

Linear Temporal Logic

Salomon Sickert

June 17, 2024

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* =

<i>True-ltlc</i>	(<i>true_c</i>)
<i>False-ltlc</i>	(<i>false_c</i>)
<i>Prop-ltlc</i> 'a	(<i>prop_c</i> '(-)')
<i>Not-ltlc</i> 'a <i>ltlc</i>	(<i>not_c</i> - [85] 85)
<i>And-ltlc</i> 'a <i>ltlc</i> 'a <i>ltlc</i>	(- <i>and_c</i> - [82,82] 81)
<i>Or-ltlc</i> 'a <i>ltlc</i> 'a <i>ltlc</i>	(- <i>or_c</i> - [81,81] 80)
<i>Implies-ltlc</i> 'a <i>ltlc</i> 'a <i>ltlc</i>	(- <i>implies_c</i> - [81,81] 80)
<i>Next-ltlc</i> 'a <i>ltlc</i>	(<i>X_c</i> - [88] 87)
<i>Final-ltlc</i> 'a <i>ltlc</i>	(<i>F_c</i> - [88] 87)
<i>Global-ltlc</i> 'a <i>ltlc</i>	(<i>G_c</i> - [88] 87)
<i>Until-ltlc</i> 'a <i>ltlc</i> 'a <i>ltlc</i>	(- <i>U_c</i> - [84,84] 83)
<i>Release-ltlc</i> 'a <i>ltlc</i> 'a <i>ltlc</i>	(- <i>R_c</i> - [84,84] 83)
<i>WeakUntil-ltlc</i> 'a <i>ltlc</i> 'a <i>ltlc</i>	(- <i>W_c</i> - [84,84] 83)
<i>StrongRelease-ltlc</i> 'a <i>ltlc</i> 'a <i>ltlc</i>	(- <i>M_c</i> - [84,84] 83)

definition *Iff-ltlc* (- *iff_c* - [81,81] 80)

where

$\varphi \text{ iff}_c \psi \equiv (\varphi \text{ implies}_c \psi) \text{ and}_c (\psi \text{ implies}_c \varphi)$

1.1.2 Semantics

primrec *semantics-ltlc* :: [*'a set word, 'a ltlc*] \Rightarrow *bool* ($- \models_c - [80,80] 80$)

where

$\xi \models_c \text{true}_c = \text{True}$
 $\xi \models_c \text{false}_c = \text{False}$
 $\xi \models_c \text{prop}_c(q) = (q \in \xi \ 0)$
 $\xi \models_c \text{not}_c \varphi = (\neg \xi \models_c \varphi)$
 $\xi \models_c \varphi \text{ and}_c \psi = (\xi \models_c \varphi \wedge \xi \models_c \psi)$
 $\xi \models_c \varphi \text{ or}_c \psi = (\xi \models_c \varphi \vee \xi \models_c \psi)$
 $\xi \models_c \varphi \text{ implies}_c \psi = (\xi \models_c \varphi \longrightarrow \xi \models_c \psi)$
 $\xi \models_c X_c \varphi = (\text{suffix } 1 \ \xi \models_c \varphi)$
 $\xi \models_c F_c \varphi = (\exists i. \text{suffix } i \ \xi \models_c \varphi)$
 $\xi \models_c G_c \varphi = (\forall i. \text{suffix } i \ \xi \models_c \varphi)$
 $\xi \models_c \varphi U_c \psi = (\exists i. \text{suffix } i \ \xi \models_c \psi \wedge (\forall j < i. \text{suffix } j \ \xi \models_c \varphi))$
 $\xi \models_c \varphi R_c \psi = (\forall i. \text{suffix } i \ \xi \models_c \psi \vee (\exists j < i. \text{suffix } j \ \xi \models_c \varphi))$
 $\xi \models_c \varphi W_c \psi = (\forall i. \text{suffix } i \ \xi \models_c \varphi \vee (\exists j \leq i. \text{suffix } j \ \xi \models_c \psi))$
 $\xi \models_c \varphi M_c \psi = (\exists i. \text{suffix } i \ \xi \models_c \varphi \wedge (\forall j \leq i. \text{suffix } j \ \xi \models_c \psi))$

lemma *semantics-ltlc-sugar* [*simp*]:

$\xi \models_c \varphi \text{ iff}_c \psi = (\xi \models_c \varphi \longleftrightarrow \xi \models_c \psi)$
 $\xi \models_c F_c \varphi = \xi \models_c (\text{true}_c U_c \varphi)$
 $\xi \models_c G_c \varphi = \xi \models_c (\text{false}_c R_c \varphi)$
<proof>

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

lemma *language-ltlc-negate* [*simp*]:

$\text{language-ltlc } (\text{not}_c \varphi) = - \text{language-ltlc } \varphi$
<proof>

lemma *ltl-true-or-con* [*simp*]:

$\xi \models_c \text{prop}_c(p) \text{ or}_c (\text{not}_c \text{prop}_c(p))$
<proof>

lemma *ltl-false-true-con* [*simp*]:

$\xi \models_c \text{not}_c \text{true}_c \longleftrightarrow \xi \models_c \text{false}_c$
<proof>

lemma *ltl-Next-Neg-con* [*simp*]:

$\xi \models_c X_c (\text{not}_c \varphi) \longleftrightarrow \xi \models_c \text{not}_c X_c \varphi$
<proof>

lemma *ltl-Until-Release-con*:

$\xi \models_c \varphi R_c \psi \iff (\neg \xi \models_c (\text{not}_c \varphi) U_c (\text{not}_c \psi))$
 $\xi \models_c \varphi U_c \psi \iff (\neg \xi \models_c (\text{not}_c \varphi) R_c (\text{not}_c \psi))$
 ⟨proof⟩

lemma *ltl-WeakUntil-StrongRelease-con:*

$\xi \models_c \varphi W_c \psi \iff (\neg \xi \models_c (\text{not}_c \varphi) M_c (\text{not}_c \psi))$
 $\xi \models_c \varphi M_c \psi \iff (\neg \xi \models_c (\text{not}_c \varphi) W_c (\text{not}_c \psi))$
 ⟨proof⟩

lemma *ltl-Release-StrongRelease-con:*

$\xi \models_c \varphi R_c \psi \iff \xi \models_c (\varphi M_c \psi) \text{ or}_c (G_c \psi)$
 $\xi \models_c \varphi M_c \psi \iff \xi \models_c (\varphi R_c \psi) \text{ and}_c (F_c \varphi)$
 ⟨proof⟩

lemma *ltl-Until-WeakUntil-con:*

$\xi \models_c \varphi U_c \psi \iff \xi \models_c (\varphi W_c \psi) \text{ and}_c (F_c \psi)$
 $\xi \models_c \varphi W_c \psi \iff \xi \models_c (\varphi U_c \psi) \text{ or}_c (G_c \varphi)$
 ⟨proof⟩

lemma *ltl-StrongRelease-Until-con:*

$\xi \models_c \varphi M_c \psi \iff \xi \models_c \psi U_c (\varphi \text{ and}_c \psi)$
 ⟨proof⟩

lemma *ltl-WeakUntil-Release-con:*

$\xi \models_c \varphi R_c \psi \iff \xi \models_c \psi W_c (\varphi \text{ and}_c \psi)$
 ⟨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]:* $pw\text{-eq-on } S w w$

and *pw-eq-on-sym:* $pw\text{-eq-on } S w w' \implies pw\text{-eq-on } S w' w$

and *pw-eq-on-trans[trans]:* $\llbracket pw\text{-eq-on } S w w'; pw\text{-eq-on } S w' w'' \rrbracket \implies pw\text{-eq-on } S w w''$

⟨proof⟩

lemma *pw-eq-on-suffix:*

$pw\text{-eq-on } S w w' \implies pw\text{-eq-on } S (\text{suffix } k w) (\text{suffix } k w')$

⟨proof⟩

lemma *pw-eq-on-subset:*

$S \subseteq S' \implies pw\text{-eq-on } S' w w' \implies pw\text{-eq-on } S w w'$

⟨proof⟩

lemma *ltlc-eq-on-aux*:

pw-eq-on (atoms-ltlc φ) w w' $\implies w \models_c \varphi \implies w' \models_c \varphi$
 \langle *proof* \rangle

lemma *ltlc-eq-on*:

pw-eq-on (atoms-ltlc φ) w w' $\implies w \models_c \varphi \longleftrightarrow w' \models_c \varphi$
 \langle *proof* \rangle

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

\langle *proof* \rangle

lemma *suffix-range*: $\bigcup (\text{range } \xi) \subseteq APs \implies \bigcup (\text{range } (\text{suffix } k \ \xi)) \subseteq APs$

\langle *proof* \rangle

lemma *map-semantic-ltlc-aux*:

assumes *inj-on f APs*

assumes $\bigcup (\text{range } w) \subseteq APs$

assumes *atoms-ltlc $\varphi \subseteq APs$*

shows $w \models_c \varphi \longleftrightarrow (\lambda i. f \ ' \ w \ i) \models_c \text{map-ltlc } f \ \varphi$

\langle *proof* \rangle

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

lemma *map-semantic-ltlc*:

assumes *INJ: inj-on f (dom f) and DOM: atoms-ltlc $\varphi \subseteq \text{dom } f$*

shows $\xi \models_c \varphi \longleftrightarrow (\text{map-props } f \ o \ \xi) \models_c \text{map-ltlc } (\text{the } o \ f) \ \varphi$

\langle *proof* \rangle

lemma *map-semantic-ltlc-inv*:

assumes *INJ: inj-on f (dom f) and DOM: atoms-ltlc $\varphi \subseteq \text{dom } f$*

shows $\xi \models_c \text{map-ltlc } (\text{the } o \ f) \ \varphi \longleftrightarrow (\lambda i. (\text{the } o \ f) \ - \ ' \ \xi \ i) \models_c \varphi$

\langle *proof* \rangle

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* =

<i>True-ltln</i>	(true_n)
<i>False-ltln</i>	(false_n)
<i>Prop-ltln 'a</i>	$(\text{prop}_n \ '(-))$
<i>Nprop-ltln 'a</i>	$(\text{nprop}_n \ '(-))$

<i>And-ltl</i> 'a ltl 'a ltl	(- and_n - [82,82] 81)
<i>Or-ltl</i> 'a ltl 'a ltl	(- or_n - [84,84] 83)
<i>Next-ltl</i> 'a ltl	(X_n - [88] 87)
<i>Until-ltl</i> 'a ltl 'a ltl	(- U_n - [84,84] 83)
<i>Release-ltl</i> 'a ltl 'a ltl	(- R_n - [84,84] 83)
<i>WeakUntil-ltl</i> 'a ltl 'a ltl	(- W_n - [84,84] 83)
<i>StrongRelease-ltl</i> 'a ltl 'a ltl	(- M_n - [84,84] 83)

abbreviation $finally_n :: 'a ltl \Rightarrow 'a ltl$ (F_n - [88] 87)

where

$$F_n \varphi \equiv true_n U_n \varphi$$

notation (*input*) $finally_n$ (\diamond_n - [88] 87)

abbreviation $globally_n :: 'a ltl \Rightarrow 'a ltl$ (G_n - [88] 87)

where

$$G_n \varphi \equiv false_n R_n \varphi$$

notation (*input*) $globally_n$ (\square_n - [88] 87)

1.2.2 Semantics

primrec $semantics-ltl :: ['a \text{ set word}, 'a ltl] \Rightarrow bool$ (- \models_n - [80,80] 80)

where

$$\begin{aligned} & \xi \models_n true_n = True \\ & | \xi \models_n false_n = False \\ & | \xi \models_n prop_n(q) = (q \in \xi \ 0) \\ & | \xi \models_n nprop_n(q) = (q \notin \xi \ 0) \\ & | \xi \models_n \varphi \ and_n \psi = (\xi \models_n \varphi \wedge \xi \models_n \psi) \\ & | \xi \models_n \varphi \ or_n \psi = (\xi \models_n \varphi \vee \xi \models_n \psi) \\ & | \xi \models_n X_n \varphi = (suffix \ 1 \ \xi \models_n \varphi) \\ & | \xi \models_n \varphi \ U_n \psi = (\exists i. suffix \ i \ \xi \models_n \psi \wedge (\forall j < i. suffix \ j \ \xi \models_n \varphi)) \\ & | \xi \models_n \varphi \ R_n \psi = (\forall i. suffix \ i \ \xi \models_n \psi \vee (\exists j < i. suffix \ j \ \xi \models_n \varphi)) \\ & | \xi \models_n \varphi \ W_n \psi = (\forall i. suffix \ i \ \xi \models_n \varphi \vee (\exists j \leq i. suffix \ j \ \xi \models_n \psi)) \\ & | \xi \models_n \varphi \ M_n \psi = (\exists i. suffix \ i \ \xi \models_n \varphi \wedge (\forall j \leq i. suffix \ j \ \xi \models_n \psi)) \end{aligned}$$

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

lemma $semantics-ltl-ite-simps[simp]$:

$$\begin{aligned} w \models_n (if \ P \ then \ true_n \ else \ false_n) &= P \\ w \models_n (if \ P \ then \ false_n \ else \ true_n) &= (\neg P) \\ \langle proof \rangle & \end{aligned}$$

