Definitive Set Semantics for LTL3

Rayhana Amjad, Rob van Glabbeek, Liam O'Connor

March 17, 2025

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

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 (‹*true*$_p$›)
  | *Prop-plogic* ‹*'a*› (‹*prop*$_p$*'*(-*')*›)
  | *Not-plogic* ‹*'a plogic*› (‹*not*$_p$ -› [*85*] *85*)
  | *Or-plogic* ‹*'a plogic*› ‹*'a plogic*› (‹- *or*$_p$ -› [*82*,*82*] *81*)
  | *And-plogic* ‹*'a plogic*› ‹*'a plogic*› (‹- *and*$_p$ -› [*82*,*82*] *81*)

fun *plogic-semantics* :: ‹*'a plogic* ⇒ *'a state AiF*› (‹⟦-⟧$_p$›) **where**
  ‹⟦ *true*$_p$ ⟧$_p$ = *T*▪›
| ‹⟦ *not*$_p$ *φ* ⟧$_p$ = ¬▪ ⟦*φ*⟧$_p$›
| ‹⟦ *prop*$_p$(*a*) ⟧$_p$ = *sat*▪ *a*›
| ‹⟦ *φ or*$_p$ *ψ* ⟧$_p$ = ⟦*φ*⟧$_p$ ∨▪ ⟦*ψ*⟧$_p$›
| ‹⟦ *φ and*$_p$ *ψ* ⟧$_p$ = ⟦*φ*⟧$_p$ ∧▪ ⟦*ψ*⟧$_p$›

definition *false-p* (‹*false*$_p$›) **where**

false-p-def [*simp*]: ‹$false_p = not_p\ true_p$›

definition *implies-p* :: ‹'*a plogic* ⇒ '*a plogic* ⇒ '*a plogic*› (‹- $implies_p$ -› [*81*,*81*] *80*) **where**
implies-p-def[*simp*]: ‹$\varphi\ implies_p\ \psi = (not_p\ \varphi\ or_p\ \psi)$›

1.1.1 Propositional logic lemmas

lemma *AiF-cases*:
assumes ‹$A\ T = B\ T$› **and** ‹$A\ F = B\ F$›
shows ‹$A = B$›
⟨*proof*⟩

lemma *or-and-negation*: ‹$[\![\ \varphi\ or_p\ \psi\ ]\!]_p = [\![\ not_p\ ((not_p\ \varphi)\ and_p\ (not_p\ \psi))\ ]\!]_p$›
⟨*proof*⟩

lemma *and-or-negation*: ‹$[\![\ \varphi\ and_p\ \psi]\!]_p = [\![\ not_p\ ((not_p\ \varphi)\ or_p\ (not_p\ \psi))\ ]\!]_p$›
⟨*proof*⟩

lemma *de-morgan-1*: ‹$[\![\ not_p\ (\varphi\ and_p\ \psi)\ ]\!]_p = [\![\ (not_p\ \varphi)\ or_p\ (not_p\ \psi)\ ]\!]_p$›
⟨*proof*⟩

lemma *de-morgan-2*: ‹$[\![\ not_p\ (\varphi\ or_p\ \psi)\ \ ]\!]_p = [\![\ (not_p\ \varphi)\ and_p\ (not_p\ \psi)\ ]\!]_p$›
⟨*proof*⟩

1.1.2 Propositional logic equivalence

fun *plogic-semantics'* :: ‹'*a state* ⇒ '*a plogic* ⇒ *bool*› (**infix** ‹$\models_p$› *60*) **where**
‹$\Gamma \models_p true_p = True$›
| ‹$\Gamma \models_p not_p\ \varphi = (\neg\ \Gamma \models_p \varphi)$›
| ‹$\Gamma \models_p prop_p(a) = (a \in L\ \Gamma)$›
| ‹$\Gamma \models_p \varphi\ or_p\ \psi = (\Gamma \models_p \varphi \vee \Gamma \models_p \psi)$›
| ‹$\Gamma \models_p \varphi\ and_p\ \psi = (\Gamma \models_p \varphi \wedge \Gamma \models_p \psi)$›

lemma *plogic-equivalence*:
shows ‹$(\ \ \Gamma\ \ \models_p \varphi \longleftrightarrow \Gamma \in [\![\varphi]\!]_p\ T)$›
and ‹$(\neg\ \Gamma\ \ \models_p \varphi \longleftrightarrow \Gamma \in [\![\varphi]\!]_p\ F)$›
⟨*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* = ‹*'a list*›
type-synonym *'a infinite-trace* = ‹*nat* ⇒ *'a*›
datatype *'a trace* = *Finite* ‹*'a finite-trace*› | *Infinite* ‹*'a infinite-trace*›

fun *thead* :: ‹*'a trace* ⇒ *'a*› **where**
‹*thead* (*Finite t*) = *t* ! *0*›
| ‹*thead* (*Infinite t*) = *t 0*›

fun *append* :: ‹*'a trace* ⇒ *'a trace* ⇒ *'a trace*› (**infixr** ‹⌢› *80*) **where**
‹(*Finite t*) ⌢ (*Infinite* ω) = *Infinite* (λ*n*. *if n* < *length t then t*!*n else* ω (*n* − *length t*))›
| ‹(*Finite t*) ⌢ (*Finite u*) = *Finite* (*t* @ *u*)›
| ‹(*Infinite t*) ⌢ *u* = *Infinite t*›

definition ε :: ‹*'a trace*› **where**
‹ε = *Finite* []›

definition *singleton* :: ‹*'a* ⇒ *'a trace*› **where**
‹*singleton* σ = *Finite* [σ]›

interpretation *trace*: *monoid-list* ‹(⌢)› ‹ε›
⟨*proof*⟩

lemma *finite-empty-suffix*:
assumes ‹*Finite xs* = *Finite xs* ⌢ *t*›
shows ‹*t* = ε›
⟨*proof*⟩

lemma *finite-empty-prefix*:
assumes ‹*Finite xs* = *t* ⌢ *Finite xs*›
shows ‹*t* = ε›
⟨*proof*⟩

