

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	10
1.2.1	Syntax	10
1.2.2	Semantics	11
1.2.3	Conversion	11
1.2.4	Negation	13
1.2.5	Subformulas	14
1.2.6	Constant Folding	15
1.2.7	Distributivity	15
1.2.8	Nested operators	17
1.2.9	Weak and strong operators	18
1.2.10	GF and FG semantics	19
1.2.11	Expansion	21
1.3	LTL in restricted Negation Normal Form	24
1.3.1	Syntax	24
1.3.2	Semantics	24
1.3.3	Conversion	25
1.3.4	Negation	26
1.3.5	Subformulas	26
1.3.6	Expansion lemmas	27

1.4	Propositional LTL	27
1.4.1	Syntax	27
1.4.2	Semantics	28
1.4.3	Conversion	29
1.4.4	Atoms	29
2	Rewrite Rules for LTL Simplification	30
2.1	Constant Eliminating Constructors	30
2.2	Constant Propagation	34
2.3	X-Normalisation	36
2.4	Pure Eventual, Pure Universal, and Suspendable Formulas . .	41
2.5	Syntactical Implication Based Simplification	48
2.6	Iterated Rewriting	51
2.7	Preservation of atoms	52
2.8	Simplifier	55
2.9	Code Generation	56
3	Equivalence Relations for LTL formulas	56
3.1	Language Equivalence	56
3.2	Propositional Equivalence	57
3.3	Constants Equivalence	59
3.4	Quotient types	62
3.5	Cardinality of propositional quotient sets	63
3.6	Substitution	67
3.7	Order of Equivalence Relations	68
4	Disjunctive Normal Form of LTL formulas	69
4.1	Definition of Minimum Sets	70
4.2	Minimal operators on sets	71
4.3	Disjunctive Normal Form	78
4.4	Folding of and_n and or_n over Finite Sets	82
4.5	DNF to LTL conversion	86
4.6	Substitution in DNF formulas	88
5	Code lemmas for abstract definitions	95
5.1	Propositional Equivalence	95
6	Example	96

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*]:

$\xi \models_c \varphi \text{ iff}_c \psi = (\xi \models_c \varphi \longleftrightarrow \xi \models_c \psi)$
 $\xi \models_c F_c \varphi = \xi \models_c (\text{true}_c U_c \varphi)$
 $\xi \models_c G_c \varphi = \xi \models_c (\text{false}_c R_c \varphi)$
by (*auto simp add: Iff-ltlc-def*)

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

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

$\text{language-ltlc } (\text{not}_c \varphi) = - \text{language-ltlc } \varphi$
unfolding *language-ltlc-def* **by** *auto*

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

$\xi \models_c \text{prop}_c(p) \text{ or}_c (\text{not}_c \text{prop}_c(p))$
by *auto*

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

$\xi \models_c \text{not}_c \text{true}_c \longleftrightarrow \xi \models_c \text{false}_c$
by *auto*

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

$\xi \models_c X_c (\text{not}_c \varphi) \longleftrightarrow \xi \models_c \text{not}_c X_c \varphi$
by *auto*

— The connection between dual operators

lemma *ltl-Until-Release-con*:

$$\xi \models_c \varphi R_c \psi \longleftrightarrow (\neg \xi \models_c (\text{not}_c \varphi) U_c (\text{not}_c \psi))$$

$$\xi \models_c \varphi U_c \psi \longleftrightarrow (\neg \xi \models_c (\text{not}_c \varphi) R_c (\text{not}_c \psi))$$

by *auto*

lemma *ltl-WeakUntil-StrongRelease-con*:

$$\xi \models_c \varphi W_c \psi \longleftrightarrow (\neg \xi \models_c (\text{not}_c \varphi) M_c (\text{not}_c \psi))$$

$$\xi \models_c \varphi M_c \psi \longleftrightarrow (\neg \xi \models_c (\text{not}_c \varphi) W_c (\text{not}_c \psi))$$

by *auto*

— The connection between weak and strong operators

lemma *ltl-Release-StrongRelease-con*:

$$\xi \models_c \varphi R_c \psi \longleftrightarrow \xi \models_c (\varphi M_c \psi) \text{ or}_c (G_c \psi)$$

$$\xi \models_c \varphi M_c \psi \longleftrightarrow \xi \models_c (\varphi R_c \psi) \text{ and}_c (F_c \varphi)$$

proof *safe*

assume *asm*: $\xi \models_c \varphi R_c \psi$

show $\xi \models_c (\varphi M_c \psi) \text{ or}_c (G_c \psi)$

proof (*cases* $\xi \models_c G_c \psi$)

case *False*

then obtain *i* **where** $\neg \text{suffix } i \xi \models_c \psi$ **and** $\forall j < i. \text{suffix } j \xi \models_c \psi$
using *exists-least-iff*[of $\lambda i. \neg \text{suffix } i \xi \models_c \psi$] **by force**

then show *?thesis*

using *asm* **by force**

qed *simp*

next

assume *asm*: $\xi \models_c (\varphi R_c \psi) \text{ and}_c (F_c \varphi)$

then show $\xi \models_c \varphi M_c \psi$

proof (*cases* $\xi \models_c F_c \varphi$)

case *True*

then obtain *i* **where** $\text{suffix } i \xi \models_c \varphi$ **and** $\forall j < i. \neg \text{suffix } j \xi \models_c \varphi$
using *exists-least-iff*[of $\lambda i. \text{suffix } i \xi \models_c \varphi$] **by force**

then show *?thesis*

using *asm* **by force**

qed *simp*

qed (*unfold semantics-ltlc.simps; insert not-less, blast*)**+**

lemma *ltl-Until-WeakUntil-con*:

$\xi \models_c \varphi U_c \psi \longleftrightarrow \xi \models_c (\varphi W_c \psi) \text{ and}_c (F_c \psi)$

