Linear Temporal Logic

Salomon Sickert

March 17, 2025

Abstract

This theory provides a formalisation of linear temporal logic (LTL) and unifies previous formalisations within the AFP. This entry establishes syntax and semantics for this logic and decouples it from existing entries, yielding a common environment for theories reasoning about LTL. Furthermore a parser written in SML and an executable simplifier are provided.

Contents

1 Linear Temporal Logic **2**
1.1 LTL with Syntactic Sugar . . . 3
1.1.1 Syntax . . . 3
1.1.2 Semantics . . . 4
1.2 LTL in Negation Normal Form . . . 6
1.2.1 Syntax . . . 6
1.2.2 Semantics . . . 7
1.2.3 Conversion . . . 8
1.2.4 Negation . . . 9
1.2.5 Subformulas . . . 10
1.2.6 Constant Folding . . . 11
1.2.7 Distributivity . . . 12
1.2.8 Nested operators . . . 13
1.2.9 Weak and strong operators . . . 13
1.2.10 GF and FG semantics . . . 14
1.2.11 Expansion . . . 15
1.3 LTL in restricted Negation Normal Form . . . 16
1.3.1 Syntax . . . 16
1.3.2 Semantics . . . 16
1.3.3 Conversion . . . 16
1.3.4 Negation . . . 17
1.3.5 Subformulas . . . 18
1.3.6 Expansion lemmas . . . 18

1.4 Propositional LTL . . . 18
1.4.1 Syntax . . . 19
1.4.2 Semantics . . . 20
1.4.3 Conversion . . . 20
1.4.4 Atoms . . . 21

2 Rewrite Rules for LTL Simplification 21
2.1 Constant Eliminating Constructors . . . 22
2.2 Constant Propagation . . . 25
2.3 X-Normalisation . . . 26
2.4 Pure Eventual, Pure Universal, and Suspendable Formulas . . . 29
2.5 Syntactical Implication Based Simplification . . . 32
2.6 Iterated Rewriting . . . 33
2.7 Preservation of atoms . . . 34
2.8 Simplifier . . . 36
2.9 Code Generation . . . 37

3 Equivalence Relations for LTL formulas 37
3.1 Language Equivalence . . . 37
3.2 Propositional Equivalence . . . 38
3.3 Constants Equivalence . . . 40
3.4 Quotient types . . . 42
3.5 Cardinality of propositional quotient sets . . . 43
3.6 Substitution . . . 46
3.7 Order of Equivalence Relations . . . 47

4 Disjunctive Normal Form of LTL formulas 48
4.1 Definition of Minimum Sets . . . 49
4.2 Minimal operators on sets . . . 49
4.3 Disjunctive Normal Form . . . 55
4.4 Folding of and_n and or_n over Finite Sets . . . 56
4.5 DNF to LTL conversion . . . 58
4.6 Substitution in DNF formulas . . . 59

5 Code lemmas for abstract definitions 63
5.1 Propositional Equivalence . . . 63

6 Example 64

1 Linear Temporal Logic

theory *LTL*
imports
Main HOL−Library.Omega-Words-Fun

begin

This theory provides a formalisation of linear temporal logic. It provides three variants:

1. LTL with syntactic sugar. This variant is the semantic reference and the included parser generates ASTs of this datatype.
2. LTL in negation normal form without syntactic sugar. This variant is used by the included rewriting engine and is used for the translation to automata (implemented in other entries).
3. LTL in restricted negation normal form without the rather uncommon operators "weak until" and "strong release". It is used by the formalization of Gerth's algorithm.
4. PLTL. A variant with a reduced set of operators.

This theory subsumes (and partly reuses) the existing formalisation found in LTL_to_GBA and Stuttering_Equivalence and unifies them.

1.1 LTL with Syntactic Sugar

In this section, we provide a formulation of LTL with explicit syntactic sugar deeply embedded. This formalization serves as a reference semantics.

1.1.1 Syntax

datatype (*atoms-ltlc*: *'a*) *ltlc* =
  True-ltlc (‹$true_c$›)
| *False-ltlc* (‹$false_c$›)
| *Prop-ltlc 'a* (‹$prop_c$'(-')›)
| *Not-ltlc 'a ltlc* (‹not_c -› [*85*] *85*)
| *And-ltlc 'a ltlc 'a ltlc* (‹- and_c -› [*82*,*82*] *81*)
| *Or-ltlc 'a ltlc 'a ltlc* (‹- or_c -› [*81*,*81*] *80*)
| *Implies-ltlc 'a ltlc 'a ltlc* (‹- $implies_c$ -› [*81*,*81*] *80*)
| *Next-ltlc 'a ltlc* (‹X_c -› [*88*] *87*)
| *Final-ltlc 'a ltlc* (‹F_c -› [*88*] *87*)
| *Global-ltlc 'a ltlc* (‹G_c -› [*88*] *87*)
| *Until-ltlc 'a ltlc 'a ltlc* (‹- U_c -› [*84*,*84*] *83*)
| *Release-ltlc 'a ltlc 'a ltlc* (‹- R_c -› [*84*,*84*] *83*)
| *WeakUntil-ltlc 'a ltlc 'a ltlc* (‹- W_c -› [*84*,*84*] *83*)
| *StrongRelease-ltlc 'a ltlc 'a ltlc* (‹- M_c -› [*84*,*84*] *83*)

definition *Iff-ltlc* (‹- iff_c -› [*81*,*81*] *80*)
where
  φ iff_c $\psi \equiv (\varphi$ $implies_c$ $\psi)$ and_c $(\psi$ $implies_c$ $\varphi)$

1.1.2 Semantics

primrec *semantics-ltlc* :: [*'a set word*, *'a ltlc*] ⇒ *bool* (‹- ⊨$_c$ -› [*80*,*80*] *80*)
where
  $\xi \models_c true_c = True$
| $\xi \models_c false_c = False$
| $\xi \models_c prop_c(q) = (q{\in}\xi\ 0)$
| $\xi \models_c not_c\ \varphi = (\neg\ \xi \models_c \varphi)$
| $\xi \models_c \varphi\ and_c\ \psi = (\xi \models_c \varphi \wedge \xi \models_c \psi)$
| $\xi \models_c \varphi\ or_c\ \psi = (\xi \models_c \varphi \vee \xi \models_c \psi)$
| $\xi \models_c \varphi\ implies_c\ \psi = (\xi \models_c \varphi \longrightarrow \xi \models_c \psi)$
| $\xi \models_c X_c\ \varphi = (suffix\ 1\ \xi \models_c \varphi)$
| $\xi \models_c F_c\ \varphi = (\exists\, i.\ suffix\ i\ \xi \models_c \varphi)$
| $\xi \models_c G_c\ \varphi = (\forall\, i.\ suffix\ i\ \xi \models_c \varphi)$
| $\xi \models_c \varphi\ U_c\ \psi = (\exists\, i.\ suffix\ i\ \xi \models_c \psi \wedge (\forall\, j{<}i.\ suffix\ j\ \xi \models_c \varphi))$
| $\xi \models_c \varphi\ R_c\ \psi = (\forall\, i.\ suffix\ i\ \xi \models_c \psi \vee (\exists\, j{<}i.\ suffix\ j\ \xi \models_c \varphi))$
| $\xi \models_c \varphi\ W_c\ \psi = (\forall\, i.\ suffix\ i\ \xi \models_c \varphi \vee (\exists\, j{\leq}i.\ suffix\ j\ \xi \models_c \psi))$
| $\xi \models_c \varphi\ M_c\ \psi = (\exists\, i.\ suffix\ i\ \xi \models_c \varphi \wedge (\forall\, j{\leq}i.\ suffix\ j\ \xi \models_c \psi))$

lemma *semantics-ltlc-sugar* [*simp*]:
  $\xi \models_c \varphi\ iff_c\ \psi = (\xi \models_c \varphi \longleftrightarrow \xi \models_c \psi)$
  $\xi \models_c F_c\ \varphi = \xi \models_c (true_c\ U_c\ \varphi)$
  $\xi \models_c G_c\ \varphi = \xi \models_c (false_c\ R_c\ \varphi)$
  ⟨*proof*⟩

definition *language-ltlc* $\varphi \equiv \{\xi.\ \xi \models_c \varphi\}$

lemma *language-ltlc-negate*[*simp*]:
  language-ltlc $(not_c\ \varphi) = -$ *language-ltlc* φ
  ⟨*proof*⟩

lemma *ltl-true-or-con*[*simp*]:
  $\xi \models_c prop_c(p)\ or_c\ (not_c\ prop_c(p))$
  ⟨*proof*⟩

lemma *ltl-false-true-con*[*simp*]:
  $\xi \models_c not_c\ true_c \longleftrightarrow \xi \models_c false_c$
  ⟨*proof*⟩

lemma *ltl-Next-Neg-con*[*simp*]:
  $\xi \models_c X_c\ (not_c\ \varphi) \longleftrightarrow \xi \models_c not_c\ X_c\ \varphi$
  ⟨*proof*⟩

lemma *ltl-Until-Release-con*:

$\xi \models_c \varphi\ R_c\ \psi \longleftrightarrow (\neg\ \xi \models_c (not_c\ \varphi)\ U_c\ (not_c\ \psi))$
$\xi \models_c \varphi\ U_c\ \psi \longleftrightarrow (\neg\ \xi \models_c (not_c\ \varphi)\ R_c\ (not_c\ \psi))$
⟨*proof*⟩

lemma *ltl-WeakUntil-StrongRelease-con*:
$\xi \models_c \varphi\ W_c\ \psi \longleftrightarrow (\neg\ \xi \models_c (not_c\ \varphi)\ M_c\ (not_c\ \psi))$
$\xi \models_c \varphi\ M_c\ \psi \longleftrightarrow (\neg\ \xi \models_c (not_c\ \varphi)\ W_c\ (not_c\ \psi))$
⟨*proof*⟩

lemma *ltl-Release-StrongRelease-con*:
$\xi \models_c \varphi\ R_c\ \psi \longleftrightarrow \xi \models_c (\varphi\ M_c\ \psi)\ or_c\ (G_c\ \psi)$
$\xi \models_c \varphi\ M_c\ \psi \longleftrightarrow \xi \models_c (\varphi\ R_c\ \psi)\ and_c\ (F_c\ \varphi)$
⟨*proof*⟩

lemma *ltl-Until-WeakUntil-con*:
$\xi \models_c \varphi\ U_c\ \psi \longleftrightarrow \xi \models_c (\varphi\ W_c\ \psi)\ and_c\ (F_c\ \psi)$
$\xi \models_c \varphi\ W_c\ \psi \longleftrightarrow \xi \models_c (\varphi\ U_c\ \psi)\ or_c\ (G_c\ \varphi)$
⟨*proof*⟩

lemma *ltl-StrongRelease-Until-con*:
$\xi \models_c \varphi\ M_c\ \psi \longleftrightarrow \xi \models_c \psi\ U_c\ (\varphi\ and_c\ \psi)$
⟨*proof*⟩

lemma *ltl-WeakUntil-Release-con*:
$\xi \models_c \varphi\ R_c\ \psi \longleftrightarrow \xi \models_c \psi\ W_c\ (\varphi\ and_c\ \psi)$
⟨*proof*⟩

definition *pw-eq-on* $S\ w\ w' \equiv \forall\, i.\ w\ i \cap S = w'\ i \cap S$

lemma *pw-eq-on-refl*[*simp*]: *pw-eq-on* $S\ w\ w$
and *pw-eq-on-sym*: *pw-eq-on* $S\ w\ w' \Longrightarrow$ *pw-eq-on* $S\ w'\ w$
and *pw-eq-on-trans*[*trans*]: ⟦*pw-eq-on* $S\ w\ w'$; *pw-eq-on* $S\ w'\ w''$⟧ $\Longrightarrow$ *pw-eq-on* $S\ w\ w''$
⟨*proof*⟩

lemma *pw-eq-on-suffix*:
pw-eq-on $S\ w\ w' \Longrightarrow$ *pw-eq-on* S (*suffix* $k\ w$) (*suffix* $k\ w'$)
⟨*proof*⟩

lemma *pw-eq-on-subset*:
$S \subseteq S' \Longrightarrow$ *pw-eq-on* $S'\ w\ w' \Longrightarrow$ *pw-eq-on* $S\ w\ w'$
⟨*proof*⟩

lemma *ltlc-eq-on-aux*:
pw-eq-on (*atoms-ltlc* φ) w $w' \Longrightarrow w \models_c \varphi \Longrightarrow w' \models_c \varphi$
⟨*proof*⟩

lemma *ltlc-eq-on*:
pw-eq-on (*atoms-ltlc* φ) w $w' \Longrightarrow w \models_c \varphi \longleftrightarrow w' \models_c \varphi$
⟨*proof*⟩

lemma *suffix-comp*: $(\lambda i.\ f\ (\mathit{suffix}\ k\ w\ i)) = \mathit{suffix}\ k\ (f\ o\ w)$
⟨*proof*⟩

lemma *suffix-range*: $\bigcup(\mathit{range}\ \xi) \subseteq \mathit{APs} \Longrightarrow \bigcup(\mathit{range}\ (\mathit{suffix}\ k\ \xi)) \subseteq \mathit{APs}$
⟨*proof*⟩

lemma *map-semantics-ltlc-aux*:
assumes *inj-on f APs*
assumes $\bigcup(\mathit{range}\ w) \subseteq \mathit{APs}$
assumes *atoms-ltlc* $\varphi \subseteq \mathit{APs}$
shows $w \models_c \varphi \longleftrightarrow (\lambda i.\ f\ `\ w\ i) \models_c$ *map-ltlc f* φ
⟨*proof*⟩

definition *map-props f APs* $\equiv \{i.\ \exists\, p{\in}\mathit{APs}.\ f\ p = \mathit{Some}\ i\}$

lemma *map-semantics-ltlc*:
assumes *INJ*: *inj-on f* (*dom f*) **and** *DOM*: *atoms-ltlc* $\varphi \subseteq$ *dom f*
shows $\xi \models_c \varphi \longleftrightarrow$ (*map-props f o* ξ) $\models_c$ *map-ltlc* (*the o f*) φ
⟨*proof*⟩

lemma *map-semantics-ltlc-inv*:
assumes *INJ*: *inj-on f* (*dom f*) **and** *DOM*: *atoms-ltlc* $\varphi \subseteq$ *dom f*
shows $\xi \models_c$ *map-ltlc* (*the o f*) $\varphi \longleftrightarrow (\lambda i.\ (\mathit{the}\ o\ f)\ -`\ \xi\ i) \models_c \varphi$
⟨*proof*⟩

1.2 LTL in Negation Normal Form

We define a type of LTL formula in negation normal form (NNF).

1.2.1 Syntax

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

True-ltln	(‹true_n›)
| *False-ltln*	(‹false_n›)
| *Prop-ltln* 'a	(‹prop_n'(-')›)
| *Nprop-ltln* 'a	(‹nprop_n'(-')›)

| *And-ltln 'a ltln 'a ltln* (‹- and_n -› [*82*,*82*] *81*)
| *Or-ltln 'a ltln 'a ltln* (‹- or_n -› [*84*,*84*] *83*)
| *Next-ltln 'a ltln* (‹X_n -› [*88*] *87*)
| *Until-ltln 'a ltln 'a ltln* (‹- U_n -› [*84*,*84*] *83*)
| *Release-ltln 'a ltln 'a ltln* (‹- R_n -› [*84*,*84*] *83*)
| *WeakUntil-ltln 'a ltln 'a ltln* (‹- W_n -› [*84*,*84*] *83*)
| *StrongRelease-ltln 'a ltln 'a ltln* (‹- M_n -› [*84*,*84*] *83*)

abbreviation *finally*$_n$ :: *'a ltln* ⇒ *'a ltln* (‹F_n -› [*88*] *87*)
where
$F_n\ \varphi \equiv true_n\ U_n\ \varphi$

notation (*input*) *finally*$_n$ (‹$\Diamond_n$ -› [*88*] *87*)

abbreviation *globally*$_n$ :: *'a ltln* ⇒ *'a ltln* (‹G_n -› [*88*] *87*)
where
$G_n\ \varphi \equiv false_n\ R_n\ \varphi$

notation (*input*) *globally*$_n$ (‹$\Box_n$ -› [*88*] *87*)

1.2.2 Semantics

primrec *semantics-ltln* :: [*'a set word*, *'a ltln*] ⇒ *bool* (‹- $\models_n$ -› [*80*,*80*] *80*)
where
$\xi \models_n true_n = True$
| $\xi \models_n false_n = False$
| $\xi \models_n prop_n(q) = (q \in \xi\ 0)$
| $\xi \models_n nprop_n(q) = (q \notin \xi\ 0)$
| $\xi \models_n \varphi\ and_n\ \psi = (\xi \models_n \varphi \wedge \xi \models_n \psi)$
| $\xi \models_n \varphi\ or_n\ \psi = (\xi \models_n \varphi \vee \xi \models_n \psi)$
| $\xi \models_n X_n\ \varphi = (suffix\ 1\ \xi \models_n \varphi)$
| $\xi \models_n \varphi\ U_n\ \psi = (\exists\, i.\ suffix\ i\ \xi \models_n \psi \wedge (\forall\, j{<}i.\ suffix\ j\ \xi \models_n \varphi))$
| $\xi \models_n \varphi\ R_n\ \psi = (\forall\, i.\ suffix\ i\ \xi \models_n \psi \vee (\exists\, j{<}i.\ suffix\ j\ \xi \models_n \varphi))$
| $\xi \models_n \varphi\ W_n\ \psi = (\forall\, i.\ suffix\ i\ \xi \models_n \varphi \vee (\exists\, j{\leq}i.\ suffix\ j\ \xi \models_n \psi))$
| $\xi \models_n \varphi\ M_n\ \psi = (\exists\, i.\ suffix\ i\ \xi \models_n \varphi \wedge (\forall\, j{\leq}i.\ suffix\ j\ \xi \models_n \psi))$

definition *language-ltln* $\varphi \equiv \{\xi.\ \xi \models_n \varphi\}$

lemma *semantics-ltln-ite-simps*[*simp*]:
$w \models_n (if\ P\ then\ true_n\ else\ false_n) = P$
$w \models_n (if\ P\ then\ false_n\ else\ true_n) = (\neg P)$
⟨*proof*⟩