1.2.3 Conversion

fun *ltlc-to-ltln'* :: *bool* \Rightarrow 'a *ltlc* \Rightarrow 'a *ltln*

where

$ltlc\text{-to-}ltln'\ False\ true_c = true_n$
 $| ltlc\text{-to-}ltln'\ False\ false_c = false_n$
 $| ltlc\text{-to-}ltln'\ False\ prop_c(q) = prop_n(q)$
 $| ltlc\text{-to-}ltln'\ False\ (\varphi\ and_c\ \psi) = (ltlc\text{-to-}ltln'\ False\ \varphi)\ and_n\ (ltlc\text{-to-}ltln'\ False\ \psi)$
 $| ltlc\text{-to-}ltln'\ False\ (\varphi\ or_c\ \psi) = (ltlc\text{-to-}ltln'\ False\ \varphi)\ or_n\ (ltlc\text{-to-}ltln'\ False\ \psi)$
 $| ltlc\text{-to-}ltln'\ False\ (\varphi\ implies_c\ \psi) = (ltlc\text{-to-}ltln'\ True\ \varphi)\ or_n\ (ltlc\text{-to-}ltln'\ False\ \psi)$
 $| ltlc\text{-to-}ltln'\ False\ (F_c\ \varphi) = true_n\ U_n\ (ltlc\text{-to-}ltln'\ False\ \varphi)$
 $| ltlc\text{-to-}ltln'\ False\ (G_c\ \varphi) = false_n\ R_n\ (ltlc\text{-to-}ltln'\ False\ \varphi)$
 $| ltlc\text{-to-}ltln'\ False\ (\varphi\ U_c\ \psi) = (ltlc\text{-to-}ltln'\ False\ \varphi)\ U_n\ (ltlc\text{-to-}ltln'\ False\ \psi)$
 $| ltlc\text{-to-}ltln'\ False\ (\varphi\ R_c\ \psi) = (ltlc\text{-to-}ltln'\ False\ \varphi)\ R_n\ (ltlc\text{-to-}ltln'\ False\ \psi)$
 $| ltlc\text{-to-}ltln'\ False\ (\varphi\ W_c\ \psi) = (ltlc\text{-to-}ltln'\ False\ \varphi)\ W_n\ (ltlc\text{-to-}ltln'\ False\ \psi)$
 $| ltlc\text{-to-}ltln'\ False\ (\varphi\ M_c\ \psi) = (ltlc\text{-to-}ltln'\ False\ \varphi)\ M_n\ (ltlc\text{-to-}ltln'\ False\ \psi)$
 $| ltlc\text{-to-}ltln'\ True\ true_c = false_n$
 $| ltlc\text{-to-}ltln'\ True\ false_c = true_n$
 $| ltlc\text{-to-}ltln'\ True\ prop_c(q) = nprop_n(q)$
 $| ltlc\text{-to-}ltln'\ True\ (\varphi\ and_c\ \psi) = (ltlc\text{-to-}ltln'\ True\ \varphi)\ or_n\ (ltlc\text{-to-}ltln'\ True\ \psi)$
 $| ltlc\text{-to-}ltln'\ True\ (\varphi\ or_c\ \psi) = (ltlc\text{-to-}ltln'\ True\ \varphi)\ and_n\ (ltlc\text{-to-}ltln'\ True\ \psi)$
 $| ltlc\text{-to-}ltln'\ True\ (\varphi\ implies_c\ \psi) = (ltlc\text{-to-}ltln'\ False\ \varphi)\ and_n\ (ltlc\text{-to-}ltln'\ True\ \psi)$
 $| ltlc\text{-to-}ltln'\ True\ (F_c\ \varphi) = false_n\ R_n\ (ltlc\text{-to-}ltln'\ True\ \varphi)$
 $| ltlc\text{-to-}ltln'\ True\ (G_c\ \varphi) = true_n\ U_n\ (ltlc\text{-to-}ltln'\ True\ \varphi)$
 $| ltlc\text{-to-}ltln'\ True\ (\varphi\ U_c\ \psi) = (ltlc\text{-to-}ltln'\ True\ \varphi)\ R_n\ (ltlc\text{-to-}ltln'\ True\ \psi)$
 $| ltlc\text{-to-}ltln'\ True\ (\varphi\ R_c\ \psi) = (ltlc\text{-to-}ltln'\ True\ \varphi)\ U_n\ (ltlc\text{-to-}ltln'\ True\ \psi)$
 $| ltlc\text{-to-}ltln'\ True\ (\varphi\ W_c\ \psi) = (ltlc\text{-to-}ltln'\ True\ \varphi)\ M_n\ (ltlc\text{-to-}ltln'\ True\ \psi)$
 $| ltlc\text{-to-}ltln'\ True\ (\varphi\ M_c\ \psi) = (ltlc\text{-to-}ltln'\ True\ \varphi)\ W_n\ (ltlc\text{-to-}ltln'\ True\ \psi)$
 $| ltlc\text{-to-}ltln'\ b\ (not_c\ \varphi) = ltlc\text{-to-}ltln'\ (\neg b)\ \varphi$
 $| ltlc\text{-to-}ltln'\ b\ (X_c\ \varphi) = X_n\ (ltlc\text{-to-}ltln'\ b\ \varphi)$

fun *ltlc-to-ltln* :: 'a *ltlc* \Rightarrow 'a *ltln*

where

$$ltlc\text{-to}\text{-}ltln \varphi = ltlc\text{-to}\text{-}ltln' \text{ False } \varphi$$

fun *ltln-to-ltlc* :: 'a *ltln* \Rightarrow 'a *ltlc*

where

$$\begin{aligned} & ltlc\text{-to}\text{-}ltln \text{ true}_n = \text{true}_c \\ | & ltlc\text{-to}\text{-}ltln \text{ false}_n = \text{false}_c \\ | & ltlc\text{-to}\text{-}ltln \text{ prop}_n(q) = \text{prop}_c(q) \\ | & ltlc\text{-to}\text{-}ltln \text{ nprop}_n(q) = \text{not}_c(\text{prop}_c(q)) \\ | & ltlc\text{-to}\text{-}ltln (\varphi \text{ and}_n \psi) = (ltlc\text{-to}\text{-}ltln \varphi \text{ and}_c ltlc\text{-to}\text{-}ltln \psi) \\ | & ltlc\text{-to}\text{-}ltln (\varphi \text{ or}_n \psi) = (ltlc\text{-to}\text{-}ltln \varphi \text{ or}_c ltlc\text{-to}\text{-}ltln \psi) \\ | & ltlc\text{-to}\text{-}ltln (X_n \varphi) = (X_c ltlc\text{-to}\text{-}ltln \varphi) \\ | & ltlc\text{-to}\text{-}ltln (\varphi U_n \psi) = (ltlc\text{-to}\text{-}ltln \varphi U_c ltlc\text{-to}\text{-}ltln \psi) \\ | & ltlc\text{-to}\text{-}ltln (\varphi R_n \psi) = (ltlc\text{-to}\text{-}ltln \varphi R_c ltlc\text{-to}\text{-}ltln \psi) \\ | & ltlc\text{-to}\text{-}ltln (\varphi W_n \psi) = (ltlc\text{-to}\text{-}ltln \varphi W_c ltlc\text{-to}\text{-}ltln \psi) \\ | & ltlc\text{-to}\text{-}ltln (\varphi M_n \psi) = (ltlc\text{-to}\text{-}ltln \varphi M_c ltlc\text{-to}\text{-}ltln \psi) \end{aligned}$$

lemma *ltlc-to-ltln'-correct*:

$$\begin{aligned} w \models_n (ltlc\text{-to}\text{-}ltln' \text{ True } \varphi) & \iff \neg w \models_c \varphi \\ w \models_n (ltlc\text{-to}\text{-}ltln' \text{ False } \varphi) & \iff w \models_c \varphi \\ \text{size } (ltlc\text{-to}\text{-}ltln' \text{ True } \varphi) & \leq 2 * \text{size } \varphi \\ \text{size } (ltlc\text{-to}\text{-}ltln' \text{ False } \varphi) & \leq 2 * \text{size } \varphi \\ \langle \text{proof} \rangle \end{aligned}$$

lemma *ltlc-to-ltln-semantics [simp]*:

$$\begin{aligned} w \models_n ltlc\text{-to}\text{-}ltln \varphi & \iff w \models_c \varphi \\ \langle \text{proof} \rangle \end{aligned}$$

lemma *ltlc-to-ltln-size*:

$$\begin{aligned} \text{size } (ltlc\text{-to}\text{-}ltln \varphi) & \leq 2 * \text{size } \varphi \\ \langle \text{proof} \rangle \end{aligned}$$

lemma *ltln-to-ltlc-semantics [simp]*:

$$\begin{aligned} w \models_c ltlc\text{-to}\text{-}ltln \varphi & \iff w \models_n \varphi \\ \langle \text{proof} \rangle \end{aligned}$$

lemma *ltlc-to-ltln-atoms*:

$$\begin{aligned} \text{atoms}\text{-}ltln (ltlc\text{-to}\text{-}ltln \varphi) & = \text{atoms}\text{-}ltlc \varphi \\ \langle \text{proof} \rangle \end{aligned}$$

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]*:

$w \models_n not_n \varphi \longleftrightarrow \neg w \models_n \varphi$
 $\langle proof \rangle$

lemma *not_n-size*:

$size (not_n \varphi) = size \varphi$
 $\langle proof \rangle$

1.2.5 Subformulas

fun *subfrmlsn* :: 'a ltln \Rightarrow '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]*:

$\varphi \in subfrmlsn \varphi$
 $\langle proof \rangle$

lemma *subfrmlsn-finite*:

$finite (subfrmlsn \varphi)$
 $\langle proof \rangle$

lemma *subfrmlsn-card*:

$card (subfrmlsn \varphi) \leq size \varphi$
 $\langle proof \rangle$

lemma *subfrmlsn-subset*:

$$\psi \in \text{subfrmlsn } \varphi \implies \text{subfrmlsn } \psi \subseteq \text{subfrmlsn } \varphi$$

<proof>

lemma *subfrmlsn-size*:

$$\psi \in \text{subfrmlsn } \varphi \implies \text{size } \psi < \text{size } \varphi \vee \psi = \varphi$$

<proof>

abbreviation

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

lemma *size-set-diff*:

$$\text{finite } S \implies S' \subseteq S \implies \text{size-set } (S - S') = \text{size-set } S - \text{size-set } S'$$

<proof>

1.2.6 Constant Folding

lemma *U-consts[intro, simp]*:

$$\begin{aligned} w \models_n \varphi \ U_n \ \text{true}_n \\ \neg (w \models_n \varphi \ U_n \ \text{false}_n) \\ (w \models_n \ \text{false}_n \ U_n \ \varphi) &= (w \models_n \ \varphi) \end{aligned}$$

<proof>

lemma *R-consts[intro, simp]*:

$$\begin{aligned} w \models_n \varphi \ R_n \ \text{true}_n \\ \neg (w \models_n \varphi \ R_n \ \text{false}_n) \\ (w \models_n \ \text{true}_n \ R_n \ \varphi) &= (w \models_n \ \varphi) \end{aligned}$$

<proof>

lemma *W-consts[intro, simp]*:

$$\begin{aligned} w \models_n \ \text{true}_n \ W_n \ \varphi \\ w \models_n \ \varphi \ W_n \ \text{true}_n \\ (w \models_n \ \text{false}_n \ W_n \ \varphi) &= (w \models_n \ \varphi) \\ (w \models_n \ \varphi \ W_n \ \text{false}_n) &= (w \models_n \ \varphi) \end{aligned}$$

<proof>

lemma *M-consts[intro, simp]*:

$$\begin{aligned} \neg (w \models_n \ \text{false}_n \ M_n \ \varphi) \\ \neg (w \models_n \ \varphi \ M_n \ \text{false}_n) \\ (w \models_n \ \text{true}_n \ M_n \ \varphi) &= (w \models_n \ \varphi) \\ (w \models_n \ \varphi \ M_n \ \text{true}_n) &= (w \models_n \ \varphi) \end{aligned}$$

<proof>

1.2.7 Distributivity

lemma *until-and-left-distrib*:

$$w \models_n (\varphi_1 \text{ and}_n \varphi_2) U_n \psi \longleftrightarrow w \models_n (\varphi_1 U_n \psi) \text{ and}_n (\varphi_2 U_n \psi)$$

<proof>

lemma *until-or-right-distrib*:

$$w \models_n \varphi U_n (\psi_1 \text{ or}_n \psi_2) \longleftrightarrow w \models_n (\varphi U_n \psi_1) \text{ or}_n (\varphi U_n \psi_2)$$

<proof>

lemma *release-and-right-distrib*:

$$w \models_n \varphi R_n (\psi_1 \text{ and}_n \psi_2) \longleftrightarrow w \models_n (\varphi R_n \psi_1) \text{ and}_n (\varphi R_n \psi_2)$$

<proof>

lemma *release-or-left-distrib*:

$$w \models_n (\varphi_1 \text{ or}_n \varphi_2) R_n \psi \longleftrightarrow w \models_n (\varphi_1 R_n \psi) \text{ or}_n (\varphi_2 R_n \psi)$$

<proof>

lemma *strong-release-and-right-distrib*:

$$w \models_n \varphi M_n (\psi_1 \text{ and}_n \psi_2) \longleftrightarrow w \models_n (\varphi M_n \psi_1) \text{ and}_n (\varphi M_n \psi_2)$$

<proof>

lemma *strong-release-or-left-distrib*:

$$w \models_n (\varphi_1 \text{ or}_n \varphi_2) M_n \psi \longleftrightarrow w \models_n (\varphi_1 M_n \psi) \text{ or}_n (\varphi_2 M_n \psi)$$

<proof>

lemma *weak-until-and-left-distrib*:

$$w \models_n (\varphi_1 \text{ and}_n \varphi_2) W_n \psi \longleftrightarrow w \models_n (\varphi_1 W_n \psi) \text{ and}_n (\varphi_2 W_n \psi)$$

<proof>

lemma *weak-until-or-right-distrib*:

$$w \models_n \varphi W_n (\psi_1 \text{ or}_n \psi_2) \longleftrightarrow w \models_n (\varphi W_n \psi_1) \text{ or}_n (\varphi W_n \psi_2)$$

<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]*:

$$w \models_n G_n (\varphi W_n \psi) \longleftrightarrow w \models_n G_n (\varphi \text{or}_n \psi)$$

<proof>

lemma *finally-strong-release[simp]*:

$$w \models_n F_n (\varphi M_n \psi) \longleftrightarrow w \models_n F_n (\varphi \text{and}_n \psi)$$

<proof>

1.2.9 Weak and strong operators

lemma *ltln-weak-strong*:

$$w \models_n \varphi W_n \psi \longleftrightarrow w \models_n (G_n \varphi) \text{or}_n (\varphi U_n \psi)$$

$$w \models_n \varphi R_n \psi \longleftrightarrow w \models_n (G_n \psi) \text{or}_n (\varphi M_n \psi)$$

<proof>

lemma *ltln-strong-weak*:

$$w \models_n \varphi U_n \psi \longleftrightarrow w \models_n (F_n \psi) \text{and}_n (\varphi W_n \psi)$$

$$w \models_n \varphi M_n \psi \longleftrightarrow w \models_n (F_n \varphi) \text{and}_n (\varphi R_n \psi)$$

<proof>

lemma *ltln-strong-to-weak*:

$$w \models_n \varphi U_n \psi \implies w \models_n \varphi W_n \psi$$

$$w \models_n \varphi M_n \psi \implies w \models_n \varphi R_n \psi$$

<proof>

lemma *ltln-weak-to-strong*:

$$\llbracket w \models_n \varphi W_n \psi; \neg w \models_n G_n \varphi \rrbracket \implies w \models_n \varphi U_n \psi$$

$$\begin{aligned} & \llbracket w \models_n \varphi \ W_n \ \psi; w \models_n F_n \ \psi \rrbracket \implies w \models_n \varphi \ U_n \ \psi \\ & \llbracket w \models_n \varphi \ R_n \ \psi; \neg w \models_n G_n \ \psi \rrbracket \implies w \models_n \varphi \ M_n \ \psi \\ & \llbracket w \models_n \varphi \ R_n \ \psi; w \models_n F_n \ \varphi \rrbracket \implies w \models_n \varphi \ M_n \ \psi \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *ltln-StrongRelease-to-Until*:

$$w \models_n \varphi \ M_n \ \psi \longleftrightarrow w \models_n \psi \ U_n \ (\varphi \ \text{and}_n \ \psi)$$

$\langle \text{proof} \rangle$

lemma *ltln-Release-to-WeakUntil*:

$$w \models_n \varphi \ R_n \ \psi \longleftrightarrow w \models_n \psi \ W_n \ (\varphi \ \text{and}_n \ \psi)$$

$\langle \text{proof} \rangle$

lemma *ltln-WeakUntil-to-Release*:

$$w \models_n \varphi \ W_n \ \psi \longleftrightarrow w \models_n \psi \ R_n \ (\varphi \ \text{or}_n \ \psi)$$

$\langle \text{proof} \rangle$

lemma *ltln-Until-to-StrongRelease*:

$$w \models_n \varphi \ U_n \ \psi \longleftrightarrow w \models_n \psi \ M_n \ (\varphi \ \text{or}_n \ \psi)$$

$\langle \text{proof} \rangle$

1.2.10 GF and FG semantics

lemma *GF-suffix*:

$$\text{suffix } i \ w \models_n G_n \ (F_n \ \psi) \longleftrightarrow w \models_n G_n \ (F_n \ \psi)$$

$\langle \text{proof} \rangle$

lemma *FG-suffix*:

$$\text{suffix } i \ w \models_n F_n \ (G_n \ \psi) \longleftrightarrow w \models_n F_n \ (G_n \ \psi)$$

$\langle \text{proof} \rangle$

lemma *GF-Inf-many*:

$$w \models_n G_n \ (F_n \ \varphi) \longleftrightarrow (\exists_{\infty} i. \text{suffix } i \ w \models_n \varphi)$$