lemma *finite-finite-suffix*:
assumes ‹*Finite xs* = *Finite ys* ⌢ *t*›
obtains *zs* **where** ‹*t* = *Finite zs*›
⟨*proof*⟩

lemma *finite-finite-prefix*:
assumes ‹*Finite xs* = *t* ⌢ *Finite ys*›

obtains *zs* **where** ‹*t* = *Finite zs*›
⟨*proof*⟩

lemma *append-is-empty*:
assumes ‹*t* ⌢ *u* = *ε*›
shows ‹*t* = *ε*›
and ‹*u* = *ε*›
⟨*proof*⟩

fun *ttake* :: ‹*nat* ⇒ *'a trace* ⇒ *'a finite-trace*› **where**
‹*ttake k* (*Finite xs*) = *take k xs*›
| ‹*ttake k* (*Infinite xs*) = *map xs* [*0*..<*k*] ›

definition *itdrop* :: ‹*nat* ⇒ *'a infinite-trace* ⇒ *'a infinite-trace*› **where**
‹*itdrop k xs* = (*λi. xs* (*i* + *k*))›

lemma *itdrop-itdrop*[*simp*]: ‹*itdrop i* (*itdrop j x*) = *itdrop* (*i* + *j*) *x*›
⟨*proof*⟩

lemma *itdrop-zero*[*simp*]: ‹*itdrop 0 x* = *x*›
⟨*proof*⟩

fun *tdrop* :: ‹*nat* ⇒ *'a trace* ⇒ *'a trace*› **where**
‹*tdrop k* (*Finite xs*) = *Finite* (*drop k xs*)›
| ‹*tdrop k* (*Infinite xs*) = *Infinite* (*itdrop k xs*) ›

lemma *ttake-simp*[*simp*]: ‹*ttake* (*length xs*) (*Finite xs* ⌢ *t*) = *xs*›
⟨*proof*⟩

lemma *ttake-tdrop*[*simp*]: ‹*Finite* (*ttake k t*) ⌢ *tdrop k t* = *t*›
⟨*proof*⟩

definition *prefixes* :: ‹*'a trace* ⇒ *'a trace set*› (‹↓ -› [*80*] *80*) **where**
‹↓ *t* = { *u* | *u v*. *t* = *u* ⌢ *v* }›

definition *extensions* :: ‹*'a trace* ⇒ *'a trace set*› (‹↑ -› [*80*] *80*) **where**
‹↑ *t* = { *t* ⌢ *u* | *u*. *True* }›

lemma *prefixes-extensions*: ‹*t* ∈ ↓ *u* ⟷ *u* ∈ ↑ *t*›
⟨*proof*⟩

interpretation *prefixes*: *order* ‹*λ t u*. *t* ∈ ↓ *u*› ‹*λ t u*. *t* ∈ ↓ *u* ∧ *t* ≠ *u*›
⟨*proof*⟩

lemma *prefixes-empty-least* : ‹*ε* ∈ ↓ *t*›
⟨*proof*⟩

lemma *prefixes-infinite-greatest* : ‹*Infinite x* ∈ ↓ *t* ⟹ *t* = *Infinite x*›
⟨*proof*⟩

lemma *prefixes-finite* : ‹*Finite xs* ∈ ↓ *Finite ys* ⟷ (∃ *zs*. *ys* = *xs* @ *zs*)›
⟨*proof*⟩

lemma *ttake-take* : ‹*take n* (*ttake m t*) = *ttake* (*min n m*) *t*›
⟨*proof*⟩

lemma *tdrop-tdrop* : ‹*tdrop n* (*tdrop m t*) = *tdrop* (*n* + *m*) *t*›
⟨*proof*⟩

lemma *tdrop-mono*: ‹*t* ∈ ↓ *u* ⟹ *tdrop k t* ∈ ↓ *tdrop k u*›
⟨*proof*⟩

lemma *ttake-finite-prefixes* : ‹*Finite xs* ∈ ↓ *t* ⟷ *xs* = *ttake* (*length xs*) *t*›
⟨*proof*⟩

lemma *ttake-prefixes* : ‹*a* ≤ *b* ⟹ *Finite* (*ttake a t*) ∈ ↓ *Finite* (*ttake b t*)›
⟨*proof*⟩

lemma *finite-directed*:
assumes ‹ *Finite xs* ∈ ↓ *t* › ‹ *Finite ys* ∈ ↓ *t* ›
shows ‹ ∃ *zs*. (*xs* = *ys* @ *zs*) ∨ (*ys* = *xs* @ *zs*) ›
⟨*proof*⟩

lemma *prefixes-directed*: ‹*u* ∈ ↓ *t* ⟹ *v* ∈ ↓ *t* ⟹ *u* ∈ ↓ *v* ∨ *v* ∈ ↓ *u*›
⟨*proof*⟩

interpretation *extensions*: *order* ‹λ *t u*. *t* ∈ ↑ *u*› ‹λ *t u*. *t* ∈ ↑ *u* ∧ *t* ≠ *u*›
⟨*proof*⟩

lemma *extensions-infinite*[*simp*]: ‹↑ *Infinite xs* = { *Infinite xs* }›
⟨*proof*⟩

lemma *extensions-empty*[*simp*]: ‹↑ *ε* = *UNIV*›
⟨*proof*⟩

lemma *prefixes-empty*: ‹↓ *ε* = {*ε*}›
⟨*proof*⟩

2.2 Prefix Closure

definition *prefix-closure* :: ‹*'a trace set* ⇒ *'a trace set*› (‹↓*s* -› [*80*] *80*) **where**
‹↓*s X* = (⋃ *t* ∈ *X*. *prefixes t*) ›

lemma *prefix-closure-subset*: ‹$X \subseteq \downarrow_s X$›
⟨*proof*⟩

lemma *prefix-closure-infinite*: ‹*Infinite* $x \in \downarrow_s X \longleftrightarrow$ *Infinite* $x \in X$›
⟨*proof*⟩

lemma *prefix-closure-idem*: ‹$\downarrow_s \downarrow_s X = \downarrow_s X$›
⟨*proof*⟩

lemma *prefix-closure-mono*: ‹$X \subseteq Y \Longrightarrow \downarrow_s X \subseteq \downarrow_s Y$›
⟨*proof*⟩