1.2.3 Conversion

fun *ltlc-to-ltln'* :: *bool* ⇒ *'a ltlc* ⇒ *'a ltln*
where
  ltlc-to-ltln' False $true_c$ = $true_n$
| *ltlc-to-ltln' False* $false_c$ = $false_n$
| *ltlc-to-ltln' False* $prop_c(q)$ = $prop_n(q)$
| *ltlc-to-ltln' False* (φ and_c ψ) = (*ltlc-to-ltln' False* φ) and_n (*ltlc-to-ltln' False* ψ)
| *ltlc-to-ltln' False* (φ or_c ψ) = (*ltlc-to-ltln' False* φ) or_n (*ltlc-to-ltln' False* ψ)
| *ltlc-to-ltln' False* (φ $implies_c$ ψ) = (*ltlc-to-ltln' True* φ) or_n (*ltlc-to-ltln' False* ψ)
| *ltlc-to-ltln' False* (F_c φ) = $true_n$ U_n (*ltlc-to-ltln' False* φ)
| *ltlc-to-ltln' False* (G_c φ) = $false_n$ R_n (*ltlc-to-ltln' False* φ)
| *ltlc-to-ltln' False* (φ U_c ψ) = (*ltlc-to-ltln' False* φ) U_n (*ltlc-to-ltln' False* ψ)
| *ltlc-to-ltln' False* (φ R_c ψ) = (*ltlc-to-ltln' False* φ) R_n (*ltlc-to-ltln' False* ψ)
| *ltlc-to-ltln' False* (φ W_c ψ) = (*ltlc-to-ltln' False* φ) W_n (*ltlc-to-ltln' False* ψ)
| *ltlc-to-ltln' False* (φ M_c ψ) = (*ltlc-to-ltln' False* φ) M_n (*ltlc-to-ltln' False* ψ)
| *ltlc-to-ltln' True* $true_c$ = $false_n$
| *ltlc-to-ltln' True* $false_c$ = $true_n$
| *ltlc-to-ltln' True* $prop_c(q)$ = $nprop_n(q)$
| *ltlc-to-ltln' True* (φ and_c ψ) = (*ltlc-to-ltln' True* φ) or_n (*ltlc-to-ltln' True* ψ)
| *ltlc-to-ltln' True* (φ or_c ψ) = (*ltlc-to-ltln' True* φ) and_n (*ltlc-to-ltln' True* ψ)
| *ltlc-to-ltln' True* (φ $implies_c$ ψ) = (*ltlc-to-ltln' False* φ) and_n (*ltlc-to-ltln' True* ψ)
| *ltlc-to-ltln' True* (F_c φ) = $false_n$ R_n (*ltlc-to-ltln' True* φ)
| *ltlc-to-ltln' True* (G_c φ) = $true_n$ U_n (*ltlc-to-ltln' True* φ)
| *ltlc-to-ltln' True* (φ U_c ψ) = (*ltlc-to-ltln' True* φ) R_n (*ltlc-to-ltln' True* ψ)
| *ltlc-to-ltln' True* (φ R_c ψ) = (*ltlc-to-ltln' True* φ) U_n (*ltlc-to-ltln' True* ψ)
| *ltlc-to-ltln' True* (φ W_c ψ) = (*ltlc-to-ltln' True* φ) M_n (*ltlc-to-ltln' True* ψ)
| *ltlc-to-ltln' True* (φ M_c ψ) = (*ltlc-to-ltln' True* φ) W_n (*ltlc-to-ltln' True* ψ)
| *ltlc-to-ltln' b* (not_c φ) = *ltlc-to-ltln'* ($\neg$ *b*) φ
| *ltlc-to-ltln' b* (X_c φ) = X_n (*ltlc-to-ltln' b* φ)

fun *ltlc-to-ltln* :: *'a ltlc* ⇒ *'a ltln*

where
ltlc-to-ltln φ = *ltlc-to-ltln'* *False* φ

fun *ltln-to-ltlc* :: *'a ltln* ⇒ *'a ltlc*
where
ltln-to-ltlc $true_n$ = $true_c$
| *ltln-to-ltlc* $false_n$ = $false_c$
| *ltln-to-ltlc* $prop_n(q)$ = $prop_c(q)$
| *ltln-to-ltlc* $nprop_n(q)$ = not_c ($prop_c(q)$)
| *ltln-to-ltlc* (φ and_n ψ) = (*ltln-to-ltlc* φ and_c *ltln-to-ltlc* ψ)
| *ltln-to-ltlc* (φ or_n ψ) = (*ltln-to-ltlc* φ or_c *ltln-to-ltlc* ψ)
| *ltln-to-ltlc* (X_n φ) = (X_c *ltln-to-ltlc* φ)
| *ltln-to-ltlc* (φ U_n ψ) = (*ltln-to-ltlc* φ U_c *ltln-to-ltlc* ψ)
| *ltln-to-ltlc* (φ R_n ψ) = (*ltln-to-ltlc* φ R_c *ltln-to-ltlc* ψ)
| *ltln-to-ltlc* (φ W_n ψ) = (*ltln-to-ltlc* φ W_c *ltln-to-ltlc* ψ)
| *ltln-to-ltlc* (φ M_n ψ) = (*ltln-to-ltlc* φ M_c *ltln-to-ltlc* ψ)

lemma *ltlc-to-ltln'-correct*:
$w \models_n$ (*ltlc-to-ltln'* *True* φ) ⟷ ¬ $w \models_c \varphi$
$w \models_n$ (*ltlc-to-ltln'* *False* φ) ⟷ $w \models_c \varphi$
size (*ltlc-to-ltln'* *True* φ) ≤ 2 ∗ *size* φ
size (*ltlc-to-ltln'* *False* φ) ≤ 2 ∗ *size* φ
⟨*proof*⟩

lemma *ltlc-to-ltln-semantics* [*simp*]:
$w \models_n$ *ltlc-to-ltln* φ ⟷ $w \models_c \varphi$
⟨*proof*⟩

lemma *ltlc-to-ltln-size*:
size (*ltlc-to-ltln* φ) ≤ 2 ∗ *size* φ
⟨*proof*⟩

lemma *ltln-to-ltlc-semantics* [*simp*]:
$w \models_c$ *ltln-to-ltlc* φ ⟷ $w \models_n \varphi$
⟨*proof*⟩

lemma *ltlc-to-ltln-atoms*:
atoms-ltln (*ltlc-to-ltln* φ) = *atoms-ltlc* φ
⟨*proof*⟩

1.2.4 Negation

fun not_n
where

$not_n\ true_n = false_n$
| $not_n\ false_n = true_n$
| $not_n\ prop_n(a) = nprop_n(a)$
| $not_n\ nprop_n(a) = prop_n(a)$
| $not_n\ (\varphi\ and_n\ \psi) = (not_n\ \varphi)\ or_n\ (not_n\ \psi)$
| $not_n\ (\varphi\ or_n\ \psi) = (not_n\ \varphi)\ and_n\ (not_n\ \psi)$
| $not_n\ (X_n\ \varphi) = X_n\ (not_n\ \varphi)$
| $not_n\ (\varphi\ U_n\ \psi) = (not_n\ \varphi)\ R_n\ (not_n\ \psi)$
| $not_n\ (\varphi\ R_n\ \psi) = (not_n\ \varphi)\ U_n\ (not_n\ \psi)$
| $not_n\ (\varphi\ W_n\ \psi) = (not_n\ \varphi)\ M_n\ (not_n\ \psi)$
| $not_n\ (\varphi\ M_n\ \psi) = (not_n\ \varphi)\ W_n\ (not_n\ \psi)$

lemma *not_n-semantics*[*simp*]:
$w \models_n not_n\ \varphi \longleftrightarrow \neg\ w \models_n \varphi$
⟨*proof*⟩

lemma *not_n-size*:
$size\ (not_n\ \varphi) = size\ \varphi$
⟨*proof*⟩

1.2.5 Subformulas

fun *subfrmlsn* :: *'a ltln* ⇒ *'a ltln set*
where
$subfrmlsn\ (\varphi\ and_n\ \psi) = \{\varphi\ and_n\ \psi\} \cup subfrmlsn\ \varphi \cup subfrmlsn\ \psi$
| $subfrmlsn\ (\varphi\ or_n\ \psi) = \{\varphi\ or_n\ \psi\} \cup subfrmlsn\ \varphi \cup subfrmlsn\ \psi$
| $subfrmlsn\ (X_n\ \varphi) = \{X_n\ \varphi\} \cup subfrmlsn\ \varphi$
| $subfrmlsn\ (\varphi\ U_n\ \psi) = \{\varphi\ U_n\ \psi\} \cup subfrmlsn\ \varphi \cup subfrmlsn\ \psi$
| $subfrmlsn\ (\varphi\ R_n\ \psi) = \{\varphi\ R_n\ \psi\} \cup subfrmlsn\ \varphi \cup subfrmlsn\ \psi$
| $subfrmlsn\ (\varphi\ W_n\ \psi) = \{\varphi\ W_n\ \psi\} \cup subfrmlsn\ \varphi \cup subfrmlsn\ \psi$
| $subfrmlsn\ (\varphi\ M_n\ \psi) = \{\varphi\ M_n\ \psi\} \cup subfrmlsn\ \varphi \cup subfrmlsn\ \psi$
| $subfrmlsn\ \varphi = \{\varphi\}$

lemma *subfrmlsn-id*[*simp*]:
$\varphi \in subfrmlsn\ \varphi$
⟨*proof*⟩

lemma *subfrmlsn-finite*:
$finite\ (subfrmlsn\ \varphi)$
⟨*proof*⟩

lemma *subfrmlsn-card*:
$card\ (subfrmlsn\ \varphi) \leq size\ \varphi$
⟨*proof*⟩

lemma *subfrmlsn-subset*:
$\psi \in \mathit{subfrmlsn}\ \varphi \Longrightarrow \mathit{subfrmlsn}\ \psi \subseteq \mathit{subfrmlsn}\ \varphi$
⟨*proof*⟩

lemma *subfrmlsn-size*:
$\psi \in \mathit{subfrmlsn}\ \varphi \Longrightarrow \mathit{size}\ \psi < \mathit{size}\ \varphi \vee \psi = \varphi$
⟨*proof*⟩

abbreviation
$\mathit{size\text{-}set}\ S \equiv \mathit{sum}\ (\lambda x.\ 2 * \mathit{size}\ x + 1)\ S$

lemma *size-set-diff*:
$\mathit{finite}\ S \Longrightarrow S' \subseteq S \Longrightarrow \mathit{size\text{-}set}\ (S - S') = \mathit{size\text{-}set}\ S - \mathit{size\text{-}set}\ S'$
⟨*proof*⟩

1.2.6 Constant Folding

lemma *U-consts*[*intro*, *simp*]:
$w \models_n \varphi\ U_n\ \mathit{true}_n$
$\neg\ (w \models_n \varphi\ U_n\ \mathit{false}_n)$
$(w \models_n \mathit{false}_n\ U_n\ \varphi) = (w \models_n \varphi)$
⟨*proof*⟩

lemma *R-consts*[*intro*, *simp*]:
$w \models_n \varphi\ R_n\ \mathit{true}_n$
$\neg\ (w \models_n \varphi\ R_n\ \mathit{false}_n)$
$(w \models_n \mathit{true}_n\ R_n\ \varphi) = (w \models_n \varphi)$
⟨*proof*⟩

lemma *W-consts*[*intro*, *simp*]:
$w \models_n \mathit{true}_n\ W_n\ \varphi$
$w \models_n \varphi\ W_n\ \mathit{true}_n$
$(w \models_n \mathit{false}_n\ W_n\ \varphi) = (w \models_n \varphi)$
$(w \models_n \varphi\ W_n\ \mathit{false}_n) = (w \models_n G_n\ \varphi)$
⟨*proof*⟩

lemma *M-consts*[*intro*, *simp*]:
$\neg\ (w \models_n \mathit{false}_n\ M_n\ \varphi)$
$\neg\ (w \models_n \varphi\ M_n\ \mathit{false}_n)$
$(w \models_n \mathit{true}_n\ M_n\ \varphi) = (w \models_n \varphi)$
$(w \models_n \varphi\ M_n\ \mathit{true}_n) = (w \models_n F_n\ \varphi)$
⟨*proof*⟩

1.2.7 Distributivity

lemma *until-and-left-distrib*:
$w \models_n (\varphi_1\ and_n\ \varphi_2)\ U_n\ \psi \longleftrightarrow w \models_n (\varphi_1\ U_n\ \psi)\ and_n\ (\varphi_2\ U_n\ \psi)$
⟨*proof*⟩

lemma *until-or-right-distrib*:
$w \models_n \varphi\ U_n\ (\psi_1\ or_n\ \psi_2) \longleftrightarrow w \models_n (\varphi\ U_n\ \psi_1)\ or_n\ (\varphi\ U_n\ \psi_2)$
⟨*proof*⟩

lemma *release-and-right-distrib*:
$w \models_n \varphi\ R_n\ (\psi_1\ and_n\ \psi_2) \longleftrightarrow w \models_n (\varphi\ R_n\ \psi_1)\ and_n\ (\varphi\ R_n\ \psi_2)$
⟨*proof*⟩

lemma *release-or-left-distrib*:
$w \models_n (\varphi_1\ or_n\ \varphi_2)\ R_n\ \psi \longleftrightarrow w \models_n (\varphi_1\ R_n\ \psi)\ or_n\ (\varphi_2\ R_n\ \psi)$
⟨*proof*⟩

lemma *strong-release-and-right-distrib*:
$w \models_n \varphi\ M_n\ (\psi_1\ and_n\ \psi_2) \longleftrightarrow w \models_n (\varphi\ M_n\ \psi_1)\ and_n\ (\varphi\ M_n\ \psi_2)$
⟨*proof*⟩

lemma *strong-release-or-left-distrib*:
$w \models_n (\varphi_1\ or_n\ \varphi_2)\ M_n\ \psi \longleftrightarrow w \models_n (\varphi_1\ M_n\ \psi)\ or_n\ (\varphi_2\ M_n\ \psi)$
⟨*proof*⟩

lemma *weak-until-and-left-distrib*:
$w \models_n (\varphi_1\ and_n\ \varphi_2)\ W_n\ \psi \longleftrightarrow w \models_n (\varphi_1\ W_n\ \psi)\ and_n\ (\varphi_2\ W_n\ \psi)$
⟨*proof*⟩

lemma *weak-until-or-right-distrib*:
$w \models_n \varphi\ W_n\ (\psi_1\ or_n\ \psi_2) \longleftrightarrow w \models_n (\varphi\ W_n\ \psi_1)\ or_n\ (\varphi\ W_n\ \psi_2)$
⟨*proof*⟩

lemma *next-until-distrib*:
$w \models_n X_n\ (\varphi\ U_n\ \psi) \longleftrightarrow w \models_n (X_n\ \varphi)\ U_n\ (X_n\ \psi)$
⟨*proof*⟩

lemma *next-release-distrib*:
$w \models_n X_n\ (\varphi\ R_n\ \psi) \longleftrightarrow w \models_n (X_n\ \varphi)\ R_n\ (X_n\ \psi)$
⟨*proof*⟩

lemma *next-weak-until-distrib*:

$w \models_n X_n\ (\varphi\ W_n\ \psi) \longleftrightarrow w \models_n (X_n\ \varphi)\ W_n\ (X_n\ \psi)$
⟨*proof*⟩

lemma *next-strong-release-distrib*:
$w \models_n X_n\ (\varphi\ M_n\ \psi) \longleftrightarrow w \models_n (X_n\ \varphi)\ M_n\ (X_n\ \psi)$
⟨*proof*⟩

1.2.8 Nested operators

lemma *finally-until*[*simp*]:
$w \models_n F_n\ (\varphi\ U_n\ \psi) \longleftrightarrow w \models_n F_n\ \psi$
⟨*proof*⟩

lemma *globally-release*[*simp*]:
$w \models_n G_n\ (\varphi\ R_n\ \psi) \longleftrightarrow w \models_n G_n\ \psi$
⟨*proof*⟩

lemma *globally-weak-until*[*simp*]:
$w \models_n G_n\ (\varphi\ W_n\ \psi) \longleftrightarrow w \models_n G_n\ (\varphi\ or_n\ \psi)$
⟨*proof*⟩

lemma *finally-strong-release*[*simp*]:
$w \models_n F_n\ (\varphi\ M_n\ \psi) \longleftrightarrow w \models_n F_n\ (\varphi\ and_n\ \psi)$
⟨*proof*⟩

1.2.9 Weak and strong operators

lemma *ltln-weak-strong*:
$w \models_n \varphi\ W_n\ \psi \longleftrightarrow w \models_n (G_n\ \varphi)\ or_n\ (\varphi\ U_n\ \psi)$
$w \models_n \varphi\ R_n\ \psi \longleftrightarrow w \models_n (G_n\ \psi)\ or_n\ (\varphi\ M_n\ \psi)$
⟨*proof*⟩

lemma *ltln-strong-weak*:
$w \models_n \varphi\ U_n\ \psi \longleftrightarrow w \models_n (F_n\ \psi)\ and_n\ (\varphi\ W_n\ \psi)$
$w \models_n \varphi\ M_n\ \psi \longleftrightarrow w \models_n (F_n\ \varphi)\ and_n\ (\varphi\ R_n\ \psi)$
⟨*proof*⟩

lemma *ltln-strong-to-weak*:
$w \models_n \varphi\ U_n\ \psi \Longrightarrow w \models_n \varphi\ W_n\ \psi$
$w \models_n \varphi\ M_n\ \psi \Longrightarrow w \models_n \varphi\ R_n\ \psi$
⟨*proof*⟩

lemma *ltln-weak-to-strong*:
$[\![w \models_n \varphi\ W_n\ \psi;\ \neg\ w \models_n G_n\ \varphi]\!] \Longrightarrow w \models_n \varphi\ U_n\ \psi$

$[\![w \models_n \varphi\ W_n\ \psi;\ w \models_n F_n\ \psi]\!] \Longrightarrow w \models_n \varphi\ U_n\ \psi$
$[\![w \models_n \varphi\ R_n\ \psi;\ \neg\ w \models_n G_n\ \psi]\!] \Longrightarrow w \models_n \varphi\ M_n\ \psi$
$[\![w \models_n \varphi\ R_n\ \psi;\ w \models_n F_n\ \varphi]\!] \Longrightarrow w \models_n \varphi\ M_n\ \psi$
⟨*proof*⟩

lemma *ltln-StrongRelease-to-Until*:
$w \models_n \varphi\ M_n\ \psi \longleftrightarrow w \models_n \psi\ U_n\ (\varphi\ and_n\ \psi)$
⟨*proof*⟩

lemma *ltln-Release-to-WeakUntil*:
$w \models_n \varphi\ R_n\ \psi \longleftrightarrow w \models_n \psi\ W_n\ (\varphi\ and_n\ \psi)$
⟨*proof*⟩

lemma *ltln-WeakUntil-to-Release*:
$w \models_n \varphi\ W_n\ \psi \longleftrightarrow w \models_n \psi\ R_n\ (\varphi\ or_n\ \psi)$
⟨*proof*⟩

lemma *ltln-Until-to-StrongRelease*:
$w \models_n \varphi\ U_n\ \psi \longleftrightarrow w \models_n \psi\ M_n\ (\varphi\ or_n\ \psi)$
⟨*proof*⟩

1.2.10 GF and FG semantics

lemma *GF-suffix*:
suffix $i\ w \models_n G_n\ (F_n\ \psi) \longleftrightarrow w \models_n G_n\ (F_n\ \psi)$
⟨*proof*⟩

lemma *FG-suffix*:
suffix $i\ w \models_n F_n\ (G_n\ \psi) \longleftrightarrow w \models_n F_n\ (G_n\ \psi)$
⟨*proof*⟩

lemma *GF-Inf-many*:
$w \models_n G_n\ (F_n\ \varphi) \longleftrightarrow (\exists_\infty\ i.\ suffix\ i\ w \models_n \varphi)$
⟨*proof*⟩

lemma *FG-Alm-all*:
$w \models_n F_n\ (G_n\ \varphi) \longleftrightarrow (\forall_\infty\ i.\ suffix\ i\ w \models_n \varphi)$
⟨*proof*⟩

lemma *MOST-nat-add*:
$(\forall_\infty i{::}nat.\ P\ i) \longleftrightarrow (\forall_\infty i.\ P\ (i + j))$

⟨*proof*⟩

lemma *INFM-nat-add*:
$(\exists_\infty i{::}nat.\ P\ i) \longleftrightarrow (\exists_\infty i.\ P\ (i + j))$
⟨*proof*⟩

lemma *FG-suffix-G*:
$w \models_n F_n\ (G_n\ \varphi) \Longrightarrow \forall_\infty i.\ suffix\ i\ w \models_n G_n\ \varphi$
⟨*proof*⟩

lemma *Alm-all-GF-F*:
$\forall_\infty i.\ suffix\ i\ w \models_n G_n\ (F_n\ \psi) \longleftrightarrow suffix\ i\ w \models_n F_n\ \psi$
⟨*proof*⟩

lemma *Alm-all-FG-G*:
$\forall_\infty i.\ suffix\ i\ w \models_n F_n\ (G_n\ \psi) \longleftrightarrow suffix\ i\ w \models_n G_n\ \psi$
⟨*proof*⟩

1.2.11 Expansion

lemma *ltln-expand-Until*:
$\xi \models_n \varphi\ U_n\ \psi \longleftrightarrow (\xi \models_n \psi\ or_n\ (\varphi\ and_n\ (X_n\ (\varphi\ U_n\ \psi))))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *ltln-expand-Release*:
$\xi \models_n \varphi\ R_n\ \psi \longleftrightarrow (\xi \models_n \psi\ and_n\ (\varphi\ or_n\ (X_n\ (\varphi\ R_n\ \psi))))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *ltln-expand-WeakUntil*:
$\xi \models_n \varphi\ W_n\ \psi \longleftrightarrow (\xi \models_n \psi\ or_n\ (\varphi\ and_n\ (X_n\ (\varphi\ W_n\ \psi))))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *ltln-expand-StrongRelease*:
$\xi \models_n \varphi\ M_n\ \psi \longleftrightarrow (\xi \models_n \psi\ and_n\ (\varphi\ or_n\ (X_n\ (\varphi\ M_n\ \psi))))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *ltln-Release-alterdef*:
$w \models_n \varphi\ R_n\ \psi \longleftrightarrow w \models_n (G_n\ \psi)\ or_n\ (\psi\ U_n\ (\varphi\ and_n\ \psi))$
⟨*proof*⟩

1.3 LTL in restricted Negation Normal Form

Some algorithms do not handle the operators W and M, hence we also provide a datatype without these two operators.