$\langle \text{proof} \rangle$

lemma *FG-All-many*:

$$w \models_n F_n \ (G_n \ \varphi) \longleftrightarrow (\forall_{\infty} i. \text{suffix } i \ w \models_n \varphi)$$

$\langle \text{proof} \rangle$

lemma *MOST-nat-add*:

$$(\forall_{\infty} i::\text{nat}. P \ i) \longleftrightarrow (\forall_{\infty} i. P \ (i + j))$$

$\langle \text{proof} \rangle$

lemma *INFM-nat-add*:

$$(\exists_{\infty} i :: \text{nat}. P\ i) \longleftrightarrow (\exists_{\infty} i. P\ (i + j))$$

$\langle \text{proof} \rangle$

lemma *FG-suffix-G*:

$$w \models_n F_n\ (G_n\ \varphi) \implies \forall_{\infty} i. \text{suffix}\ i\ w \models_n G_n\ \varphi$$

$\langle \text{proof} \rangle$

lemma *Alm-all-GF-F*:

$$\forall_{\infty} i. \text{suffix}\ i\ w \models_n G_n\ (F_n\ \psi) \longleftrightarrow \text{suffix}\ i\ w \models_n F_n\ \psi$$

$\langle \text{proof} \rangle$

lemma *Alm-all-FG-G*:

$$\forall_{\infty} i. \text{suffix}\ i\ w \models_n F_n\ (G_n\ \psi) \longleftrightarrow \text{suffix}\ i\ w \models_n G_n\ \psi$$

$\langle \text{proof} \rangle$

1.2.11 Expansion

lemma *ltln-expand-Until*:

$$\xi \models_n \varphi\ U_n\ \psi \longleftrightarrow (\xi \models_n \psi\ \text{or}_n\ (\varphi\ \text{and}_n\ (X_n\ (\varphi\ U_n\ \psi))))$$

(**is** ?lhs = ?rhs)

$\langle \text{proof} \rangle$

lemma *ltln-expand-Release*:

$$\xi \models_n \varphi\ R_n\ \psi \longleftrightarrow (\xi \models_n \psi\ \text{and}_n\ (\varphi\ \text{or}_n\ (X_n\ (\varphi\ R_n\ \psi))))$$

(**is** ?lhs = ?rhs)

$\langle \text{proof} \rangle$

lemma *ltln-expand-WeakUntil*:

$$\xi \models_n \varphi\ W_n\ \psi \longleftrightarrow (\xi \models_n \psi\ \text{or}_n\ (\varphi\ \text{and}_n\ (X_n\ (\varphi\ W_n\ \psi))))$$

(**is** ?lhs = ?rhs)

$\langle \text{proof} \rangle$

lemma *ltln-expand-StrongRelease*:

$$\xi \models_n \varphi\ M_n\ \psi \longleftrightarrow (\xi \models_n \psi\ \text{and}_n\ (\varphi\ \text{or}_n\ (X_n\ (\varphi\ M_n\ \psi))))$$

(**is** ?lhs = ?rhs)

$\langle \text{proof} \rangle$

lemma *ltln-Release-alterdef*:

$$w \models_n \varphi\ R_n\ \psi \longleftrightarrow w \models_n (G_n\ \psi)\ \text{or}_n\ (\psi\ U_n\ (\varphi\ \text{and}_n\ \psi))$$

$\langle \text{proof} \rangle$

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* =

<i>True-ltlr</i>	$(true_r)$
<i>False-ltlr</i>	$(false_r)$
<i>Prop-ltlr</i> 'a	$(prop_r '(-))$
<i>Nprop-ltlr</i> 'a	$(nprop_r '(-))$
<i>And-ltlr</i> 'a <i>ltlr</i> 'a <i>ltlr</i>	$(- \text{and}_r - [82,82] 81)$
<i>Or-ltlr</i> 'a <i>ltlr</i> 'a <i>ltlr</i>	$(- \text{or}_r - [84,84] 83)$
<i>Next-ltlr</i> 'a <i>ltlr</i>	$(X_r - [88] 87)$
<i>Until-ltlr</i> 'a <i>ltlr</i> 'a <i>ltlr</i>	$(- U_r - [84,84] 83)$
<i>Release-ltlr</i> 'a <i>ltlr</i> 'a <i>ltlr</i>	$(- R_r - [84,84] 83)$

1.3.2 Semantics

primrec *semantics-ltlr* :: [*a set word*, 'a *ltlr*] \Rightarrow *bool* ($- \models_r - [80,80] 80$)

where

$\xi \models_r true_r = True$
$\xi \models_r false_r = False$
$\xi \models_r prop_r(q) = (q \in \xi 0)$
$\xi \models_r nprop_r(q) = (q \notin \xi 0)$
$\xi \models_r \varphi \text{and}_r \psi = (\xi \models_r \varphi \wedge \xi \models_r \psi)$
$\xi \models_r \varphi \text{or}_r \psi = (\xi \models_r \varphi \vee \xi \models_r \psi)$
$\xi \models_r X_r \varphi = (\text{suffix } 1 \xi \models_r \varphi)$
$\xi \models_r \varphi U_r \psi = (\exists i. \text{suffix } i \xi \models_r \psi \wedge (\forall j < i. \text{suffix } j \xi \models_r \varphi))$
$\xi \models_r \varphi R_r \psi = (\forall i. \text{suffix } i \xi \models_r \psi \vee (\exists j < i. \text{suffix } j \xi \models_r \varphi))$

1.3.3 Conversion

fun *ltln-to-ltlr* :: 'a *ltln* \Rightarrow 'a *ltlr*

where

$ltln\text{-to-ltlr } true_n = true_r$
$ltln\text{-to-ltlr } false_n = false_r$
$ltln\text{-to-ltlr } prop_n(a) = prop_r(a)$
$ltln\text{-to-ltlr } nprop_n(a) = nprop_r(a)$
$ltln\text{-to-ltlr } (\varphi \text{and}_n \psi) = (ltln\text{-to-ltlr } \varphi) \text{and}_r (ltln\text{-to-ltlr } \psi)$
$ltln\text{-to-ltlr } (\varphi \text{or}_n \psi) = (ltln\text{-to-ltlr } \varphi) \text{or}_r (ltln\text{-to-ltlr } \psi)$
$ltln\text{-to-ltlr } (X_n \varphi) = X_r (ltln\text{-to-ltlr } \varphi)$
$ltln\text{-to-ltlr } (\varphi U_n \psi) = (ltln\text{-to-ltlr } \varphi) U_r (ltln\text{-to-ltlr } \psi)$
$ltln\text{-to-ltlr } (\varphi R_n \psi) = (ltln\text{-to-ltlr } \varphi) R_r (ltln\text{-to-ltlr } \psi)$

$| \text{ltln-to-ltlr } (\varphi \ W_n \ \psi) = (\text{ltln-to-ltlr } \psi) \ R_r \ ((\text{ltln-to-ltlr } \varphi) \ \text{or}_r \ (\text{ltln-to-ltlr } \psi))$
 $| \text{ltln-to-ltlr } (\varphi \ M_n \ \psi) = (\text{ltln-to-ltlr } \psi) \ U_r \ ((\text{ltln-to-ltlr } \varphi) \ \text{and}_r \ (\text{ltln-to-ltlr } \psi))$

fun *ltlr-to-ltln* :: 'a *ltlr* \Rightarrow 'a *ltln*

where

$\text{ltlr-to-ltln } \text{true}_r = \text{true}_n$
 $| \text{ltlr-to-ltln } \text{false}_r = \text{false}_n$
 $| \text{ltlr-to-ltln } \text{prop}_r(a) = \text{prop}_n(a)$
 $| \text{ltlr-to-ltln } \text{nprop}_r(a) = \text{nprop}_n(a)$
 $| \text{ltlr-to-ltln } (\varphi \ \text{and}_r \ \psi) = (\text{ltlr-to-ltln } \varphi) \ \text{and}_n \ (\text{ltlr-to-ltln } \psi)$
 $| \text{ltlr-to-ltln } (\varphi \ \text{or}_r \ \psi) = (\text{ltlr-to-ltln } \varphi) \ \text{or}_n \ (\text{ltlr-to-ltln } \psi)$
 $| \text{ltlr-to-ltln } (X_r \ \varphi) = X_n \ (\text{ltlr-to-ltln } \varphi)$
 $| \text{ltlr-to-ltln } (\varphi \ U_r \ \psi) = (\text{ltlr-to-ltln } \varphi) \ U_n \ (\text{ltlr-to-ltln } \psi)$
 $| \text{ltlr-to-ltln } (\varphi \ R_r \ \psi) = (\text{ltlr-to-ltln } \varphi) \ R_n \ (\text{ltlr-to-ltln } \psi)$

lemma *ltln-to-ltlr-semantics*:

$w \models_r \text{ltln-to-ltlr } \varphi \longleftrightarrow w \models_n \varphi$
<proof>

lemma *ltlr-to-ltln-semantics*:

$w \models_n \text{ltlr-to-ltln } \varphi \longleftrightarrow w \models_r \varphi$
<proof>

1.3.4 Negation

fun *not_r*

where

$\text{not}_r \text{true}_r = \text{false}_r$
 $| \text{not}_r \text{false}_r = \text{true}_r$
 $| \text{not}_r \text{prop}_r(a) = \text{nprop}_r(a)$
 $| \text{not}_r \text{nprop}_r(a) = \text{prop}_r(a)$
 $| \text{not}_r (\varphi \ \text{and}_r \ \psi) = (\text{not}_r \ \varphi) \ \text{or}_r \ (\text{not}_r \ \psi)$
 $| \text{not}_r (\varphi \ \text{or}_r \ \psi) = (\text{not}_r \ \varphi) \ \text{and}_r \ (\text{not}_r \ \psi)$
 $| \text{not}_r (X_r \ \varphi) = X_r \ (\text{not}_r \ \varphi)$
 $| \text{not}_r (\varphi \ U_r \ \psi) = (\text{not}_r \ \varphi) \ R_r \ (\text{not}_r \ \psi)$
 $| \text{not}_r (\varphi \ R_r \ \psi) = (\text{not}_r \ \varphi) \ U_r \ (\text{not}_r \ \psi)$

lemma *not_r-semantics* [*simp*]:

$w \models_r \text{not}_r \ \varphi \longleftrightarrow \neg w \models_r \varphi$
<proof>

1.3.5 Subformulas

fun *subfrmlsr* :: 'a ltlr \Rightarrow 'a ltlr set

where

$subfrmlsr (\varphi \text{ and}_r \psi) = \{\varphi \text{ and}_r \psi\} \cup subfrmlsr \varphi \cup subfrmlsr \psi$
 $| subfrmlsr (\varphi \text{ or}_r \psi) = \{\varphi \text{ or}_r \psi\} \cup subfrmlsr \varphi \cup subfrmlsr \psi$
 $| subfrmlsr (\varphi U_r \psi) = \{\varphi U_r \psi\} \cup subfrmlsr \varphi \cup subfrmlsr \psi$
 $| subfrmlsr (\varphi R_r \psi) = \{\varphi R_r \psi\} \cup subfrmlsr \varphi \cup subfrmlsr \psi$
 $| subfrmlsr (X_r \varphi) = \{X_r \varphi\} \cup subfrmlsr \varphi$
 $| subfrmlsr x = \{x\}$

lemma *subfrmlsr-id[simp]*:

$\varphi \in subfrmlsr \varphi$
 $\langle proof \rangle$

lemma *subfrmlsr-finite*:

$finite (subfrmlsr \varphi)$
 $\langle proof \rangle$

lemma *subfrmlsr-subset*:

$\psi \in subfrmlsr \varphi \implies subfrmlsr \psi \subseteq subfrmlsr \varphi$
 $\langle proof \rangle$

lemma *subfrmlsr-size*:

$\psi \in subfrmlsr \varphi \implies size \psi < size \varphi \vee \psi = \varphi$
 $\langle proof \rangle$

1.3.6 Expansion lemmas

lemma *ltlr-expand-Until*:

$\xi \models_r \varphi U_r \psi \longleftrightarrow (\xi \models_r \psi \text{ or}_r (\varphi \text{ and}_r (X_r (\varphi U_r \psi))))$
 $\langle proof \rangle$

lemma *ltlr-expand-Release*:

$\xi \models_r \varphi R_r \psi \longleftrightarrow (\xi \models_r \psi \text{ and}_r (\varphi \text{ or}_r (X_r (\varphi R_r \psi))))$
 $\langle proof \rangle$

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 $'a$ *pttl* =

<i>False-ltlp</i>	$(false_p)$
<i>Atom-ltlp</i> $'a \Rightarrow bool$	$(atom_p '(-))$
<i>Implies-ltlp</i> $'a$ <i>pttl</i> $'a$ <i>pttl</i>	$(- \textit{implies}_p - [81,81] 80)$
<i>Next-ltlp</i> $'a$ <i>pttl</i>	$(X_p - [88] 87)$
<i>Until-ltlp</i> $'a$ <i>pttl</i> $'a$ <i>pttl</i>	$(- U_p - [84,84] 83)$

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

definition *Not-ltlp* ($not_p - [85] 85$)

where

$$not_p \varphi \equiv \varphi \textit{implies}_p false_p$$

definition *True-ltlp* ($true_p$)

where

$$true_p \equiv not_p false_p$$

definition *Or-ltlp* ($- or_p - [81,81] 80$)

where

$$\varphi or_p \psi \equiv (not_p \varphi) \textit{implies}_p \psi$$

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

$$\varphi M_p \psi \equiv \psi U_p (\varphi \text{ and}_p \psi)$$

1.4.2 Semantics

fun *semantics-pltl* :: [*'a word, 'a pltl*] \Rightarrow *bool* ($- \models_p - [80,80] 80$)

where

$$\begin{aligned} & w \models_p \text{false}_p = \text{False} \\ | & w \models_p \text{atom}_p(p) = (p (w 0)) \\ | & w \models_p \varphi \text{ implies}_p \psi = (w \models_p \varphi \longrightarrow w \models_p \psi) \\ | & w \models_p X_p \varphi = (\text{suffix } 1 w \models_p \varphi) \\ | & w \models_p \varphi U_p \psi = (\exists i. \text{suffix } i w \models_p \psi \wedge (\forall j < i. \text{suffix } j w \models_p \varphi)) \end{aligned}$$

lemma *semantics-pltl-sugar* [*simp*]:

$$\begin{aligned} & w \models_p \text{not}_p \varphi = (\neg w \models_p \varphi) \\ & w \models_p \text{true}_p = \text{True} \\ & w \models_p \varphi \text{ or}_p \psi = (w \models_p \varphi \vee w \models_p \psi) \\ & w \models_p \varphi \text{ and}_p \psi = (w \models_p \varphi \wedge w \models_p \psi) \\ & w \models_p F_p \varphi = (\exists i. \text{suffix } i w \models_p \varphi) \\ & w \models_p G_p \varphi = (\forall i. \text{suffix } i w \models_p \varphi) \\ & w \models_p \varphi R_p \psi = (\forall i. \text{suffix } i w \models_p \psi \vee (\exists j < i. \text{suffix } j w \models_p \varphi)) \\ & w \models_p \varphi W_p \psi = (\forall i. \text{suffix } i w \models_p \varphi \vee (\exists j \leq i. \text{suffix } j w \models_p \psi)) \\ & w \models_p \varphi M_p \psi = (\exists i. \text{suffix } i w \models_p \varphi \wedge (\forall j \leq i. \text{suffix } j w \models_p \psi)) \\ & \langle \text{proof} \rangle \end{aligned}$$