lemma *prefix-closure-union-distrib*: ‹$\downarrow_s (X \cup Y) = \downarrow_s X \cup \downarrow_s Y$›
⟨*proof*⟩

lemma *prefix-closure-Union-distrib*: ‹$\downarrow_s (\bigcup S) = \bigcup$ (*prefix-closure* ‘ S)›
⟨*proof*⟩

lemma *prefix-closure-Inter*: ‹$\downarrow_s (\bigcap$ (*prefix-closure* ‘ S)) $= \bigcap$ (*prefix-closure* ‘ S) ›
⟨*proof*⟩

lemma *prefix-closure-inter*: ‹$\downarrow_s (\downarrow_s X \cap \downarrow_s Y) = \downarrow_s X \cap \downarrow_s Y$›
⟨*proof*⟩

lemma *prefix-closure-UNIV*: ‹$\downarrow_s UNIV = UNIV$›
⟨*proof*⟩

lemma *prefix-closure-empty*: ‹$\downarrow_s \{\} = \{\}$›
⟨*proof*⟩

lemma *prefix-closure-extensions*: ‹ $\downarrow_s (\uparrow t) = \uparrow t \cup \downarrow t$›
⟨*proof*⟩

lemma *prefix-closure-prefixes*: ‹$\downarrow_s (\downarrow t) = \downarrow t$›
⟨*proof*⟩

2.3 Definitive Prefixes

definition *dprefixes* :: ‹*'a trace set* ⇒ *'a trace set*› (‹$\downarrow_d$ -› [80] 80) **where**
‹$\downarrow_d X = \{ t \mid t. \uparrow t \subseteq \downarrow_s X \}$›

lemma *dprefixes-are-prefixes* : ‹$\downarrow_d X \subseteq \downarrow_s X$›
⟨*proof*⟩

lemma *prefix-closure-dprefixes* : ‹$\downarrow_s (\downarrow_d X) \subseteq \downarrow_s X$›
⟨*proof*⟩

lemma *dprefixes-idem*: ‹$\downarrow_d \downarrow_d X = \downarrow_d X$›

⟨*proof*⟩

lemma *dprefixes-contains-extensions*: ‹$t \in \downarrow_d X \Longrightarrow \uparrow t \subseteq \downarrow_d X$›
⟨*proof*⟩

lemma *dprefixes-infinite*: ‹*Infinite* $x \in \downarrow_d X \longleftrightarrow$ *Infinite* $x \in X$›
⟨*proof*⟩

lemma *dprefixes-UNIV*: ‹$\downarrow_d$ *UNIV* = *UNIV*›
⟨*proof*⟩

lemma *dprefixes-empty*: ‹$\downarrow_d \{\} = \{\}$›
⟨*proof*⟩

lemma *dprefixes-Inter-distrib*: ‹$\downarrow_d (\bigcap S) \subseteq \bigcap$ (*dprefixes* ‘ *S*)›
⟨*proof*⟩

lemma *dprefixes-Inter*: ‹$\downarrow_d (\bigcap$ (*dprefixes* ‘ *S*)) = $\bigcap$ (*dprefixes* ‘ *S*)›
⟨*proof*⟩

lemma *dprefixes-mono*:
assumes ‹$X \subseteq Y$›
shows ‹$\downarrow_d X \subseteq \downarrow_d Y$›
⟨*proof*⟩

lemma *dprefixes-inter*: ‹$\downarrow_d (\downarrow_d X \cap \downarrow_d Y) = (\downarrow_d X \cap \downarrow_d Y)$›
⟨*proof*⟩

lemma *dprefixes-inter-distrib*: ‹$\downarrow_d (X \cap Y) \subseteq \downarrow_d X \cap \downarrow_d Y$›
⟨*proof*⟩

2.4 Definitive Sets

definition *definitive*:: ‹*'a trace set* ⇒ *bool*› **where**
‹*definitive* $X \longleftrightarrow \downarrow_d X = X$›

lemma *definitive-image*: ‹$\forall X \in S$. *definitive* $X \Longrightarrow$ *dprefixes* ‘ $S = S$›
⟨*proof*⟩

lemma *definitive-dprefixes*: ‹*definitive* ($\downarrow_d X$)›
⟨*proof*⟩

lemma *definitive-contains-extensions*: ‹*definitive* $X \Longrightarrow t \in X \Longrightarrow \uparrow t \subseteq X$›
⟨*proof*⟩

lemma *definitive-UNIV*: ‹*definitive UNIV*›
⟨*proof*⟩

lemma *definitive-empty*: ‹*definitive* {}›
⟨*proof*⟩

lemma *definitive-Inter*: ‹∀ $X \in S$. *definitive* $X \Longrightarrow$ *definitive* $(\bigcap S)$›
⟨*proof*⟩

lemma *definitive-inter*: ‹*definitive* $X \Longrightarrow$ *definitive* $Y \Longrightarrow$ *definitive* $(X \cap Y)$›
⟨*proof*⟩

lemma *definitive-infinite-extension*:
assumes ‹*definitive* X› **and** ‹$t \in X$›
shows ‹∃ f. *Infinite* $f \in X \wedge t \in \downarrow$ *Infinite* f›
⟨*proof*⟩

lemma *definitive-elemI*:
assumes ‹*definitive* X› ‹$\uparrow t \subseteq \downarrow_s X$›
shows ‹$t \in X$›
⟨*proof*⟩

definition *dUnion* :: ‹'*a trace set set* ⇒ '*a trace set*› (‹$\bigcup_d$›) **where**
‹$\bigcup_d X = \downarrow_d \bigcup X$›

abbreviation *dunion* :: ‹'*a trace set* ⇒ '*a trace set* ⇒ '*a trace set*› (**infixl** ‹$\cup_d$› *65*) **where**
‹$X \cup_d Y \equiv \bigcup_d \{X, Y\}$›

lemma *dprefixes-dUnion*: ‹$\downarrow_d \bigcup_d S = \bigcup_d S$›
⟨*proof*⟩

lemma *definitive-dUnion*: ‹*definitive* $(\bigcup_d S)$›
⟨*proof*⟩

lemma *dUnion-contains-dprefixes*: ‹$t \in S \Longrightarrow \downarrow_d t \subseteq \bigcup_d S$›
⟨*proof*⟩