$\xi \models_c \varphi W_c \psi \longleftrightarrow \xi \models_c (\varphi U_c \psi) \text{ or}_c (G_c \varphi)$

proof *safe*

assume *asm*: $\xi \models_c (\varphi W_c \psi) \text{ and}_c (F_c \psi)$

then show $\xi \models_c \varphi U_c \psi$

proof (*cases* $\xi \models_c F_c \psi$)

case *True*

then obtain *i* **where** *suffix* *i* $\xi \models_c \psi$ **and** $\forall j < i. \neg \text{suffix } j \xi \models_c \psi$

using *exists-least-iff*[of $\lambda i. \text{suffix } i \xi \models_c \psi$] **by force**

then show *?thesis*

using *asm* **by force**

qed *simp*

next

assume *asm*: $\xi \models_c \varphi W_c \psi$

then show $\xi \models_c (\varphi U_c \psi) \text{ or}_c (G_c \varphi)$

proof (*cases* $\xi \models_c G_c \varphi$)

case *False*

then obtain *i* **where** $\neg \text{suffix } i \xi \models_c \varphi$ **and** $\forall j < i. \text{suffix } j \xi \models_c \varphi$

using *exists-least-iff*[of $\lambda i. \neg \text{suffix } i \xi \models_c \varphi$] **by force**

then show *?thesis*

using *asm* **by force**

qed *simp*

qed (*unfold semantics-ltlc.simps; insert not-less, blast*)⁺

lemma *ltl-StrongRelease-Until-con*:

$\xi \models_c \varphi M_c \psi \longleftrightarrow \xi \models_c \psi U_c (\varphi \text{ and}_c \psi)$

using *order.order-iff-strict* **by auto**

lemma *ltl-WeakUntil-Release-con*:

$\xi \models_c \varphi R_c \psi \longleftrightarrow \xi \models_c \psi W_c (\varphi \text{ and}_c \psi)$

by (*meson* *ltl-Release-StrongRelease-con*(1) *ltl-StrongRelease-Until-con* *ltl-Until-WeakUntil-con*(2) *semantics-ltlc.simps*(6))

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

lemma *pw-eq-on-refl*[*simp*]: *pw-eq-on* *S* *w* *w*

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

lemma *pw-eq-on-suffix*:
 $pw\text{-eq-on } S w w' \implies pw\text{-eq-on } S (\text{suffix } k w) (\text{suffix } k w')$
by (*simp add: pw-eq-on-def*)

lemma *pw-eq-on-subset*:
 $S \subseteq S' \implies pw\text{-eq-on } S' w w' \implies pw\text{-eq-on } S w w'$
by (*auto simp add: pw-eq-on-def*)

lemma *ltlc-eq-on-aux*:
 $pw\text{-eq-on } (\text{atoms-ltlc } \varphi) w w' \implies w \models_c \varphi \implies w' \models_c \varphi$
proof (*induction* φ *arbitrary*: $w w'$)
case *Until-ltlc*
thus *?case*
by *simp (meson Un-upper1 Un-upper2 pw-eq-on-subset pw-eq-on-suffix)*
next
case *Release-ltlc*
thus *?case*
by *simp (metis Un-upper1 pw-eq-on-subset pw-eq-on-suffix sup-commute)*
next
case *WeakUntil-ltlc*
thus *?case*
by *simp (meson pw-eq-on-subset pw-eq-on-suffix sup.cobounded1 sup-ge2)*
next
case *StrongRelease-ltlc*
thus *?case*
by *simp (metis Un-upper1 pw-eq-on-subset pw-eq-on-suffix pw-eq-on-sym sup-ge2)*
next
case (*And-ltlc* $\varphi \psi$)
thus *?case*
by *simp (meson Un-upper1 inf-sup-ord(4) pw-eq-on-subset)*
next
case (*Or-ltlc* $\varphi \psi$)
thus *?case*
by *simp (meson Un-upper2 pw-eq-on-subset sup-ge1)*
next
case (*Implies-ltlc* $\varphi \psi$)
thus *?case*
by *simp (meson Un-upper1 Un-upper2 pw-eq-on-subset[of atoms-ltlc -*

atoms-ltlc $\varphi \cup \text{atoms-ltlc } \psi$] *pw-eq-on-sym*)
qed (*auto simp add: pw-eq-on-def;metis suffix-nth*)+

lemma *ltlc-eq-on*:

pw-eq-on (*atoms-ltlc* φ) $w w' \implies w \models_c \varphi \longleftrightarrow w' \models_c \varphi$
using *ltlc-eq-on-aux pw-eq-on-sym* **by** *blast*

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

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

lemma *map-semantic-ltlc-aux*:

assumes *inj-on* $f APs$
assumes $\bigcup (\text{range } w) \subseteq APs$
assumes *atoms-ltlc* $\varphi \subseteq APs$
shows $w \models_c \varphi \longleftrightarrow (\lambda i. f ' w i) \models_c \text{map-ltlc } f \varphi$
using *assms(2,3)*

proof (*induction* φ *arbitrary*: w)

case (*Prop-ltlc* x)

thus *?case* **using** *assms(1)*

by (*simp add: SUP-le-iff inj-on-image-mem-iff*)

next

case (*Next-ltlc* φ)

show *?case*

using *Next-ltlc(1)[of suffix 1 w, unfolded suffix-comp comp-def]* *Next-ltlc(2,3)*

apply *simp*

by (*metis Next-ltlc.prem(1) One-nat-def* $\langle [\bigcup (\text{range } (\text{suffix } 1 w)) \subseteq APs; \text{atoms-ltlc } \varphi \subseteq APs] \implies \text{suffix } 1 w \models_c \varphi = \text{suffix } 1 (\lambda x. f ' w x) \models_c \text{map-ltlc } f \varphi \rangle$ *suffix-range*)