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

1.4.3 Conversion

fun *ltlc-to-pltl* :: [*'a ltlc*] \Rightarrow [*'a set pltl*]

where

$$\begin{aligned} & \text{ltlc-to-pltl } \text{true}_c = \text{true}_p \\ | & \text{ltlc-to-pltl } \text{false}_c = \text{false}_p \\ | & \text{ltlc-to-pltl } (\text{prop}_c(q)) = \text{atom}_p((\in) q) \\ | & \text{ltlc-to-pltl } (\text{not}_c \varphi) = \text{not}_p (\text{ltlc-to-pltl } \varphi) \\ | & \text{ltlc-to-pltl } (\varphi \text{ and}_c \psi) = (\text{ltlc-to-pltl } \varphi) \text{ and}_p (\text{ltlc-to-pltl } \psi) \\ | & \text{ltlc-to-pltl } (\varphi \text{ or}_c \psi) = (\text{ltlc-to-pltl } \varphi) \text{ or}_p (\text{ltlc-to-pltl } \psi) \\ | & \text{ltlc-to-pltl } (\varphi \text{ implies}_c \psi) = (\text{ltlc-to-pltl } \varphi) \text{ implies}_p (\text{ltlc-to-pltl } \psi) \\ | & \text{ltlc-to-pltl } (X_c \varphi) = X_p (\text{ltlc-to-pltl } \varphi) \\ | & \text{ltlc-to-pltl } (F_c \varphi) = F_p (\text{ltlc-to-pltl } \varphi) \\ | & \text{ltlc-to-pltl } (G_c \varphi) = G_p (\text{ltlc-to-pltl } \varphi) \\ | & \text{ltlc-to-pltl } (\varphi U_c \psi) = (\text{ltlc-to-pltl } \varphi) U_p (\text{ltlc-to-pltl } \psi) \\ | & \text{ltlc-to-pltl } (\varphi R_c \psi) = (\text{ltlc-to-pltl } \varphi) R_p (\text{ltlc-to-pltl } \psi) \end{aligned}$$

| $ltlc\text{-}to\text{-}pltl (\varphi W_c \psi) = (ltlc\text{-}to\text{-}pltl \varphi) W_p (ltlc\text{-}to\text{-}pltl \psi)$
| $ltlc\text{-}to\text{-}pltl (\varphi M_c \psi) = (ltlc\text{-}to\text{-}pltl \varphi) M_p (ltlc\text{-}to\text{-}pltl \psi)$

lemma *ltlc-to-pltl-semantic* [*simp*]:
 $w \models_p (ltlc\text{-}to\text{-}pltl \varphi) \longleftrightarrow w \models_c \varphi$
 $\langle proof \rangle$

1.4.4 Atoms

fun *atoms-pltl* :: 'a pltl \Rightarrow ('a \Rightarrow bool) set
where

$atoms\text{-}pltl\ false_p = \{\}$
| $atoms\text{-}pltl\ atom_p(p) = \{p\}$
| $atoms\text{-}pltl (\varphi\ implies_p \psi) = atoms\text{-}pltl \varphi \cup atoms\text{-}pltl \psi$
| $atoms\text{-}pltl (X_p \varphi) = atoms\text{-}pltl \varphi$
| $atoms\text{-}pltl (\varphi\ U_p \psi) = atoms\text{-}pltl \varphi \cup atoms\text{-}pltl \psi$

lemma *atoms-finite* [*iff*]:
 $finite (atoms\text{-}pltl \varphi)$
 $\langle proof \rangle$

lemma *atoms-pltl-sugar* [*simp*]:
 $atoms\text{-}pltl (not_p \varphi) = atoms\text{-}pltl \varphi$
 $atoms\text{-}pltl\ true_p = \{\}$
 $atoms\text{-}pltl (\varphi\ or_p \psi) = atoms\text{-}pltl \varphi \cup atoms\text{-}pltl \psi$
 $atoms\text{-}pltl (\varphi\ and_p \psi) = atoms\text{-}pltl \varphi \cup atoms\text{-}pltl \psi$
 $atoms\text{-}pltl (F_p \varphi) = atoms\text{-}pltl \varphi$
 $atoms\text{-}pltl (G_p \varphi) = atoms\text{-}pltl \varphi$
 $\langle proof \rangle$

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-ltl*-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 *ltn*.

2.1 Constant Eliminating Constructors

definition *mk-and*

where

$mk\text{-and } x y \equiv case\ x\ of\ false_n \Rightarrow false_n \mid true_n \Rightarrow y \mid - \Rightarrow (case\ y\ of\ false_n \Rightarrow false_n \mid true_n \Rightarrow x \mid - \Rightarrow x\ and_n\ y)$

definition *mk-or*

where

$mk\text{-or } x y \equiv case\ x\ of\ false_n \Rightarrow y \mid true_n \Rightarrow true_n \mid - \Rightarrow (case\ y\ of\ true_n \Rightarrow true_n \mid false_n \Rightarrow x \mid - \Rightarrow x\ or_n\ y)$

fun *remove-strong-ops*

where

$remove\text{-strong-ops } (x\ U_n\ y) = remove\text{-strong-ops } y$
 $\mid remove\text{-strong-ops } (x\ M_n\ y) = x\ and_n\ y$
 $\mid remove\text{-strong-ops } (x\ or_n\ y) = remove\text{-strong-ops } x\ or_n\ remove\text{-strong-ops } y$
 $\mid remove\text{-strong-ops } x = x$

fun *remove-weak-ops*

where

$remove\text{-weak-ops } (x\ R_n\ y) = remove\text{-weak-ops } y$
 $\mid remove\text{-weak-ops } (x\ W_n\ y) = x\ or_n\ y$
 $\mid remove\text{-weak-ops } (x\ and_n\ y) = remove\text{-weak-ops } x\ and_n\ remove\text{-weak-ops } y$
 $\mid remove\text{-weak-ops } x = x$

definition *mk-finally*

where

$mk\text{-finally } x \equiv case\ x\ of\ true_n \Rightarrow true_n \mid false_n \Rightarrow false_n \mid - \Rightarrow F_n\ (remove\text{-strong-ops } x)$

definition *mk-globally*

where

$mk\text{-globally } x \equiv case\ x\ of\ true_n \Rightarrow true_n \mid false_n \Rightarrow false_n \mid - \Rightarrow G_n\ (remove\text{-weak-ops } x)$

definition *mk-until*

where

$mk\text{-until } x y \equiv \text{case } x \text{ of } false_n \Rightarrow y$
| $true_n \Rightarrow mk\text{-finally } y$
| $- \Rightarrow (\text{case } y \text{ of } true_n \Rightarrow true_n \mid false_n \Rightarrow false_n \mid - \Rightarrow x U_n y)$

definition *mk-release*

where

$mk\text{-release } x y \equiv \text{case } x \text{ of } true_n \Rightarrow y$
| $false_n \Rightarrow mk\text{-globally } y$
| $- \Rightarrow (\text{case } y \text{ of } true_n \Rightarrow true_n \mid false_n \Rightarrow false_n \mid - \Rightarrow x R_n y)$

definition *mk-weak-until*

where

$mk\text{-weak-until } x y \equiv \text{case } y \text{ of } true_n \Rightarrow true_n$
| $false_n \Rightarrow mk\text{-globally } x$
| $- \Rightarrow (\text{case } x \text{ of } true_n \Rightarrow true_n \mid false_n \Rightarrow y \mid - \Rightarrow x W_n y)$

definition *mk-strong-release*

where

$mk\text{-strong-release } x y \equiv \text{case } y \text{ of } false_n \Rightarrow false_n$
| $true_n \Rightarrow mk\text{-finally } x$
| $- \Rightarrow (\text{case } x \text{ of } true_n \Rightarrow y \mid false_n \Rightarrow false_n \mid - \Rightarrow x M_n y)$

definition *mk-next*

where

$mk\text{-next } x \equiv \text{case } x \text{ of } true_n \Rightarrow true_n \mid false_n \Rightarrow false_n \mid - \Rightarrow X_n x$

definition *mk-next-pow* (X_n'')

where

$mk\text{-next-pow } n x \equiv \text{case } x \text{ of } true_n \Rightarrow true_n \mid false_n \Rightarrow false_n \mid - \Rightarrow$
 $(Next\text{-ltn } \widetilde{\sim} n) x$

lemma *mk-and-semantics* [*simp*]:

$w \models_n mk\text{-and } x y \longleftrightarrow w \models_n x \text{ and}_n y$
 $\langle proof \rangle$

lemma *mk-or-semantics* [*simp*]:

$w \models_n mk\text{-or } x y \longleftrightarrow w \models_n x \text{ or}_n y$
 $\langle proof \rangle$

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

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

lemma *mk-finally-semantic* [*simp*]:
 $w \models_n \text{mk-finally } x \longleftrightarrow w \models_n F_n x$
 ⟨*proof*⟩

lemma *mk-globally-semantic* [*simp*]:
 $w \models_n \text{mk-globally } x \longleftrightarrow w \models_n G_n x$
 ⟨*proof*⟩

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

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

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

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

lemma *mk-next-semantic* [*simp*]:
 $w \models_n \text{mk-next } x \longleftrightarrow w \models_n X_n x$
 ⟨*proof*⟩

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

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

2.2 Constant Propagation

fun *is-constant* :: 'a ltl_n ⇒ 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-antics*:

$is_constant\ \varphi \implies ((\forall w. w \models_n \varphi) \vee \neg(\exists w. w \models_n \varphi))$
 ⟨proof⟩

lemma *until-constant-simp*:

$is_constant\ \psi \implies w \models_n \varphi\ U_n\ \psi \longleftrightarrow w \models_n\ \psi$
 ⟨proof⟩

lemma *release-constant-simp*:

$is_constant\ \psi \implies w \models_n \varphi\ R_n\ \psi \longleftrightarrow w \models_n\ \psi$
 ⟨proof⟩

lemma *mk-next-pow-dist*:

$mk_next_pow\ (i + j)\ \varphi = mk_next_pow\ i\ (mk_next_pow\ j\ \varphi)$
 ⟨proof⟩

lemma *mk-next-pow-until*:

$suffix\ (min\ i\ j)\ w \models_n\ (mk_next_pow\ (i - j)\ \varphi)\ U_n\ (mk_next_pow\ (j - i)$
 $\psi) \longleftrightarrow w \models_n\ (mk_next_pow\ i\ \varphi)\ U_n\ (mk_next_pow\ j\ \psi)$
 ⟨proof⟩

lemma *mk-next-pow-release*:

$suffix\ (min\ i\ j)\ w \models_n\ (mk_next_pow\ (i - j)\ \varphi)\ R_n\ (mk_next_pow\ (j - i)$
 $\psi) \longleftrightarrow w \models_n\ (mk_next_pow\ i\ \varphi)\ R_n\ (mk_next_pow\ j\ \psi)$
 ⟨proof⟩

2.3 X-Normalisation

The following rewrite functions pull 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* \Rightarrow *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

\langle *proof* \rangle

fun *combine* :: ('*a* *ltln* \Rightarrow '*a* *ltln* \Rightarrow '*a* *ltln*) \Rightarrow ('*a* *ltln* * *enat*) \Rightarrow ('*a* *ltln* * *enat*) \Rightarrow ('*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)

\langle *proof* \rangle

abbreviation *to-ltln* :: ('*a* *ltln* * *enat*) \Rightarrow '*a* *ltln*

where

to-ltln *x* \equiv *mk-next-pow* (*the-enat-0* (*snd* *x*)) (*fst* *x*)

fun *rewrite-X-enat* :: '*a* *ltln* \Rightarrow ('*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)

$(\text{rewrite-}X\text{-enat } \psi)$
 $| \text{rewrite-}X\text{-enat } (\varphi M_n \psi) = \text{combine mk-strong-release } (\text{rewrite-}X\text{-enat } \varphi)$
 $(\text{rewrite-}X\text{-enat } \psi)$
 $| \text{rewrite-}X\text{-enat } (X_n \varphi) = (\lambda(\varphi, n). (\varphi, e\text{Suc } n)) (\text{rewrite-}X\text{-enat } \varphi)$

definition

$\text{rewrite-}X \varphi = \text{to-ltln } (\text{rewrite-}X\text{-enat } \varphi)$

lemma combine-infinity-invariant:

assumes $i = \infty \longleftrightarrow \text{is-constant } x$
assumes $j = \infty \longleftrightarrow \text{is-constant } y$
shows $\text{combine mk-and } (x, i) (y, j) = (z, k) \implies (k = \infty \longleftrightarrow \text{is-constant } z)$
and $\text{combine mk-or } (x, i) (y, j) = (z, k) \implies (k = \infty \longleftrightarrow \text{is-constant } z)$
and $\text{combine mk-until } (x, i) (y, j) = (z, k) \implies (k = \infty \longleftrightarrow \text{is-constant } z)$
and $\text{combine mk-release } (x, i) (y, j) = (z, k) \implies (k = \infty \longleftrightarrow \text{is-constant } z)$
and $\text{combine mk-weak-until } (x, i) (y, j) = (z, k) \implies (k = \infty \longleftrightarrow \text{is-constant } z)$
and $\text{combine mk-strong-release } (x, i) (y, j) = (z, k) \implies (k = \infty \longleftrightarrow \text{is-constant } z)$
 $\langle \text{proof} \rangle$

lemma combine-and-or-semantics:

assumes $i = \infty \longleftrightarrow \text{is-constant } \varphi$
assumes $j = \infty \longleftrightarrow \text{is-constant } \psi$
shows $w \models_n \text{to-ltln } (\text{combine mk-and } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n \text{to-ltln } (\varphi, i) \text{ and}_n \text{to-ltln } (\psi, j)$
and $w \models_n \text{to-ltln } (\text{combine mk-or } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n \text{to-ltln } (\varphi, i) \text{ or}_n \text{to-ltln } (\psi, j)$
 $\langle \text{proof} \rangle$

lemma combine-until-release-semantics:

assumes $i = \infty \longleftrightarrow \text{is-constant } \varphi$
assumes $j = \infty \longleftrightarrow \text{is-constant } \psi$
shows $w \models_n \text{to-ltln } (\text{combine mk-until } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n \text{to-ltln } (\varphi, i) U_n \text{to-ltln } (\psi, j)$
and $w \models_n \text{to-ltln } (\text{combine mk-release } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n \text{to-ltln } (\varphi, i) R_n \text{to-ltln } (\psi, j)$
 $\langle \text{proof} \rangle$

lemma *combine-weak-until-strong-release-semantics:*

assumes $i = \infty \longleftrightarrow \text{is-constant } \varphi$
assumes $j = \infty \longleftrightarrow \text{is-constant } \psi$
shows $w \models_n \text{to-ltln } (\text{combine mk-weak-until } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n$
 $\text{to-ltln } (\varphi, i) W_n \text{to-ltln } (\psi, j)$
and $w \models_n \text{to-ltln } (\text{combine mk-strong-release } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n$
 $\text{to-ltln } (\varphi, i) M_n \text{to-ltln } (\psi, j)$
 $\langle \text{proof} \rangle$

lemma *rewrite-X-enat-infinity-invariant:*

$\text{snd } (\text{rewrite-X-enat } \varphi) = \infty \longleftrightarrow \text{is-constant } (\text{fst } (\text{rewrite-X-enat } \varphi))$
 $\langle \text{proof} \rangle$

lemma *rewrite-X-enat-correct:*

$w \models_n \varphi \longleftrightarrow w \models_n \text{to-ltln } (\text{rewrite-X-enat } \varphi)$
 $\langle \text{proof} \rangle$

lemma *rewrite-X-sound [simp]:*

$w \models_n \text{rewrite-X } \varphi \longleftrightarrow w \models_n \varphi$
 $\langle \text{proof} \rangle$