lemma *dUnion-contains-definitive*: ‹$X \in S \Longrightarrow$ *definitive* $X \Longrightarrow X \subseteq \bigcup_d S$›
⟨*proof*⟩

lemma *dUnion-empty*[*simp*]: ‹$\bigcup_d \{\} = \{\}$›
⟨*proof*⟩

lemma *dUnion-least-dprefixes*: ‹$(\bigwedge X.\ X \in S \Longrightarrow X \subseteq \downarrow_d Z) \Longrightarrow \downarrow_d (\bigcup (\mathit{dprefixes}\ `\ S)) \subseteq \downarrow_d Z$›
⟨*proof*⟩

lemma *dUnion-least-definitive*:
assumes *all-defn*: ‹∀ $X \in S$. *definitive* X›
shows ‹$(\bigwedge X.\ X \in S \Longrightarrow X \subseteq Z) \Longrightarrow$ *definitive* $Z \Longrightarrow \downarrow_d \bigcup S \subseteq Z$›

⟨*proof*⟩

2.5 A type for definitive sets

typedef *'a dset* = ‹{*p* :: *'a trace set*. *definitive p* }›
⟨*proof*⟩

setup-lifting *type-definition-dset*

lift-definition *Inter-dset* :: ‹*'a dset set* ⇒ *'a dset*› (‹$\bigsqcap$›) **is** ‹λ *ss*. $\bigcap$ *ss*›
⟨*proof*⟩

abbreviation *inter-dset* :: ‹*'a dset* ⇒ *'a dset* ⇒ *'a dset*› (**infixl** ‹⊓› *66*) **where**
‹*X* ⊓ *Y* ≡ $\bigsqcap$ {*X*, *Y*}›

lift-definition *Union-cset* :: ‹*'a dset set* ⇒ *'a dset*› (‹$\bigsqcup$›) **is** ‹λ *ss*. $\bigcup_d$ *ss*›
⟨*proof*⟩

abbreviation *union-dset* :: ‹*'a dset* ⇒ *'a dset* ⇒ *'a dset*› (**infixl** ‹⊔› *65*) **where**
‹*X* ⊔ *Y* ≡ $\bigsqcup$ {*X*, *Y*}›

lift-definition *empty-dset* :: ‹*'a dset*› (‹∅›) **is** ‹{}›
⟨*proof*⟩

lift-definition *univ-dset* :: ‹*'a dset*› (‹Σ∞›) **is** ‹*UNIV*›
⟨*proof*⟩

lift-definition *subset-dset* :: ‹*'a dset* ⇒ *'a dset* ⇒ *bool*› (**infix** ‹⊑› *50*) **is** ‹(⊆)›
⟨*proof*⟩

lift-definition *strict-subset-cset* :: ‹*'a dset* ⇒ *'a dset* ⇒ *bool*› (**infix** ‹⊏› *50*) **is** ‹(⊂)›
⟨*proof*⟩

lift-definition *in-dset* :: ‹*'a trace* ⇒ *'a dset* ⇒ *bool*› **is** ‹(∈)›
⟨*proof*⟩

lift-definition *notin-dset* :: ‹*'a trace* ⇒ *'a dset* ⇒ *bool*› **is** ‹(∉)›
⟨*proof*⟩

lemma *in-dset-ε*: ‹*in-dset ε A* ⟹ *A* = Σ∞›
⟨*proof*⟩

lemma *in-dset-UNIV*: ‹*in-dset x* Σ∞›
⟨*proof*⟩

lemma *in-dset-subset*: ‹*A* ⊑ *B* ⟹ *in-dset x A* ⟹ *in-dset x B*›
⟨*proof*⟩

lemma *in-dset-inter*: ‹*in-dset x A* ⟹ *in-dset x B* ⟹ *in-dset x* (*A* ⊓ *B*)›
⟨*proof*⟩

interpretation *dset*: *complete-lattice* ‹⨅› ‹⨆› ‹(⊓)› ‹(⊑)› ‹(⊏)› ‹(⊔)› ‹∅› ‹Σ∞›
⟨*proof*⟩

2.6 Isomorphism of definitive sets and LTL properties

definition *infinites* :: ‹'*a trace set* ⇒ '*a infinite-trace set*› **where**
‹*infinites X* = (⋃ *x* ∈ *X*. *case x of Finite xs* ⇒ {} | *Infinite xs* ⇒ {*xs*})›

lemma *infinites-alt*: ‹*Infinite* ‘ *infinites A* = *A* ∩ *range Infinite*›
⟨*proof*⟩

lemma *infinites-append-right*: ‹*t* ⌢ *Infinite ω* ∈ *range Infinite*›
⟨*proof*⟩

lemma *infinites-prefix-closure*:
assumes ‹*definitive X*›
shows ‹↓*s Infinite* ‘ *infinites X* = ↓*s X*›
⟨*proof*⟩

lemma *infinites-UNIV*[*simp*]: ‹*infinites UNIV* = *UNIV*›
⟨*proof*⟩

lemma *infinites-empty*[*simp*]: ‹*infinites* {} = {}›
⟨*proof*⟩

lemma *infinites-Inter*: ‹*infinites* (⋂ *S*) = ⋂ (*infinites* ‘ *S*)›
⟨*proof*⟩

lemma *infinites-Union*: ‹*infinites* (⋃ *S*) = ⋃ (*infinites* ‘ *S*)›
⟨*proof*⟩

lemma *infinites-dprefixes*: ‹*infinites* (↓*d X*) = *infinites X*›
⟨*proof*⟩

lemma *infinites-dprefixes-Infinite*: ‹*infinites* (↓*d Infinite* ‘ *X*) = *X*›
⟨*proof*⟩

lift-definition *property* :: ‹'*a dset* ⇒ '*a infinite-trace set*› **is** ‹*infinites*›
⟨*proof*⟩