next

case (*Final-ltlc* φ)

thus *?case*

using *Final-ltlc(1)[of suffix - , unfolded suffix-comp comp-def, OF suffix-range]* **by** *fastforce*

next

case (*Global-ltlc*)

thus *?case*

using *Global-ltlc(1)[of suffix - w, unfolded suffix-comp comp-def, OF suffix-range]* **by** *fastforce*

next

case (*Until-ltlc*)

thus *?case*

using *Until-ltlc*(1,2)[of suffix - w, unfolded suffix-comp comp-def, OF suffix-range] **by** *fastforce*
next
case (*Release-ltlc*)
thus ?*case*
using *Release-ltlc*(1,2)[of suffix - w, unfolded suffix-comp comp-def, OF suffix-range] **by** *fastforce*
next
case (*WeakUntil-ltlc*)
thus ?*case*
using *WeakUntil-ltlc*(1,2)[of suffix - w, unfolded suffix-comp comp-def, OF suffix-range] **by** *fastforce*
next
case (*StrongRelease-ltlc*)
thus ?*case*
using *StrongRelease-ltlc*(1,2)[of suffix - w, unfolded suffix-comp comp-def, OF suffix-range] **by** *fastforce*
qed *simp+*

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

lemma *map-semantics-ltlc*:

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

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

proof –

let ? $\xi_r = \lambda i. \xi i \cap \text{atoms-ltlc } \varphi$

let ? $\xi_{r'} = \lambda i. \xi i \cap \text{dom } f$

have 1: $\bigcup (\text{range } ?\xi_r) \subseteq \text{atoms-ltlc } \varphi$ **by** *auto*

have *INJ-the-dom*: *inj-on (the o f) (dom f)*

using *assms*

by (*auto simp: inj-on-def domIff*)

note 2 = *subset-inj-on*[OF *this DOM*]

have 3: $(\lambda i. (\text{the } o f) ' ?\xi_{r'} i) = \text{map-props } f \circ \xi$ **using** *DOM INJ*

apply (*auto intro!: ext simp: map-props-def domIff image-iff*)

by (*metis Int-iff domI option.sel*)

have $\xi \models_c \varphi \longleftrightarrow ?\xi_r \models_c \varphi$

apply (*rule ltlc-eq-on*)

apply (*auto simp: pw-eq-on-def*)

done

also from *map-semantics-ltlc-aux*[OF 2 1 *subset-refl*]

have ... $\longleftrightarrow (\lambda i. (the\ of\ f)\ ' \ ?\xi r\ i) \models_c\ map\ ltlc\ (the\ of\ f)\ \varphi$.
also have ... $\longleftrightarrow (\lambda i. (the\ of\ f)\ ' \ ?\xi r'\ i) \models_c\ map\ ltlc\ (the\ of\ f)\ \varphi$
apply (rule ltlc-eq-on) **using** *DOM INJ*
apply (auto simp: pw-eq-on-def ltlc.set-map domIff image-iff)
by (metis Int-iff contra-subsetD domD domI inj-on-eq-iff option.sel)
also note 3
finally show ?thesis .
qed

lemma *map-semantic-ltlc-inv*:

assumes *INJ*: *inj-on f (dom f)* **and** *DOM*: *atoms-ltlc $\varphi \subseteq dom\ f$*
shows $\xi \models_c\ map\ ltlc\ (the\ of\ f)\ \varphi \longleftrightarrow (\lambda i. (the\ of\ f)\ -'\ \xi\ i) \models_c\ \varphi$
using *map-semantic-ltlc[OF assms]*
apply *simp*
apply (*intro ltlc-eq-on*)
apply (*auto simp add: pw-eq-on-def ltlc.set-map map-props-def*)
by (*metis DOM comp-apply contra-subsetD domD option.sel vimage-eq*)

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>	(<i>true_n</i>)
<i>False-ltln</i>	(<i>false_n</i>)
<i>Prop-ltln</i> 'a	(<i>prop_n'(-)</i>)
<i>Nprop-ltln</i> 'a	(<i>nprop_{n}'(-^)}</i>)
<i>And-ltln</i> 'a <i>ltln</i> 'a <i>ltln</i>	(- <i>and_n</i> - [82,82] 81)
<i>Or-ltln</i> 'a <i>ltln</i> 'a <i>ltln</i>	(- <i>or_n</i> - [84,84] 83)
<i>Next-ltln</i> 'a <i>ltln</i>	(<i>X_n</i> - [88] 87)
<i>Until-ltln</i> 'a <i>ltln</i> 'a <i>ltln</i>	(- <i>U_n</i> - [84,84] 83)
<i>Release-ltln</i> 'a <i>ltln</i> 'a <i>ltln</i>	(- <i>R_n</i> - [84,84] 83)
<i>WeakUntil-ltln</i> 'a <i>ltln</i> 'a <i>ltln</i>	(- <i>W_n</i> - [84,84] 83)
<i>StrongRelease-ltln</i> 'a <i>ltln</i> 'a <i>ltln</i>	(- <i>M_n</i> - [84,84] 83)

abbreviation *finally_n* :: 'a *ltln* \Rightarrow 'a *ltln* (*F_n* - [88] 87)

where

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

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

abbreviation *globally_n* :: 'a *ltln* \Rightarrow 'a *ltln* (*G_n* - [88] 87)