2.4 Pure Eventual, Pure Universal, and Suspending Formulas

fun *pure-eventual* :: 'a ltln \Rightarrow bool

where

$\text{pure-eventual true}_n = \text{True}$
 $|\ \text{pure-eventual false}_n = \text{True}$
 $|\ \text{pure-eventual } (\mu \text{ and}_n \mu') = (\text{pure-eventual } \mu \wedge \text{pure-eventual } \mu')$
 $|\ \text{pure-eventual } (\mu \text{ or}_n \mu') = (\text{pure-eventual } \mu \wedge \text{pure-eventual } \mu')$
 $|\ \text{pure-eventual } (\mu U_n \mu') = (\mu = \text{true}_n \vee \text{pure-eventual } \mu')$
 $|\ \text{pure-eventual } (\mu R_n \mu') = (\text{pure-eventual } \mu \wedge \text{pure-eventual } \mu')$
 $|\ \text{pure-eventual } (\mu W_n \mu') = (\text{pure-eventual } \mu \wedge \text{pure-eventual } \mu')$
 $|\ \text{pure-eventual } (\mu M_n \mu') = (\text{pure-eventual } \mu \wedge \text{pure-eventual } \mu' \vee \mu' = \text{true}_n)$
 $|\ \text{pure-eventual } (X_n \mu) = \text{pure-eventual } \mu$
 $|\ \text{pure-eventual } - = \text{False}$

fun *pure-universal* :: 'a ltln \Rightarrow bool

where

$\text{pure-universal true}_n = \text{True}$
 $|\ \text{pure-universal false}_n = \text{True}$

| *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') = ($\nu = \text{false}_n \vee$ *pure-universal* ν')
 | *pure-universal* (ν *W*_{*n*} ν') = (*pure-universal* ν \wedge *pure-universal* $\nu' \vee \nu' = \text{false}_n$)
 | *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) = ($\varphi = \text{true}_n \wedge$ *pure-universal* $\xi \vee$ *suspendable* ξ)
 | *suspendable* (φ *R*_{*n*} ξ) = ($\varphi = \text{false}_n \wedge$ *pure-eventual* $\xi \vee$ *suspendable* ξ)
 | *suspendable* (φ *W*_{*n*} ξ) = (*suspendable* φ \wedge *suspendable* $\xi \vee$ *pure-eventual* $\varphi \wedge \xi = \text{false}_n$)
 | *suspendable* (φ *M*_{*n*} ξ) = (*suspendable* φ \wedge *suspendable* $\xi \vee$ *pure-universal* $\varphi \wedge \xi = \text{true}_n$)
 | *suspendable* (*X*_{*n*} ξ) = *suspendable* ξ
 | *suspendable* - = *False*

lemma *pure-eventual-left-append*:

pure-eventual $\mu \Rightarrow w \models_n \mu \Rightarrow (u \frown w) \models_n \mu$
 <proof>

lemma *pure-universal-suffix-closed*:

pure-universal $\nu \Rightarrow (u \frown w) \models_n \nu \Rightarrow w \models_n \nu$
 <proof>

lemma *suspendable-prefix-invariant*:

suspendable $\xi \Rightarrow (u \frown w) \models_n \xi \longleftrightarrow w \models_n \xi$
 <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 $w \models_n \varphi W_n \psi \longleftrightarrow w \models_n \varphi \text{ or}_n \psi$
 ⟨proof⟩

theorem *pure-eventual-strong-release-simp*:

assumes *pure-eventual* φ **and** *pure-eventual* ψ
shows $w \models_n \varphi M_n \psi \longleftrightarrow w \models_n \varphi \text{ and}_n \psi$
 ⟨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 $w \models_n \varphi W_n \psi \longleftrightarrow w \models_n \varphi \text{ or}_n \psi$ (**is** ?t1)
and $w \models_n \varphi M_n \psi \longleftrightarrow w \models_n \varphi \text{ and}_n \psi$ (**is** ?t2)
 ⟨proof⟩

fun *rewrite-modal* :: 'a ltl_n \Rightarrow 'a ltl_n

where

rewrite-modal *true*_n = *true*_n
 | *rewrite-modal* *false*_n = *false*_n
 | *rewrite-modal* ($\varphi \text{ and}_n \psi$) = (*rewrite-modal* $\varphi \text{ and}_n \text{rewrite-modal } \psi$)
 | *rewrite-modal* ($\varphi \text{ or}_n \psi$) = (*rewrite-modal* $\varphi \text{ or}_n \text{rewrite-modal } \psi$)
 | *rewrite-modal* ($\varphi U_n \psi$) = (*if* *pure-eventual* $\psi \vee \text{suspendable } \psi$ *then*
rewrite-modal ψ *else* (*rewrite-modal* $\varphi U_n \text{rewrite-modal } \psi$)
 | *rewrite-modal* ($\varphi R_n \psi$) = (*if* *pure-universal* $\psi \vee \text{suspendable } \psi$ *then*
rewrite-modal ψ *else* (*rewrite-modal* $\varphi R_n \text{rewrite-modal } \psi$)
 | *rewrite-modal* ($\varphi W_n \psi$) = (*if* *pure-universal* $\varphi \wedge \text{pure-universal } \psi \vee$
suspendable $\varphi \wedge \text{suspendable } \psi$ *then* (*rewrite-modal* $\varphi \text{ or}_n \text{rewrite-modal } \psi$)
else (*rewrite-modal* $\varphi W_n \text{rewrite-modal } \psi$)
 | *rewrite-modal* ($\varphi M_n \psi$) = (*if* *pure-eventual* $\varphi \wedge \text{pure-eventual } \psi \vee \text{sus-$
pendable $\varphi \wedge \text{suspendable } \psi$ *then* (*rewrite-modal* $\varphi \text{ and}_n \text{rewrite-modal } \psi$)
else (*rewrite-modal* $\varphi M_n \text{rewrite-modal } \psi$)
 | *rewrite-modal* ($X_n \varphi$) = (*if* *suspendable* φ *then* *rewrite-modal* φ *else* X_n

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

lemma *rewrite-modal-sound* [*simp*]:
 $w \models_n \text{rewrite-modal } \varphi \longleftrightarrow w \models_n \varphi$
 $\langle \text{proof} \rangle$

lemma *rewrite-modal-size*:
 $\text{size } (\text{rewrite-modal } \varphi) \leq \text{size } \varphi$
 $\langle \text{proof} \rangle$

2.5 Syntactical Implication Based Simplification

inductive *syntactical-implies* :: 'a *ltn* \Rightarrow 'a *ltn* \Rightarrow bool (- \vdash_s - [*80*, *80*])

where

- $\vdash_s \text{true}_n$
| *false*_{*n*} \vdash_s -
| $\varphi = \varphi \Rightarrow \varphi \vdash_s \varphi$

| $\varphi \vdash_s \chi \Rightarrow (\varphi \text{and}_n \psi) \vdash_s \chi$
| $\psi \vdash_s \chi \Rightarrow (\varphi \text{and}_n \psi) \vdash_s \chi$
| $\varphi \vdash_s \psi \Rightarrow \varphi \vdash_s \chi \Rightarrow \varphi \vdash_s (\psi \text{and}_n \chi)$

| $\varphi \vdash_s \psi \Rightarrow \varphi \vdash_s (\psi \text{or}_n \chi)$
| $\varphi \vdash_s \chi \Rightarrow \varphi \vdash_s (\psi \text{or}_n \chi)$
| $\varphi \vdash_s \chi \Rightarrow \psi \vdash_s \chi \Rightarrow (\varphi \text{or}_n \psi) \vdash_s \chi$

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

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

| (*false*_{*n*} *R*_{*n*} φ) $\vdash_s \psi \Rightarrow (\text{false}_n R_n \varphi) \vdash_s X_n \psi$
| $\varphi \vdash_s (\text{true}_n U_n \psi) \Rightarrow (X_n \varphi) \vdash_s (\text{true}_n U_n \psi)$
| $\varphi \vdash_s \psi \Rightarrow (X_n \varphi) \vdash_s (X_n \psi)$

lemma *syntactical-implies-correct*:
 $\varphi \vdash_s \psi \Rightarrow w \models_n \varphi \Rightarrow w \models_n \psi$
 $\langle \text{proof} \rangle$

fun *rewrite-syn-imp*

where

```
rewrite-syn-imp ( $\varphi$  andn  $\psi$ ) = (  
  if  $\varphi \vdash_s \psi$  then  
    rewrite-syn-imp  $\varphi$   
  else if  $\psi \vdash_s \varphi$  then  
    rewrite-syn-imp  $\psi$   
  else if  $\varphi \vdash_s (\text{not}_n \psi) \vee \psi \vdash_s (\text{not}_n \varphi)$  then  
    falsen  
  else  
    mk-and (rewrite-syn-imp  $\varphi$ ) (rewrite-syn-imp  $\psi$ )  
| rewrite-syn-imp ( $\varphi$  orn  $\psi$ ) = (  
  if  $\varphi \vdash_s \psi$  then  
    rewrite-syn-imp  $\psi$   
  else if  $\psi \vdash_s \varphi$  then  
    rewrite-syn-imp  $\varphi$   
  else if  $(\text{not}_n \varphi) \vdash_s \psi \vee (\text{not}_n \psi) \vdash_s \varphi$  then  
    truen  
  else  
    mk-or (rewrite-syn-imp  $\varphi$ ) (rewrite-syn-imp  $\psi$ )  
| rewrite-syn-imp ( $\varphi$  Un  $\psi$ ) = (  
  if  $\varphi \vdash_s \psi$  then  
    rewrite-syn-imp  $\psi$   
  else if  $(\text{not}_n \varphi) \vdash_s \psi$  then  
    mk-finally (rewrite-syn-imp  $\psi$ )  
  else  
    mk-until (rewrite-syn-imp  $\varphi$ ) (rewrite-syn-imp  $\psi$ )  
| rewrite-syn-imp ( $\varphi$  Rn  $\psi$ ) = (  
  if  $\psi \vdash_s \varphi$  then  
    rewrite-syn-imp  $\psi$   
  else if  $\psi \vdash_s (\text{not}_n \varphi)$  then  
    mk-globally (rewrite-syn-imp  $\psi$ )  
  else  
    mk-release (rewrite-syn-imp  $\varphi$ ) (rewrite-syn-imp  $\psi$ )  
| rewrite-syn-imp ( $X_n \varphi$ ) = mk-next (rewrite-syn-imp  $\varphi$ )  
| rewrite-syn-imp  $\varphi$  =  $\varphi$ 
```