1.3.1 Syntax

datatype (*atoms-ltlr*: *'a*) *ltlr* =
  True-ltlr (‹$true_r$›)
| *False-ltlr* (‹$false_r$›)
| *Prop-ltlr 'a* (‹$prop_r$ *'*(-*'*)›)
| *Nprop-ltlr 'a* (‹$nprop_r$ *'*(-*'*)›)
| *And-ltlr 'a ltlr 'a ltlr* (‹- and_r -› [*82*,*82*] *81*)
| *Or-ltlr 'a ltlr 'a ltlr* (‹- or_r -› [*84*,*84*] *83*)
| *Next-ltlr 'a ltlr* (‹X_r -› [*88*] *87*)
| *Until-ltlr 'a ltlr 'a ltlr* (‹- U_r -› [*84*,*84*] *83*)
| *Release-ltlr 'a ltlr 'a ltlr* (‹- R_r -› [*84*,*84*] *83*)

1.3.2 Semantics

primrec *semantics-ltlr* :: [*'a set word*, *'a ltlr*] ⇒ *bool* (‹- $\models_r$ -› [*80*,*80*] *80*)
where
  $\xi \models_r true_r = True$
| $\xi \models_r false_r = False$
| $\xi \models_r prop_r(q) = (q \in \xi\ 0)$
| $\xi \models_r nprop_r(q) = (q \notin \xi\ 0)$
| $\xi \models_r \varphi\ and_r\ \psi = (\xi \models_r \varphi \wedge \xi \models_r \psi)$
| $\xi \models_r \varphi\ or_r\ \psi = (\xi \models_r \varphi \vee \xi \models_r \psi)$
| $\xi \models_r X_r\ \varphi = (suffix\ 1\ \xi \models_r \varphi)$
| $\xi \models_r \varphi\ U_r\ \psi = (\exists i.\ suffix\ i\ \xi \models_r \psi \wedge (\forall j{<}i.\ suffix\ j\ \xi \models_r \varphi))$
| $\xi \models_r \varphi\ R_r\ \psi = (\forall i.\ suffix\ i\ \xi \models_r \psi \vee (\exists j{<}i.\ suffix\ j\ \xi \models_r \varphi))$

1.3.3 Conversion

fun *ltln-to-ltlr* :: *'a ltln* ⇒ *'a ltlr*
where
  ltln-to-ltlr $true_n = true_r$
| *ltln-to-ltlr* $false_n = false_r$
| *ltln-to-ltlr* $prop_n(a) = prop_r(a)$
| *ltln-to-ltlr* $nprop_n(a) = nprop_r(a)$
| *ltln-to-ltlr* $(\varphi\ and_n\ \psi) =$ (*ltln-to-ltlr* φ) and_r (*ltln-to-ltlr* ψ)
| *ltln-to-ltlr* $(\varphi\ or_n\ \psi) =$ (*ltln-to-ltlr* φ) or_r (*ltln-to-ltlr* ψ)
| *ltln-to-ltlr* $(X_n\ \varphi) = X_r$ (*ltln-to-ltlr* φ)
| *ltln-to-ltlr* $(\varphi\ U_n\ \psi) =$ (*ltln-to-ltlr* φ) U_r (*ltln-to-ltlr* ψ)
| *ltln-to-ltlr* $(\varphi\ R_n\ \psi) =$ (*ltln-to-ltlr* φ) R_r (*ltln-to-ltlr* ψ)

| *ltln-to-ltlr* (φ W_n ψ) = (*ltln-to-ltlr* ψ) R_r ((*ltln-to-ltlr* φ) or_r (*ltln-to-ltlr* ψ))
| *ltln-to-ltlr* (φ M_n ψ) = (*ltln-to-ltlr* ψ) U_r ((*ltln-to-ltlr* φ) and_r (*ltln-to-ltlr* ψ))

fun *ltlr-to-ltln* :: *'a ltlr* ⇒ *'a ltln*
where
ltlr-to-ltln $true_r$ = $true_n$
| *ltlr-to-ltln* $false_r$ = $false_n$
| *ltlr-to-ltln* $prop_r(a)$ = $prop_n(a)$
| *ltlr-to-ltln* $nprop_r(a)$ = $nprop_n(a)$
| *ltlr-to-ltln* (φ and_r ψ) = (*ltlr-to-ltln* φ) and_n (*ltlr-to-ltln* ψ)
| *ltlr-to-ltln* (φ or_r ψ) = (*ltlr-to-ltln* φ) or_n (*ltlr-to-ltln* ψ)
| *ltlr-to-ltln* (X_r φ) = X_n (*ltlr-to-ltln* φ)
| *ltlr-to-ltln* (φ U_r ψ) = (*ltlr-to-ltln* φ) U_n (*ltlr-to-ltln* ψ)
| *ltlr-to-ltln* (φ R_r ψ) = (*ltlr-to-ltln* φ) R_n (*ltlr-to-ltln* ψ)

lemma *ltln-to-ltlr-semantics*:
$w \models_r$ *ltln-to-ltlr* $\varphi \longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

lemma *ltlr-to-ltln-semantics*:
$w \models_n$ *ltlr-to-ltln* $\varphi \longleftrightarrow w \models_r \varphi$
⟨*proof*⟩

1.3.4 Negation

fun not_r
where
not_r $true_r$ = $false_r$
| not_r $false_r$ = $true_r$
| not_r $prop_r(a)$ = $nprop_r(a)$
| not_r $nprop_r(a)$ = $prop_r(a)$
| not_r (φ and_r ψ) = (not_r φ) or_r (not_r ψ)
| not_r (φ or_r ψ) = (not_r φ) and_r (not_r ψ)
| not_r (X_r φ) = X_r (not_r φ)
| not_r (φ U_r ψ) = (not_r φ) R_r (not_r ψ)
| not_r (φ R_r ψ) = (not_r φ) U_r (not_r ψ)

lemma not_r-*semantics* [*simp*]:
$w \models_r$ not_r $\varphi \longleftrightarrow \neg$ $w \models_r \varphi$
⟨*proof*⟩

1.3.5 Subformulas

fun *subfrmlsr* :: *'a ltlr* ⇒ *'a ltlr set*
where
  $subfrmlsr\ (\varphi\ and_r\ \psi) = \{\varphi\ and_r\ \psi\} \cup subfrmlsr\ \varphi \cup subfrmlsr\ \psi$
| $subfrmlsr\ (\varphi\ or_r\ \psi) = \{\varphi\ or_r\ \psi\} \cup subfrmlsr\ \varphi \cup subfrmlsr\ \psi$
| $subfrmlsr\ (\varphi\ U_r\ \psi) = \{\varphi\ U_r\ \psi\} \cup subfrmlsr\ \varphi \cup subfrmlsr\ \psi$
| $subfrmlsr\ (\varphi\ R_r\ \psi) = \{\varphi\ R_r\ \psi\} \cup subfrmlsr\ \varphi \cup subfrmlsr\ \psi$
| $subfrmlsr\ (X_r\ \varphi) = \{X_r\ \varphi\} \cup subfrmlsr\ \varphi$
| $subfrmlsr\ x = \{x\}$

lemma *subfrmlsr-id*[*simp*]:
  $\varphi \in subfrmlsr\ \varphi$
  ⟨*proof*⟩

lemma *subfrmlsr-finite*:
  $finite\ (subfrmlsr\ \varphi)$
  ⟨*proof*⟩

lemma *subfrmlsr-subset*:
  $\psi \in subfrmlsr\ \varphi \Longrightarrow subfrmlsr\ \psi \subseteq subfrmlsr\ \varphi$
  ⟨*proof*⟩

lemma *subfrmlsr-size*:
  $\psi \in subfrmlsr\ \varphi \Longrightarrow size\ \psi < size\ \varphi \vee \psi = \varphi$
  ⟨*proof*⟩

1.3.6 Expansion lemmas

lemma *ltlr-expand-Until*:
  $\xi \models_r \varphi\ U_r\ \psi \longleftrightarrow (\xi \models_r \psi\ or_r\ (\varphi\ and_r\ (X_r\ (\varphi\ U_r\ \psi))))$
  ⟨*proof*⟩

lemma *ltlr-expand-Release*:
  $\xi \models_r \varphi\ R_r\ \psi \longleftrightarrow (\xi \models_r \psi\ and_r\ (\varphi\ or_r\ (X_r\ (\varphi\ R_r\ \psi))))$
  ⟨*proof*⟩

1.4 Propositional LTL

We define the syntax and semantics of propositional linear-time temporal logic PLTL. PLTL formulas are built from atomic formulas, propositional connectives, and the temporal operators "next" and "until". The following data type definition is parameterized by the type of states over which formulas are evaluated.

1.4.1 Syntax

datatype *'a pltl* =
  False-ltlp (‹$false_p$›)
| *Atom-ltlp 'a ⇒ bool* (‹$atom_p$'(-')›)
| *Implies-ltlp 'a pltl 'a pltl* (‹- $implies_p$ -› [*81*,*81*] *80*)
| *Next-ltlp 'a pltl* (‹X_p -› [*88*] *87*)
| *Until-ltlp 'a pltl 'a pltl* (‹- U_p -› [*84*,*84*] *83*)

— Further connectives of PLTL can be defined in terms of the existing syntax.

definition *Not-ltlp* (‹not_p -› [*85*] *85*)
where
  $not_p\ \varphi \equiv \varphi\ implies_p\ false_p$

definition *True-ltlp* (‹$true_p$›)
where
  $true_p \equiv not_p\ false_p$

definition *Or-ltlp* (‹- or_p -› [*81*,*81*] *80*)
where
  $\varphi\ or_p\ \psi \equiv (not_p\ \varphi)\ implies_p\ \psi$

definition *And-ltlp* (‹- and_p -› [*82*,*82*] *81*)
where
  $\varphi\ and_p\ \psi \equiv not_p\ ((not_p\ \varphi)\ or_p\ (not_p\ \psi))$

definition *Eventually-ltlp* (‹F_p -› [*88*] *87*)
where
  $F_p\ \varphi \equiv true_p\ U_p\ \varphi$

definition *Always-ltlp* (‹G_p -› [*88*] *87*)
where
  $G_p\ \varphi \equiv not_p\ (F_p\ (not_p\ \varphi))$

definition *Release-ltlp* (‹- R_p -› [*84*,*84*] *83*)
where
  $\varphi\ R_p\ \psi \equiv not_p\ ((not_p\ \varphi)\ U_p\ (not_p\ \psi))$

definition *WeakUntil-ltlp* (‹- W_p -› [*84*,*84*] *83*)
where
  $\varphi\ W_p\ \psi \equiv \psi\ R_p\ (\varphi\ or_p\ \psi)$

definition *StrongRelease-ltlp* (‹- M_p -› [*84*,*84*] *83*)
where
$\varphi\ M_p\ \psi \equiv \psi\ U_p\ (\varphi\ and_p\ \psi)$

1.4.2 Semantics

fun *semantics-pltl* :: ['*a word*, '*a pltl*] ⇒ *bool* (‹- $\models_p$ -› [*80*,*80*] *80*)
where
$w \models_p false_p = False$
| $w \models_p atom_p(p) = (p\ (w\ 0))$
| $w \models_p \varphi\ implies_p\ \psi = (w \models_p \varphi \longrightarrow w \models_p \psi)$
| $w \models_p X_p\ \varphi = (suffix\ 1\ w \models_p \varphi)$
| $w \models_p \varphi\ U_p\ \psi = (\exists\, i.\ suffix\ i\ w \models_p \psi \wedge (\forall\, j{<}i.\ suffix\ j\ w \models_p \varphi))$

lemma *semantics-pltl-sugar* [*simp*]:
$w \models_p not_p\ \varphi = (\neg w \models_p \varphi)$
$w \models_p true_p = True$
$w \models_p \varphi\ or_p\ \psi = (w \models_p \varphi \vee w \models_p \psi)$
$w \models_p \varphi\ and_p\ \psi = (w \models_p \varphi \wedge w \models_p \psi)$
$w \models_p F_p\ \varphi = (\exists\, i.\ suffix\ i\ w \models_p \varphi)$
$w \models_p G_p\ \varphi = (\forall\, i.\ suffix\ i\ w \models_p \varphi)$
$w \models_p \varphi\ R_p\ \psi = (\forall\, i.\ suffix\ i\ w \models_p \psi \vee (\exists\, j{<}i.\ suffix\ j\ w \models_p \varphi))$
$w \models_p \varphi\ W_p\ \psi = (\forall\, i.\ suffix\ i\ w \models_p \varphi \vee (\exists\, j{\leq}i.\ suffix\ j\ w \models_p \psi))$
$w \models_p \varphi\ M_p\ \psi = (\exists\, i.\ suffix\ i\ w \models_p \varphi \wedge (\forall\, j{\leq}i.\ suffix\ j\ w \models_p \psi))$
⟨*proof*⟩

definition *language-ltlp* $\varphi \equiv \{\xi.\ \xi \models_p \varphi\}$

1.4.3 Conversion

fun *ltlc-to-pltl* :: '*a ltlc* ⇒ '*a set pltl*
where
ltlc-to-pltl $true_c = true_p$
| *ltlc-to-pltl* $false_c = false_p$
| *ltlc-to-pltl* $(prop_c(q)) = atom_p((\in)\ q)$
| *ltlc-to-pltl* $(not_c\ \varphi) = not_p\ (ltlc\text{-}to\text{-}pltl\ \varphi)$
| *ltlc-to-pltl* $(\varphi\ and_c\ \psi) = (ltlc\text{-}to\text{-}pltl\ \varphi)\ and_p\ (ltlc\text{-}to\text{-}pltl\ \psi)$
| *ltlc-to-pltl* $(\varphi\ or_c\ \psi) = (ltlc\text{-}to\text{-}pltl\ \varphi)\ or_p\ (ltlc\text{-}to\text{-}pltl\ \psi)$
| *ltlc-to-pltl* $(\varphi\ implies_c\ \psi) = (ltlc\text{-}to\text{-}pltl\ \varphi)\ implies_p\ (ltlc\text{-}to\text{-}pltl\ \psi)$
| *ltlc-to-pltl* $(X_c\ \varphi) = X_p\ (ltlc\text{-}to\text{-}pltl\ \varphi)$
| *ltlc-to-pltl* $(F_c\ \varphi) = F_p\ (ltlc\text{-}to\text{-}pltl\ \varphi)$
| *ltlc-to-pltl* $(G_c\ \varphi) = G_p\ (ltlc\text{-}to\text{-}pltl\ \varphi)$
| *ltlc-to-pltl* $(\varphi\ U_c\ \psi) = (ltlc\text{-}to\text{-}pltl\ \varphi)\ U_p\ (ltlc\text{-}to\text{-}pltl\ \psi)$
| *ltlc-to-pltl* $(\varphi\ R_c\ \psi) = (ltlc\text{-}to\text{-}pltl\ \varphi)\ R_p\ (ltlc\text{-}to\text{-}pltl\ \psi)$

| *ltlc-to-pltl* (φ W_c ψ) = (*ltlc-to-pltl* φ) W_p (*ltlc-to-pltl* ψ)
| *ltlc-to-pltl* (φ M_c ψ) = (*ltlc-to-pltl* φ) M_p (*ltlc-to-pltl* ψ)

lemma *ltlc-to-pltl-semantics* [*simp*]:
$w \models_p$ (*ltlc-to-pltl* φ) $\longleftrightarrow$ $w \models_c \varphi$
⟨*proof*⟩

1.4.4 Atoms

fun *atoms-pltl* :: $'a$ *pltl* $\Rightarrow$ ($'a$ $\Rightarrow$ *bool*) *set*
where
atoms-pltl $false_p$ = {}
| *atoms-pltl* $atom_p(p)$ = {p}
| *atoms-pltl* (φ $implies_p$ ψ) = *atoms-pltl* φ ∪ *atoms-pltl* ψ
| *atoms-pltl* (X_p φ) = *atoms-pltl* φ
| *atoms-pltl* (φ U_p ψ) = *atoms-pltl* φ ∪ *atoms-pltl* ψ

lemma *atoms-finite* [*iff*]:
finite (*atoms-pltl* φ)
⟨*proof*⟩

lemma *atoms-pltl-sugar* [*simp*]:
atoms-pltl (not_p φ) = *atoms-pltl* φ
atoms-pltl $true_p$ = {}
atoms-pltl (φ or_p ψ) = *atoms-pltl* φ ∪ *atoms-pltl* ψ
atoms-pltl (φ and_p ψ) = *atoms-pltl* φ ∪ *atoms-pltl* ψ
atoms-pltl (F_p φ) = *atoms-pltl* φ
atoms-pltl (G_p φ) = *atoms-pltl* φ
⟨*proof*⟩

end

2 Rewrite Rules for LTL Simplification

theory *Rewriting*
imports
LTL HOL−Library.Extended-Nat
begin

This theory provides rewrite rules for the simplification of LTL formulas. It supports:

- Constants Removal
- *Next-ltln*-Normalisation

- Modal Simplification (based on pure eventual, pure universal, or suspendable formulas)
- Syntactic Implication Checking

It reuses parts of LTL_Rewrite.thy (CAVA, LTL_TO_GBA). Furthermore, some rules were taken from [2] and [1]. All functions are defined for *ltln*.

