least-dprefixes*: $\langle (\bigwedge X. X \in S \implies X \subseteq \downarrow_d Z) \implies \downarrow_d (\bigcup (\text{dprefixes } ' S)) \subseteq \downarrow_d Z \rangle$
 $\langle \text{proof} \rangle$

lemma *dUnion-least-definitive*:
assumes *all-defn*: $\langle \forall X \in S. \text{definitive } X \rangle$
shows $\langle (\bigwedge X. X \in S \implies X \subseteq Z) \implies \text{definitive } Z \implies \downarrow_d \bigcup S \subseteq Z \rangle$

$\langle proof \rangle$

2.5 A type for definitive sets

typedef $'a\ dset = \langle \{p :: 'a\ trace\ set.\ definitive\ p\} \rangle$

$\langle proof \rangle$

setup-lifting $type\ definition\ dset$

lift-definition $Inter\ dset :: \langle 'a\ dset\ set \Rightarrow 'a\ dset \rangle (\langle \sqcap \rangle)$ **is** $\langle \lambda\ ss.\ \bigcap\ ss \rangle$

$\langle proof \rangle$

abbreviation $inter\ dset :: \langle 'a\ dset \Rightarrow 'a\ dset \Rightarrow 'a\ dset \rangle$ (**infixl** $\langle \sqcap \rangle$ 66) **where**

$\langle X\ \sqcap\ Y \equiv \bigcap\ \{X, Y\} \rangle$

lift-definition $Union\ cset :: \langle 'a\ dset\ set \Rightarrow 'a\ dset \rangle (\langle \sqcup \rangle)$ **is** $\langle \lambda\ ss.\ \bigcup_d\ ss \rangle$

$\langle proof \rangle$

abbreviation $union\ dset :: \langle 'a\ dset \Rightarrow 'a\ dset \Rightarrow 'a\ dset \rangle$ (**infixl** $\langle \sqcup \rangle$ 65) **where**

$\langle X\ \sqcup\ Y \equiv \bigcup\ \{X, Y\} \rangle$

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

$\langle proof \rangle$

lift-definition $univ\ dset :: \langle 'a\ dset \rangle (\langle \Sigma_\infty \rangle)$ **is** $\langle UNIV \rangle$

$\langle proof \rangle$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma $in\ dset\ \varepsilon: \langle in\ dset\ \varepsilon\ A \Longrightarrow A = \Sigma_\infty \rangle$

$\langle proof \rangle$

lemma $in\ dset\ UNIV: \langle in\ dset\ x\ \Sigma_\infty \rangle$

$\langle proof \rangle$

lemma $in\ dset\ subset: \langle A \sqsubseteq B \Longrightarrow in\ dset\ x\ A \Longrightarrow in\ dset\ x\ B \rangle$

$\langle proof \rangle$

lemma *in-dset-inter*: $\langle \text{in-dset } x \ A \implies \text{in-dset } x \ B \implies \text{in-dset } x \ (A \sqcap B) \rangle$
 $\langle \text{proof} \rangle$

interpretation *dset*: *complete-lattice* $\langle \sqcap \rangle \langle \sqcup \rangle \langle (\sqcap) \rangle \langle (\sqsubseteq) \rangle \langle (\sqsupset) \rangle \langle (\sqcup) \rangle \langle \emptyset \rangle \langle \Sigma_\infty \rangle$
 $\langle \text{proof} \rangle$

2.6 Isomorphism of definitive sets and LTL properties

definition *infinites* :: $\langle 'a \ \text{trace set} \implies 'a \ \text{infinite-trace set} \rangle$ **where**
 $\langle \text{infinites } X = (\bigcup x \in X. \text{ case } x \text{ of } \text{Finite } xs \implies \{\} \mid \text{Infinite } xs \implies \{xs\}) \rangle$