lift-definition *definitives* :: ‹'*a infinite-trace set* ⇒ '*a dset*› **is** ‹λ*x*. ↓*d* (*Infinite* ‘ *x*)›
⟨*proof*⟩

lemma *property-inverse*: ‹*property* (*definitives X*) = *X*›

⟨*proof*⟩

lemma *definitives-inverse*: ‹*definitives* (*property X*) = *X*›
⟨*proof*⟩

lemma *definitives-mono*: ‹*A* ⊆ *B* ⟹ *definitives A* ⊑ *definitives B*›
⟨*proof*⟩

lemma *property-mono*: ‹*A* ⊑ *B* ⟹ *property A* ⊆ *property B*›
⟨*proof*⟩

lemma *definitives-reflecting*: ‹*definitives A* ⊑ *definitives B* ⟹ *A* ⊆ *B*›
⟨*proof*⟩

lemma *completions-reflecting*: ‹*property A* ⊆ *property B* ⟹ *A* ⊑ *B*›
⟨*proof*⟩

lemma *property-Inter*: ‹*property* (⨅ *S*) = ⋂ (*property* ‘ *S*)›
⟨*proof*⟩

lemma *property-Union*: ‹*property* (⨆ *S*) = ⋃ (*property* ‘ *S*)›
⟨*proof*⟩

interpretation *dset*: *complete-distrib-lattice* ‹⨅› ‹⨆› ‹(⊓)› ‹(⊑)› ‹(⊏)› ‹(⊔)› ‹∅› ‹Σ∞›
⟨*proof*⟩

definition *iprepend* :: ‹'*a infinite-trace set* ⇒ '*a infinite-trace set*› **where**
‹*iprepend X* = {*t*. *itdrop 1 t* ∈ *X* }›

lemma *iprepend-itdrop*: ‹*itdrop k x* ∈ *iprepend B* ⟷ *itdrop* (*Suc k*) *x* ∈ *B*›
⟨*proof*⟩

lemmas *iprepend-itdrop-0*[*simp*] = *iprepend-itdrop*[**where** *k* = ‹*0*›,*simplified*]

definition *prepend'* :: ‹'*a trace set* ⇒ '*a trace set*› **where**
‹*prepend' X* = {*t*. *tdrop 1 t* ∈ *X* }›

lemma *trace-uncons-cases* [*case-names Cons Nil*]:
assumes ‹⋀σ *t*. *x* = *singleton* σ ⌢ *t* ⟹ *P*›
and ‹*x* = ε ⟹ *P*›
shows ‹*P*›
⟨*proof*⟩

lemma *append-prefixes-left*: ‹*a* ∈ ↓ *b* ⟹ *c* ⌢ *a* ∈ ↓ *c* ⌢ *b*›
⟨*proof*⟩

lemma *tdrop-singleton-append*[*simp*]: ‹*tdrop* (*Suc n*) (*singleton* σ $\frown$ *t*) = *tdrop n t*›
⟨*proof*⟩
lemma *tdrop-zero*[*simp*]: ‹*tdrop 0 t* = *t*›
⟨*proof*⟩
lemma *tdrop-ε*[*simp*]: ‹*tdrop k* ε = ε›
⟨*proof*⟩

lemma *prepend′-prefix-closure*: ‹$\downarrow_s$ (*prepend′ X*) ⊆ *prepend′* ($\downarrow_s$ *X*)›
⟨*proof*⟩

lemma *prepend′-dprefixes* :
assumes ‹*definitive X*›
shows ‹$\downarrow_d$ *prepend′ X* = *prepend′ X*›
⟨*proof*⟩

lemma *prepend′-definitive* :
assumes ‹*definitive X*›
shows ‹*definitive* (*prepend′ X*)›
⟨*proof*⟩

lift-definition *prepend* :: ‹′*a dset* ⇒ ′*a dset*› **is** ‹*prepend′*›
⟨*proof*⟩

lemma *prepend-Inter*: ‹$\bigsqcap$ (*prepend* ‘ *S*) = *prepend* ($\bigsqcap$ *S*)›
⟨*proof*⟩

lemma *in-dset-prependD*: ‹*in-dset* (*Finite* [*a*] $\frown$ *x*) (*prepend A*) ⟹ *in-dset x A*›
⟨*proof*⟩

lemma *in-dset-prependI*: ‹*in-dset x A* ⟹ *in-dset* (*Finite* [*a*] $\frown$ *x*) (*prepend A*)›
⟨*proof*⟩

lemma *prepend′-mono*:
assumes ‹*A* ⊆ *B*›
shows ‹*prepend′ A* ⊆ *prepend′ B*›
⟨*proof*⟩

lemma *property-prepend*: ‹*property* (*prepend X*) = *iprepend* (*property X*)›
⟨*proof*⟩

lemma *iprepend-Union*: ‹$\bigcup$ (*iprepend* ‘ *S*) = *iprepend* ($\bigcup$ *S*)›
⟨*proof*⟩

lemma *definitives-inverse-eqI*: ‹*definitives* (*property X*) = *definitives* (*property Y*) ⟹ *X* = *Y*›
⟨*proof*⟩

lemma *prepend-Union*: ‹$\bigsqcup$ (*prepend* ‘ *S*) = *prepend* ($\bigsqcup$ *S*)›
⟨*proof*⟩

lemma *non-empty-trace*: ‹ $x \neq \varepsilon \longleftrightarrow (\exists \sigma\ x'.\ x = Finite\ [\sigma] \frown x')$›
⟨*proof*⟩

lemma *thead-append*: ‹$x \neq \varepsilon \Longrightarrow thead\ (x \frown y) = thead\ x$›
⟨*proof*⟩

lemma *thead-prefix*: ‹ $x \in \downarrow y \Longrightarrow x \neq \varepsilon \Longrightarrow thead\ x = thead\ y$›
⟨*proof*⟩

lemma *compr'-inter-thead*:
‹$\downarrow_d \{x.\ x \neq \varepsilon \wedge P\ (thead\ x)\} \cap \downarrow_d \{x.\ \ x \neq \varepsilon \wedge Q\ (thead\ x)\}$
$= \downarrow_d \{x.\ x \neq \varepsilon \wedge P\ (thead\ x) \wedge Q\ (thead\ x)\}$›
⟨*proof*⟩

lift-definition *compr* :: ‹('*a trace* ⇒ *bool*) ⇒ '*a dset*› **is** ‹$\lambda p.\ \downarrow_d \{x.\ p\ x\ \}$›
⟨*proof*⟩

lift-definition *complement* :: ‹'*a dset* ⇒ '*a dset*› **is** ‹$\lambda p.\ \downarrow_d (range\ Infinite - p)$›
⟨*proof*⟩

lemma *property-complement*[*simp*]: ‹*property* (*complement X*) = *UNIV* − *property X*›
⟨*proof*⟩