2.1 Constant Eliminating Constructors

definition *mk-and*
where
mk-and x y ≡ *case x of false*$_n$ ⇒ *false*$_n$ | *true*$_n$ ⇒ *y* | - ⇒ (*case y of false*$_n$ ⇒ *false*$_n$ | *true*$_n$ ⇒ *x* | - ⇒ *x and*$_n$ *y*)

definition *mk-or*
where
mk-or x y ≡ *case x of false*$_n$ ⇒ *y* | *true*$_n$ ⇒ *true*$_n$ | - ⇒ (*case y of true*$_n$ ⇒ *true*$_n$ | *false*$_n$ ⇒ *x* | - ⇒ *x or*$_n$ *y*)

fun *remove-strong-ops*
where
remove-strong-ops (*x U*$_n$ *y*) = *remove-strong-ops y*
| *remove-strong-ops* (*x M*$_n$ *y*) = *x and*$_n$ *y*
| *remove-strong-ops* (*x or*$_n$ *y*) = *remove-strong-ops x or*$_n$ *remove-strong-ops y*
| *remove-strong-ops x* = *x*

fun *remove-weak-ops*
where
remove-weak-ops (*x R*$_n$ *y*) = *remove-weak-ops y*
| *remove-weak-ops* (*x W*$_n$ *y*) = *x or*$_n$ *y*
| *remove-weak-ops* (*x and*$_n$ *y*) = *remove-weak-ops x and*$_n$ *remove-weak-ops y*
| *remove-weak-ops x* = *x*

definition *mk-finally*
where
mk-finally x ≡ *case x of true*$_n$ ⇒ *true*$_n$ | *false*$_n$ ⇒ *false*$_n$ | - ⇒ *F*$_n$ (*remove-strong-ops x*)

definition *mk-globally*
where
mk-globally x ≡ *case x of true*$_n$ ⇒ *true*$_n$ | *false*$_n$ ⇒ *false*$_n$ | - ⇒ *G*$_n$ (*remove-weak-ops x*)

definition *mk-until*
where
mk-until x y ≡ *case x of false*$_n$ ⇒ *y*
| *true*$_n$ ⇒ *mk-finally y*
| - ⇒ (*case y of true*$_n$ ⇒ *true*$_n$ | *false*$_n$ ⇒ *false*$_n$ | - ⇒ *x U*$_n$ *y*)

definition *mk-release*
where
mk-release x y ≡ *case x of true*$_n$ ⇒ *y*
| *false*$_n$ ⇒ *mk-globally y*
| - ⇒ (*case y of true*$_n$ ⇒ *true*$_n$ | *false*$_n$ ⇒ *false*$_n$ | - ⇒ *x R*$_n$ *y*)

definition *mk-weak-until*
where
mk-weak-until x y ≡ *case y of true*$_n$ ⇒ *true*$_n$
| *false*$_n$ ⇒ *mk-globally x*
| - ⇒ (*case x of true*$_n$ ⇒ *true*$_n$ | *false*$_n$ ⇒ *y* | - ⇒ *x W*$_n$ *y*)

definition *mk-strong-release*
where
mk-strong-release x y ≡ *case y of false*$_n$ ⇒ *false*$_n$
| *true*$_n$ ⇒ *mk-finally x*
| - ⇒ (*case x of true*$_n$ ⇒ *y* | *false*$_n$ ⇒ *false*$_n$ | - ⇒ *x M*$_n$ *y*)

definition *mk-next*
where
mk-next x ≡ *case x of true*$_n$ ⇒ *true*$_n$ | *false*$_n$ ⇒ *false*$_n$ | - ⇒ *X*$_n$ *x*

definition *mk-next-pow* (‹*X*$_n$ ″›)
where
mk-next-pow n x ≡ *case x of true*$_n$ ⇒ *true*$_n$ | *false*$_n$ ⇒ *false*$_n$ | - ⇒ (*Next-ltln* ^^ *n*) *x*

lemma *mk-and-semantics* [*simp*]:
w ⊨$_n$ *mk-and x y* ⟷ *w* ⊨$_n$ *x and*$_n$ *y*
⟨*proof*⟩

lemma *mk-or-semantics* [*simp*]:
w ⊨$_n$ *mk-or x y* ⟷ *w* ⊨$_n$ *x or*$_n$ *y*
⟨*proof*⟩

lemma *remove-strong-ops-sound* [*simp*]:
$w \models_n F_n$ (*remove-strong-ops* y) $\longleftrightarrow$ $w \models_n F_n\ y$
⟨*proof*⟩

lemma *remove-weak-ops-sound* [*simp*]:
$w \models_n G_n$ (*remove-weak-ops* y) $\longleftrightarrow$ $w \models_n G_n\ y$
⟨*proof*⟩

lemma *mk-finally-semantics* [*simp*]:
$w \models_n$ *mk-finally* $x \longleftrightarrow w \models_n F_n\ x$
⟨*proof*⟩

lemma *mk-globally-semantics* [*simp*]:
$w \models_n$ *mk-globally* $x \longleftrightarrow w \models_n G_n\ x$
⟨*proof*⟩

lemma *mk-until-semantics* [*simp*]:
$w \models_n$ *mk-until* $x\ y \longleftrightarrow w \models_n x\ U_n\ y$
⟨*proof*⟩

lemma *mk-release-semantics* [*simp*]:
$w \models_n$ *mk-release* $x\ y \longleftrightarrow w \models_n x\ R_n\ y$
⟨*proof*⟩

lemma *mk-weak-until-semantics* [*simp*]:
$w \models_n$ *mk-weak-until* $x\ y \longleftrightarrow w \models_n x\ W_n\ y$
⟨*proof*⟩

lemma *mk-strong-release-semantics* [*simp*]:
$w \models_n$ *mk-strong-release* $x\ y \longleftrightarrow w \models_n x\ M_n\ y$
⟨*proof*⟩

lemma *mk-next-semantics* [*simp*]:
$w \models_n$ *mk-next* $x \longleftrightarrow w \models_n X_n\ x$
⟨*proof*⟩

lemma *mk-next-pow-semantics* [*simp*]:
$w \models_n$ *mk-next-pow* $i\ x \longleftrightarrow$ *suffix* $i\ w \models_n x$
⟨*proof*⟩

lemma *mk-next-pow-simp* [*simp*, *code-unfold*]:
mk-next-pow $0\ x = x$
mk-next-pow $1\ x =$ *mk-next* x
⟨*proof*⟩

2.2 Constant Propagation

fun *is-constant* :: *'a ltln* ⇒ *bool*
where
is-constant $true_n$ = *True*
| *is-constant* $false_n$ = *True*
| *is-constant* - = *False*

lemma *is-constant-constructorsI*:
is-constant x ⟹ *is-constant y* ⟹ *is-constant* (*mk-and x y*)
¬*is-constant x* ⟹ ¬*is-constant y* ⟹ ¬*is-constant* (*mk-and x y*)
is-constant x ⟹ *is-constant y* ⟹ *is-constant* (*mk-or x y*)
¬*is-constant x* ⟹ ¬*is-constant y* ⟹ ¬*is-constant* (*mk-or x y*)
is-constant x ⟹ *is-constant* (*mk-finally x*)
¬*is-constant x* ⟹ ¬*is-constant* (*mk-finally x*)
is-constant x ⟹ *is-constant* (*mk-globally x*)
¬*is-constant x* ⟹ ¬*is-constant* (*mk-globally x*)
is-constant x ⟹ *is-constant* (*mk-until y x*)
¬*is-constant x* ⟹ ¬*is-constant* (*mk-until y x*)
is-constant x ⟹ *is-constant* (*mk-release y x*)
¬*is-constant x* ⟹ ¬*is-constant* (*mk-release y x*)
is-constant x ⟹ *is-constant y* ⟹ *is-constant* (*mk-weak-until x y*)
¬*is-constant x* ⟹ ¬*is-constant y* ⟹ ¬*is-constant* (*mk-weak-until x y*)
is-constant x ⟹ *is-constant y* ⟹ *is-constant* (*mk-strong-release x y*)
¬*is-constant x* ⟹ ¬*is-constant y* ⟹ ¬*is-constant* (*mk-strong-release x y*)
is-constant x ⟹ *is-constant* (*mk-next x*)
¬*is-constant x* ⟹ ¬*is-constant* (*mk-next x*)
is-constant x ⟹ *is-constant* (*mk-next-pow n x*)
⟨*proof*⟩

lemma *is-constant-constructors-simps*:
mk-next-pow n x = $false_n$ ⟷ *x* = $false_n$
mk-next-pow n x = $true_n$ ⟷ *x* = $true_n$
is-constant (*mk-next-pow n x*) ⟷ *is-constant x*
⟨*proof*⟩

lemma *is-constant-constructors-simps2*:
is-constant (*mk-and x y*) ⟷ (*x* = $true_n$ ∧ *y* = $true_n$ ∨ *x* = $false_n$ ∨ *y* = $false_n$)
is-constant (*mk-or x y*) ⟷ (*x* = $false_n$ ∧ *y* = $false_n$ ∨ *x* = $true_n$ ∨ *y* = $true_n$)
is-constant (*mk-finally x*) ⟷ *is-constant x*
is-constant (*mk-globally x*) ⟷ *is-constant x*

is-constant (*mk-until y x*) $\longleftrightarrow$ *is-constant x*
is-constant (*mk-release y x*) $\longleftrightarrow$ *is-constant x*
is-constant (*mk-next x*) $\longleftrightarrow$ *is-constant x*
⟨*proof*⟩

lemma *is-constant-constructors-simps3*:
is-constant (*mk-weak-until x y*) $\longleftrightarrow$ ($x = false_n \wedge y = false_n \vee x = true_n \vee y = true_n$)
is-constant (*mk-strong-release x y*) $\longleftrightarrow$ ($x = true_n \wedge y = true_n \vee x = false_n \vee y = false_n$)
⟨*proof*⟩

lemma *is-constant-semantics*:
is-constant $\varphi \Longrightarrow ((\forall\, w.\ w \models_n \varphi) \vee \neg(\exists\, w.\ w \models_n \varphi))$
⟨*proof*⟩

lemma *until-constant-simp*:
is-constant $\psi \Longrightarrow w \models_n \varphi\ U_n\ \psi \longleftrightarrow w \models_n \psi$
⟨*proof*⟩

lemma *release-constant-simp*:
is-constant $\psi \Longrightarrow w \models_n \varphi\ R_n\ \psi \longleftrightarrow w \models_n \psi$
⟨*proof*⟩

lemma *mk-next-pow-dist*:
mk-next-pow $(i + j)\ \varphi =$ *mk-next-pow i* (*mk-next-pow j* φ)
⟨*proof*⟩

lemma *mk-next-pow-until*:
suffix (*min i j*) $w \models_n$ (*mk-next-pow* $(i - j)\ \varphi)\ U_n$ (*mk-next-pow* $(j - i)\ \psi) \longleftrightarrow w \models_n$ (*mk-next-pow i* $\varphi)\ U_n$ (*mk-next-pow j* ψ)
⟨*proof*⟩

lemma *mk-next-pow-release*:
suffix (*min i j*) $w \models_n$ (*mk-next-pow* $(i - j)\ \varphi)\ R_n$ (*mk-next-pow* $(j - i)\ \psi) \longleftrightarrow w \models_n$ (*mk-next-pow i* $\varphi)\ R_n$ (*mk-next-pow j* ψ)
⟨*proof*⟩

2.3 X-Normalisation

The following rewrite functions pulls the X-operator up in the syntax tree. This preprocessing step enables the removal of X-operators in front of suspendable formulas. Furthermore constants are removed as far as possible.

fun *the-enat-0* :: *enat* ⇒ *nat*
where
  the-enat-0 i = *i*
| *the-enat-0* ∞ = *0*

lemma *the-enat-0-simp* [*simp*]:
  the-enat-0 0 = *0*
  the-enat-0 1 = *1*
  ⟨*proof*⟩

fun *combine* :: (′*a ltln* ⇒ ′*a ltln* ⇒ ′*a ltln*) ⇒ (′*a ltln* ∗ *enat*) ⇒ (′*a ltln* ∗ *enat*) ⇒ (′*a ltln* ∗ *enat*)
where
  combine binop (φ, *i*) (ψ, *j*) = (
    let
     χ = *binop* (*mk-next-pow* (*the-enat-0* (*i* − *j*)) φ) (*mk-next-pow* (*the-enat-0* (*j* − *i*)) ψ)
    in
      (χ, *if is-constant* χ *then* ∞ *else min i j*))

lemma *fst-combine*:
  fst (*combine binop* (φ, *i*) (ψ, *j*)) = *binop* (*mk-next-pow* (*the-enat-0* (*i* − *j*)) φ) (*mk-next-pow* (*the-enat-0* (*j* − *i*)) ψ)
  ⟨*proof*⟩

abbreviation *to-ltln* :: (′*a ltln* ∗ *enat*) ⇒ ′*a ltln*
where
  to-ltln x ≡ *mk-next-pow* (*the-enat-0* (*snd x*)) (*fst x*)

fun *rewrite-X-enat* :: ′*a ltln* ⇒ (′*a ltln* ∗ *enat*)
where
  rewrite-X-enat $true_n$ = ($true_n$, ∞)
| *rewrite-X-enat* $false_n$ = ($false_n$, ∞)
| *rewrite-X-enat* $prop_n(a)$ = ($prop_n(a)$, *0*)
| *rewrite-X-enat* $nprop_n(a)$ = ($nprop_n(a)$, *0*)
| *rewrite-X-enat* (φ and_n ψ) = *combine mk-and* (*rewrite-X-enat* φ) (*rewrite-X-enat* ψ)
| *rewrite-X-enat* (φ or_n ψ) = *combine mk-or* (*rewrite-X-enat* φ) (*rewrite-X-enat* ψ)
| *rewrite-X-enat* (φ U_n ψ) = *combine mk-until* (*rewrite-X-enat* φ) (*rewrite-X-enat* ψ)
| *rewrite-X-enat* (φ R_n ψ) = *combine mk-release* (*rewrite-X-enat* φ) (*rewrite-X-enat* ψ)
| *rewrite-X-enat* (φ W_n ψ) = *combine mk-weak-until* (*rewrite-X-enat* φ)

(*rewrite-X-enat* ψ)
| *rewrite-X-enat* (φ M_n ψ) = *combine mk-strong-release* (*rewrite-X-enat* φ) (*rewrite-X-enat* ψ)
| *rewrite-X-enat* (X_n φ) = ($\lambda(\varphi,\ n).\ (\varphi,\ eSuc\ n)$) (*rewrite-X-enat* φ)

definition
rewrite-X φ = *to-ltln* (*rewrite-X-enat* φ)

lemma *combine-infinity-invariant*:
assumes $i = \infty \longleftrightarrow$ *is-constant* x
assumes $j = \infty \longleftrightarrow$ *is-constant* y
shows *combine mk-and* (x, i) (y, j) = (z, k) $\Longrightarrow$ ($k = \infty \longleftrightarrow$ *is-constant* z)
and *combine mk-or* (x, i) (y, j) = (z, k) $\Longrightarrow$ ($k = \infty \longleftrightarrow$ *is-constant* z)
and *combine mk-until* (x, i) (y, j) = (z, k) $\Longrightarrow$ ($k = \infty \longleftrightarrow$ *is-constant* z)
and *combine mk-release* (x, i) (y, j) = (z, k) $\Longrightarrow$ ($k = \infty \longleftrightarrow$ *is-constant* z)
and *combine mk-weak-until* (x, i) (y, j) = (z, k) $\Longrightarrow$ ($k = \infty \longleftrightarrow$ *is-constant* z)
and *combine mk-strong-release* (x, i) (y, j) = (z, k) $\Longrightarrow$ ($k = \infty \longleftrightarrow$ *is-constant* z)
⟨*proof*⟩

lemma *combine-and-or-semantics*:
assumes $i = \infty \longleftrightarrow$ *is-constant* φ
assumes $j = \infty \longleftrightarrow$ *is-constant* ψ
shows $w \models_n$ *to-ltln* (*combine mk-and* (φ, i) (ψ, j)) $\longleftrightarrow$ $w \models_n$ *to-ltln* (φ, i) and_n *to-ltln* (ψ, j)
and $w \models_n$ *to-ltln* (*combine mk-or* (φ, i) (ψ, j)) $\longleftrightarrow$ $w \models_n$ *to-ltln* (φ, i) or_n *to-ltln* (ψ, j)
⟨*proof*⟩

lemma *combine-until-release-semantics*:
assumes $i = \infty \longleftrightarrow$ *is-constant* φ
assumes $j = \infty \longleftrightarrow$ *is-constant* ψ
shows $w \models_n$ *to-ltln* (*combine mk-until* (φ, i) (ψ, j)) $\longleftrightarrow$ $w \models_n$ *to-ltln* (φ, i) U_n *to-ltln* (ψ, j)
and $w \models_n$ *to-ltln* (*combine mk-release* (φ, i) (ψ, j)) $\longleftrightarrow$ $w \models_n$ *to-ltln* (φ, i) R_n *to-ltln* (ψ, j)
⟨*proof*⟩

lemma *combine-weak-until-strong-release-semantics*:
assumes $i = \infty \longleftrightarrow$ *is-constant* φ
assumes $j = \infty \longleftrightarrow$ *is-constant* ψ
shows $w \models_n$ *to-ltln* (*combine mk-weak-until* (φ, i) (ψ, j)) $\longleftrightarrow$ $w \models_n$ *to-ltln* (φ, i) W_n *to-ltln* (ψ, j)
and $w \models_n$ *to-ltln* (*combine mk-strong-release* (φ, i) (ψ, j)) $\longleftrightarrow$ $w \models_n$ *to-ltln* (φ, i) M_n *to-ltln* (ψ, j)
⟨*proof*⟩

lemma *rewrite-X-enat-infinity-invariant*:
snd (*rewrite-X-enat* φ) $= \infty \longleftrightarrow$ *is-constant* (*fst* (*rewrite-X-enat* φ))
⟨*proof*⟩

lemma *rewrite-X-enat-correct*:
$w \models_n \varphi \longleftrightarrow w \models_n$ *to-ltln* (*rewrite-X-enat* φ)
⟨*proof*⟩

lemma *rewrite-X-sound* [*simp*]:
$w \models_n$ *rewrite-X* $\varphi \longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