where

$$G_n \varphi \equiv \text{false}_n R_n \varphi$$

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

1.2.2 Semantics

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

where

$$\begin{aligned} & \xi \models_n \text{true}_n = \text{True} \\ | & \xi \models_n \text{false}_n = \text{False} \\ | & \xi \models_n \text{prop}_n(q) = (q \in \xi \ 0) \\ | & \xi \models_n \text{nprop}_n(q) = (q \notin \xi \ 0) \\ | & \xi \models_n \varphi \ \text{and}_n \ \psi = (\xi \models_n \varphi \wedge \xi \models_n \psi) \\ | & \xi \models_n \varphi \ \text{or}_n \ \psi = (\xi \models_n \varphi \vee \xi \models_n \psi) \\ | & \xi \models_n X_n \varphi = (\text{suffix } 1 \ \xi \models_n \varphi) \\ | & \xi \models_n \varphi \ U_n \ \psi = (\exists i. \text{suffix } i \ \xi \models_n \psi \wedge (\forall j < i. \text{suffix } j \ \xi \models_n \varphi)) \\ | & \xi \models_n \varphi \ R_n \ \psi = (\forall i. \text{suffix } i \ \xi \models_n \psi \vee (\exists j < i. \text{suffix } j \ \xi \models_n \varphi)) \\ | & \xi \models_n \varphi \ W_n \ \psi = (\forall i. \text{suffix } i \ \xi \models_n \varphi \vee (\exists j \leq i. \text{suffix } j \ \xi \models_n \psi)) \\ | & \xi \models_n \varphi \ M_n \ \psi = (\exists i. \text{suffix } i \ \xi \models_n \varphi \wedge (\forall j \leq i. \text{suffix } j \ \xi \models_n \psi)) \end{aligned}$$

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

lemma *semantics-ltln-ite-simps[simp]*:

$$\begin{aligned} w \models_n (\text{if } P \ \text{then } \text{true}_n \ \text{else } \text{false}_n) &= P \\ w \models_n (\text{if } P \ \text{then } \text{false}_n \ \text{else } \text{true}_n) &= (\neg P) \\ &\text{by } \text{simp-all} \end{aligned}$$

1.2.3 Conversion

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

where

$$\begin{aligned} & \text{ltlc-to-ltln}' \ \text{False} \ \text{true}_c = \text{true}_n \\ | & \text{ltlc-to-ltln}' \ \text{False} \ \text{false}_c = \text{false}_n \\ | & \text{ltlc-to-ltln}' \ \text{False} \ \text{prop}_c(q) = \text{prop}_n(q) \\ | & \text{ltlc-to-ltln}' \ \text{False} \ (\varphi \ \text{and}_c \ \psi) = (\text{ltlc-to-ltln}' \ \text{False} \ \varphi) \ \text{and}_n \ (\text{ltlc-to-ltln}' \ \text{False} \ \psi) \\ | & \text{ltlc-to-ltln}' \ \text{False} \ (\varphi \ \text{or}_c \ \psi) = (\text{ltlc-to-ltln}' \ \text{False} \ \varphi) \ \text{or}_n \ (\text{ltlc-to-ltln}' \ \text{False} \ \psi) \\ | & \text{ltlc-to-ltln}' \ \text{False} \ (\varphi \ \text{implies}_c \ \psi) = (\text{ltlc-to-ltln}' \ \text{True} \ \varphi) \ \text{or}_n \ (\text{ltlc-to-ltln}' \ \text{False} \ \psi) \\ | & \text{ltlc-to-ltln}' \ \text{False} \ (F_c \ \varphi) = \text{true}_n \ U_n \ (\text{ltlc-to-ltln}' \ \text{False} \ \varphi) \\ | & \text{ltlc-to-ltln}' \ \text{False} \ (G_c \ \varphi) = \text{false}_n \ R_n \ (\text{ltlc-to-ltln}' \ \text{False} \ \varphi) \\ | & \text{ltlc-to-ltln}' \ \text{False} \ (\varphi \ U_c \ \psi) = (\text{ltlc-to-ltln}' \ \text{False} \ \varphi) \ U_n \ (\text{ltlc-to-ltln}' \ \text{False} \ \psi) \end{aligned}$$

ψ)
 $| \text{ltlc-to-ltln}' \text{ False } (\varphi R_c \psi) = (\text{ltlc-to-ltln}' \text{ False } \varphi) R_n (\text{ltlc-to-ltln}' \text{ False } \psi)$
 $| \text{ltlc-to-ltln}' \text{ False } (\varphi W_c \psi) = (\text{ltlc-to-ltln}' \text{ False } \varphi) W_n (\text{ltlc-to-ltln}' \text{ False } \psi)$
 $| \text{ltlc-to-ltln}' \text{ False } (\varphi M_c \psi) = (\text{ltlc-to-ltln}' \text{ False } \varphi) M_n (\text{ltlc-to-ltln}' \text{ False } \psi)$
 $| \text{ltlc-to-ltln}' \text{ True } \text{true}_c = \text{false}_n$
 $| \text{ltlc-to-ltln}' \text{ True } \text{false}_c = \text{true}_n$
 $| \text{ltlc-to-ltln}' \text{ True } \text{prop}_c(q) = \text{nprop}_n(q)$
 $| \text{ltlc-to-ltln}' \text{ True } (\varphi \text{and}_c \psi) = (\text{ltlc-to-ltln}' \text{ True } \varphi) \text{or}_n (\text{ltlc-to-ltln}' \text{ True } \psi)$
 $| \text{ltlc-to-ltln}' \text{ True } (\varphi \text{or}_c \psi) = (\text{ltlc-to-ltln}' \text{ True } \varphi) \text{and}_n (\text{ltlc-to-ltln}' \text{ True } \psi)$
 $| \text{ltlc-to-ltln}' \text{ True } (\varphi \text{implies}_c \psi) = (\text{ltlc-to-ltln}' \text{ False } \varphi) \text{and}_n (\text{ltlc-to-ltln}' \text{ True } \psi)$
 $| \text{ltlc-to-ltln}' \text{ True } (F_c \varphi) = \text{false}_n R_n (\text{ltlc-to-ltln}' \text{ True } \varphi)$
 $| \text{ltlc-to-ltln}' \text{ True } (G_c \varphi) = \text{true}_n U_n (\text{ltlc-to-ltln}' \text{ True } \varphi)$
 $| \text{ltlc-to-ltln}' \text{ True } (\varphi U_c \psi) = (\text{ltlc-to-ltln}' \text{ True } \varphi) R_n (\text{ltlc-to-ltln}' \text{ True } \psi)$
 $| \text{ltlc-to-ltln}' \text{ True } (\varphi R_c \psi) = (\text{ltlc-to-ltln}' \text{ True } \varphi) U_n (\text{ltlc-to-ltln}' \text{ True } \psi)$
 $| \text{ltlc-to-ltln}' \text{ True } (\varphi W_c \psi) = (\text{ltlc-to-ltln}' \text{ True } \varphi) M_n (\text{ltlc-to-ltln}' \text{ True } \psi)$
 $| \text{ltlc-to-ltln}' \text{ True } (\varphi M_c \psi) = (\text{ltlc-to-ltln}' \text{ True } \varphi) W_n (\text{ltlc-to-ltln}' \text{ True } \psi)$
 $| \text{ltlc-to-ltln}' b (\text{not}_c \varphi) = \text{ltlc-to-ltln}' (\neg b) \varphi$
 $| \text{ltlc-to-ltln}' b (X_c \varphi) = X_n (\text{ltlc-to-ltln}' b \varphi)$