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

3 Linear-time Temporal Logic

datatype (*atoms-ltl*: *'a*) *ltl* =
  True-ltl (‹$true_l$›)
| *Not-ltl* ‹*'a ltl*› (‹not_l -› [*85*] *85*)
| *Prop-ltl* ‹*'a*› (‹$prop_l$ *'*(-*'*)›)
| *Or-ltl* ‹*'a ltl*› ‹*'a ltl*› (‹- or_l -› [*82*,*82*] *81*)
| *And-ltl* ‹*'a ltl*› ‹*'a ltl*› (‹- and_l - › [*82*,*82*] *81*)
| *Next-ltl* ‹*'a ltl*› (‹X_l -› [*88*] *87*)
| *Until-ltl* ‹*'a ltl*› ‹*'a ltl*› (‹- U_l -› [*84*,*84*] *83*)

fun *lsatisfying-AiF* :: ‹*'a* ⇒ *'a state infinite-trace AiF*› (‹*lsat*•›) **where**
‹*lsat*• *x T* = {*t*. *x* ∈ *L* (*t* *0*)}› |
‹*lsat*• *x F* = {*t*. *x* ∉ *L* (*t* *0*)}›

fun *x-operator* :: ‹*'a infinite-trace AiF* ⇒ *'a infinite-trace AiF*› (‹*X*▪›) **where**
‹*X*▪ *t T* = {*x* | *x*. *itdrop* *1* *x* ∈ (*t T*)}›|
‹*X*▪ *t F* = {*x* | *x*. *itdrop* *1* *x* ∈ (*t F*)}›

fun *u-operator* :: ‹*'a infinite-trace AiF* ⇒ *'a infinite-trace AiF* ⇒ *'a infinite-trace AiF*› (**infix** ‹*U*▪› *61*) **where**
‹(*a U*▪ *b*) *T* = {*x* | *x*. ∃ *k*. (∀ *i*<*k*. *itdrop* *i* *x* ∈ (*a T*)) ∧ *itdrop* *k* *x* ∈ (*b T*)}›|
‹(*a U*▪ *b*) *F* = {*x* | *x*. ∀ *k*. (∃ *i*<*k*. *itdrop* *i* *x* ∈ (*a F*)) ∨ *itdrop* *k* *x* ∈ (*b F*)}›

fun *ltl-semantics* :: ‹*'a ltl* ⇒ *'a state infinite-trace AiF*› (‹⟦-⟧$_l$›) **where**
‹⟦ $true_l$ ⟧$_l$ = *T*•›|
‹⟦ not_l φ ⟧$_l$ = ¬▪ ⟦ φ ⟧$_l$›|
‹⟦ $prop_l$(*a*) ⟧$_l$ = *lsat*• *a*›|
‹⟦ φ or_l ψ ⟧$_l$ = ⟦ φ ⟧$_l$ ∨▪ ⟦ ψ ⟧$_l$›|
‹⟦ φ and_l ψ ⟧$_l$ = ⟦ φ ⟧$_l$ ∧▪ ⟦ ψ ⟧$_l$›|
‹⟦ X_l φ ⟧$_l$ = *X*▪ ⟦ φ ⟧$_l$›|
‹⟦ φ U_l ψ ⟧$_l$ = ⟦ φ ⟧$_l$ *U*▪ ⟦ ψ ⟧$_l$›

lemma *excluded-middle-ltl'* :
  shows ‹(Γ ∉ ⟦φ⟧$_l$ *T*) ⟷ (Γ ∈ ⟦φ⟧$_l$ *F*)›
  and ‹(Γ ∉ ⟦φ⟧$_l$ *F*) ⟷ (Γ ∈ ⟦φ⟧$_l$ *T*)›
⟨*proof*⟩

lemma *excluded-middle-ltl*: ‹Γ ∈ ⟦φ⟧$_l$ *T* ∨ Γ ∈ ⟦φ⟧$_l$ *F*›
  ⟨*proof*⟩

definition *false-ltl* (‹$false_l$›) **where**
  false-ltl-def[*simp*]: ‹$false_l$ = not_l $true_l$›

definition *implies-ltl* :: ‹'*a ltl* ⇒ '*a ltl* ⇒ '*a ltl*› (**infix** ‹*implies*$_l$› *80*) **where**
implies-ltl-def[*simp*]: ‹φ *implies*$_l$ ψ = (*not*$_l$ φ *or*$_l$ ψ)›

definition *final-ltl* :: ‹'*a ltl* ⇒ '*a ltl*› (‹F_l -›) **where**
final-ltl-def[*simp*]: ‹(F_l φ) = (*true*$_l$ U_l φ)›

definition *global-ltl* :: ‹'*a ltl* ⇒ '*a ltl*› (‹G_l -›) **where**
global-ltl-def[*simp*]: ‹(G_l φ) = (*not*$_l$ F_l (*not*$_l$ φ))›

3.1 Linear temporal logic equivalence

fun *ltl-semantics'* :: ‹'*a state infinite-trace* ⇒ '*a ltl* ⇒ *bool*› (**infix** ‹$\models_l$› *60*) **where**
‹Γ $\models_l$ *true*$_l$ = *True*›
| ‹Γ $\models_l$ *not*$_l$ φ = ($\neg$ Γ $\models_l$ φ)›
| ‹Γ $\models_l$ *prop*$_l$(a) = (a ∈ L (Γ *0*))›
| ‹Γ $\models_l$ φ *or*$_l$ ψ = (Γ $\models_l$ φ ∨ Γ $\models_l$ ψ)›
| ‹Γ $\models_l$ φ *and*$_l$ ψ = (Γ $\models_l$ φ ∧ Γ $\models_l$ ψ)›
| ‹Γ $\models_l$ (X_l φ) = *itdrop 1* Γ $\models_l$ φ›
| ‹Γ $\models_l$ (φ U_l ψ) = (∃ k. (∀ i<k. *itdrop* i Γ $\models_l$ φ) ∧ *itdrop* k Γ $\models_l$ ψ)›