2.4 Pure Eventual, Pure Universal, and Suspendable Formulas

fun *pure-eventual* :: $'a$ *ltln* $\Rightarrow$ *bool*
where
pure-eventual $true_n$ = *True*
| *pure-eventual* $false_n$ = *True*
| *pure-eventual* (μ and_n μ') = (*pure-eventual* $\mu \wedge$ *pure-eventual* μ')
| *pure-eventual* (μ or_n μ') = (*pure-eventual* $\mu \wedge$ *pure-eventual* μ')
| *pure-eventual* (μ U_n μ') = ($\mu = true_n \vee$ *pure-eventual* μ')
| *pure-eventual* (μ R_n μ') = (*pure-eventual* $\mu \wedge$ *pure-eventual* μ')
| *pure-eventual* (μ W_n μ') = (*pure-eventual* $\mu \wedge$ *pure-eventual* μ')
| *pure-eventual* (μ M_n μ') = (*pure-eventual* $\mu \wedge$ *pure-eventual* $\mu' \vee \mu' = true_n$)
| *pure-eventual* (X_n μ) = *pure-eventual* μ
| *pure-eventual* - = *False*

fun *pure-universal* :: $'a$ *ltln* $\Rightarrow$ *bool*
where
pure-universal $true_n$ = *True*
| *pure-universal* $false_n$ = *True*

| *pure-universal* (ν *and*$_n$ ν') = (*pure-universal* ν $\wedge$ *pure-universal* ν')
| *pure-universal* (ν *or*$_n$ ν') = (*pure-universal* ν $\wedge$ *pure-universal* ν')
| *pure-universal* (ν U_n ν') = (*pure-universal* ν $\wedge$ *pure-universal* ν')
| *pure-universal* (ν R_n ν') = (ν = *false*$_n$ $\vee$ *pure-universal* ν')
| *pure-universal* (ν W_n ν') = (*pure-universal* ν $\wedge$ *pure-universal* ν' $\vee$ ν' = *false*$_n$)
| *pure-universal* (ν M_n ν') = (*pure-universal* ν $\wedge$ *pure-universal* ν')
| *pure-universal* (X_n ν) = *pure-universal* ν
| *pure-universal* - = *False*

fun *suspendable* :: *'a ltln* $\Rightarrow$ *bool*
where
suspendable *true*$_n$ = *True*
| *suspendable* *false*$_n$ = *True*
| *suspendable* (ξ *and*$_n$ ξ') = (*suspendable* ξ $\wedge$ *suspendable* ξ')
| *suspendable* (ξ *or*$_n$ ξ') = (*suspendable* ξ $\wedge$ *suspendable* ξ')
| *suspendable* (φ U_n ξ) = (φ = *true*$_n$ $\wedge$ *pure-universal* ξ $\vee$ *suspendable* ξ)
| *suspendable* (φ R_n ξ) = (φ = *false*$_n$ $\wedge$ *pure-eventual* ξ $\vee$ *suspendable* ξ)
| *suspendable* (φ W_n ξ) = (*suspendable* φ $\wedge$ *suspendable* ξ $\vee$ *pure-eventual* φ $\wedge$ ξ = *false*$_n$)
| *suspendable* (φ M_n ξ) = (*suspendable* φ $\wedge$ *suspendable* ξ $\vee$ *pure-universal* φ $\wedge$ ξ = *true*$_n$)
| *suspendable* (X_n ξ) = *suspendable* ξ
| *suspendable* - = *False*

lemma *pure-eventual-left-append*:
pure-eventual μ $\Longrightarrow$ $w \models_n \mu$ $\Longrightarrow$ $(u \frown w) \models_n \mu$
⟨*proof*⟩

lemma *pure-universal-suffix-closed*:
pure-universal ν $\Longrightarrow$ $(u \frown w) \models_n \nu$ $\Longrightarrow$ $w \models_n \nu$
⟨*proof*⟩

lemma *suspendable-prefix-invariant*:
suspendable ξ $\Longrightarrow$ $(u \frown w) \models_n \xi \longleftrightarrow w \models_n \xi$
⟨*proof*⟩

theorem *pure-eventual-until-simp*:
assumes *pure-eventual* μ
shows $w \models_n \varphi\ U_n\ \mu \longleftrightarrow w \models_n \mu$
⟨*proof*⟩

theorem *pure-universal-release-simp*:
assumes *pure-universal* ν

shows $w \models_n \varphi\ R_n\ \nu \longleftrightarrow w \models_n \nu$
⟨*proof*⟩

theorem *pure-universal-weak-until-simp*:
assumes *pure-universal* φ **and** *pure-universal* ψ
shows $w \models_n \varphi\ W_n\ \psi \longleftrightarrow w \models_n \varphi\ or_n\ \psi$
⟨*proof*⟩

theorem *pure-eventual-strong-release-simp*:
assumes *pure-eventual* φ **and** *pure-eventual* ψ
shows $w \models_n \varphi\ M_n\ \psi \longleftrightarrow w \models_n \varphi\ and_n\ \psi$
⟨*proof*⟩

theorem *suspendable-formula-simp*:
assumes *suspendable* ξ
shows $w \models_n X_n\ \xi \longleftrightarrow w \models_n \xi$ (**is** *?t1*)
and $w \models_n \varphi\ U_n\ \xi \longleftrightarrow w \models_n \xi$ (**is** *?t2*)
and $w \models_n \varphi\ R_n\ \xi \longleftrightarrow w \models_n \xi$ (**is** *?t3*)
⟨*proof*⟩

theorem *suspendable-formula-simp2*:
assumes *suspendable* φ **and** *suspendable* ψ
shows $w \models_n \varphi\ W_n\ \psi \longleftrightarrow w \models_n \varphi\ or_n\ \psi$ (**is** *?t1*)
and $w \models_n \varphi\ M_n\ \psi \longleftrightarrow w \models_n \varphi\ and_n\ \psi$ (**is** *?t2*)
⟨*proof*⟩

fun *rewrite-modal* :: *'a ltln* ⇒ *'a ltln*
where
rewrite-modal $true_n$ = $true_n$
| *rewrite-modal* $false_n$ = $false_n$
| *rewrite-modal* (φ and_n ψ) = (*rewrite-modal* φ and_n *rewrite-modal* ψ)
| *rewrite-modal* (φ or_n ψ) = (*rewrite-modal* φ or_n *rewrite-modal* ψ)
| *rewrite-modal* (φ U_n ψ) = (*if pure-eventual* ψ ∨ *suspendable* ψ *then rewrite-modal* ψ *else* (*rewrite-modal* φ U_n *rewrite-modal* ψ))
| *rewrite-modal* (φ R_n ψ) = (*if pure-universal* ψ ∨ *suspendable* ψ *then rewrite-modal* ψ *else* (*rewrite-modal* φ R_n *rewrite-modal* ψ))
| *rewrite-modal* (φ W_n ψ) = (*if pure-universal* φ ∧ *pure-universal* ψ ∨ *suspendable* φ ∧ *suspendable* ψ *then* (*rewrite-modal* φ or_n *rewrite-modal* ψ) *else* (*rewrite-modal* φ W_n *rewrite-modal* ψ))
| *rewrite-modal* (φ M_n ψ) = (*if pure-eventual* φ ∧ *pure-eventual* ψ ∨ *suspendable* φ ∧ *suspendable* ψ *then* (*rewrite-modal* φ and_n *rewrite-modal* ψ) *else* (*rewrite-modal* φ M_n *rewrite-modal* ψ))
| *rewrite-modal* (X_n φ) = (*if suspendable* φ *then rewrite-modal* φ *else* X_n

(*rewrite-modal* φ))
| *rewrite-modal* $\varphi = \varphi$

lemma *rewrite-modal-sound* [*simp*]:
$w \models_n$ *rewrite-modal* $\varphi \longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

lemma *rewrite-modal-size*:
size (*rewrite-modal* φ) $\leq$ *size* φ
⟨*proof*⟩

2.5 Syntactical Implication Based Simplification

inductive *syntactical-implies* :: $'a$ *ltln* $\Rightarrow$ $'a$ *ltln* $\Rightarrow$ *bool* (‹- $\vdash_s$ -› [*80*, *80*])
where
- $\vdash_s$ $true_n$
| $false_n \vdash_s$ -
| $\varphi = \varphi \Longrightarrow \varphi \vdash_s \varphi$

| $\varphi \vdash_s \chi \Longrightarrow (\varphi\ and_n\ \psi) \vdash_s \chi$
| $\psi \vdash_s \chi \Longrightarrow (\varphi\ and_n\ \psi) \vdash_s \chi$
| $\varphi \vdash_s \psi \Longrightarrow \varphi \vdash_s \chi \Longrightarrow \varphi \vdash_s (\psi\ and_n\ \chi)$

| $\varphi \vdash_s \psi \Longrightarrow \varphi \vdash_s (\psi\ or_n\ \chi)$
| $\varphi \vdash_s \chi \Longrightarrow \varphi \vdash_s (\psi\ or_n\ \chi)$
| $\varphi \vdash_s \chi \Longrightarrow \psi \vdash_s \chi \Longrightarrow (\varphi\ or_n\ \psi) \vdash_s \chi$

| $\varphi \vdash_s \chi \Longrightarrow \varphi \vdash_s (\psi\ U_n\ \chi)$
| $\varphi \vdash_s \chi \Longrightarrow \psi \vdash_s \chi \Longrightarrow (\varphi\ U_n\ \psi) \vdash_s \chi$
| $\varphi \vdash_s \chi \Longrightarrow \psi \vdash_s \nu \Longrightarrow (\varphi\ U_n\ \psi) \vdash_s (\chi\ U_n\ \nu)$

| $\chi \vdash_s \varphi \Longrightarrow (\psi\ R_n\ \chi) \vdash_s \varphi$
| $\chi \vdash_s \varphi \Longrightarrow \chi \vdash_s \psi \Longrightarrow \chi \vdash_s (\varphi\ R_n\ \psi)$
| $\varphi \vdash_s \chi \Longrightarrow \psi \vdash_s \nu \Longrightarrow (\varphi\ R_n\ \psi) \vdash_s (\chi\ R_n\ \nu)$

| $(false_n\ R_n\ \varphi) \vdash_s \psi \Longrightarrow (false_n\ R_n\ \varphi) \vdash_s X_n\ \psi$
| $\varphi \vdash_s (true_n\ U_n\ \psi) \Longrightarrow (X_n\ \varphi) \vdash_s (true_n\ U_n\ \psi)$
| $\varphi \vdash_s \psi \Longrightarrow (X_n\ \varphi) \vdash_s (X_n\ \psi)$

lemma *syntactical-implies-correct*:
$\varphi \vdash_s \psi \Longrightarrow w \models_n \varphi \Longrightarrow w \models_n \psi$
⟨*proof*⟩

fun *rewrite-syn-imp*

where
rewrite-syn-imp (φ and_n ψ) = (
if $\varphi \vdash_s \psi$ *then*
rewrite-syn-imp φ
else if $\psi \vdash_s \varphi$ *then*
rewrite-syn-imp ψ
else if $\varphi \vdash_s (not_n\ \psi) \vee \psi \vdash_s (not_n\ \varphi)$ *then*
$false_n$
else
mk-and (*rewrite-syn-imp* φ) (*rewrite-syn-imp* ψ))
| *rewrite-syn-imp* (φ or_n ψ) = (
if $\varphi \vdash_s \psi$ *then*
rewrite-syn-imp ψ
else if $\psi \vdash_s \varphi$ *then*
rewrite-syn-imp φ
else if $(not_n\ \varphi) \vdash_s \psi \vee (not_n\ \psi) \vdash_s \varphi$ *then*
$true_n$
else
mk-or (*rewrite-syn-imp* φ) (*rewrite-syn-imp* ψ))
| *rewrite-syn-imp* (φ U_n ψ) = (
if $\varphi \vdash_s \psi$ *then*
rewrite-syn-imp ψ
else if $(not_n\ \varphi) \vdash_s \psi$ *then*
mk-finally (*rewrite-syn-imp* ψ)
else
mk-until (*rewrite-syn-imp* φ) (*rewrite-syn-imp* ψ))
| *rewrite-syn-imp* (φ R_n ψ) = (
if $\psi \vdash_s \varphi$ *then*
rewrite-syn-imp ψ
else if $\psi \vdash_s (not_n\ \varphi)$ *then*
mk-globally (*rewrite-syn-imp* ψ)
else
mk-release (*rewrite-syn-imp* φ) (*rewrite-syn-imp* ψ))
| *rewrite-syn-imp* (X_n φ) = *mk-next* (*rewrite-syn-imp* φ)
| *rewrite-syn-imp* φ = φ

lemma *rewrite-syn-imp-sound*:
$w \models_n$ *rewrite-syn-imp* $\varphi \longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

2.6 Iterated Rewriting

fun *iterate*
where

iterate f x 0 = x
| *iterate f x (Suc n) = (let x′ = f x in if x = x′ then x else iterate f x′ n)*

definition
rewrite-iter-fast φ ≡ iterate (rewrite-modal o rewrite-X) φ (size φ)

definition
rewrite-iter-slow φ ≡ iterate (rewrite-syn-imp o rewrite-modal o rewrite-X) φ (size φ)

The rewriting functions defined in the previous subsections can be used as-is. However, in the most cases one wants to iterate these rules until the formula cannot be simplified further. *rewrite-iter-fast* pulls X operators up in the syntax tree and the uses the "modal" simplification rules. *rewrite-iter-slow* additionally tries to simplify the formula using syntactic implication checking.

lemma *iterate-sound*:
assumes $\bigwedge\varphi.\ w \models_n f\ \varphi \longleftrightarrow w \models_n \varphi$
shows $w \models_n$ *iterate f φ n* $\longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

theorem *rewrite-iter-fast-sound* [*simp*]:
$w \models_n$ *rewrite-iter-fast φ* $\longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

theorem *rewrite-iter-slow-sound* [*simp*]:
$w \models_n$ *rewrite-iter-slow φ* $\longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

2.7 Preservation of atoms

lemma *iterate-atoms*:
assumes
$\bigwedge\varphi.$ *atoms-ltln (f φ) ⊆ atoms-ltln φ*
shows
atoms-ltln (iterate f φ n) ⊆ atoms-ltln φ
⟨*proof*⟩

lemma *rewrite-modal-atoms*:
atoms-ltln (rewrite-modal φ) ⊆ atoms-ltln φ
⟨*proof*⟩

lemma *mk-and-atoms*:
atoms-ltln (mk-and φ ψ) ⊆ atoms-ltln φ ∪ atoms-ltln ψ

⟨*proof*⟩

lemma *mk-or-atoms*:
atoms-ltln (*mk-or* φ ψ) $\subseteq$ *atoms-ltln* φ $\cup$ *atoms-ltln* ψ
⟨*proof*⟩

lemma *remove-strong-ops-atoms*:
atoms-ltln (*remove-strong-ops* φ) $\subseteq$ *atoms-ltln* φ
⟨*proof*⟩

lemma *remove-weak-ops-atoms*:
atoms-ltln (*remove-weak-ops* φ) $\subseteq$ *atoms-ltln* φ
⟨*proof*⟩

lemma *mk-finally-atoms*:
atoms-ltln (*mk-finally* φ) $\subseteq$ *atoms-ltln* φ
⟨*proof*⟩

lemma *mk-globally-atoms*:
atoms-ltln (*mk-globally* φ) $\subseteq$ *atoms-ltln* φ
⟨*proof*⟩

lemma *mk-until-atoms*:
atoms-ltln (*mk-until* φ ψ) $\subseteq$ *atoms-ltln* φ $\cup$ *atoms-ltln* ψ
⟨*proof*⟩

lemma *mk-release-atoms*:
atoms-ltln (*mk-release* φ ψ) $\subseteq$ *atoms-ltln* φ $\cup$ *atoms-ltln* ψ
⟨*proof*⟩

lemma *mk-weak-until-atoms*:
atoms-ltln (*mk-weak-until* φ ψ) $\subseteq$ *atoms-ltln* φ $\cup$ *atoms-ltln* ψ
⟨*proof*⟩

lemma *mk-strong-release-atoms*:
atoms-ltln (*mk-strong-release* φ ψ) $\subseteq$ *atoms-ltln* φ $\cup$ *atoms-ltln* ψ
⟨*proof*⟩

lemma *mk-next-atoms*:
atoms-ltln (*mk-next* φ) = *atoms-ltln* φ
⟨*proof*⟩

lemma *mk-next-pow-atoms*:
atoms-ltln (*mk-next-pow* n φ) = *atoms-ltln* φ

⟨*proof*⟩

lemma *combine-atoms*:
assumes
⋀φ ψ. *atoms-ltln* (*f* φ ψ) ⊆ *atoms-ltln* φ ∪ *atoms-ltln* ψ
shows
atoms-ltln (*fst* (*combine f x y*)) ⊆ *atoms-ltln* (*fst x*) ∪ *atoms-ltln* (*fst y*)
⟨*proof*⟩

lemmas *combine-mk-atoms* =
combine-atoms[*OF mk-and-atoms*]
combine-atoms[*OF mk-or-atoms*]
combine-atoms[*OF mk-until-atoms*]
combine-atoms[*OF mk-release-atoms*]
combine-atoms[*OF mk-weak-until-atoms*]
combine-atoms[*OF mk-strong-release-atoms*]

lemma *rewrite-X-enat-atoms*:
atoms-ltln (*fst* (*rewrite-X-enat* φ)) ⊆ *atoms-ltln* φ
⟨*proof*⟩

lemma *rewrite-X-atoms*:
atoms-ltln (*rewrite-X* φ) ⊆ *atoms-ltln* φ
⟨*proof*⟩

lemma *rewrite-syn-imp-atoms*:
atoms-ltln (*rewrite-syn-imp* φ) ⊆ *atoms-ltln* φ
⟨*proof*⟩

lemma *rewrite-iter-fast-atoms*:
atoms-ltln (*rewrite-iter-fast* φ) ⊆ *atoms-ltln* φ
⟨*proof*⟩

lemma *rewrite-iter-slow-atoms*:
atoms-ltln (*rewrite-iter-slow* φ) ⊆ *atoms-ltln* φ
⟨*proof*⟩

2.8 Simplifier

We now define a convenience wrapper for the rewriting engine

datatype *mode* = *Nop* | *Fast* | *Slow*

fun *simplify* :: *mode* ⇒ *'a ltln* ⇒ *'a ltln*

where
simplify Nop = *id*
| *simplify Fast* = *rewrite-iter-fast*
| *simplify Slow* = *rewrite-iter-slow*

theorem *simplify-correct*:
$w \models_n$ *simplify* $m\ \varphi \longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

lemma *simplify-atoms*:
atoms-ltln (*simplify* $m\ \varphi$) $\subseteq$ *atoms-ltln* φ
⟨*proof*⟩