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

lemma *infinites-append-right*: $\langle t \frown \text{Infinite } \omega \in \text{range Infinite} \rangle$
 $\langle \text{proof} \rangle$

lemma *infinites-prefix-closure*:
assumes $\langle \text{definitive } X \rangle$
shows $\langle \downarrow_s \text{Infinite } \text{'infinites } X = \downarrow_s X \rangle$
 $\langle \text{proof} \rangle$

lemma *infinites-UNIV[simp]*: $\langle \text{infinites } \text{UNIV} = \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *infinites-empty[simp]*: $\langle \text{infinites } \{\} = \{\} \rangle$
 $\langle \text{proof} \rangle$

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

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

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

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

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

lift-definition *definitives* :: $\langle 'a \ \text{infinite-trace set} \implies 'a \ \text{dset} \rangle$ **is** $\langle \lambda x. \downarrow_d (\text{Infinite } \text{' } x) \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

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

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

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

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

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

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

interpretation *dset*: *complete-distrib-lattice* $\langle \prod \rangle \langle \sqcup \rangle \langle (\prod) \rangle \langle (\sqsubseteq) \rangle \langle (\sqsupset) \rangle \langle (\sqcup) \rangle \langle \emptyset \rangle \langle \Sigma_\infty \rangle$
 $\langle \text{proof} \rangle$

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*: $\langle \text{itdrop } k x \in \text{iprepend } B \iff \text{itdrop } (\text{Suc } k) x \in B \rangle$
 $\langle \text{proof} \rangle$

lemmas *iprepend-itdrop-0[simp]* = *iprepend-itdrop[where k = 0,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 $\langle \bigwedge \sigma t. x = \text{singleton } \sigma \frown t \implies P \rangle$
and $\langle x = \varepsilon \implies P \rangle$
shows $\langle P \rangle$
 $\langle \text{proof} \rangle$

lemma *append-prefixes-left*: $\langle a \in \downarrow b \implies c \frown a \in \downarrow c \frown b \rangle$
 $\langle \text{proof} \rangle$

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

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

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

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

lemma *prepend'-dprefixes* :
assumes $\langle \text{definitive } X \rangle$
shows $\langle \downarrow_d \text{prepend}' X = \text{prepend}' X \rangle$
<proof>

lemma *prepend'-definitive* :
assumes $\langle \text{definitive } X \rangle$
shows $\langle \text{definitive } (\text{prepend}' X) \rangle$
<proof>

lift-definition *prepend* :: $\langle 'a \text{ dset} \Rightarrow 'a \text{ dset} \rangle$ **is** $\langle \text{prepend}' \rangle$
<proof>

lemma *prepend-Inter*: $\langle \sqcap (\text{prepend}' S) = \text{prepend}' (\sqcap S) \rangle$
<proof>

lemma *in-dset-prependD*: $\langle \text{in-dset } (\text{Finite } [a] \frown x) (\text{prepend}' A) \Longrightarrow \text{in-dset } x A \rangle$
<proof>

lemma *in-dset-prependI*: $\langle \text{in-dset } x A \Longrightarrow \text{in-dset } (\text{Finite } [a] \frown x) (\text{prepend}' A) \rangle$
<proof>

lemma *prepend'-mono*:
assumes $\langle A \subseteq B \rangle$
shows $\langle \text{prepend}' A \subseteq \text{prepend}' B \rangle$
<proof>

lemma *property-prepend*: $\langle \text{property } (\text{prepend}' X) = \text{iprepend}' (\text{property } X) \rangle$
<proof>

lemma *iprepend-Union*: $\langle \bigcup (\text{iprepend}' S) = \text{iprepend}' (\bigcup S) \rangle$
<proof>

lemma *definitives-inverse-eqI*: $\langle \text{definitives } (\text{property } X) = \text{definitives } (\text{property } Y) \Longrightarrow X = Y \rangle$
<proof>

lemma *prepend-Union*: $\langle \sqcup (\text{prepend}' S) = \text{prepend}' (\sqcup S) \rangle$
<proof>

lemma *non-empty-trace*: $\langle x \neq \varepsilon \longleftrightarrow (\exists \sigma x'. x = \text{Finite } [\sigma] \frown x') \rangle$
<proof>

lemma *thead-append*: $\langle x \neq \varepsilon \implies \text{thead } (x \frown y) = \text{thead } x \rangle$
<proof>

lemma *thead-prefix*: $\langle x \in \downarrow y \implies x \neq \varepsilon \implies \text{thead } x = \text{thead } y \rangle$
<proof>