3.2 Linear temporal logic lemmas

lemma ‹⟦ F_l (F_l φ) ⟧$_l$ = ⟦ F_l φ ⟧$_l$›
⟨*proof*⟩

lemma *ltl-equivalence*:
shows ‹ Γ $\models_l$ φ = (Γ ∈ ⟦ φ ⟧$_l$ T)›
and ‹($\neg$ Γ $\models_l$ φ) = (Γ ∈ ⟦ φ ⟧$_l$ F)›
⟨*proof*⟩

end
theory *LTL3*
imports *Main Traces AnswerIndexedFamilies LinearTemporalLogic*
begin

4 LTL3: Semantics, Equivalence and Formula Progression

type-synonym $'a\ AiF_3 = \langle answer \Rightarrow 'a\ state\ dset\rangle$

primrec *and-AiF*$_3$ :: $\langle 'a\ AiF_3 \Rightarrow 'a\ AiF_3 \Rightarrow 'a\ AiF_3\rangle$ (**infixl** $\langle\wedge_3\cdot\rangle$ *60*) **where**
  $\langle(a \wedge_3\cdot\ b)\ T = a\ T \sqcap b\ T\rangle$
| $\langle(a \wedge_3\cdot\ b)\ F = a\ F \sqcup b\ F\rangle$

primrec *or-AiF*$_3$ :: $\langle 'a\ AiF_3 \Rightarrow 'a\ AiF_3 \Rightarrow 'a\ AiF_3\rangle$ (**infixl** $\langle\vee_3\cdot\rangle$ *59*) **where**
  $\langle(a \vee_3\cdot\ b)\ T = a\ T \sqcup b\ T\rangle$
| $\langle(a \vee_3\cdot\ b)\ F = a\ F \sqcap b\ F\rangle$

fun *not-AiF*$_3$ :: $\langle 'a\ AiF_3 \Rightarrow 'a\ AiF_3\rangle$ ($\langle\neg_3\cdot$ -$\rangle$) **where**
  $\langle(\neg_3\cdot\ a)\ T = a\ F\rangle$
| $\langle(\neg_3\cdot\ a)\ F = a\ T\rangle$

fun *univ-AiF*$_3$ :: $\langle 'a\ AiF_3\rangle$ ($\langle T_3\cdot\rangle$) **where**
  $\langle T_3\cdot\ T = \Sigma\infty\rangle$
| $\langle T_3\cdot\ F = \emptyset\rangle$

fun *lsatisfying-AiF*$_3$ :: $\langle 'a \Rightarrow 'a\ AiF_3\rangle$ ($\langle lsat_3\cdot\rangle$) **where**
  $\langle lsat_3\cdot\ x\ T = compr\ (\lambda t.\ t \neq \varepsilon \wedge x \in L\ (thead\ t))\rangle$
| $\langle lsat_3\cdot\ x\ F = compr\ (\lambda t.\ t \neq \varepsilon \wedge x \notin L\ (thead\ t))\rangle$

fun *x*$_3$*-operator* :: $\langle 'a\ AiF_3 \Rightarrow 'a\ AiF_3\rangle$ ($\langle X_3\cdot\rangle$) **where**
  $\langle X_3\cdot\ t\ T = prepend\ (t\ T)\rangle$
| $\langle X_3\cdot\ t\ F = prepend\ (t\ F)\rangle$

fun *iterate* :: $\langle('a \Rightarrow 'a) \Rightarrow nat \Rightarrow ('a \Rightarrow 'a)\rangle$ **where**
$\langle iterate\ f\ 0\ x = x\rangle|$
$\langle iterate\ f\ (Suc\ n)\ x = f\ (iterate\ f\ n\ x)\rangle$

primrec *u*$_3$*-operator* :: $\langle 'a\ AiF_3 \Rightarrow 'a\ AiF_3 \Rightarrow 'a\ AiF_3\rangle$ (**infix** $\langle U_3\cdot\rangle$ *61*) **where**
  $\langle(a\ U_3\cdot\ b)\ T = \bigsqcup\ (\ range\ (\lambda i.\ iterate\ (\lambda x.\ prepend\ x \sqcap a\ T)\ i\ (b\ T)\ ))\rangle$
| $\langle(a\ U_3\cdot\ b)\ F = \bigsqcap\ (\ range\ (\lambda i.\ iterate\ (\lambda x.\ prepend\ x \sqcup a\ F)\ i\ (b\ F)\ ))\rangle$

fun *triv-true* :: $\langle 'a \Rightarrow bool\rangle$ **where**
$\langle triv\text{-}true\ x = (\forall s.\ x{\in}\ L\ s)\rangle$

fun *triv-false* :: $\langle 'a \Rightarrow bool\rangle$ **where**
$\langle triv\text{-}false\ x = (\forall s.\ x{\notin}\ L\ s)\rangle$

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-semantics*$_3$ :: ‹'*a ltl* ⇒ '*a AiF*$_3$› (‹⟦-⟧$_3$›) **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 ‹*property* ($[\![\varphi]\!]_3\ T$) = $[\![\varphi]\!]_l\ T$›
and ‹*property* ($[\![\varphi]\!]_3\ F$) = $[\![\varphi]\!]_l\ F$›
⟨*proof*⟩

4.2 Equivalence to LTL3 of Bauer et al.

lemma *extension-lemma*: ‹*in-dset t A* = (∀ω. *t* ⌢ *Infinite* ω ∈ *Infinite* ‘ *property A*)›
⟨*proof*⟩

lemma *extension*:
shows ‹*in-dset t* (*ltl-semantics*$_3$ φ *T*) = (∀ω. (*t* ⌢ *Infinite* ω) ∈ *Infinite* ‘ (*ltl-semantics* φ *T*))›
and ‹*in-dset t* (*ltl-semantics*$_3$ φ *F*) = (∀ω. (*t* ⌢ *Infinite* ω) ∈ *Infinite* ‘ (*ltl-semantics* φ *F*))›
⟨*proof*⟩