2.9 Code Generation

code-pred *syntactical-implies* ⟨*proof*⟩

export-code *simplify* **checking**

lemma *rewrite-iter-fast* (F_n (G_n (X_n $prop_n(''a'')$))) = (F_n (G_n $prop_n(''a'')$))
⟨*proof*⟩

lemma *rewrite-iter-fast* (X_n $prop_n(''a'')$ U_n (X_n $nprop_n(''a'')$)) = X_n ($prop_n(''a'')$ U_n $nprop_n(''a'')$)
⟨*proof*⟩

lemma *rewrite-iter-slow* (X_n $prop_n(''a'')$ U_n (X_n $nprop_n(''a'')$)) = X_n (F_n $nprop_n(''a'')$)
⟨*proof*⟩

end

3 Equivalence Relations for LTL formulas

theory *Equivalence-Relations*
imports
LTL
begin

3.1 Language Equivalence

definition *ltl-lang-equiv* :: *'a ltln* ⇒ *'a ltln* ⇒ *bool* (**infix** ‹$\sim_L$› 75)
where
$\varphi \sim_L \psi \equiv \forall\, w.\ w \models_n \varphi \longleftrightarrow w \models_n \psi$

lemma *ltl-lang-equiv-equivp*:
$\mathit{equivp}\ (\sim_L)$
⟨*proof*⟩

lemma *ltl-lang-equiv-and-true*[*simp*]:
$\varphi_1\ \mathit{and}_n\ \varphi_2 \sim_L \mathit{true}_n \longleftrightarrow \varphi_1 \sim_L \mathit{true}_n \wedge \varphi_2 \sim_L \mathit{true}_n$
⟨*proof*⟩

lemma *ltl-lang-equiv-and-false*[*intro*, *simp*]:
$\varphi_1 \sim_L \mathit{false}_n \Longrightarrow \varphi_1\ \mathit{and}_n\ \varphi_2 \sim_L \mathit{false}_n$
$\varphi_2 \sim_L \mathit{false}_n \Longrightarrow \varphi_1\ \mathit{and}_n\ \varphi_2 \sim_L \mathit{false}_n$
⟨*proof*⟩

lemma *ltl-lang-equiv-or-false*[*simp*]:
$\varphi_1\ \mathit{or}_n\ \varphi_2 \sim_L \mathit{false}_n \longleftrightarrow \varphi_1 \sim_L \mathit{false}_n \wedge \varphi_2 \sim_L \mathit{false}_n$
⟨*proof*⟩

lemma *ltl-lang-equiv-or-const*[*intro*, *simp*]:
$\varphi_1 \sim_L \mathit{true}_n \Longrightarrow \varphi_1\ \mathit{or}_n\ \varphi_2 \sim_L \mathit{true}_n$
$\varphi_2 \sim_L \mathit{true}_n \Longrightarrow \varphi_1\ \mathit{or}_n\ \varphi_2 \sim_L \mathit{true}_n$
⟨*proof*⟩

3.2 Propositional Equivalence

fun *ltl-prop-entailment* :: *'a ltln set* ⇒ *'a ltln* ⇒ *bool* (**infix** ‹$\models_P$› *80*)
where
$\mathcal{A} \models_P \mathit{true}_n = \mathit{True}$
| $\mathcal{A} \models_P \mathit{false}_n = \mathit{False}$
| $\mathcal{A} \models_P \varphi\ \mathit{and}_n\ \psi = (\mathcal{A} \models_P \varphi \wedge \mathcal{A} \models_P \psi)$
| $\mathcal{A} \models_P \varphi\ \mathit{or}_n\ \psi = (\mathcal{A} \models_P \varphi \vee \mathcal{A} \models_P \psi)$
| $\mathcal{A} \models_P \varphi = (\varphi \in \mathcal{A})$

lemma *ltl-prop-entailment-monotonI*[*intro*]:
$S \models_P \varphi \Longrightarrow S \subseteq S' \Longrightarrow S' \models_P \varphi$
⟨*proof*⟩

lemma *ltl-models-equiv-prop-entailment*:
$w \models_n \varphi \longleftrightarrow \{\psi.\ w \models_n \psi\} \models_P \varphi$
⟨*proof*⟩

definition *ltl-prop-equiv* :: *'a ltln* ⇒ *'a ltln* ⇒ *bool* (**infix** ‹$\sim_P$› *75*)
where
$\varphi \sim_P \psi \equiv \forall \mathcal{A}.\ \mathcal{A} \models_P \varphi \longleftrightarrow \mathcal{A} \models_P \psi$

definition *ltl-prop-implies* :: *'a ltln* ⇒ *'a ltln* ⇒ *bool* (**infix** ‹⟶$_P$› *75*)
where
$\varphi \longrightarrow_P \psi \equiv \forall \mathcal{A}.\ \mathcal{A} \models_P \varphi \longrightarrow \mathcal{A} \models_P \psi$

lemma *ltl-prop-implies-equiv*:
$\varphi \sim_P \psi \longleftrightarrow (\varphi \longrightarrow_P \psi \wedge \psi \longrightarrow_P \varphi)$
⟨*proof*⟩

lemma *ltl-prop-equiv-equivp*:
equivp $(\sim_P)$
⟨*proof*⟩

lemma *ltl-prop-equiv-trans*[*trans*]:
$\varphi \sim_P \psi \Longrightarrow \psi \sim_P \chi \Longrightarrow \varphi \sim_P \chi$
⟨*proof*⟩

lemma *ltl-prop-equiv-true*:
$\varphi \sim_P true_n \longleftrightarrow \{\} \models_P \varphi$
⟨*proof*⟩

lemma *ltl-prop-equiv-false*:
$\varphi \sim_P false_n \longleftrightarrow \neg\ UNIV \models_P \varphi$
⟨*proof*⟩

lemma *ltl-prop-equiv-true-implies-true*:
$x \sim_P true_n \Longrightarrow x \longrightarrow_P y \Longrightarrow y \sim_P true_n$
⟨*proof*⟩

lemma *ltl-prop-equiv-false-implied-by-false*:
$y \sim_P false_n \Longrightarrow x \longrightarrow_P y \Longrightarrow x \sim_P false_n$
⟨*proof*⟩

lemma *ltl-prop-implication-implies-ltl-implication*:
$w \models_n \varphi \Longrightarrow \varphi \longrightarrow_P \psi \Longrightarrow w \models_n \psi$
⟨*proof*⟩

lemma *ltl-prop-equiv-implies-ltl-lang-equiv*:
$\varphi \sim_P \psi \Longrightarrow \varphi \sim_L \psi$
⟨*proof*⟩

lemma *ltl-prop-equiv-lt-ltl-lang-equiv*[*simp*]:
$(\sim_P) \leq (\sim_L)$
⟨*proof*⟩

3.3 Constants Equivalence

datatype *tvl* = *Yes* | *No* | *Maybe*

definition *eval-and* :: *tvl* ⇒ *tvl* ⇒ *tvl*
where
eval-and φ ψ =
(*case* (φ, ψ) *of*
(*Yes*, *Yes*) ⇒ *Yes*
| (*No*, -) ⇒ *No*
| (-, *No*) ⇒ *No*
| - ⇒ *Maybe*)

definition *eval-or* :: *tvl* ⇒ *tvl* ⇒ *tvl*
where
eval-or φ ψ =
(*case* (φ, ψ) *of*
(*No*, *No*) ⇒ *No*
| (*Yes*, -) ⇒ *Yes*
| (-, *Yes*) ⇒ *Yes*
| - ⇒ *Maybe*)

fun *eval* :: *'a ltln* ⇒ *tvl*
where
eval $true_n$ = *Yes*
| *eval* $false_n$ = *No*
| *eval* (φ and_n ψ) = *eval-and* (*eval* φ) (*eval* ψ)
| *eval* (φ or_n ψ) = *eval-or* (*eval* φ) (*eval* ψ)
| *eval* φ = *Maybe*

lemma *eval-and-const*[*simp*]:
eval-and φ ψ = *No* ⟷ φ = *No* ∨ ψ = *No*
eval-and φ ψ = *Yes* ⟷ φ = *Yes* ∧ ψ = *Yes*
⟨*proof*⟩

lemma *eval-or-const*[*simp*]:
eval-or φ ψ = *Yes* ⟷ φ = *Yes* ∨ ψ = *Yes*
eval-or φ ψ = *No* ⟷ φ = *No* ∧ ψ = *No*
⟨*proof*⟩

lemma *eval-prop-entailment*:
eval φ = *Yes* ⟷ {} $\models_P$ φ
eval φ = *No* ⟷ ¬ *UNIV* $\models_P$ φ
⟨*proof*⟩

definition *ltl-const-equiv* :: *'a ltln* ⇒ *'a ltln* ⇒ *bool* (**infix** ‹∼$_C$› *75*)
where
$\varphi \sim_C \psi \equiv \varphi = \psi \vee (eval\ \varphi = eval\ \psi \wedge eval\ \psi \neq Maybe)$

lemma *ltl-const-equiv-equivp*:
equivp $(\sim_C)$
⟨*proof*⟩

lemma *ltl-const-equiv-const*:
$\varphi \sim_C true_n \longleftrightarrow eval\ \varphi = Yes$
$\varphi \sim_C false_n \longleftrightarrow eval\ \varphi = No$
⟨*proof*⟩

lemma *ltl-const-equiv-and-const*[*simp*]:
$\varphi_1\ and_n\ \varphi_2 \sim_C true_n \longleftrightarrow \varphi_1 \sim_C true_n \wedge \varphi_2 \sim_C true_n$
$\varphi_1\ and_n\ \varphi_2 \sim_C false_n \longleftrightarrow \varphi_1 \sim_C false_n \vee \varphi_2 \sim_C false_n$
⟨*proof*⟩

lemma *ltl-const-equiv-or-const*[*simp*]:
$\varphi_1\ or_n\ \varphi_2 \sim_C true_n \longleftrightarrow \varphi_1 \sim_C true_n \vee \varphi_2 \sim_C true_n$
$\varphi_1\ or_n\ \varphi_2 \sim_C false_n \longleftrightarrow \varphi_1 \sim_C false_n \wedge \varphi_2 \sim_C false_n$
⟨*proof*⟩

lemma *ltl-const-equiv-other*[*simp*]:
$\varphi \sim_C prop_n(a) \longleftrightarrow \varphi = prop_n(a)$
$\varphi \sim_C nprop_n(a) \longleftrightarrow \varphi = nprop_n(a)$
$\varphi \sim_C X_n\ \psi \longleftrightarrow \varphi = X_n\ \psi$
$\varphi \sim_C \psi_1\ U_n\ \psi_2 \longleftrightarrow \varphi = \psi_1\ U_n\ \psi_2$
$\varphi \sim_C \psi_1\ R_n\ \psi_2 \longleftrightarrow \varphi = \psi_1\ R_n\ \psi_2$
$\varphi \sim_C \psi_1\ W_n\ \psi_2 \longleftrightarrow \varphi = \psi_1\ W_n\ \psi_2$
$\varphi \sim_C \psi_1\ M_n\ \psi_2 \longleftrightarrow \varphi = \psi_1\ M_n\ \psi_2$
⟨*proof*⟩

lemma *ltl-const-equiv-no-const-singleton*:
$eval\ \psi = Maybe \Longrightarrow \varphi \sim_C \psi \Longrightarrow \varphi = \psi$
⟨*proof*⟩

lemma *ltl-const-equiv-implies-prop-equiv*:
$\varphi \sim_C true_n \longleftrightarrow \varphi \sim_P true_n$
$\varphi \sim_C false_n \longleftrightarrow \varphi \sim_P false_n$
⟨*proof*⟩

lemma *ltl-const-equiv-no-const-prop-equiv*:

eval ψ = *Maybe* $\Longrightarrow$ φ $\sim_C$ ψ $\Longrightarrow$ φ $\sim_P$ ψ
⟨*proof*⟩

lemma *ltl-const-equiv-implies-ltl-prop-equiv*:
φ $\sim_C$ ψ $\Longrightarrow$ φ $\sim_P$ ψ
⟨*proof*⟩

lemma *ltl-const-equiv-lt-ltl-prop-equiv*[*simp*]:
($\sim_C$) $\leq$ ($\sim_P$)
⟨*proof*⟩

3.4 Quotient types

quotient-type $'a$ $ltln_L$ = $'a$ *ltln* / ($\sim_L$)
⟨*proof*⟩

instantiation $ltln_L$ :: (*type*) *equal*
begin

lift-definition $ltln_L$-*eq-test* :: $'a$ $ltln_L$ $\Rightarrow$ $'a$ $ltln_L$ $\Rightarrow$ *bool* **is** λx y. x $\sim_L$ y
⟨*proof*⟩

definition
eq_L: *equal-class.equal* $\equiv$ $ltln_L$-*eq-test*

instance
⟨*proof*⟩

end

quotient-type $'a$ $ltln_P$ = $'a$ *ltln* / ($\sim_P$)
⟨*proof*⟩

instantiation $ltln_P$ :: (*type*) *equal*
begin

lift-definition $ltln_P$-*eq-test* :: $'a$ $ltln_P$ $\Rightarrow$ $'a$ $ltln_P$ $\Rightarrow$ *bool* **is** λx y. x $\sim_P$ y
⟨*proof*⟩

definition
eq_P: *equal-class.equal* $\equiv$ $ltln_P$-*eq-test*

instance

⟨*proof*⟩

end

quotient-type $'a\ ltln_C = 'a\ ltln\ /\ (\sim_C)$
⟨*proof*⟩

instantiation $ltln_C :: (type)\ equal$
begin

lift-definition $ltln_C\text{-}eq\text{-}test :: 'a\ ltln_C \Rightarrow 'a\ ltln_C \Rightarrow bool$ **is** $\lambda x\ y.\ x \sim_C y$
⟨*proof*⟩

definition
eq_C: $equal\text{-}class.equal \equiv ltln_C\text{-}eq\text{-}test$

instance
⟨*proof*⟩

end

3.5 Cardinality of propositional quotient sets

definition $sat\text{-}models :: 'a\ ltln_P \Rightarrow 'a\ ltln\ set\ set$
where
$sat\text{-}models\ \varphi = \{\mathcal{A}.\ \mathcal{A} \models_P rep\text{-}ltln_P\ \varphi\}$

lemma *Rep-Abs-prop-entailment*[*simp*]:
$\mathcal{A} \models_P rep\text{-}ltln_P\ (abs\text{-}ltln_P\ \varphi) = \mathcal{A} \models_P \varphi$
⟨*proof*⟩

lemma *sat-models-Abs*:
$\mathcal{A} \in sat\text{-}models\ (abs\text{-}ltln_P\ \varphi) = \mathcal{A} \models_P \varphi$
⟨*proof*⟩

lemma *sat-models-inj*:
inj sat-models
⟨*proof*⟩

fun $prop\text{-}atoms :: 'a\ ltln \Rightarrow 'a\ ltln\ set$
where
$prop\text{-}atoms\ true_n = \{\}$

| *prop-atoms* $false_n = \{\}$
| *prop-atoms* $(\varphi\ and_n\ \psi) =$ *prop-atoms* $\varphi \cup$ *prop-atoms* ψ
| *prop-atoms* $(\varphi\ or_n\ \psi) =$ *prop-atoms* $\varphi \cup$ *prop-atoms* ψ
| *prop-atoms* $\varphi = \{\varphi\}$

fun *nested-prop-atoms* :: *'a ltln* ⇒ *'a ltln set*
where
nested-prop-atoms $true_n = \{\}$
| *nested-prop-atoms* $false_n = \{\}$
| *nested-prop-atoms* $(\varphi\ and_n\ \psi) =$ *nested-prop-atoms* $\varphi \cup$ *nested-prop-atoms* ψ
| *nested-prop-atoms* $(\varphi\ or_n\ \psi) =$ *nested-prop-atoms* $\varphi \cup$ *nested-prop-atoms* ψ
| *nested-prop-atoms* $(X_n\ \varphi) = \{X_n\ \varphi\} \cup$ *nested-prop-atoms* φ
| *nested-prop-atoms* $(\varphi\ U_n\ \psi) = \{\varphi\ U_n\ \psi\} \cup$ *nested-prop-atoms* $\varphi \cup$ *nested-prop-atoms* ψ
| *nested-prop-atoms* $(\varphi\ R_n\ \psi) = \{\varphi\ R_n\ \psi\} \cup$ *nested-prop-atoms* $\varphi \cup$ *nested-prop-atoms* ψ
| *nested-prop-atoms* $(\varphi\ W_n\ \psi) = \{\varphi\ W_n\ \psi\} \cup$ *nested-prop-atoms* $\varphi \cup$ *nested-prop-atoms* ψ
| *nested-prop-atoms* $(\varphi\ M_n\ \psi) = \{\varphi\ M_n\ \psi\} \cup$ *nested-prop-atoms* $\varphi \cup$ *nested-prop-atoms* ψ
| *nested-prop-atoms* $\varphi = \{\varphi\}$

lemma *prop-atoms-nested-prop-atoms*:
prop-atoms $\varphi \subseteq$ *nested-prop-atoms* φ
⟨*proof*⟩

lemma *prop-atoms-subfrmlsn*:
prop-atoms $\varphi \subseteq$ *subfrmlsn* φ
⟨*proof*⟩

lemma *nested-prop-atoms-subfrmlsn*:
nested-prop-atoms $\varphi \subseteq$ *subfrmlsn* φ
⟨*proof*⟩

lemma *prop-atoms-notin*[*simp*]:
$true_n \notin$ *prop-atoms* φ
$false_n \notin$ *prop-atoms* φ
$\varphi_1\ and_n\ \varphi_2 \notin$ *prop-atoms* φ
$\varphi_1\ or_n\ \varphi_2 \notin$ *prop-atoms* φ
⟨*proof*⟩

lemma *nested-prop-atoms-notin*[*simp*]:

$true_n \notin$ *nested-prop-atoms* φ
$false_n \notin$ *nested-prop-atoms* φ
φ_1 and_n $\varphi_2 \notin$ *nested-prop-atoms* φ
φ_1 or_n $\varphi_2 \notin$ *nested-prop-atoms* φ
⟨*proof*⟩

lemma *prop-atoms-finite*:
finite (*prop-atoms* φ)
⟨*proof*⟩

lemma *nested-prop-atoms-finite*:
finite (*nested-prop-atoms* φ)
⟨*proof*⟩

lemma *prop-atoms-entailment-iff*:
$\varphi \in$ *prop-atoms* $\psi \Longrightarrow \mathcal{A} \models_P \varphi \longleftrightarrow \varphi \in \mathcal{A}$
⟨*proof*⟩

lemma *prop-atoms-entailment-inter*:
prop-atoms $\varphi \subseteq P \Longrightarrow (\mathcal{A} \cap P) \models_P \varphi = \mathcal{A} \models_P \varphi$
⟨*proof*⟩

lemma *nested-prop-atoms-entailment-inter*:
nested-prop-atoms $\varphi \subseteq P \Longrightarrow (\mathcal{A} \cap P) \models_P \varphi = \mathcal{A} \models_P \varphi$
⟨*proof*⟩

lemma *sat-models-inter-inj-helper*:
assumes
prop-atoms $\varphi \subseteq P$
and
prop-atoms $\psi \subseteq P$
and
sat-models (*abs-ltln*$_P$ φ) $\cap$ *Pow* P = *sat-models* (*abs-ltln*$_P$ ψ) $\cap$ *Pow* P
shows
$\varphi \sim_P \psi$
⟨*proof*⟩

lemma *sat-models-inter-inj*:
inj-on ($\lambda\varphi$. *sat-models* $\varphi \cap$ *Pow* P) {*abs-ltln*$_P$ φ |φ. *prop-atoms* $\varphi \subseteq P$}
⟨*proof*⟩

lemma *sat-models-pow-pow*:
{*sat-models* (*abs-ltln*$_P$ φ) $\cap$ *Pow* P | φ. *prop-atoms* $\varphi \subseteq P$} $\subseteq$ *Pow* (*Pow* P)