lemma *rewrite-syn-imp-sound*:

```
 $w \models_n \text{rewrite-syn-imp } \varphi \longleftrightarrow w \models_n \varphi$   
<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

$rewrite-iter-fast\ \varphi \equiv iterate\ (rewrite-modal\ o\ rewrite-X)\ \varphi\ (size\ \varphi)$

definition

$rewrite-iter-slow\ \varphi \equiv iterate\ (rewrite-syn-imp\ o\ rewrite-modal\ o\ rewrite-X)\ \varphi\ (size\ \varphi)$

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 $w \models_n iterate\ f\ \varphi\ n \longleftrightarrow w \models_n \varphi$
 $\langle proof \rangle$

theorem *rewrite-iter-fast-sound* [*simp*]:

$w \models_n rewrite-iter-fast\ \varphi \longleftrightarrow w \models_n \varphi$
 $\langle proof \rangle$

theorem *rewrite-iter-slow-sound* [*simp*]:

$w \models_n rewrite-iter-slow\ \varphi \longleftrightarrow w \models_n \varphi$
 $\langle proof \rangle$

2.7 Preservation of atoms

lemma *iterate-atoms*:

assumes
 $\bigwedge\varphi. atoms-ltln\ (f\ \varphi) \subseteq atoms-ltln\ \varphi$
shows
 $atoms-ltln\ (iterate\ f\ \varphi\ n) \subseteq atoms-ltln\ \varphi$
 $\langle proof \rangle$

lemma *rewrite-modal-atoms*:

$atoms-ltln\ (rewrite-modal\ \varphi) \subseteq atoms-ltln\ \varphi$
 $\langle proof \rangle$

lemma *mk-and-atoms*:

$atoms-ltln\ (mk-and\ \varphi\ \psi) \subseteq atoms-ltln\ \varphi \cup atoms-ltln\ \psi$

$\langle \text{proof} \rangle$

lemma *mk-or-atoms*:

$\text{atoms-ltln} (\text{mk-or } \varphi \ \psi) \subseteq \text{atoms-ltln } \varphi \cup \text{atoms-ltln } \psi$
 $\langle \text{proof} \rangle$

lemma *remove-strong-ops-atoms*:

$\text{atoms-ltln} (\text{remove-strong-ops } \varphi) \subseteq \text{atoms-ltln } \varphi$
 $\langle \text{proof} \rangle$

lemma *remove-weak-ops-atoms*:

$\text{atoms-ltln} (\text{remove-weak-ops } \varphi) \subseteq \text{atoms-ltln } \varphi$
 $\langle \text{proof} \rangle$

lemma *mk-finally-atoms*:

$\text{atoms-ltln} (\text{mk-finally } \varphi) \subseteq \text{atoms-ltln } \varphi$
 $\langle \text{proof} \rangle$

lemma *mk-globally-atoms*:

$\text{atoms-ltln} (\text{mk-globally } \varphi) \subseteq \text{atoms-ltln } \varphi$
 $\langle \text{proof} \rangle$

lemma *mk-until-atoms*:

$\text{atoms-ltln} (\text{mk-until } \varphi \ \psi) \subseteq \text{atoms-ltln } \varphi \cup \text{atoms-ltln } \psi$
 $\langle \text{proof} \rangle$

lemma *mk-release-atoms*:

$\text{atoms-ltln} (\text{mk-release } \varphi \ \psi) \subseteq \text{atoms-ltln } \varphi \cup \text{atoms-ltln } \psi$
 $\langle \text{proof} \rangle$

lemma *mk-weak-until-atoms*:

$\text{atoms-ltln} (\text{mk-weak-until } \varphi \ \psi) \subseteq \text{atoms-ltln } \varphi \cup \text{atoms-ltln } \psi$
 $\langle \text{proof} \rangle$

lemma *mk-strong-release-atoms*:

$\text{atoms-ltln} (\text{mk-strong-release } \varphi \ \psi) \subseteq \text{atoms-ltln } \varphi \cup \text{atoms-ltln } \psi$
 $\langle \text{proof} \rangle$

lemma *mk-next-atoms*:

$\text{atoms-ltln} (\text{mk-next } \varphi) = \text{atoms-ltln } \varphi$
 $\langle \text{proof} \rangle$

lemma *mk-next-pow-atoms*:

$\text{atoms-ltln} (\text{mk-next-pow } n \ \varphi) = \text{atoms-ltln } \varphi$

$\langle \text{proof} \rangle$

lemma *combine-atoms*:

assumes

$\bigwedge \varphi \psi. \text{atoms-ltln } (f \varphi \psi) \subseteq \text{atoms-ltln } \varphi \cup \text{atoms-ltln } \psi$

shows

$\text{atoms-ltln } (\text{fst } (\text{combine } f \ x \ y)) \subseteq \text{atoms-ltln } (\text{fst } x) \cup \text{atoms-ltln } (\text{fst } y)$

$\langle \text{proof} \rangle$

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*:

$\text{atoms-ltln } (\text{fst } (\text{rewrite-X-enat } \varphi)) \subseteq \text{atoms-ltln } \varphi$

$\langle \text{proof} \rangle$

lemma *rewrite-X-atoms*:

$\text{atoms-ltln } (\text{rewrite-X } \varphi) \subseteq \text{atoms-ltln } \varphi$

$\langle \text{proof} \rangle$

lemma *rewrite-syn-imp-atoms*:

$\text{atoms-ltln } (\text{rewrite-syn-imp } \varphi) \subseteq \text{atoms-ltln } \varphi$

$\langle \text{proof} \rangle$

lemma *rewrite-iter-fast-atoms*:

$\text{atoms-ltln } (\text{rewrite-iter-fast } \varphi) \subseteq \text{atoms-ltln } \varphi$

$\langle \text{proof} \rangle$

lemma *rewrite-iter-slow-atoms*:

$\text{atoms-ltln } (\text{rewrite-iter-slow } \varphi) \subseteq \text{atoms-ltln } \varphi$

$\langle \text{proof} \rangle$

2.8 Simplifier

We now define a convenience wrapper for the rewriting engine

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

fun *simplify* :: *mode* \Rightarrow 'a *ltln* \Rightarrow 'a *ltln*

where

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

theorem *simplify-correct*:

$w \models_n \text{simplify } m \ \varphi \longleftrightarrow w \models_n \varphi$
<proof>

lemma *simplify-atoms*:

$\text{atoms-ltln } (\text{simplify } m \ \varphi) \subseteq \text{atoms-ltln } \varphi$
<proof>

2.9 Code Generation

code-pred *syntactical-implies* *<proof>*

export-code *simplify* **checking**

lemma *rewrite-iter-fast* $(F_n (G_n (X_n \text{prop}_n("a")))) = (F_n (G_n \text{prop}_n("a")))$
<proof>

lemma *rewrite-iter-fast* $(X_n \text{prop}_n("a") \ U_n (X_n \text{nprop}_n("a"))) = X_n (\text{prop}_n("a") \ U_n \text{nprop}_n("a"))$
<proof>

lemma *rewrite-iter-slow* $(X_n \text{prop}_n("a") \ U_n (X_n \text{nprop}_n("a"))) = X_n (F_n \text{nprop}_n("a"))$
<proof>

end

3 Equivalence Relations for LTL formulas

theory *Equivalence-Relations*

imports

LTL

begin

3.1 Language Equivalence

definition *ltl-lang-equiv* :: $'a \ \text{ltln} \Rightarrow 'a \ \text{ltln} \Rightarrow \text{bool}$ (**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*:

equivp (\sim_L)
 $\langle \text{proof} \rangle$

lemma *ltl-lang-equiv-and-true[simp]*:

$\varphi_1 \text{ and}_n \varphi_2 \sim_L \text{true}_n \iff \varphi_1 \sim_L \text{true}_n \wedge \varphi_2 \sim_L \text{true}_n$
 $\langle \text{proof} \rangle$

lemma *ltl-lang-equiv-and-false[intro, simp]*:

$\varphi_1 \sim_L \text{false}_n \implies \varphi_1 \text{ and}_n \varphi_2 \sim_L \text{false}_n$
 $\varphi_2 \sim_L \text{false}_n \implies \varphi_1 \text{ and}_n \varphi_2 \sim_L \text{false}_n$
 $\langle \text{proof} \rangle$

lemma *ltl-lang-equiv-or-false[simp]*:

$\varphi_1 \text{ or}_n \varphi_2 \sim_L \text{false}_n \iff \varphi_1 \sim_L \text{false}_n \wedge \varphi_2 \sim_L \text{false}_n$
 $\langle \text{proof} \rangle$

lemma *ltl-lang-equiv-or-const[intro, simp]*:

$\varphi_1 \sim_L \text{true}_n \implies \varphi_1 \text{ or}_n \varphi_2 \sim_L \text{true}_n$
 $\varphi_2 \sim_L \text{true}_n \implies \varphi_1 \text{ or}_n \varphi_2 \sim_L \text{true}_n$
 $\langle \text{proof} \rangle$

3.2 Propositional Equivalence

fun *ltl-prop-entailment* :: 'a ltl_n set \Rightarrow 'a ltl_n \Rightarrow bool (**infix** \models_P 80)

where

$\mathcal{A} \models_P \text{true}_n = \text{True}$
 $\mathcal{A} \models_P \text{false}_n = \text{False}$
 $\mathcal{A} \models_P \varphi \text{ and}_n \psi = (\mathcal{A} \models_P \varphi \wedge \mathcal{A} \models_P \psi)$
 $\mathcal{A} \models_P \varphi \text{ or}_n \psi = (\mathcal{A} \models_P \varphi \vee \mathcal{A} \models_P \psi)$
 $\mathcal{A} \models_P \varphi = (\varphi \in \mathcal{A})$

lemma *ltl-prop-entailment-monotonI[intro]*:

$S \models_P \varphi \implies S \subseteq S' \implies S' \models_P \varphi$
 $\langle \text{proof} \rangle$

lemma *ltl-models-equiv-prop-entailment*:

$w \models_n \varphi \iff \{\psi. w \models_n \psi\} \models_P \varphi$
 $\langle \text{proof} \rangle$

definition *ltl-prop-equiv* :: 'a ltl_n \Rightarrow 'a ltl_n \Rightarrow bool (**infix** \sim_P 75)

where

$\varphi \sim_P \psi \equiv \forall \mathcal{A}. \mathcal{A} \models_P \varphi \iff \mathcal{A} \models_P \psi$

definition *ltl-prop-implies* :: 'a ltl_n ⇒ 'a ltl_n ⇒ bool (**infix** \longrightarrow_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*:

$$\text{equivp } (\sim_P)$$

<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*:

$$\varphi \sim_P \text{true}_n \longleftrightarrow \{\} \models_P \varphi$$

<proof>

lemma *ltl-prop-equiv-false*:

$$\varphi \sim_P \text{false}_n \longleftrightarrow \neg \text{UNIV} \models_P \varphi$$

<proof>

lemma *ltl-prop-equiv-true-implies-true*:

$$x \sim_P \text{true}_n \Longrightarrow x \longrightarrow_P y \Longrightarrow y \sim_P \text{true}_n$$

<proof>

lemma *ltl-prop-equiv-false-implied-by-false*:

$$y \sim_P \text{false}_n \Longrightarrow x \longrightarrow_P y \Longrightarrow x \sim_P \text{false}_n$$

<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 $ttl = Yes \mid No \mid Maybe$

definition $eval\text{-}and :: ttl \Rightarrow ttl \Rightarrow ttl$

where

$eval\text{-}and \ \varphi \ \psi =$
 (case (φ, ψ) of
 (Yes, Yes) \Rightarrow Yes
 | (No, -) \Rightarrow No
 | (-, No) \Rightarrow No
 | - \Rightarrow Maybe)

definition $eval\text{-}or :: ttl \Rightarrow ttl \Rightarrow ttl$

where

$eval\text{-}or \ \varphi \ \psi =$
 (case (φ, ψ) of
 (No, No) \Rightarrow No
 | (Yes, -) \Rightarrow Yes
 | (-, Yes) \Rightarrow Yes
 | - \Rightarrow Maybe)

fun $eval :: 'a \text{ ltn} \Rightarrow ttl$

where

$eval \ true_n = Yes$
| $eval \ false_n = No$
| $eval \ (\varphi \text{ and}_n \ \psi) = eval\text{-}and \ (eval \ \varphi) \ (eval \ \psi)$
| $eval \ (\varphi \text{ or}_n \ \psi) = eval\text{-}or \ (eval \ \varphi) \ (eval \ \psi)$
| $eval \ \varphi = Maybe$

lemma $eval\text{-}and\text{-}const[simp]$:

$eval\text{-}and \ \varphi \ \psi = No \longleftrightarrow \varphi = No \vee \psi = No$
 $eval\text{-}and \ \varphi \ \psi = Yes \longleftrightarrow \varphi = Yes \wedge \psi = Yes$
<proof>

lemma $eval\text{-}or\text{-}const[simp]$:

$eval\text{-}or \ \varphi \ \psi = Yes \longleftrightarrow \varphi = Yes \vee \psi = Yes$
 $eval\text{-}or \ \varphi \ \psi = No \longleftrightarrow \varphi = No \wedge \psi = No$
<proof>

lemma $eval\text{-}prop\text{-}entailment$:

$eval \ \varphi = Yes \longleftrightarrow \{\} \models_P \varphi$
 $eval \ \varphi = No \longleftrightarrow \neg \text{UNIV} \models_P \varphi$
<proof>

definition *ltl-const-equiv* :: 'a ltl_n ⇒ 'a ltl_n ⇒ bool (**infix** \sim_C 75)

where

$$\varphi \sim_C \psi \equiv \varphi = \psi \vee (\text{eval } \varphi = \text{eval } \psi \wedge \text{eval } \psi \neq \text{Maybe})$$

lemma *ltl-const-equiv-equivp*:

$$\text{equivp } (\sim_C)$$

<proof>

lemma *ltl-const-equiv-const*:

$$\varphi \sim_C \text{true}_n \iff \text{eval } \varphi = \text{Yes}$$

$$\varphi \sim_C \text{false}_n \iff \text{eval } \varphi = \text{No}$$

<proof>

lemma *ltl-const-equiv-and-const[simp]*:

$$\varphi_1 \text{ and}_n \varphi_2 \sim_C \text{true}_n \iff \varphi_1 \sim_C \text{true}_n \wedge \varphi_2 \sim_C \text{true}_n$$

$$\varphi_1 \text{ and}_n \varphi_2 \sim_C \text{false}_n \iff \varphi_1 \sim_C \text{false}_n \vee \varphi_2 \sim_C \text{false}_n$$

<proof>

lemma *ltl-const-equiv-or-const[simp]*:

$$\varphi_1 \text{ or}_n \varphi_2 \sim_C \text{true}_n \iff \varphi_1 \sim_C \text{true}_n \vee \varphi_2 \sim_C \text{true}_n$$

$$\varphi_1 \text{ or}_n \varphi_2 \sim_C \text{false}_n \iff \varphi_1 \sim_C \text{false}_n \wedge \varphi_2 \sim_C \text{false}_n$$

<proof>

lemma *ltl-const-equiv-other[simp]*:

$$\varphi \sim_C \text{prop}_n(a) \iff \varphi = \text{prop}_n(a)$$

$$\varphi \sim_C \text{nprop}_n(a) \iff \varphi = \text{nprop}_n(a)$$

$$\varphi \sim_C X_n \psi \iff \varphi = X_n \psi$$

$$\varphi \sim_C \psi_1 U_n \psi_2 \iff \varphi = \psi_1 U_n \psi_2$$

$$\varphi \sim_C \psi_1 R_n \psi_2 \iff \varphi = \psi_1 R_n \psi_2$$

$$\varphi \sim_C \psi_1 W_n \psi_2 \iff \varphi = \psi_1 W_n \psi_2$$

$$\varphi \sim_C \psi_1 M_n \psi_2 \iff \varphi = \psi_1 M_n \psi_2$$

<proof>

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

$$\text{eval } \psi = \text{Maybe} \implies \varphi \sim_C \psi \implies \varphi = \psi$$

<proof>

lemma *ltl-const-equiv-implies-prop-equiv*:

$$\varphi \sim_C \text{true}_n \iff \varphi \sim_P \text{true}_n$$

$$\varphi \sim_C \text{false}_n \iff \varphi \sim_P \text{false}_n$$

<proof>

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

eval $\psi = \text{Maybe} \implies \varphi \sim_C \psi \implies \varphi \sim_P \psi$
<proof>

lemma *ltl-const-equiv-implies-ltl-prop-equiv*:
 $\varphi \sim_C \psi \implies \varphi \sim_P \psi$
<proof>

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

3.4 Quotient types

quotient-type $'a \text{ ltl}_L = 'a \text{ ltl} / (\sim_L)$
<proof>

instantiation *ltl_L* :: (type) equal
begin

lift-definition *ltl_L-eq-test* :: $'a \text{ ltl}_L \Rightarrow 'a \text{ ltl}_L \Rightarrow \text{bool}$ **is** $\lambda x y. x \sim_L y$
<proof>

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

instance
<proof>

end

quotient-type $'a \text{ ltl}_P = 'a \text{ ltl} / (\sim_P)$
<proof>

instantiation *ltl_P* :: (type) equal
begin

lift-definition *ltl_P-eq-test* :: $'a \text{ ltl}_P \Rightarrow 'a \text{ ltl}_P \Rightarrow \text{bool}$ **is** $\lambda x y. x \sim_P y$
<proof>

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

instance

<proof>

end

quotient-type $'a\ ltl_n_C = 'a\ ltl_n / (\sim_C)$
<proof>

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

lift-definition $ltl_n_C\text{-eq-test} :: 'a\ ltl_n_C \Rightarrow 'a\ ltl_n_C \Rightarrow bool$ **is** $\lambda x\ y. x \sim_C y$
<proof>

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

instance
<proof>

end

3.5 Cardinality of propositional quotient sets

definition $sat\text{-models} :: 'a\ ltl_n_P \Rightarrow 'a\ ltl_n\ set\ set$
where

$sat\text{-models}\ \varphi = \{\mathcal{A}. \mathcal{A} \models_P rep\text{-}ltl_n_P\ \varphi\}$

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

lemma *sat-models-Abs*:
 $\mathcal{A} \in sat\text{-models}\ (abs\text{-}ltl_n_P\ \varphi) = \mathcal{A} \models_P \varphi$
<proof>

lemma *sat-models-inj*:
 $inj\ sat\text{-models}$
<proof>

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

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

fun $nested\text{-}prop\text{-}atoms :: 'a\ ltn \Rightarrow 'a\ ltn\ set$

where

$nested\text{-}prop\text{-}atoms\ true_n = \{\}$
 | $nested\text{-}prop\text{-}atoms\ false_n = \{\}$
 | $nested\text{-}prop\text{-}atoms (\varphi\ and_n\ \psi) = nested\text{-}prop\text{-}atoms\ \varphi \cup nested\text{-}prop\text{-}atoms\ \psi$
 | $nested\text{-}prop\text{-}atoms (\varphi\ or_n\ \psi) = nested\text{-}prop\text{-}atoms\ \varphi \cup nested\text{-}prop\text{-}atoms\ \psi$
 | $nested\text{-}prop\text{-}atoms (X_n\ \varphi) = \{X_n\ \varphi\} \cup nested\text{-}prop\text{-}atoms\ \varphi$
 | $nested\text{-}prop\text{-}atoms (\varphi\ U_n\ \psi) = \{\varphi\ U_n\ \psi\} \cup nested\text{-}prop\text{-}atoms\ \varphi \cup nested\text{-}prop\text{-}atoms\ \psi$
 | $nested\text{-}prop\text{-}atoms (\varphi\ R_n\ \psi) = \{\varphi\ R_n\ \psi\} \cup nested\text{-}prop\text{-}atoms\ \varphi \cup nested\text{-}prop\text{-}atoms\ \psi$
 | $nested\text{-}prop\text{-}atoms (\varphi\ W_n\ \psi) = \{\varphi\ W_n\ \psi\} \cup nested\text{-}prop\text{-}atoms\ \varphi \cup nested\text{-}prop\text{-}atoms\ \psi$
 | $nested\text{-}prop\text{-}atoms (\varphi\ M_n\ \psi) = \{\varphi\ M_n\ \psi\} \cup nested\text{-}prop\text{-}atoms\ \varphi \cup nested\text{-}prop\text{-}atoms\ \psi$
 | $nested\text{-}prop\text{-}atoms\ \varphi = \{\varphi\}$

lemma $prop\text{-}atoms\text{-}nested\text{-}prop\text{-}atoms$:

$prop\text{-}atoms\ \varphi \subseteq nested\text{-}prop\text{-}atoms\ \varphi$
 <proof>

lemma $prop\text{-}atoms\text{-}subfrmlsn$:

$prop\text{-}atoms\ \varphi \subseteq subfrmlsn\ \varphi$
 <proof>

lemma $nested\text{-}prop\text{-}atoms\text{-}subfrmlsn$:

$nested\text{-}prop\text{-}atoms\ \varphi \subseteq subfrmlsn\ \varphi$
 <proof>

lemma $prop\text{-}atoms\text{-}notin[simp]$:

$true_n \notin prop\text{-}atoms\ \varphi$
 $false_n \notin prop\text{-}atoms\ \varphi$
 $\varphi_1\ and_n\ \varphi_2 \notin prop\text{-}atoms\ \varphi$
 $\varphi_1\ or_n\ \varphi_2 \notin prop\text{-}atoms\ \varphi$
 <proof>

lemma $nested\text{-}prop\text{-}atoms\text{-}notin[simp]$:

$true_n \notin \text{nested-prop-atoms } \varphi$
 $false_n \notin \text{nested-prop-atoms } \varphi$
 $\varphi_1 \text{ and}_n \varphi_2 \notin \text{nested-prop-atoms } \varphi$
 $\varphi_1 \text{ or}_n \varphi_2 \notin \text{nested-prop-atoms } \varphi$
 ⟨proof⟩

lemma *prop-atoms-finite*:

$finite (\text{prop-atoms } \varphi)$
 ⟨proof⟩

lemma *nested-prop-atoms-finite*:

$finite (\text{nested-prop-atoms } \varphi)$
 ⟨proof⟩

lemma *prop-atoms-entailment-iff*:

$\varphi \in \text{prop-atoms } \psi \implies \mathcal{A} \models_P \varphi \iff \varphi \in \mathcal{A}$
 ⟨proof⟩

lemma *prop-atoms-entailment-inter*:

$\text{prop-atoms } \varphi \subseteq P \implies (\mathcal{A} \cap P) \models_P \varphi = \mathcal{A} \models_P \varphi$
 ⟨proof⟩

lemma *nested-prop-atoms-entailment-inter*:

$\text{nested-prop-atoms } \varphi \subseteq P \implies (\mathcal{A} \cap P) \models_P \varphi = \mathcal{A} \models_P \varphi$
 ⟨proof⟩

lemma *sat-models-inter-inj-helper*:

assumes

$\text{prop-atoms } \varphi \subseteq P$

and

$\text{prop-atoms } \psi \subseteq P$

and

$\text{sat-models } (\text{abs-ltln}_P \varphi) \cap \text{Pow } P = \text{sat-models } (\text{abs-ltln}_P \psi) \cap \text{Pow } P$

shows

$\varphi \sim_P \psi$

⟨proof⟩

lemma *sat-models-inter-inj*:

$\text{inj-on } (\lambda\varphi. \text{sat-models } \varphi \cap \text{Pow } P) \{ \text{abs-ltln}_P \varphi \mid \varphi. \text{prop-atoms } \varphi \subseteq P \}$
 ⟨proof⟩

lemma *sat-models-pow-pow*:

$\{ \text{sat-models } (\text{abs-ltln}_P \varphi) \cap \text{Pow } P \mid \varphi. \text{prop-atoms } \varphi \subseteq P \} \subseteq \text{Pow } (\text{Pow } P)$

$\langle \text{proof} \rangle$

lemma *sat-models-finite*:

$\text{finite } P \implies \text{finite } \{\text{sat-models } (\text{abs-ltln}_P \varphi) \cap \text{Pow } P \mid \varphi. \text{prop-atoms } \varphi \subseteq P\}$

$\langle \text{proof} \rangle$

lemma *sat-models-card*:

$\text{finite } P \implies \text{card } (\{\text{sat-models } (\text{abs-ltln}_P \varphi) \cap \text{Pow } P \mid \varphi. \text{prop-atoms } \varphi \subseteq P\}) \leq 2 \wedge 2 \wedge \text{card } P$

$\langle \text{proof} \rangle$

lemma *image-filter*:

$f \text{ ` } \{g \ a \mid a. P \ a\} = \{f \ (g \ a) \mid a. P \ a\}$

$\langle \text{proof} \rangle$

lemma *prop-equiv-finite*:

$\text{finite } P \implies \text{finite } \{\text{abs-ltln}_P \ \psi \mid \psi. \text{prop-atoms } \psi \subseteq P\}$

$\langle \text{proof} \rangle$

lemma *prop-equiv-card*:

$\text{finite } P \implies \text{card } \{\text{abs-ltln}_P \ \psi \mid \psi. \text{prop-atoms } \psi \subseteq P\} \leq 2 \wedge 2 \wedge \text{card } P$

$\langle \text{proof} \rangle$

lemma *prop-equiv-subset*:

$\{\text{abs-ltln}_P \ \psi \mid \psi. \text{nested-prop-atoms } \psi \subseteq P\} \subseteq \{\text{abs-ltln}_P \ \psi \mid \psi. \text{prop-atoms } \psi \subseteq P\}$

$\langle \text{proof} \rangle$

lemma *prop-equiv-finite'*:

$\text{finite } P \implies \text{finite } \{\text{abs-ltln}_P \ \psi \mid \psi. \text{nested-prop-atoms } \psi \subseteq P\}$

$\langle \text{proof} \rangle$

lemma *prop-equiv-card'*:

$\text{finite } P \implies \text{card } \{\text{abs-ltln}_P \ \psi \mid \psi. \text{nested-prop-atoms } \psi \subseteq P\} \leq 2 \wedge 2 \wedge \text{card } P$

$\langle \text{proof} \rangle$

3.6 Substitution

fun *subst* :: 'a ltltn \Rightarrow ('a ltltn \rightarrow 'a ltltn) \Rightarrow 'a ltltn

where

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

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

lemma *ltl-prop-equiv-subst-S*:

$S \models_P subst\ \varphi\ m = ((S - dom\ m) \cup \{\chi \mid \chi\ \chi'.\ \chi \in dom\ m \wedge m\ \chi = Some\ \chi' \wedge S \models_P \chi'\}) \models_P \varphi$
 $\langle proof \rangle$

lemma *subst-respects-ltl-prop-entailment*:

$\varphi \longrightarrow_P \psi \implies subst\ \varphi\ m \longrightarrow_P subst\ \psi\ m$
 $\varphi \sim_P \psi \implies subst\ \varphi\ m \sim_P subst\ \psi\ m$
 $\langle proof \rangle$

lemma *eval-subst*:

$eval\ \varphi = Yes \implies eval\ (subst\ \varphi\ m) = Yes$
 $eval\ \varphi = No \implies eval\ (subst\ \varphi\ m) = No$
 $\langle proof \rangle$

lemma *subst-respects-ltl-const-entailment*:

$\varphi \sim_C \psi \implies subst\ \varphi\ m \sim_C subst\ \psi\ m$
 $\langle proof \rangle$

3.7 Order of Equivalence Relations

locale *ltl-equivalence* =

fixes

$eq :: 'a\ ltl_n \Rightarrow 'a\ ltl_n \Rightarrow bool$ (**infix** \sim 75)

assumes

$eq_equivp: equivp\ (\sim)$

and

$ge_const_equiv: (\sim_C) \leq (\sim)$

and

$le_lang_equiv: (\sim) \leq (\sim_L)$

begin

lemma *eq-implies-ltl-equiv*:

$\varphi \sim \psi \implies w \models_n \varphi = w \models_n \psi$
 $\langle proof \rangle$

lemma *const-implies-eq*:

$\varphi \sim_C \psi \implies \varphi \sim \psi$
<proof>

lemma *eq-implies-lang*:

$\varphi \sim \psi \implies \varphi \sim_L \psi$
<proof>

lemma *eq-refl[simp]*:

$\varphi \sim \varphi$
<proof>

lemma *eq-sym[sym]*:

$\varphi \sim \psi \implies \psi \sim \varphi$
<proof>

lemma *eq-trans[trans]*:

$\varphi \sim \psi \implies \psi \sim \chi \implies \varphi \sim \chi$
<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 \Rightarrow 'a set **where**
 $\text{min-set } X = \{y \in X. \forall x \in X. x \leq y \longrightarrow x = y\}$

lemma *min-set-iff*:

$x \in \text{min-set } X \longleftrightarrow x \in X \wedge (\forall y \in X. y \leq x \longrightarrow y = x)$
<proof>

lemma *min-set-subset*:

$\text{min-set } X \subseteq X$
<proof>

lemma *min-set-idem*[*simp*]:

$\text{min-set } (\text{min-set } X) = \text{min-set } X$
<proof>

lemma *min-set-empty*[*simp*]:

$\text{min-set } \{\} = \{\}$
<proof>

lemma *min-set-singleton*[*simp*]:

$\text{min-set } \{x\} = \{x\}$
<proof>

lemma *min-set-finite*:

$\text{finite } X \Longrightarrow \text{finite } (\text{min-set } X)$
<proof>

lemma *min-set-obtains-helper*:

$A \in B \Longrightarrow \exists C. C \sqsubseteq A \wedge C \in \text{min-set } B$
<proof>

lemma *min-set-obtains*:

assumes $A \in B$
obtains C **where** $C \sqsubseteq A$ **and** $C \in \text{min-set } B$
<proof>

4.2 Minimal operators on sets

definition *product* :: 'a fset set \Rightarrow 'a fset set \Rightarrow 'a fset set (**infixr** \otimes 65)
where $A \otimes B = \{a \mid\cup\mid b \mid a \ b. a \in A \wedge b \in B\}$

definition *min-product* :: 'a fset set \Rightarrow 'a fset set \Rightarrow 'a fset set (**infixr** \otimes_m 65)

where $A \otimes_m B = \text{min-set } (A \otimes B)$

definition *min-union* :: 'a fset set \Rightarrow 'a fset set \Rightarrow 'a fset set (**infixr** \cup_m 65)

where $A \cup_m B = \text{min-set } (A \cup B)$

definition *product-set* :: 'a fset set set \Rightarrow 'a fset set (\otimes)

where $\otimes X = \text{Finite-Set.fold product } \{\{\|\|\}\} X$

definition *min-product-set* :: 'a fset set set \Rightarrow 'a fset set (\otimes_m)

where $\otimes_m X = \text{Finite-Set.fold min-product } \{\{\|\|\}\} X$

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

$A \otimes_m A = \text{min-set } A$

<proof>

lemma *min-union-idem[simp]*:

$A \cup_m A = \text{min-set } A$

<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]*:

$A \cup_m \{\} = \text{min-set } A$

$\{\} \cup_m A = \text{min-set } A$

<proof>

lemma *product-empty-singleton[simp]*:

$A \otimes \{\{\|\|\}\} = A$

$\{\{\|\|\}\} \otimes A = A$

<proof>

lemma *min-product-empty-singleton[simp]*:

$$A \otimes_m \{\{\|\}\} = \text{min-set } A$$

$$\{\{\|\}\} \otimes_m A = \text{min-set } A$$

<proof>

lemma *product-singleton-singleton*:

$$A \otimes \{\{|x|\}\} = \text{finsert } x \text{ ' } A$$

$$\{\{|x|\}\} \otimes A = \text{finsert } x \text{ ' } A$$

<proof>

lemma *product-mono*:

$$A \subseteq B \implies A \otimes C \subseteq B \otimes C$$

$$B \subseteq C \implies A \otimes B \subseteq A \otimes C$$

<proof>

lemma *product-finite*:

$$\text{finite } A \implies \text{finite } B \implies \text{finite } (A \otimes B)$$

<proof>

lemma *min-product-finite*:

$$\text{finite } A \implies \text{finite } B \implies \text{finite } (A \otimes_m B)$$

<proof>

lemma *min-union-finite*:

$$\text{finite } A \implies \text{finite } B \implies \text{finite } (A \cup_m B)$$

<proof>

lemma *product-set-infinite[simp]*:

$$\text{infinite } X \implies \bigotimes X = \{\{\|\}\}$$

<proof>

lemma *min-product-set-infinite[simp]*:

$$\text{infinite } X \implies \bigotimes_m X = \{\{\|\}\}$$

<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*:

$$x \in A \otimes B \iff (\exists a \in A. \exists b \in B. x = a \mid \cup \mid b)$$

<proof>

lemma *min-product-iff*:

$$x \in A \otimes_m B \iff (\exists a \in A. \exists b \in B. x = a \mid \cup \mid b) \wedge (\forall a \in A. \forall b \in B. a \mid \cup \mid b \mid \subseteq \mid x \longrightarrow a \mid \cup \mid b = x)$$

<proof>

lemma *min-union-iff*:

$$x \in A \cup_m B \iff x \in A \cup B \wedge (\forall a \in A. a \mid \subseteq \mid x \longrightarrow a = x) \wedge (\forall b \in B. b \mid \subseteq \mid x \longrightarrow b = x)$$

<proof>

lemma *min-set-min-product-helper*:

$$x \in (\text{min-set } A) \otimes_m B \iff x \in A \otimes_m B$$

<proof>

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

$$(\text{min-set } A) \otimes_m B = A \otimes_m B$$

$$A \otimes_m (\text{min-set } B) = A \otimes_m B$$

<proof>

lemma *min-set-min-union[simp]*:

$$(\text{min-set } A) \cup_m B = A \cup_m B$$

$$A \cup_m (\text{min-set } B) = A \cup_m B$$

<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*:

$$a \in A \implies b \in B \implies \exists c. c \mid\subseteq (a \mid\cup b) \wedge c \in A \otimes_m B$$

<proof>

lemma *min-union-comp*:

$$a \in A \implies \exists c. c \mid\subseteq a \wedge c \in A \cup_m B$$

<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]*:

$$\begin{aligned} \otimes \{\} &= \{\{\|\}\} \\ \otimes \{\{\}\} &= \{\} \\ \otimes \{\{\{\|\}\}\} &= \{\{\|\}\} \end{aligned}$$

<proof>

lemma *min-product-set-empty[simp]*:

$$\begin{aligned} \otimes_m \{\} &= \{\{\|\}\} \\ \otimes_m \{\{\}\} &= \{\} \\ \otimes_m \{\{\{\|\}\}\} &= \{\{\|\}\} \end{aligned}$$

<proof>

lemma *product-set-code[code]*:

$$\otimes (\text{set } xs) = \text{fold product (remdups } xs) \{\{\|\}\}$$

<proof>

lemma *min-product-set-code*[code]:

$$\bigotimes_m (\text{set } xs) = \text{fold } \text{min-product } (\text{remdups } xs) \{\{\|\|\}\}$$

<proof>

lemma *product-set-insert*[simp]:

$$\text{finite } X \implies \bigotimes (\text{insert } x X) = x \otimes (\bigotimes (X - \{x\}))$$

<proof>

lemma *min-product-set-insert*[simp]:

$$\text{finite } X \implies \bigotimes_m (\text{insert } x X) = x \otimes_m (\bigotimes_m X)$$

<proof>

lemma *min-product-subseteq*:

$$x \in A \otimes_m B \implies \exists a. a \sqsubseteq x \wedge a \in A$$

<proof>

lemma *min-product-set-subseteq*:

$$\text{finite } X \implies x \in \bigotimes_m X \implies A \in X \implies \exists a \in A. a \sqsubseteq x$$

<proof>

lemma *min-set-product-set*:

$$\bigotimes_m A = \text{min-set } (\bigotimes A)$$

<proof>

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

$$\text{min-set } (A \otimes_m B) = A \otimes_m B$$

<proof>

lemma *min-union-min-set*[simp]:

$$\text{min-set } (A \cup_m B) = A \cup_m B$$

<proof>

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

$$\text{finite } X \implies \text{min-set } (\bigotimes_m X) = \bigotimes_m X$$

<proof>

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

$$\text{finite } X \implies \bigotimes_m (\text{min-set } X) = \bigotimes_m X$$

<proof>

lemma *min-product-set-union*[simp]:

$$\text{finite } X \implies \text{finite } Y \implies \bigotimes_m (X \cup Y) = (\bigotimes_m X) \otimes_m (\bigotimes_m Y)$$

<proof>

lemma *product-set-finite*:

$(\bigwedge x. x \in X \implies \text{finite } x) \implies \text{finite } (\bigotimes X)$
<proof>

lemma *min-product-set-finite*:

$(\bigwedge x. x \in X \implies \text{finite } x) \implies \text{finite } (\bigotimes_m X)$
<proof>

4.3 Disjunctive Normal Form

fun *dnf* :: 'a ltn \Rightarrow 'a ltn fset set

where

dnf true_n = $\{\{\|\|\}\}$
| *dnf false_n* = $\{\}$
| *dnf* (φ and_n ψ) = (*dnf* φ) \otimes (*dnf* ψ)
| *dnf* (φ or_n ψ) = (*dnf* φ) \cup (*dnf* ψ)
| *dnf* φ = $\{\{|\varphi|\}\}$

fun *min-dnf* :: 'a ltn \Rightarrow 'a ltn fset set

where

min-dnf true_n = $\{\{\|\|\}\}$
| *min-dnf false_n* = $\{\}$
| *min-dnf* (φ and_n ψ) = (*min-dnf* φ) \otimes_m (*min-dnf* ψ)
| *min-dnf* (φ or_n ψ) = (*min-dnf* φ) \cup_m (*min-dnf* ψ)
| *min-dnf* φ = $\{\{|\varphi|\}\}$

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]*:

$$fset (Abs-fset (min-dnf \varphi)) = min-dnf \varphi$$

<proof>

lemma *dnf-prop-atoms*:

$$\Phi \in dnf \varphi \implies fset \Phi \subseteq prop-atoms \varphi$$

<proof>

lemma *min-dnf-prop-atoms*:

$$\Phi \in min-dnf \varphi \implies fset \Phi \subseteq prop-atoms \varphi$$

<proof>

lemma *min-dnf-atoms-dnf*:

$$\Phi \in min-dnf \psi \implies \varphi \in fset \Phi \implies dnf \varphi = \{ \{ |\varphi| \} \}$$

<proof>

lemma *min-dnf-min-set[simp]*:

$$min-set (min-dnf \varphi) = min-dnf \varphi$$

<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*:

$$\varphi \longrightarrow_P \psi = (\forall A \in min-dnf \varphi. \exists B \in min-dnf \psi. B \subseteq A)$$

<proof>

lemma *ltl-prop-equiv-min-dnf*:

$$\varphi \sim_P \psi = (min-dnf \varphi = min-dnf \psi)$$

<proof>

lemma *min-dnf-rep-abs[simp]*:

$$min-dnf (rep-ltln_P (abs-ltln_P \varphi)) = min-dnf \varphi$$

<proof>

4.4 Folding of and_n and or_n over Finite Sets

definition $And_n :: 'a\ ltl\ set \Rightarrow 'a\ ltl$

where

$$And_n \Phi \equiv SOME \varphi. fold-graph\ And-ltln\ True-ltln\ \Phi\ \varphi$$

definition $Or_n :: 'a\ ltl\ n\ set \Rightarrow 'a\ ltl\ n$

where

$Or_n\ \Phi \equiv SOME\ \varphi.\ fold_graph\ Or_ltln\ False_ltln\ \Phi\ \varphi$

lemma *fold-graph-And_n*:

$finite\ \Phi \Longrightarrow fold_graph\ And_ltln\ True_ltln\ \Phi\ (And_n\ \Phi)$
<proof>

lemma *fold-graph-Or_n*:

$finite\ \Phi \Longrightarrow fold_graph\ Or_ltln\ False_ltln\ \Phi\ (Or_n\ \Phi)$
<proof>

lemma *Or_n-empty[simp]*:

$Or_n\ \{\} = False_ltln$
<proof>

lemma *And_n-empty[simp]*:

$And_n\ \{\} = True_ltln$
<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 $And_n \{s. \text{finite } s\}$
 ⟨proof⟩

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

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

lemma *And_n-semantics*:
 $\text{finite } \Phi \implies w \models_n And_n \Phi \iff (\forall \varphi \in \Phi. w \models_n \varphi)$
 ⟨proof⟩

lemma *Or_n-semantics*:
 $\text{finite } \Phi \implies w \models_n Or_n \Phi \iff (\exists \varphi \in \Phi. w \models_n \varphi)$
 ⟨proof⟩

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

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

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

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

4.5 DNF to LTL conversion

definition *ltln-of-dnf* :: 'a ltln fset set \Rightarrow 'a ltln
where

$$\text{ltln-of-dnf } \mathcal{A} = Or_n (And_n \text{ ' } \text{fset } \text{ ' } \mathcal{A})$$

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

lemma *ltln-of-dnf-prop-semantics:*

assumes *finite* \mathcal{A}

shows $\mathcal{I} \models_P \text{ltln-of-dnf } \mathcal{A} \longleftrightarrow (\exists \Phi \in \mathcal{A}. \forall \varphi. \varphi \in \Phi \longrightarrow \mathcal{I} \models_P \varphi)$

<proof>

lemma *ltln-of-dnf-prop-equiv:*

$\text{ltln-of-dnf } (\text{min-dnf } \varphi) \sim_P \varphi$

<proof>

lemma *min-dnf-ltn-of-dnf[simp]:*

$\text{min-dnf } (\text{ltln-of-dnf } (\text{min-dnf } \varphi)) = \text{min-dnf } \varphi$

<proof>

4.6 Substitution in DNF formulas

definition *subst-clause* :: $'a \text{ ltn fset} \Rightarrow ('a \text{ ltn} \rightarrow 'a \text{ ltn}) \Rightarrow 'a \text{ ltn fset set}$

where

$\text{subst-clause } \Phi \ m = \bigotimes_m \{ \text{min-dnf } (\text{subst } \varphi \ m) \mid \varphi. \varphi \in \text{fset } \Phi \}$

definition *subst-dnf* :: $'a \text{ ltn fset set} \Rightarrow ('a \text{ ltn} \rightarrow 'a \text{ ltn}) \Rightarrow 'a \text{ ltn fset set}$

where

$\text{subst-dnf } \mathcal{A} \ m = (\bigcup \Phi \in \mathcal{A}. \text{subst-clause } \Phi \ m)$

lemma *subst-clause-empty[simp]:*

$\text{subst-clause } \{\{\}\} \ m = \{\{\{\}\}\}$

<proof>

lemma *subst-dnf-empty[simp]:*

$\text{subst-dnf } \{\} \ m = \{\}$

<proof>

lemma *subst-clause-inner-finite:*

$\text{finite } \{ \text{min-dnf } (\text{subst } \varphi \ m) \mid \varphi. \varphi \in \Phi \}$ **if** $\text{finite } \Phi$

<proof>

lemma *subst-clause-finite:*

$\text{finite } (\text{subst-clause } \Phi \ m)$

<proof>

lemma *subst-dnf-finite:*

$\text{finite } \mathcal{A} \Longrightarrow \text{finite } (\text{subst-dnf } \mathcal{A} \ m)$

<proof>

lemma *subst-dnf-mono*:

$$\mathcal{A} \subseteq \mathcal{B} \implies \text{subst-dnf } \mathcal{A} \ m \subseteq \text{subst-dnf } \mathcal{B} \ m$$

<proof>

lemma *subst-clause-min-set[simp]*:

$$\text{min-set } (\text{subst-clause } \Phi \ m) = \text{subst-clause } \Phi \ m$$

<proof>

lemma *subst-clause-finsert[simp]*:

$$\text{subst-clause } (\text{finsert } \varphi \ \Phi) \ m = (\text{min-dnf } (\text{subst } \varphi \ m)) \otimes_m (\text{subst-clause } \Phi \ m)$$

<proof>

lemma *subst-clause-funion[simp]*:

$$\text{subst-clause } (\Phi \ \cup \ \Psi) \ m = (\text{subst-clause } \Phi \ m) \otimes_m (\text{subst-clause } \Psi \ m)$$

<proof>

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

definition *list-product* :: 'a list set \Rightarrow 'a list set \Rightarrow 'a list set (**infixl** \otimes_l 65)

where

$$A \otimes_l B = \{a \ @ \ b \mid a \ b. \ a \in A \wedge b \in B\}$$

lemma *list-product-fset-of-list[simp]*:

$$\text{fset-of-list } ' (A \otimes_l B) = (\text{fset-of-list } ' A) \otimes (\text{fset-of-list } ' B)$$

<proof>

lemma *list-product-finite*:

$$\text{finite } A \implies \text{finite } B \implies \text{finite } (A \otimes_l B)$$

<proof>

lemma *list-product-iff*:

$$x \in A \otimes_l B \longleftrightarrow (\exists a \ b. \ a \in A \wedge b \in B \wedge x = a \ @ \ b)$$

<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 ltl n \Rightarrow 'a ltl n list set

where

$$\begin{aligned} & \text{list-dnf } \text{true}_n = \{\{\}\} \\ & \text{list-dnf } \text{false}_n = \{\} \\ & \text{list-dnf } (\varphi \ \text{and}_n \ \psi) = (\text{list-dnf } \varphi) \otimes_l (\text{list-dnf } \psi) \end{aligned}$$

| $list-dnf (\varphi \text{ or}_n \psi) = (list-dnf \varphi) \cup (list-dnf \psi)$
| $list-dnf \varphi = \{[\varphi]\}$

definition $list-dnf-to-dnf :: 'a list set \Rightarrow 'a fset set$
where

$list-dnf-to-dnf X = fset-of-list ' X$

lemma $list-dnf-to-dnf-list-dnf[simp]$:
 $list-dnf-to-dnf (list-dnf \varphi) = dnf \varphi$
 $\langle proof \rangle$

lemma $list-dnf-finite$:
 $finite (list-dnf \varphi)$
 $\langle proof \rangle$

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

definition $subst-clause' :: 'a ltn list \Rightarrow ('a ltn \rightarrow 'a ltn) \Rightarrow 'a ltn list set$
where

$subst-clause' \Phi m = fold (\lambda \varphi acc. acc \otimes_l list-dnf (subst \varphi m)) \Phi \{\}\}$

definition $subst-dnf' :: 'a ltn list set \Rightarrow ('a ltn \rightarrow 'a ltn) \Rightarrow 'a ltn list set$
where

$subst-dnf' \mathcal{A} m = (\bigcup \Phi \in \mathcal{A}. subst-clause' \Phi m)$

lemma $subst-clause'-finite$:
 $finite (subst-clause' \Phi m)$
 $\langle proof \rangle$

lemma $subst-clause'-nil[simp]$:
 $subst-clause' [] m = \{\}\}$
 $\langle proof \rangle$

lemma $subst-clause'-cons[simp]$:
 $subst-clause' (xs @ [x]) m = subst-clause' xs m \otimes_l list-dnf (subst x m)$
 $\langle proof \rangle$

lemma $subst-clause'-append[simp]$:
 $subst-clause' (A @ B) m = subst-clause' A m \otimes_l subst-clause' B m$
 $\langle proof \rangle$

lemma $subst-dnf'-iff$:
 $x \in subst-dnf' A m \iff (\exists \Phi \in A. x \in subst-clause' \Phi m)$
 $\langle proof \rangle$

lemma *subst-dnf'-product:*

$subst\text{-}dnf' (A \otimes_l B) m = (subst\text{-}dnf' A m) \otimes_l (subst\text{-}dnf' B m)$ (is ?lhs
= ?rhs)
<proof>

lemma *subst-dnf'-list-dnf:*

$subst\text{-}dnf' (list\text{-}dnf \varphi) m = list\text{-}dnf (subst \varphi m)$
<proof>

lemma *min-set-Union:*

$finite X \implies min\text{-}set (\bigcup (min\text{-}set ' X)) = min\text{-}set (\bigcup X)$ for $X :: 'a fset$
set set
<proof>

lemma *min-set-Union-image:*

$finite X \implies min\text{-}set (\bigcup x \in X. min\text{-}set (f x)) = min\text{-}set (\bigcup x \in X. f x)$
for $f :: 'b \Rightarrow 'a fset$ set
<proof>

lemma *subst-clause-fset-of-list:*

$subst\text{-}clause (fset\text{-}of\text{-}list \Phi) m = min\text{-}set (list\text{-}dnf\text{-}to\text{-}dnf (subst\text{-}clause' \Phi m))$
<proof>

lemma *min-set-list-dnf-to-dnf-subst-dnf':*

$finite X \implies min\text{-}set (list\text{-}dnf\text{-}to\text{-}dnf (subst\text{-}dnf' X m)) = min\text{-}set (subst\text{-}dnf (list\text{-}dnf\text{-}to\text{-}dnf X) m)$
<proof>

lemma *subst-dnf-dnf:*

$min\text{-}set (subst\text{-}dnf (dnf \varphi) m) = min\text{-}dnf (subst \varphi m)$
<proof>

This is almost the lemma we need. However, we need to show that the same holds for $min\text{-}dnf \varphi$, too.

lemma *fold-product:*

$Finite\text{-}Set.fold (\lambda x. (\otimes) \{\{x\}\}) \{\{\}\} (fset x) = \{x\}$
<proof>

lemma *fold-union:*

$Finite\text{-}Set.fold (\lambda x. (\cup) \{x\}) \{\} (fset x) = fset x$
<proof>

lemma *fold-union-fold-product*:

assumes *finite X and* $\bigwedge \Psi \psi. \Psi \in X \implies \psi \in \text{fset } \Psi \implies \text{dnf } \psi = \{\{|\psi|\}\}$
shows *Finite-Set.fold* ($\lambda x. (\cup)$) (*Finite-Set.fold* ($\lambda \varphi. (\otimes)$) (*dnf* φ)) $\{\{\|\}\}$
(*fset x*)) $\{\}$ $X = X$ (**is** *?lhs = X*)
 $\langle \text{proof} \rangle$

lemma *dnf-ltln-of-dnf-min-dnf*:

dnf (*ltln-of-dnf* (*min-dnf* φ)) = *min-dnf* φ
 $\langle \text{proof} \rangle$

lemma *min-dnf-subst*:

min-set (*subst-dnf* (*min-dnf* φ) m) = *min-dnf* (*subst* φ m) (**is** *?lhs = ?rhs*)
 $\langle \text{proof} \rangle$

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* :: $'a \text{ ltl}n \Rightarrow 'a \text{ ltl}n \text{ ifex}$

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) $s = \{x. s\ x\} \models_P b$

$\langle \text{proof} \rangle$

lemma *reduced-ifex*:

```
reduced (ifex-of-ltl b) {}  
<proof>
```

```
lemma ifex-of-ltl-reduced-bdt-checker:  
  reduced-bdt-checkers ifex-of-ltl ( $\lambda y s. \{x. s x\} \models_P y$ )  
<proof>
```

```
lemma ltl-prop-equiv-impl[code]:  
  ( $\varphi \sim_P \psi$ ) = equiv-test ifex-of-ltl  $\varphi \psi$   
<proof>
```

```
lemma ltl-prop-implies-impl[code]:  
  ( $\varphi \longrightarrow_P \psi$ ) = impl-test ifex-of-ltl  $\varphi \psi$   
<proof>
```

```
export-code ( $\sim_P$ ) ( $\longrightarrow_P$ ) 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  $\Rightarrow$  String.literal ltlc  
where
```

```
  rewrite  $\equiv$  ltlc-to-ltlc o rewrite-iter-slow o ltlc-to-ltlc o (map-ltlc ( $\lambda s. \text{String.implode}$   
  "prop(" + s + String.implode ")"))
```

— Export rewriting engine (and also constructors)

```
export-code truec Iff-ltlc rewrite in SML file-prefix <rewrite-example>
```

```
end
```


References

- [1] T. Babiak, M. Kretínský, V. Reháč, 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.