4.3 Formula Progression

fun *progress* :: ‹'*a ltl* ⇒ '*a state* ⇒ '*a ltl*› **where**
‹*progress true*$_l$ σ = *true*$_l$›
| ‹*progress* (*not*$_l$ φ) σ = *not*$_l$ (*progress* φ) σ›
| ‹*progress* (*prop*$_l$(*a*)) σ = (*if a* ∈ *L* σ *then true*$_l$ *else not*$_l$ *true*$_l$)›
| ‹*progress* (φ *or*$_l$ ψ) σ = (*progress* φ σ) *or*$_l$ (*progress* ψ σ)›
| ‹*progress* (φ *and*$_l$ ψ) σ = (*progress* φ σ) *and*$_l$ (*progress* ψ σ)›
| ‹*progress* (*X*$_l$ φ) σ = φ›
| ‹*progress* (φ *U*$_l$ ψ) σ = (*progress* ψ σ) *or*$_l$ ((*progress* φ σ) *and*$_l$ (φ *U*$_l$ ψ))›

lemma *unroll-Union*: ‹⨆ (*range P*) = *P 0* ⊔ (⨆ (*range* (*P* ∘ *Suc*)))›
⟨*proof*⟩

lemma *unroll-Inter*: ‹⨅ (*range P*) = *P 0* ⊓ (⨅ (*range* (*P* ∘ *Suc*)))›
⟨*proof*⟩

lemma *iterates-nonempty*: ‹*range* (λ*i*. *iterate f i X*) ≠ {}›
⟨*proof*⟩

lemma *until-cont*: ‹*A* ≠ {} ⟹ *prepend* (⨆ *A*) ⊓ *X* = ⨆ ((λ*x*. *prepend x* ⊓ *X*) ‘ *A*)›
⟨*proof*⟩

lemma *release-cont*: ‹*A* ≠ {} ⟹ *prepend* (⨅ *A*) ⊔ *X* = ⨅ ((λ*x*. *prepend x* ⊔ *X*) ‘ *A*)›
⟨*proof*⟩

lemma *iterate-unroll-Inter*:
assumes ‹⋀*A*. *A* ≠ {} ⟹ *f* (⨅ *A*) = ⨅ (*f* ‘ *A*)›
shows ‹⨅ (*range* (λ*i*. *iterate f i X*)) = *X* ⊓ *f* (⨅ (*range* (λ*i*. *iterate f i X*)))›
⟨*proof*⟩

lemma *iterate-unroll-Union*:
assumes ‹$\bigwedge A.\ A \neq \{\} \Longrightarrow f\ (\bigsqcup\ A) = \bigsqcup\ (f\ `\ A)$›
shows ‹$\bigsqcup\ (range\ (\lambda i.\ iterate\ f\ i\ X)) = X \sqcup f\ (\bigsqcup\ (range\ (\lambda i.\ iterate\ f\ i\ X\ )))$›
⟨*proof*⟩

lemma *inf-inf*: ‹$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap (x \sqcap z)$›
⟨*proof*⟩

theorem *progression-tf* :
‹$prepend\ (\llbracket progress\ \varphi\ \sigma\ \rrbracket_3\ T) \sqcap compr\ (\lambda t.\ t \neq \varepsilon \wedge thead\ t = \sigma) \sqsubseteq \llbracket\ \varphi\ \rrbracket_3\ T$ ›
‹$prepend\ (\llbracket progress\ \varphi\ \sigma\ \rrbracket_3\ F) \sqcap compr\ (\lambda t.\ t \neq \varepsilon \wedge thead\ t = \sigma) \sqsubseteq \llbracket\ \varphi\ \rrbracket_3\ F$ ›
⟨*proof*⟩

theorem *progression-tf′* :
‹$\llbracket\ \varphi\ \rrbracket_3\ T \sqcap compr\ (\lambda t.\ t \neq \varepsilon \wedge thead\ t = \sigma) \sqsubseteq prepend\ (\llbracket progress\ \varphi\ \sigma\ \rrbracket_3\ T)$ ›
‹$\llbracket\ \varphi\ \rrbracket_3\ F \sqcap compr\ (\lambda t.\ t \neq \varepsilon \wedge thead\ t = \sigma) \sqsubseteq prepend\ (\llbracket progress\ \varphi\ \sigma\ \rrbracket_3\ F)$ ›
⟨*proof*⟩

theorem *progression-tf′-u*:
shows ‹$\llbracket\ \varphi\ \rrbracket_3\ A \sqcap compr\ (\lambda t.\ t \neq \varepsilon \wedge thead\ t = \sigma) \sqsubseteq prepend\ (\llbracket progress\ \varphi\ \sigma\ \rrbracket_3\ A)$›
⟨*proof*⟩

theorem *progression-tf-u*:
shows ‹$prepend\ (\llbracket progress\ \varphi\ \sigma\ \rrbracket_3\ A) \sqcap compr\ (\lambda t.\ t \neq \varepsilon \wedge thead\ t = \sigma) \sqsubseteq \llbracket\ \varphi\ \rrbracket_3\ A$›
⟨*proof*⟩

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

theorem *fp*:
shows ‹$in\text{-}dset\ (Finite\ t)\ (\llbracket\ \varphi\ \rrbracket_3\ A) \longleftrightarrow \llbracket\ foldl\ progress\ \varphi\ t\ \rrbracket_3\ A = \Sigma\infty$ ›
⟨*proof*⟩

lemma *em-ltl*: ‹$\llbracket\ \varphi\ \rrbracket_l\ T = UNIV - (\llbracket\ \varphi\ \rrbracket_l\ F)$›
⟨*proof*⟩

theorem *em*:
shows ‹$\llbracket\ \varphi\ \rrbracket_3\ T = complement\ (\llbracket\ \varphi\ \rrbracket_3\ F)$›
⟨*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.