⟨*proof*⟩

lemma *sat-models-finite*:
finite $P \Longrightarrow$ *finite* $\{$*sat-models* (*abs-ltln*$_P$ φ) $\cap$ *Pow* P | φ. *prop-atoms* $\varphi \subseteq P\}$
⟨*proof*⟩

lemma *sat-models-card*:
finite $P \Longrightarrow$ *card* $(\{$*sat-models* (*abs-ltln*$_P$ φ) $\cap$ *Pow* P | φ. *prop-atoms* $\varphi \subseteq P\}) \leq 2 \mathbin{\hat{}} 2 \mathbin{\hat{}}$ *card* P
⟨*proof*⟩

lemma *image-filter*:
f ‘ $\{g\ a \mid a.\ P\ a\} = \{f\ (g\ a) \mid a.\ P\ a\}$
⟨*proof*⟩

lemma *prop-equiv-finite*:
finite $P \Longrightarrow$ *finite* $\{$*abs-ltln*$_P$ ψ | ψ. *prop-atoms* $\psi \subseteq P\}$
⟨*proof*⟩

lemma *prop-equiv-card*:
finite $P \Longrightarrow$ *card* $\{$*abs-ltln*$_P$ ψ | ψ. *prop-atoms* $\psi \subseteq P\} \leq 2 \mathbin{\hat{}} 2 \mathbin{\hat{}}$ *card* P
⟨*proof*⟩

lemma *prop-equiv-subset*:
$\{$*abs-ltln*$_P$ ψ |ψ. *nested-prop-atoms* $\psi \subseteq P\} \subseteq \{$*abs-ltln*$_P$ ψ |ψ. *prop-atoms* $\psi \subseteq P\}$
⟨*proof*⟩

lemma *prop-equiv-finite′*:
finite $P \Longrightarrow$ *finite* $\{$*abs-ltln*$_P$ ψ | ψ. *nested-prop-atoms* $\psi \subseteq P\}$
⟨*proof*⟩

lemma *prop-equiv-card′*:
finite $P \Longrightarrow$ *card* $\{$*abs-ltln*$_P$ ψ | ψ. *nested-prop-atoms* $\psi \subseteq P\} \leq 2 \mathbin{\hat{}} 2 \mathbin{\hat{}}$ *card* P
⟨*proof*⟩

3.6 Substitution

fun *subst* :: $'a$ *ltln* $\Rightarrow$ ($'a$ *ltln* $\rightharpoonup$ $'a$ *ltln*) $\Rightarrow$ $'a$ *ltln*
where

subst $true_n$ m = $true_n$
| *subst* $false_n$ m = $false_n$
| *subst* (φ and_n ψ) m = *subst* φ m and_n *subst* ψ m
| *subst* (φ or_n ψ) m = *subst* φ m or_n *subst* ψ m
| *subst* φ m = (*case* m φ *of Some* $\psi \Rightarrow \psi$ | *None* $\Rightarrow \varphi$)

Based on Uwe Schoening's Translation Lemma (Logic for CS, p. 54)

lemma *ltl-prop-equiv-subst-S*:
$S \models_P$ *subst* φ m = ((S − *dom* m) ∪ {χ | χ χ'. $\chi \in$ *dom* $m \wedge m\ \chi =$ *Some* $\chi' \wedge S \models_P \chi'$}) $\models_P \varphi$
⟨*proof*⟩

lemma *subst-respects-ltl-prop-entailment*:
$\varphi \longrightarrow_P \psi \Longrightarrow$ *subst* φ $m \longrightarrow_P$ *subst* ψ m
$\varphi \sim_P \psi \Longrightarrow$ *subst* φ $m \sim_P$ *subst* ψ m
⟨*proof*⟩

lemma *eval-subst*:
eval φ = *Yes* $\Longrightarrow$ *eval* (*subst* φ m) = *Yes*
eval φ = *No* $\Longrightarrow$ *eval* (*subst* φ m) = *No*
⟨*proof*⟩

lemma *subst-respects-ltl-const-entailment*:
$\varphi \sim_C \psi \Longrightarrow$ *subst* φ $m \sim_C$ *subst* ψ m
⟨*proof*⟩

3.7 Order of Equivalence Relations

locale *ltl-equivalence* =
fixes
eq :: *'a ltln* ⇒ *'a ltln* ⇒ *bool* (**infix** ‹∼› 75)
assumes
eq-equivp: *equivp* (∼)
and
ge-const-equiv: ($\sim_C$) ≤ (∼)
and
le-lang-equiv: (∼) ≤ ($\sim_L$)
begin

lemma *eq-implies-ltl-equiv*:
$\varphi \sim \psi \Longrightarrow w \models_n \varphi = w \models_n \psi$
⟨*proof*⟩

lemma *const-implies-eq*:
$\varphi \sim_C \psi \Longrightarrow \varphi \sim \psi$
⟨*proof*⟩

lemma *eq-implies-lang*:
$\varphi \sim \psi \Longrightarrow \varphi \sim_L \psi$
⟨*proof*⟩

lemma *eq-refl*[*simp*]:
$\varphi \sim \varphi$
⟨*proof*⟩

lemma *eq-sym*[*sym*]:
$\varphi \sim \psi \Longrightarrow \psi \sim \varphi$
⟨*proof*⟩

lemma *eq-trans*[*trans*]:
$\varphi \sim \psi \Longrightarrow \psi \sim \chi \Longrightarrow \varphi \sim \chi$
⟨*proof*⟩

end

interpretation *ltl-lang-equivalence*: *ltl-equivalence* $(\sim_L)$
⟨*proof*⟩

interpretation *ltl-prop-equivalence*: *ltl-equivalence* $(\sim_P)$
⟨*proof*⟩

interpretation *ltl-const-equivalence*: *ltl-equivalence* $(\sim_C)$
⟨*proof*⟩

end

4 Disjunctive Normal Form of LTL formulas

theory *Disjunctive-Normal-Form*
imports
LTL Equivalence-Relations HOL−Library.FSet
begin

We use the propositional representation of LTL formulas to define the minimal disjunctive normal form of our formulas. For this purpose we define the minimal product $\otimes_m$ and union $\cup_m$. In the end we show that for a set $\mathcal{A}$ of literals, $\mathcal{A} \models_P \varphi$ if, and only if, there exists a subset of $\mathcal{A}$ in the minimal

DNF of φ.

4.1 Definition of Minimum Sets

definition (**in** *ord*) *min-set* :: *'a set* ⇒ *'a set* **where**
min-set X = {*y* ∈ *X*. ∀ *x* ∈ *X*. *x* ≤ *y* ⟶ *x* = *y*}

lemma *min-set-iff*:
x ∈ *min-set X* ⟷ *x* ∈ *X* ∧ (∀ *y* ∈ *X*. *y* ≤ *x* ⟶ *y* = *x*)
⟨*proof*⟩

lemma *min-set-subset*:
min-set X ⊆ *X*
⟨*proof*⟩

lemma *min-set-idem*[*simp*]:
min-set (*min-set X*) = *min-set X*
⟨*proof*⟩

lemma *min-set-empty*[*simp*]:
min-set {} = {}
⟨*proof*⟩

lemma *min-set-singleton*[*simp*]:
min-set {*x*} = {*x*}
⟨*proof*⟩

lemma *min-set-finite*:
finite X ⟹ *finite* (*min-set X*)
⟨*proof*⟩

lemma *min-set-obtains-helper*:
A ∈ *B* ⟹ ∃ *C*. *C* |⊆| *A* ∧ *C* ∈ *min-set B*
⟨*proof*⟩

lemma *min-set-obtains*:
assumes *A* ∈ *B*
obtains *C* **where** *C* |⊆| *A* **and** *C* ∈ *min-set B*
⟨*proof*⟩

4.2 Minimal operators on sets

definition *product* :: *'a fset set* ⇒ *'a fset set* ⇒ *'a fset set* (**infixr** ‹⊗› *65*)
where *A* ⊗ *B* = {*a* |∪| *b* | *a b*. *a* ∈ *A* ∧ *b* ∈ *B*}

definition *min-product* :: *'a fset set* ⇒ *'a fset set* ⇒ *'a fset set* (**infixr** ‹$\otimes_m$› *65*)
where $A \otimes_m B = min\text{-}set\ (A \otimes B)$

definition *min-union* :: *'a fset set* ⇒ *'a fset set* ⇒ *'a fset set* (**infixr** ‹$\cup_m$› *65*)
where $A \cup_m B = min\text{-}set\ (A \cup B)$

definition *product-set* :: *'a fset set set* ⇒ *'a fset set* (‹$\bigotimes$ ›)
where $\bigotimes\ X = Finite\text{-}Set.fold\ product\ \{\{||\}\}\ X$

definition *min-product-set* :: *'a fset set set* ⇒ *'a fset set* (‹$\bigotimes_m$›)
where $\bigotimes_m\ X = Finite\text{-}Set.fold\ min\text{-}product\ \{\{||\}\}\ X$

lemma *min-product-idem*[*simp*]:
$A \otimes_m A = min\text{-}set\ A$
⟨*proof*⟩

lemma *min-union-idem*[*simp*]:
$A \cup_m A = min\text{-}set\ A$
⟨*proof*⟩

lemma *product-empty*[*simp*]:
$A \otimes \{\} = \{\}$
$\{\} \otimes A = \{\}$
⟨*proof*⟩

lemma *min-product-empty*[*simp*]:
$A \otimes_m \{\} = \{\}$
$\{\} \otimes_m A = \{\}$
⟨*proof*⟩

lemma *min-union-empty*[*simp*]:
$A \cup_m \{\} = min\text{-}set\ A$
$\{\} \cup_m A = min\text{-}set\ A$
⟨*proof*⟩

lemma *product-empty-singleton*[*simp*]:
$A \otimes \{\{||\}\} = A$
$\{\{||\}\} \otimes A = A$
⟨*proof*⟩

lemma *min-product-empty-singleton*[*simp*]:
$A \otimes_m \{\{||\}\} = \textit{min-set}\ A$
$\{\{||\}\} \otimes_m A = \textit{min-set}\ A$
⟨*proof*⟩

lemma *product-singleton-singleton*:
$A \otimes \{\{|x|\}\} = \textit{finsert}\ x\ `\ A$
$\{\{|x|\}\} \otimes A = \textit{finsert}\ x\ `\ A$
⟨*proof*⟩

lemma *product-mono*:
$A \subseteq B \Longrightarrow A \otimes C \subseteq B \otimes C$
$B \subseteq C \Longrightarrow A \otimes B \subseteq A \otimes C$
⟨*proof*⟩

lemma *product-finite*:
$\textit{finite}\ A \Longrightarrow \textit{finite}\ B \Longrightarrow \textit{finite}\ (A \otimes B)$
⟨*proof*⟩

lemma *min-product-finite*:
$\textit{finite}\ A \Longrightarrow \textit{finite}\ B \Longrightarrow \textit{finite}\ (A \otimes_m B)$
⟨*proof*⟩

lemma *min-union-finite*:
$\textit{finite}\ A \Longrightarrow \textit{finite}\ B \Longrightarrow \textit{finite}\ (A \cup_m B)$
⟨*proof*⟩

lemma *product-set-infinite*[*simp*]:
$\textit{infinite}\ X \Longrightarrow \bigotimes\ X = \{\{||\}\}$
⟨*proof*⟩

lemma *min-product-set-infinite*[*simp*]:
$\textit{infinite}\ X \Longrightarrow \bigotimes_m\ X = \{\{||\}\}$
⟨*proof*⟩

lemma *product-comm*:
$A \otimes B = B \otimes A$
⟨*proof*⟩

lemma *min-product-comm*:
$A \otimes_m B = B \otimes_m A$
⟨*proof*⟩

lemma *min-union-comm*:
$A \cup_m B = B \cup_m A$
⟨*proof*⟩

lemma *product-iff*:
$x \in A \otimes B \longleftrightarrow (\exists\, a \in A.\ \exists\, b \in B.\ x = a \mid\cup\mid b)$
⟨*proof*⟩

lemma *min-product-iff*:
$x \in A \otimes_m B \longleftrightarrow (\exists\, a \in A.\ \exists\, b \in B.\ x = a \mid\cup\mid b) \wedge (\forall\, a \in A.\ \forall\, b \in B.\ a \mid\cup\mid b \mid\subseteq\mid x \longrightarrow a \mid\cup\mid b = x)$
⟨*proof*⟩

lemma *min-union-iff*:
$x \in A \cup_m B \longleftrightarrow x \in A \cup B \wedge (\forall\, a \in A.\ a \mid\subseteq\mid x \longrightarrow a = x) \wedge (\forall\, b \in B.\ b \mid\subseteq\mid x \longrightarrow b = x)$
⟨*proof*⟩

lemma *min-set-min-product-helper*:
$x \in (\textit{min-set}\ A) \otimes_m B \longleftrightarrow x \in A \otimes_m B$
⟨*proof*⟩

lemma *min-set-min-product*[*simp*]:
$(\textit{min-set}\ A) \otimes_m B = A \otimes_m B$
$A \otimes_m (\textit{min-set}\ B) = A \otimes_m B$
⟨*proof*⟩

lemma *min-set-min-union*[*simp*]:
$(\textit{min-set}\ A) \cup_m B = A \cup_m B$
$A \cup_m (\textit{min-set}\ B) = A \cup_m B$
⟨*proof*⟩

lemma *product-assoc*[*simp*]:
$(A \otimes B) \otimes C = A \otimes (B \otimes C)$
⟨*proof*⟩

lemma *min-product-assoc*[*simp*]:

$(A \otimes_m B) \otimes_m C = A \otimes_m (B \otimes_m C)$
⟨*proof*⟩

lemma *min-union-assoc*[*simp*]:
$(A \cup_m B) \cup_m C = A \cup_m (B \cup_m C)$
⟨*proof*⟩

lemma *min-product-comp*:
$a \in A \Longrightarrow b \in B \Longrightarrow \exists\, c.\ c \mid\subseteq\mid (a \mid\cup\mid b) \wedge c \in A \otimes_m B$
⟨*proof*⟩

lemma *min-union-comp*:
$a \in A \Longrightarrow \exists\, c.\ c \mid\subseteq\mid a \wedge c \in A \cup_m B$
⟨*proof*⟩

interpretation *product-set-thms*: *Finite-Set.comp-fun-commute product*
⟨*proof*⟩

interpretation *min-product-set-thms*: *Finite-Set.comp-fun-idem min-product*
⟨*proof*⟩

interpretation *min-union-set-thms*: *Finite-Set.comp-fun-idem min-union*
⟨*proof*⟩

lemma *product-set-empty*[*simp*]:
$\bigotimes\ \{\} = \{\{||\}\}$
$\bigotimes\ \{\{\}\} = \{\}$
$\bigotimes\ \{\{\{||\}\}\} = \{\{||\}\}$
⟨*proof*⟩

lemma *min-product-set-empty*[*simp*]:
$\bigotimes_m\ \{\} = \{\{||\}\}$
$\bigotimes_m\ \{\{\}\} = \{\}$
$\bigotimes_m\ \{\{\{||\}\}\} = \{\{||\}\}$
⟨*proof*⟩

lemma *product-set-code*[*code*]:
$\bigotimes\ (\mathit{set}\ \mathit{xs}) = \mathit{fold}\ \mathit{product}\ (\mathit{remdups}\ \mathit{xs})\ \{\{||\}\}$
⟨*proof*⟩

lemma *min-product-set-code*[*code*]:
$\bigotimes_m$ (*set xs*) = *fold min-product* (*remdups xs*) {{||}}
⟨*proof*⟩

lemma *product-set-insert*[*simp*]:
finite X ⟹ $\bigotimes$ (*insert x X*) = $x \otimes (\bigotimes\ (X - \{x\}))$
⟨*proof*⟩

lemma *min-product-set-insert*[*simp*]:
finite X ⟹ $\bigotimes_m$ (*insert x X*) = $x \otimes_m (\bigotimes_m\ X)$
⟨*proof*⟩

lemma *min-product-subseteq*:
$x \in A \otimes_m B \Longrightarrow \exists\, a.\ a \mid\subseteq\mid x \wedge a \in A$
⟨*proof*⟩

lemma *min-product-set-subseteq*:
finite X ⟹ $x \in \bigotimes_m X \Longrightarrow A \in X \Longrightarrow \exists\, a \in A.\ a \mid\subseteq\mid x$
⟨*proof*⟩

lemma *min-set-product-set*:
$\bigotimes_m A$ = *min-set* ($\bigotimes\ A$)
⟨*proof*⟩

lemma *min-product-min-set*[*simp*]:
min-set ($A \otimes_m B$) = $A \otimes_m B$
⟨*proof*⟩

lemma *min-union-min-set*[*simp*]:
min-set ($A \cup_m B$) = $A \cup_m B$
⟨*proof*⟩

lemma *min-product-set-min-set*[*simp*]:
finite X ⟹ *min-set* ($\bigotimes_m X$) = $\bigotimes_m X$
⟨*proof*⟩

lemma *min-set-min-product-set*[*simp*]:
finite X ⟹ $\bigotimes_m$ (*min-set* ‘ *X*) = $\bigotimes_m X$
⟨*proof*⟩

lemma *min-product-set-union*[*simp*]:
finite X ⟹ *finite Y* ⟹ $\bigotimes_m (X \cup Y) = (\bigotimes_m X) \otimes_m (\bigotimes_m Y)$
⟨*proof*⟩

lemma *product-set-finite*:
$(\bigwedge x.\ x \in X \Longrightarrow \mathit{finite}\ x) \Longrightarrow \mathit{finite}\ (\bigotimes\ X)$
⟨*proof*⟩

lemma *min-product-set-finite*:
$(\bigwedge x.\ x \in X \Longrightarrow \mathit{finite}\ x) \Longrightarrow \mathit{finite}\ (\bigotimes_m\ X)$
⟨*proof*⟩

4.3 Disjunctive Normal Form

fun *dnf* :: *'a ltln* ⇒ *'a ltln fset set*
where
dnf true$_n$ = {{||}}
| *dnf false*$_n$ = {}
| *dnf* (φ *and*$_n$ ψ) = (*dnf* φ) ⊗ (*dnf* ψ)
| *dnf* (φ *or*$_n$ ψ) = (*dnf* φ) ∪ (*dnf* ψ)
| *dnf* φ = {{|φ|}}

fun *min-dnf* :: *'a ltln* ⇒ *'a ltln fset set*
where
min-dnf true$_n$ = {{||}}
| *min-dnf false*$_n$ = {}
| *min-dnf* (φ *and*$_n$ ψ) = (*min-dnf* φ) ⊗$_m$ (*min-dnf* ψ)
| *min-dnf* (φ *or*$_n$ ψ) = (*min-dnf* φ) ∪$_m$ (*min-dnf* ψ)
| *min-dnf* φ = {{|φ|}}

lemma *dnf-min-set*:
min-dnf φ = *min-set* (*dnf* φ)
⟨*proof*⟩

lemma *dnf-finite*:
finite (*dnf* φ)
⟨*proof*⟩

lemma *min-dnf-finite*:
finite (*min-dnf* φ)
⟨*proof*⟩

lemma *dnf-Abs-fset*[*simp*]:
fset (*Abs-fset* (*dnf* φ)) = *dnf* φ
⟨*proof*⟩

lemma *min-dnf-Abs-fset*[*simp*]:
fset (*Abs-fset* (*min-dnf* φ)) = *min-dnf* φ
⟨*proof*⟩

lemma *dnf-prop-atoms*:
$\Phi \in$ *dnf* $\varphi \Longrightarrow$ *fset* $\Phi \subseteq$ *prop-atoms* φ
⟨*proof*⟩

lemma *min-dnf-prop-atoms*:
$\Phi \in$ *min-dnf* $\varphi \Longrightarrow$ *fset* $\Phi \subseteq$ *prop-atoms* φ
⟨*proof*⟩

lemma *min-dnf-atoms-dnf*:
$\Phi \in$ *min-dnf* $\psi \Longrightarrow \varphi \in$ *fset* $\Phi \Longrightarrow$ *dnf* $\varphi = \{\{|\varphi|\}\}$
⟨*proof*⟩

lemma *min-dnf-min-set*[*simp*]:
min-set (*min-dnf* φ) = *min-dnf* φ
⟨*proof*⟩

lemma *min-dnf-iff-prop-assignment-subset*:
$\mathcal{A} \models_P \varphi \longleftrightarrow (\exists\, B.$ *fset* $B \subseteq \mathcal{A} \wedge B \in$ *min-dnf* $\varphi)$
⟨*proof*⟩

lemma *ltl-prop-implies-min-dnf*:
$\varphi \longrightarrow_P \psi = (\forall\, A \in$ *min-dnf* $\varphi.\ \exists\, B \in$ *min-dnf* $\psi.\ B \mid\subseteq\mid A)$
⟨*proof*⟩

lemma *ltl-prop-equiv-min-dnf*:
$\varphi \sim_P \psi =$ (*min-dnf* $\varphi =$ *min-dnf* ψ)
⟨*proof*⟩

lemma *min-dnf-rep-abs*[*simp*]:
min-dnf (*rep-ltln*$_P$ (*abs-ltln*$_P$ φ)) = *min-dnf* φ
⟨*proof*⟩