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>

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

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

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

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* $\langle \text{lsat} \cdot \rangle$ **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* $\langle X \cdot \rangle$ **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* $\langle \text{infix } U \cdot \rangle$ **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* $\langle \llbracket - \rrbracket_l \rangle$ **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 $\langle (\Gamma \notin \llbracket \varphi \rrbracket_l F) \longleftrightarrow (\Gamma \in \llbracket \varphi \rrbracket_l T) \rangle$
 $\langle \text{proof} \rangle$

lemma *excluded-middle-ltl*: $\langle \Gamma \in \llbracket \varphi \rrbracket_l T \vee \Gamma \in \llbracket \varphi \rrbracket_l F \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \llbracket F_l (F_l \varphi) \rrbracket_l = \llbracket F_l \varphi \rrbracket_l \rangle$
 $\langle \text{proof} \rangle$

lemma *ctl-equivalence*:

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

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**
 ⟨[[*true*_l]₃ = *T*₃⟩
 | ⟨[[*not*_l *φ*]₃ = ¬₃• [[*φ*]₃⟩
 | ⟨[[*prop*_l(*a*)]₃ = *lsat*₃• *a*⟩
 | ⟨[[*φ or*_l *ψ*]₃ = [[*φ*]₃ ∨₃• [[*ψ*]₃⟩
 | ⟨[[*φ and*_l *ψ*]₃ = [[*φ*]₃ ∧₃• [[*ψ*]₃⟩
 | ⟨[[*X*_l *φ*]₃ = *X*₃• [[*φ*]₃⟩
 | ⟨[[*φ U*_l *ψ*]₃ = [[*φ*]₃ *U*₃• [[*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*⟩
 ⟨*proof*⟩

lemma *itdrop-exists-split*[*simp*]:
shows ⟨(∃ *i* < *Suc k*. *itdrop i x* ∈ *A*) ⟷ (∃ *i* < *k*. *itdrop* (*Suc i*) *x* ∈ *A*) ∨ *x* ∈ *A*⟩
 ⟨*proof*⟩

lemma *until-iterate* :
 ⟨{*x*. ∃ *k*. (∀ *i* < *k*. *itdrop i x* ∈ *A*) ∧ *itdrop k x* ∈ *B*} = ∪ (range (λ*k*. *iterate* (λ*x*. *iprepend x* ∩ *A*) *k B*)⟩
 ⟨*proof*⟩

lemma *release-iterate*:
 ⟨{*u*. ∀ *k*. (∃ *i* < *k*. *itdrop i u* ∈ *A*) ∨ *itdrop k u* ∈ *B*} = ∩ (range (λ*i*. *iterate* (λ*x*. *iprepend x* ∪ *A*) *i B*)⟩
 ⟨*proof*⟩

lemma *property-until-iterate*:
 ⟨*property* (*iterate* (λ*x*. *prepend x* ∩ *A*) *k B*) = *iterate* (λ*x*. *iprepend x* ∩ *property A*) *k* (*property B*)⟩
 ⟨*proof*⟩

lemma *property-release-iterate*:
 ⟨*property* (*iterate* (λ*x*. *prepend x* ∪ *A*) *k B*) = *iterate* (λ*x*. *iprepend x* ∪ *property A*) *k* (*property B*)⟩
 ⟨*proof*⟩

lemma *ltl3-equiv-ltl*:

shows $\langle \text{property } (\llbracket \varphi \rrbracket_3 T) = \llbracket \varphi \rrbracket_l T \rangle$

and $\langle \text{property } (\llbracket \varphi \rrbracket_3 F) = \llbracket \varphi \rrbracket_l F \rangle$