4.4 Folding of and_n and or_n over Finite Sets

definition And_n :: *'a ltln set* ⇒ *'a ltln*
where
$And_n\ \Phi \equiv$ *SOME* φ. *fold-graph And-ltln True-ltln* $\Phi\ \varphi$

definition $Or_n :: 'a\ ltln\ set \Rightarrow 'a\ ltln$
where
$Or_n\ \Phi \equiv SOME\ \varphi.\ fold\text{-}graph\ Or\text{-}ltln\ False\text{-}ltln\ \Phi\ \varphi$

lemma *fold-graph-*And_n:
finite $\Phi \Longrightarrow$ *fold-graph And-ltln True-ltln* Φ $(And_n\ \Phi)$
⟨*proof*⟩

lemma *fold-graph-*Or_n:
finite $\Phi \Longrightarrow$ *fold-graph Or-ltln False-ltln* Φ $(Or_n\ \Phi)$
⟨*proof*⟩

lemma Or_n*-empty*[*simp*]:
$Or_n\ \{\} = False\text{-}ltln$
⟨*proof*⟩

lemma And_n*-empty*[*simp*]:
$And_n\ \{\} = True\text{-}ltln$
⟨*proof*⟩

interpretation *dnf-union-thms*: *Finite-Set.comp-fun-commute* $\lambda\varphi.\ (\cup)\ (f\ \varphi)$
⟨*proof*⟩

interpretation *dnf-product-thms*: *Finite-Set.comp-fun-commute* $\lambda\varphi.\ (\otimes)\ (f\ \varphi)$
⟨*proof*⟩
lemma *fold-graph-finite*:
assumes *fold-graph f z A y*
shows *finite A*
⟨*proof*⟩

Taking the DNF of And_n and Or_n is the same as folding over the individual DNFs.

lemma And_n*-dnf*:
finite $\Phi \Longrightarrow$ *dnf* $(And_n\ \Phi) =$ *Finite-Set.fold* $(\lambda\varphi.\ (\otimes)\ (dnf\ \varphi))\ \{\{||\}\}\ \Phi$
⟨*proof*⟩

lemma Or_n*-dnf*:
finite $\Phi \Longrightarrow$ *dnf* $(Or_n\ \Phi) =$ *Finite-Set.fold* $(\lambda\varphi.\ (\cup)\ (dnf\ \varphi))\ \{\}\ \Phi$
⟨*proof*⟩

And_n and Or_n are injective on finite sets.

lemma And_n*-inj*:

inj-on And_n $\{s.$ *finite* $s\}$
⟨*proof*⟩

lemma Or_n*-inj*:
inj-on Or_n $\{s.$ *finite* $s\}$
⟨*proof*⟩

The semantics of And_n and Or_n can be expressed using quantifiers.

lemma And_n*-semantics*:
finite $\Phi \Longrightarrow w \models_n And_n\ \Phi \longleftrightarrow (\forall \varphi \in \Phi.\ w \models_n \varphi)$
⟨*proof*⟩

lemma Or_n*-semantics*:
finite $\Phi \Longrightarrow w \models_n Or_n\ \Phi \longleftrightarrow (\exists \varphi \in \Phi.\ w \models_n \varphi)$
⟨*proof*⟩

lemma And_n*-prop-semantics*:
finite $\Phi \Longrightarrow \mathcal{A} \models_P And_n\ \Phi \longleftrightarrow (\forall \varphi \in \Phi.\ \mathcal{A} \models_P \varphi)$
⟨*proof*⟩

lemma Or_n*-prop-semantics*:
finite $\Phi \Longrightarrow \mathcal{A} \models_P Or_n\ \Phi \longleftrightarrow (\exists \varphi \in \Phi.\ \mathcal{A} \models_P \varphi)$
⟨*proof*⟩

lemma Or_n*-*And_n*-image-semantics*:
assumes *finite* $\mathcal{A}$ **and** $\bigwedge \Phi.\ \Phi \in \mathcal{A} \Longrightarrow$ *finite* Φ
shows $w \models_n Or_n\ (And_n\ `\ \mathcal{A}) \longleftrightarrow (\exists \Phi \in \mathcal{A}.\ \forall \varphi \in \Phi.\ w \models_n \varphi)$
⟨*proof*⟩

lemma Or_n*-*And_n*-image-prop-semantics*:
assumes *finite* $\mathcal{A}$ **and** $\bigwedge \Phi.\ \Phi \in \mathcal{A} \Longrightarrow$ *finite* Φ
shows $\mathcal{I} \models_P Or_n\ (And_n\ `\ \mathcal{A}) \longleftrightarrow (\exists \Phi \in \mathcal{A}.\ \forall \varphi \in \Phi.\ \mathcal{I} \models_P \varphi)$
⟨*proof*⟩

4.5 DNF to LTL conversion

definition *ltln-of-dnf* :: *'a ltln fset set* ⇒ *'a ltln*
where
ltln-of-dnf $\mathcal{A} =$ $Or_n\ (And_n\ `$ *fset* $`\ \mathcal{A})$

lemma *ltln-of-dnf-semantics*:
assumes *finite* $\mathcal{A}$
shows $w \models_n$ *ltln-of-dnf* $\mathcal{A} \longleftrightarrow (\exists \Phi \in \mathcal{A}.\ \forall \varphi.\ \varphi\ |\in|\ \Phi \longrightarrow w \models_n \varphi)$
⟨*proof*⟩

lemma *ltln-of-dnf-prop-semantics*:
assumes *finite* $\mathcal{A}$
shows $\mathcal{I} \models_P$ *ltln-of-dnf* $\mathcal{A} \longleftrightarrow (\exists\, \Phi \in \mathcal{A}.\ \forall\, \varphi.\ \varphi \mid\in\mid \Phi \longrightarrow \mathcal{I} \models_P \varphi)$
⟨*proof*⟩

lemma *ltln-of-dnf-prop-equiv*:
ltln-of-dnf (*min-dnf* φ) $\sim_P \varphi$
⟨*proof*⟩

lemma *min-dnf-ltln-of-dnf*[*simp*]:
min-dnf (*ltln-of-dnf* (*min-dnf* φ)) = *min-dnf* φ
⟨*proof*⟩

4.6 Substitution in DNF formulas

definition *subst-clause* :: *'a ltln fset* ⇒ (*'a ltln* ⇀ *'a ltln*) ⇒ *'a ltln fset set*
where
subst-clause Φ m = $\bigotimes_m$ {*min-dnf* (*subst* φ m) | φ. $\varphi \in$ *fset* Φ}

definition *subst-dnf* :: *'a ltln fset set* ⇒ (*'a ltln* ⇀ *'a ltln*) ⇒ *'a ltln fset set*
where
subst-dnf $\mathcal{A}$ m = $(\bigcup \Phi \in \mathcal{A}.$ *subst-clause* Φ $m)$

lemma *subst-clause-empty*[*simp*]:
subst-clause {||} m = {{||}}
⟨*proof*⟩

lemma *subst-dnf-empty*[*simp*]:
subst-dnf {} m = {}
⟨*proof*⟩

lemma *subst-clause-inner-finite*:
finite {*min-dnf* (*subst* φ m) | φ. $\varphi \in \Phi$} **if** *finite* Φ
⟨*proof*⟩

lemma *subst-clause-finite*:
finite (*subst-clause* Φ m)
⟨*proof*⟩

lemma *subst-dnf-finite*:
finite $\mathcal{A}$ ⟹ *finite* (*subst-dnf* $\mathcal{A}$ m)
⟨*proof*⟩

lemma *subst-dnf-mono*:
$\mathcal{A} \subseteq \mathcal{B} \Longrightarrow$ *subst-dnf* $\mathcal{A}$ $m \subseteq$ *subst-dnf* $\mathcal{B}$ m
⟨*proof*⟩

lemma *subst-clause-min-set*[*simp*]:
min-set (*subst-clause* Φ m) = *subst-clause* Φ m
⟨*proof*⟩

lemma *subst-clause-finsert*[*simp*]:
subst-clause (*finsert* φ Φ) m = (*min-dnf* (*subst* φ m)) $\otimes_m$ (*subst-clause* Φ m)
⟨*proof*⟩

lemma *subst-clause-funion*[*simp*]:
subst-clause (Φ |∪| Ψ) m = (*subst-clause* Φ m) $\otimes_m$ (*subst-clause* Ψ m)
⟨*proof*⟩

For the proof of correctness, we redefine the ($\otimes$) operator on lists.

definition *list-product* :: *'a list set* ⇒ *'a list set* ⇒ *'a list set* (**infixl** ‹$\otimes_l$› *65*)
where
$A \otimes_l B = \{a \mathbin{@} b \mid a\ b.\ a \in A \wedge b \in B\}$

lemma *list-product-fset-of-list*[*simp*]:
fset-of-list ‘ ($A \otimes_l B$) = (*fset-of-list* ‘ A) $\otimes$ (*fset-of-list* ‘ B)
⟨*proof*⟩

lemma *list-product-finite*:
finite $A \Longrightarrow$ *finite* $B \Longrightarrow$ *finite* ($A \otimes_l B$)
⟨*proof*⟩

lemma *list-product-iff*:
$x \in A \otimes_l B \longleftrightarrow (\exists\, a\ b.\ a \in A \wedge b \in B \wedge x = a \mathbin{@} b)$
⟨*proof*⟩

lemma *list-product-assoc*[*simp*]:
$A \otimes_l (B \otimes_l C) = A \otimes_l B \otimes_l C$
⟨*proof*⟩

Furthermore, we introduct DNFs where the clauses are represented as lists.

fun *list-dnf* :: *'a ltln* ⇒ *'a ltln list set*
where
list-dnf $true_n = \{[]\}$
| *list-dnf* $false_n = \{\}$

| *list-dnf* $(\varphi\ and_n\ \psi) = (list\text{-}dnf\ \varphi) \otimes_l (list\text{-}dnf\ \psi)$
| *list-dnf* $(\varphi\ or_n\ \psi) = (list\text{-}dnf\ \varphi) \cup (list\text{-}dnf\ \psi)$
| *list-dnf* $\varphi = \{[\varphi]\}$

definition *list-dnf-to-dnf* :: $'a\ list\ set \Rightarrow 'a\ fset\ set$
where
list-dnf-to-dnf $X = fset\text{-}of\text{-}list\ `\ X$

lemma *list-dnf-to-dnf-list-dnf*[*simp*]:
list-dnf-to-dnf $(list\text{-}dnf\ \varphi) = dnf\ \varphi$
⟨*proof*⟩

lemma *list-dnf-finite*:
finite $(list\text{-}dnf\ \varphi)$
⟨*proof*⟩

We use this to redefine *subst-clause* and *subst-dnf* on list DNFs.

definition *subst-clause'* :: $'a\ ltln\ list \Rightarrow ('a\ ltln \rightharpoonup 'a\ ltln) \Rightarrow 'a\ ltln\ list\ set$
where
subst-clause' $\Phi\ m = fold\ (\lambda\varphi\ acc.\ acc \otimes_l list\text{-}dnf\ (subst\ \varphi\ m))\ \Phi\ \{[]\}$

definition *subst-dnf'* :: $'a\ ltln\ list\ set \Rightarrow ('a\ ltln \rightharpoonup 'a\ ltln) \Rightarrow 'a\ ltln\ list\ set$
where
subst-dnf' $\mathcal{A}\ m = (\bigcup \Phi \in \mathcal{A}.\ subst\text{-}clause'\ \Phi\ m)$

lemma *subst-clause'-finite*:
finite $(subst\text{-}clause'\ \Phi\ m)$
⟨*proof*⟩

lemma *subst-clause'-nil*[*simp*]:
subst-clause' $[]\ m = \{[]\}$
⟨*proof*⟩

lemma *subst-clause'-cons*[*simp*]:
subst-clause' $(xs\ @\ [x])\ m = subst\text{-}clause'\ xs\ m \otimes_l list\text{-}dnf\ (subst\ x\ m)$
⟨*proof*⟩

lemma *subst-clause'-append*[*simp*]:
subst-clause' $(A\ @\ B)\ m = subst\text{-}clause'\ A\ m \otimes_l subst\text{-}clause'\ B\ m$
⟨*proof*⟩

lemma *subst-dnf'-iff*:
$x \in subst\text{-}dnf'\ A\ m \longleftrightarrow (\exists \Phi \in A.\ x \in subst\text{-}clause'\ \Phi\ m)$

⟨*proof*⟩

lemma *subst-dnf'-product*:
subst-dnf' (*A* ⊗$_l$ *B*) *m* = (*subst-dnf'* *A* *m*) ⊗$_l$ (*subst-dnf'* *B* *m*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *subst-dnf'-list-dnf*:
subst-dnf' (*list-dnf* φ) *m* = *list-dnf* (*subst* φ *m*)
⟨*proof*⟩

lemma *min-set-Union*:
finite X ⟹ *min-set* (⋃ (*min-set* ' *X*)) = *min-set* (⋃ *X*) **for** *X* :: *'a fset set set*
⟨*proof*⟩

lemma *min-set-Union-image*:
finite X ⟹ *min-set* (⋃*x* ∈ *X*. *min-set* (*f x*)) = *min-set* (⋃*x* ∈ *X*. *f x*) **for** *f* :: *'b* ⇒ *'a fset set*
⟨*proof*⟩

lemma *subst-clause-fset-of-list*:
subst-clause (*fset-of-list* Φ) *m* = *min-set* (*list-dnf-to-dnf* (*subst-clause'* Φ *m*))
⟨*proof*⟩

lemma *min-set-list-dnf-to-dnf-subst-dnf'*:
finite X ⟹ *min-set* (*list-dnf-to-dnf* (*subst-dnf'* *X* *m*)) = *min-set* (*subst-dnf* (*list-dnf-to-dnf* *X*) *m*)
⟨*proof*⟩

lemma *subst-dnf-dnf*:
min-set (*subst-dnf* (*dnf* φ) *m*) = *min-dnf* (*subst* φ *m*)
⟨*proof*⟩

This is almost the lemma we need. However, we need to show that the same holds for *min-dnf* φ, too.

lemma *fold-product*:
Finite-Set.fold (λ*x*. (⊗) {{|*x*|}}) {{||}} (*fset x*) = {*x*}
⟨*proof*⟩

lemma *fold-union*:
Finite-Set.fold (λ*x*. (∪) {*x*}) {} (*fset x*) = *fset x*

⟨*proof*⟩

lemma *fold-union-fold-product*:
assumes *finite X* **and** $\bigwedge \Psi\ \psi.\ \Psi \in X \Longrightarrow \psi \in fset\ \Psi \Longrightarrow dnf\ \psi = \{\{|\psi|\}\}$
shows *Finite-Set.fold* $(\lambda x.\ (\cup)\ (Finite\text{-}Set.fold\ (\lambda\varphi.\ (\otimes)\ (dnf\ \varphi))\ \{\{||\}\}\ (fset\ x)))\ \{\}\ X = X$ (**is** *?lhs* = *X*)
⟨*proof*⟩

lemma *dnf-ltln-of-dnf-min-dnf*:
dnf (*ltln-of-dnf* (*min-dnf* φ)) = *min-dnf* φ
⟨*proof*⟩

lemma *min-dnf-subst*:
min-set (*subst-dnf* (*min-dnf* φ) *m*) = *min-dnf* (*subst* φ *m*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

end

5 Code lemmas for abstract definitions

theory *Code-Equations*
imports
LTL Equivalence-Relations
Boolean-Expression-Checkers.Boolean-Expression-Checkers
Boolean-Expression-Checkers.Boolean-Expression-Checkers-AList-Mapping
begin

5.1 Propositional Equivalence

fun *ifex-of-ltl* :: *'a ltln* ⇒ *'a ltln ifex*
where
ifex-of-ltl $true_n$ = *Trueif*
| *ifex-of-ltl* $false_n$ = *Falseif*
| *ifex-of-ltl* (φ and_n ψ) = *normif Mapping.empty* (*ifex-of-ltl* φ) (*ifex-of-ltl* ψ) *Falseif*
| *ifex-of-ltl* (φ or_n ψ) = *normif Mapping.empty* (*ifex-of-ltl* φ) *Trueif* (*ifex-of-ltl* ψ)
| *ifex-of-ltl* φ = *IF* φ *Trueif Falseif*

lemma *val-ifex*:
val-ifex (*ifex-of-ltl b*) *s* = $\{x.\ s\ x\} \models_P b$
⟨*proof*⟩

lemma *reduced-ifex*:
reduced (*ifex-of-ltl b*) {}
⟨*proof*⟩

lemma *ifex-of-ltl-reduced-bdt-checker*:
reduced-bdt-checkers ifex-of-ltl (λ*y s*. {*x*. *s x*} ⊨$_P$ *y*)
⟨*proof*⟩

lemma *ltl-prop-equiv-impl*[*code*]:
(*φ* ∼$_P$ *ψ*) = *equiv-test ifex-of-ltl φ ψ*
⟨*proof*⟩

lemma *ltl-prop-implies-impl*[*code*]:
(*φ* ⟶$_P$ *ψ*) = *impl-test ifex-of-ltl φ ψ*
⟨*proof*⟩
export-code (∼$_P$) (⟶$_P$) **checking**

end

6 Example

theory *Example*
imports
../LTL ../Rewriting HOL−Library.Code-Target-Numeral
begin

— The included parser always returns a *String.literal ltlc*. If a different labelling is needed one can use *map-ltlc* to relabel the leafs. In our example we prepend a string to each atom.

definition *rewrite* :: *String.literal ltlc* ⇒ *String.literal ltlc*
where
rewrite ≡ *ltln-to-ltlc o rewrite-iter-slow o ltlc-to-ltln o* (*map-ltlc* (λ*s*. *String.implode* ″*prop*(″ + *s* + *String.implode* ″)″))

— Export rewriting engine (and also constructors)

export-code *true$_c$ Iff-ltlc rewrite* **in** *SML* **file-prefix** ‹*rewrite-example*›

end

References

[1] T. Babiak, M. Kretínský, V. Rehák, and J. Strejcek. LTL to büchi automata translation: Fast and more deterministic. In C. Flanagan and B. König, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7214 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2012.

[2] F. Somenzi and R. Bloem. Efficient büchi automata from LTL formulae. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2000.