$\langle \text{proof} \rangle$

4.2 Equivalence to LTL3 of Bauer et al.

lemma *extension-lemma*: $\langle \text{in-dset } t A = (\forall \omega. t \frown \text{Infinite } \omega \in \text{Infinite } \langle \text{property } A \rangle) \rangle$

$\langle \text{proof} \rangle$

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 $\langle \text{in-dset } t (\text{ltl-semantic}_3 \varphi F) = (\forall \omega. (t \frown \text{Infinite } \omega) \in \text{Infinite } \langle \text{ltl-semantic } \varphi F \rangle) \rangle$

$\langle \text{proof} \rangle$

4.3 Formula Progression

fun *progress* :: $\langle 'a \text{ ltl} \Rightarrow 'a \text{ state} \Rightarrow 'a \text{ ltl} \rangle$ **where**

$\langle \text{progress } \text{true}_l \sigma = \text{true}_l \rangle$

$\langle \text{progress } (\text{not}_l \varphi) \sigma = \text{not}_l (\text{progress } \varphi) \sigma \rangle$

$\langle \text{progress } (\text{prop}_l(a)) \sigma = (\text{if } a \in L \sigma \text{ then } \text{true}_l \text{ else } \text{not}_l \text{true}_l) \rangle$

$\langle \text{progress } (\varphi \text{ or}_l \psi) \sigma = (\text{progress } \varphi \sigma) \text{ or}_l (\text{progress } \psi \sigma) \rangle$

$\langle \text{progress } (\varphi \text{ and}_l \psi) \sigma = (\text{progress } \varphi \sigma) \text{ and}_l (\text{progress } \psi \sigma) \rangle$

$\langle \text{progress } (X_l \varphi) \sigma = \varphi \rangle$

$\langle \text{progress } (\varphi U_l \psi) \sigma = (\text{progress } \psi \sigma) \text{ or}_l ((\text{progress } \varphi \sigma) \text{ and}_l (\varphi U_l \psi)) \rangle$

lemma *unroll-Union*: $\langle \bigsqcup (\text{range } P) = P \ 0 \sqcup (\bigsqcup (\text{range } (P \circ \text{Suc}))) \rangle$

$\langle \text{proof} \rangle$

lemma *unroll-Inter*: $\langle \bigsqcap (\text{range } P) = P \ 0 \sqcap (\bigsqcap (\text{range } (P \circ \text{Suc}))) \rangle$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *until-cont*: $\langle A \neq \{\} \Longrightarrow \text{prepend } (\bigsqcup A) \sqcap X = \bigsqcup ((\lambda x. \text{prepend } x \sqcap X) \langle A \rangle) \rangle$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *iterate-unroll-Inter*:

assumes $\langle \bigwedge A. A \neq \{\} \Longrightarrow f (\bigsqcap A) = \bigsqcap (f \langle A \rangle) \rangle$

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

$\langle \text{proof} \rangle$

lemma *iterate-unroll-Union*:

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

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

<proof>

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

<proof>

theorem *progression-tf* :

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

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

<proof>

theorem *progression-tf'* :

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

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

<proof>

theorem *progression-tf'-u*:

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

<proof>

theorem *progression-tf-u*:

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

<proof>

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

<proof>

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>

lemma *em-ltl*: $\langle \llbracket \varphi \rrbracket_l T = \text{UNIV} - (\llbracket \varphi \rrbracket_l F) \rangle$

<proof>

theorem *em*:

shows $\langle \llbracket \varphi \rrbracket_3 T = \text{complement } (\llbracket \varphi \rrbracket_3 F) \rangle$

<proof>

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.