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

where

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

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

where

ltln-to-ltlc $\text{true}_n = \text{true}_c$

$| \text{ltln-to-ltlc } \text{false}_n = \text{false}_c$

$| \text{ltln-to-ltlc } \text{prop}_n(q) = \text{prop}_c(q)$

$| \text{ltln-to-ltlc } \text{nprop}_n(q) = \text{not}_c (\text{prop}_c(q))$

$| \text{ltln-to-ltlc } (\varphi \text{and}_n \psi) = (\text{ltln-to-ltlc } \varphi \text{and}_c \text{ltln-to-ltlc } \psi)$

$| \text{ltln-to-ltlc } (\varphi \text{or}_n \psi) = (\text{ltln-to-ltlc } \varphi \text{or}_c \text{ltln-to-ltlc } \psi)$

$| \text{ltln-to-ltlc } (X_n \varphi) = (X_c \text{ltln-to-ltlc } \varphi)$

$| \text{ltln-to-ltlc } (\varphi U_n \psi) = (\text{ltln-to-ltlc } \varphi U_c \text{ltln-to-ltlc } \psi)$

$| \text{ltln-to-ltlc } (\varphi R_n \psi) = (\text{ltln-to-ltlc } \varphi R_c \text{ltln-to-ltlc } \psi)$

$| \text{ltln-to-ltlc } (\varphi W_n \psi) = (\text{ltln-to-ltlc } \varphi W_c \text{ltln-to-ltlc } \psi)$

$| \text{ltln-to-ltlc } (\varphi M_n \psi) = (\text{ltln-to-ltlc } \varphi M_c \text{ltln-to-ltlc } \psi)$

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

$w \models_n (\text{ltlc-to-ltln}' \text{ True } \varphi) \longleftrightarrow \neg w \models_c \varphi$
 $w \models_n (\text{ltlc-to-ltln}' \text{ False } \varphi) \longleftrightarrow w \models_c \varphi$
 $\text{size } (\text{ltlc-to-ltln}' \text{ True } \varphi) \leq 2 * \text{size } \varphi$
 $\text{size } (\text{ltlc-to-ltln}' \text{ False } \varphi) \leq 2 * \text{size } \varphi$
by (*induction* φ *arbitrary*: w) *simp+*

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

$w \models_n \text{ltlc-to-ltln } \varphi \longleftrightarrow w \models_c \varphi$
using *ltlc-to-ltln'-correct* **by** *auto*

lemma *ltlc-to-ltln-size*:

$\text{size } (\text{ltlc-to-ltln } \varphi) \leq 2 * \text{size } \varphi$
using *ltlc-to-ltln'-correct* **by** *simp*

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

$w \models_c \text{ltln-to-ltlc } \varphi \longleftrightarrow w \models_n \varphi$
by (*induction* φ *arbitrary*: w) *simp+*

lemma *ltlc-to-ltln-atoms*:

$\text{atoms-ltln } (\text{ltlc-to-ltln } \varphi) = \text{atoms-ltlc } \varphi$

proof –

have $\text{atoms-ltln } (\text{ltlc-to-ltln}' \text{ True } \varphi) = \text{atoms-ltlc } \varphi$
 $\text{atoms-ltln } (\text{ltlc-to-ltln}' \text{ False } \varphi) = \text{atoms-ltlc } \varphi$
by (*induction* φ) *simp+*
thus *?thesis*
by *simp*

qed

1.2.4 Negation

fun *not_n*

where

$\text{not}_n \text{true}_n = \text{false}_n$
 $\text{not}_n \text{false}_n = \text{true}_n$
 $\text{not}_n \text{prop}_n(a) = \text{nprop}_n(a)$
 $\text{not}_n \text{nprop}_n(a) = \text{prop}_n(a)$
 $\text{not}_n (\varphi \text{and}_n \psi) = (\text{not}_n \varphi) \text{or}_n (\text{not}_n \psi)$
 $\text{not}_n (\varphi \text{or}_n \psi) = (\text{not}_n \varphi) \text{and}_n (\text{not}_n \psi)$
 $\text{not}_n (X_n \varphi) = X_n (\text{not}_n \varphi)$
 $\text{not}_n (\varphi U_n \psi) = (\text{not}_n \varphi) R_n (\text{not}_n \psi)$
 $\text{not}_n (\varphi R_n \psi) = (\text{not}_n \varphi) U_n (\text{not}_n \psi)$
 $\text{not}_n (\varphi W_n \psi) = (\text{not}_n \varphi) M_n (\text{not}_n \psi)$

| $not_n (\varphi M_n \psi) = (not_n \varphi) W_n (not_n \psi)$

lemma *not_n-semantics[simp]*:

$w \models_n not_n \varphi \longleftrightarrow \neg w \models_n \varphi$
by (*induction* φ *arbitrary: w*) *auto*

lemma *not_n-size*:

$size (not_n \varphi) = size \varphi$
by (*induction* φ) *auto*

1.2.5 Subformulas

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

$\varphi \in subfrmlsn \varphi$
by (*induction* φ) *auto*

lemma *subfrmlsn-finite*:

$finite (subfrmlsn \varphi)$
by (*induction* φ) *auto*

lemma *subfrmlsn-card*:

$card (subfrmlsn \varphi) \leq size \varphi$
by (*induction* φ) (*simp-all add: card-insert-if subfrmlsn-finite, (meson add-le-mono card-Un-le dual-order.trans le-SucI)*)

lemma *subfrmlsn-subset*:

$\psi \in subfrmlsn \varphi \Longrightarrow subfrmlsn \psi \subseteq subfrmlsn \varphi$
by (*induction* φ) *auto*

lemma *subfrmlsn-size*:

$\psi \in subfrmlsn \varphi \Longrightarrow size \psi < size \varphi \vee \psi = \varphi$
by (*induction* φ) *auto*

abbreviation

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

lemma $size\text{-}set\text{-}diff$:

$finite\ S \implies S' \subseteq S \implies size\text{-}set\ (S - S') = size\text{-}set\ S - size\text{-}set\ S'$
using $sum\text{-}diff\text{-}nat\ finite\text{-}subset$ by $metis$

1.2.6 Constant Folding

lemma $U\text{-consts}$ [$intro, simp$]:

$w \models_n \varphi\ U_n\ true_n$
 $\neg (w \models_n \varphi\ U_n\ false_n)$
 $(w \models_n false_n\ U_n\ \varphi) = (w \models_n \varphi)$
by $force+$

lemma $R\text{-consts}$ [$intro, simp$]:

$w \models_n \varphi\ R_n\ true_n$
 $\neg (w \models_n \varphi\ R_n\ false_n)$
 $(w \models_n true_n\ R_n\ \varphi) = (w \models_n \varphi)$
by $force+$

lemma $W\text{-consts}$ [$intro, simp$]:

$w \models_n true_n\ W_n\ \varphi$
 $w \models_n \varphi\ W_n\ true_n$
 $(w \models_n false_n\ W_n\ \varphi) = (w \models_n \varphi)$
 $(w \models_n \varphi\ W_n\ false_n) = (w \models_n G_n\ \varphi)$
by $force+$

lemma $M\text{-consts}$ [$intro, simp$]:

$\neg (w \models_n false_n\ M_n\ \varphi)$
 $\neg (w \models_n \varphi\ M_n\ false_n)$
 $(w \models_n true_n\ M_n\ \varphi) = (w \models_n \varphi)$
 $(w \models_n \varphi\ M_n\ true_n) = (w \models_n F_n\ \varphi)$
by $force+$

1.2.7 Distributivity

lemma $until\text{-}and\text{-}left\text{-}distrib$:

$w \models_n (\varphi_1\ and_n\ \varphi_2)\ U_n\ \psi \iff w \models_n (\varphi_1\ U_n\ \psi)\ and_n\ (\varphi_2\ U_n\ \psi)$

proof

assume $w \models_n \varphi_1\ U_n\ \psi\ and_n\ \varphi_2\ U_n\ \psi$

then obtain $i1\ i2$ where $suffix\ i1\ w \models_n \psi \wedge (\forall j < i1. suffix\ j\ w \models_n \varphi_1)$
and $suffix\ i2\ w \models_n \psi \wedge (\forall j < i2. suffix\ j\ w \models_n \varphi_2)$

by auto

then have $\text{suffix } (\text{min } i1 \ i2) \ w \models_n \psi \wedge (\forall j < \text{min } i1 \ i2. \text{suffix } j \ w \models_n \varphi_1 \text{ and}_n \varphi_2)$

by (*simp add: min-def*)

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

by force

qed auto

lemma *until-or-right-distrib*:

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

by auto

lemma *release-and-right-distrib*:

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

by auto

lemma *release-or-left-distrib*:

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

by (*metis not_n.simps(6) not_n.simps(9) not_n-semantic until-and-left-distrib*)

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

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

then obtain $i1 \ i2$ **where** $\text{suffix } i1 \ w \models_n \varphi \wedge (\forall j \leq i1. \text{suffix } j \ w \models_n \psi_1)$
and $\text{suffix } i2 \ w \models_n \varphi \wedge (\forall j \leq i2. \text{suffix } j \ w \models_n \psi_2)$

by auto

then have $\text{suffix } (\text{min } i1 \ i2) \ w \models_n \varphi \wedge (\forall j \leq \text{min } i1 \ i2. \text{suffix } j \ w \models_n \psi_1 \text{ and}_n \psi_2)$

by (*simp add: min-def*)

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

by force

qed auto

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

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

by auto

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

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

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

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

by (*metis not_n.simps(10) not_n.simps(6) not_n-semantics strong-release-and-right-distrib*)

lemma *next-until-distrib*:

$w \models_n X_n (\varphi U_n \psi) \longleftrightarrow w \models_n (X_n \varphi) U_n (X_n \psi)$

by auto

lemma *next-release-distrib*:

$w \models_n X_n (\varphi R_n \psi) \longleftrightarrow w \models_n (X_n \varphi) R_n (X_n \psi)$

by auto

lemma *next-weak-until-distrib*:

$w \models_n X_n (\varphi W_n \psi) \longleftrightarrow w \models_n (X_n \varphi) W_n (X_n \psi)$

by auto

lemma *next-strong-release-distrib*:

$w \models_n X_n (\varphi M_n \psi) \longleftrightarrow w \models_n (X_n \varphi) M_n (X_n \psi)$

by auto

1.2.8 Nested operators

lemma *finally-until[simp]*:

$w \models_n F_n (\varphi U_n \psi) \longleftrightarrow w \models_n F_n \psi$

by auto force

lemma *globally-release[simp]*:

$w \models_n G_n (\varphi R_n \psi) \longleftrightarrow w \models_n G_n \psi$

by auto force

lemma *globally-weak-until[simp]*:

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

by auto force

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

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

by auto force

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 *auto*

fix i

assume $\forall i. \text{suffix } i \ w \models_n \varphi \vee (\exists j \leq i. \text{suffix } j \ w \models_n \psi)$
and $\forall i. \text{suffix } i \ w \models_n \psi \longrightarrow (\exists j < i. \neg \text{suffix } j \ w \models_n \varphi)$

then show $\text{suffix } i \ w \models_n \varphi$

by (*induction i rule: less-induct*) *force*

next

fix $i \ k$

assume $\forall j \leq i. \neg \text{suffix } j \ w \models_n \psi$
and $\text{suffix } k \ w \models_n \psi$
and $\forall j < k. \text{suffix } j \ w \models_n \varphi$

then show $\text{suffix } i \ w \models_n \varphi$

by (*cases i < k*) *simp-all*

next

fix i

assume $\forall i. \text{suffix } i \ w \models_n \psi \vee (\exists j < i. \text{suffix } j \ w \models_n \varphi)$
and $\forall i. \text{suffix } i \ w \models_n \varphi \longrightarrow (\exists j \leq i. \neg \text{suffix } j \ w \models_n \psi)$

then show $\text{suffix } i \ w \models_n \psi$

by (*induction i rule: less-induct*) *force*

next

fix $i \ k$

assume $\forall j < i. \neg \text{suffix } j \ w \models_n \varphi$
and $\text{suffix } k \ w \models_n \varphi$
and $\forall j \leq k. \text{suffix } j \ w \models_n \psi$

then show $\text{suffix } i \ w \models_n \psi$

by (*cases i ≤ k*) *simp-all*

qed

lemma *ltln-strong-weak*:

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

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

by (*metis ltln-weak-strong not_n.simps(1,5,8-11) not_n-semantics*)⁺

lemma *ltln-strong-to-weak*:

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

$w \models_n \varphi M_n \psi \implies w \models_n \varphi R_n \psi$
using *ltln-weak-strong* **by** *simp-all blast+*

lemma *ltln-weak-to-strong*:

$\llbracket w \models_n \varphi W_n \psi; \neg w \models_n G_n \varphi \rrbracket \implies w \models_n \varphi U_n \psi$
 $\llbracket w \models_n \varphi W_n \psi; w \models_n F_n \psi \rrbracket \implies w \models_n \varphi U_n \psi$
 $\llbracket w \models_n \varphi R_n \psi; \neg w \models_n G_n \psi \rrbracket \implies w \models_n \varphi M_n \psi$
 $\llbracket w \models_n \varphi R_n \psi; w \models_n F_n \varphi \rrbracket \implies w \models_n \varphi M_n \psi$
unfolding *ltln-weak-strong*[of $w \varphi \psi$] **by** *auto*

lemma *ltln-StrongRelease-to-Until*:

$w \models_n \varphi M_n \psi \longleftrightarrow w \models_n \psi U_n (\varphi \text{ and}_n \psi)$
using *order.order-iff-strict* **by** *auto*

lemma *ltln-Release-to-WeakUntil*:

$w \models_n \varphi R_n \psi \longleftrightarrow w \models_n \psi W_n (\varphi \text{ and}_n \psi)$
by (*meson ltln-StrongRelease-to-Until ltln-weak-strong semantics-ltln.simps(6)*)

lemma *ltln-WeakUntil-to-Release*:

$w \models_n \varphi W_n \psi \longleftrightarrow w \models_n \psi R_n (\varphi \text{ or}_n \psi)$
by (*metis ltln-StrongRelease-to-Until not_n.simps(6,9,10) not_n-semantics*)

lemma *ltln-Until-to-StrongRelease*:

$w \models_n \varphi U_n \psi \longleftrightarrow w \models_n \psi M_n (\varphi \text{ or}_n \psi)$
by (*metis ltln-Release-to-WeakUntil not_n.simps(6,8,11) not_n-semantics*)

1.2.10 GF and FG semantics

lemma *GF-suffix*:

$\text{suffix } i \ w \models_n G_n (F_n \psi) \longleftrightarrow w \models_n G_n (F_n \psi)$
by *auto* (*metis ab-semigroup-add-class.add-ac(1) add.left-commute*)

lemma *FG-suffix*:

$\text{suffix } i \ w \models_n F_n (G_n \psi) \longleftrightarrow w \models_n F_n (G_n \psi)$
by (*auto simp: algebra-simps*) (*metis add.commute add.left-commute*)

lemma *GF-Inf-many*:

$w \models_n G_n (F_n \varphi) \longleftrightarrow (\exists \infty i. \text{suffix } i \ w \models_n \varphi)$
unfolding *INFM-nat-le*
by *simp* (*blast dest: le-Suc-ex intro: le-add1*)

lemma *FG-Alm-all*:

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

unfolding *MOST-nat-le*
by *simp* (*blast dest: le-Suc-ex intro: le-add1*)

lemma *MOST-nat-add*:
 $(\forall_{\infty} i :: \text{nat}. P\ i) \longleftrightarrow (\forall_{\infty} i. P\ (i + j))$
by (*simp add: cofinite-eq-sequentially*)

lemma *INFM-nat-add*:
 $(\exists_{\infty} i :: \text{nat}. P\ i) \longleftrightarrow (\exists_{\infty} i. P\ (i + j))$
using *MOST-nat-add not-MOST not-INFM* **by** *blast*

lemma *FG-suffix-G*:
 $w \models_n F_n (G_n \varphi) \implies \forall_{\infty} i. \text{suffix } i\ w \models_n G_n \varphi$
proof –
assume $w \models_n F_n (G_n \varphi)$
then have $w \models_n F_n (G_n (G_n \varphi))$
by (*meson globally-release semantics-ltln.simps(8)*)
then show $\forall_{\infty} i. \text{suffix } i\ w \models_n G_n \varphi$
unfolding *FG-Alm-all* .

qed

lemma *Alm-all-GF-F*:
 $\forall_{\infty} i. \text{suffix } i\ w \models_n G_n (F_n \psi) \longleftrightarrow \text{suffix } i\ w \models_n F_n \psi$
unfolding *MOST-nat*

proof *standard+*

fix $i :: \text{nat}$
assume $\text{suffix } i\ w \models_n G_n (F_n \psi)$
then show $\text{suffix } i\ w \models_n F_n \psi$
unfolding *GF-Inf-many INFM-nat* **by** *fastforce*

next

fix $i :: \text{nat}$
assume $\text{suffix } i\ w \models_n F_n \psi$
assume $\text{max}: i > \text{Max } \{i. \text{suffix } i\ w \models_n \psi\}$

with *suffix* **obtain** j **where** $j \geq i$ **and** $j\text{-suffix}: \text{suffix } j\ w \models_n \psi$
by *simp* (*blast intro: le-add1*)

with *max* **have** $j\text{-max}: j > \text{Max } \{i. \text{suffix } i\ w \models_n \psi\}$
by *fastforce*

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

proof (*cases* $w \models_n G_n (F_n \psi)$)
case *False*
then have $\neg (\exists_{\infty} i. \text{suffix } i \ w \models_n \psi)$
unfolding *GF-Inf-many* **by** *simp*
then have *finite* $\{i. \text{suffix } i \ w \models_n \psi\}$
by (*simp add: INFM-iff-infinite*)
then have $\forall i > \text{Max } \{i. \text{suffix } i \ w \models_n \psi\}. \neg \text{suffix } i \ w \models_n \psi$
using *Max-ge not-le* **by** *auto*
then show *?thesis*
using *j-suffix j-max* **by** *blast*
qed *force*
qed

lemma *Alm-all-FG-G:*

$\forall_{\infty} i. \text{suffix } i \ w \models_n F_n (G_n \psi) \longleftrightarrow \text{suffix } i \ w \models_n G_n \psi$
unfolding *MOST-nat*

proof *standard+*

fix $i :: \text{nat}$
assume $\text{suffix } i \ w \models_n G_n \psi$
then show $\text{suffix } i \ w \models_n F_n (G_n \psi)$
unfolding *FG-Alm-all INFM-nat* **by** *fastforce*

next

fix $i :: \text{nat}$
assume *suffix:* $\text{suffix } i \ w \models_n F_n (G_n \psi)$
assume *max:* $i > \text{Max } \{i. \neg \text{suffix } i \ w \models_n G_n \psi\}$

with *suffix* **have** $\forall_{\infty} j. \text{suffix } (i + j) \ w \models_n G_n \psi$
using *FG-suffix-G[of suffix i w] suffix-suffix*
by *fastforce*

then have $\neg (\exists_{\infty} j. \neg \text{suffix } j \ w \models_n G_n \psi)$
using *MOST-nat-add[of $\lambda i. \text{suffix } i \ w \models_n G_n \psi$ i]*
by (*simp add: algebra-simps*)

then have *finite* $\{i. \neg \text{suffix } i \ w \models_n G_n \psi\}$
by (*simp add: INFM-iff-infinite*)

with *max* **show** $\text{suffix } i \ w \models_n G_n \psi$
using *Max-ge leD* **by** *blast*

qed

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)

proof
 assume $?lhs$
 then obtain i where $\text{suffix } i \ \xi \models_n \psi$
 and $\forall j < i. \text{suffix } j \ \xi \models_n \varphi$
 by *auto*
 thus $?rhs$
 by (*cases* i) *auto*
next
 assume $?rhs$
 show $?lhs$
proof (*cases* $\xi \models_n \psi$)
 case *False*
 then have $\xi \models_n \varphi$ and $\xi \models_n X_n (\varphi \ U_n \ \psi)$
 using $\langle ?rhs \rangle$ by *auto*
 thus $?lhs$
 using *less-Suc-eq-0-disj suffix-singleton-suffix* by *force*
qed *force*
qed

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 $?lhs = ?rhs$)

proof
 assume $?lhs$
 thus $?rhs$
 using *less-Suc-eq-0-disj* by *force*
next
 assume $?rhs$
 {
 fix i
 assume $\neg \text{suffix } i \ \xi \models_n \psi$
 then have $\exists j < i. \text{suffix } j \ \xi \models_n \varphi$
 using $\langle ?rhs \rangle$ by (*cases* i) *force+*
 }
 thus $?lhs$
 by *auto*
qed

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 $?lhs = ?rhs$)

proof

```

assume ?lhs
thus ?rhs
  by (metis ltn-expand-Release ltn-expand-Until ltn-weak-strong(1) se-
mantics-ltn.simps(2,5,6,7))
next
  assume ?rhs

  {
    fix i
    assume  $\neg$  suffix i  $\xi \models_n \varphi$ 
    then have  $\exists j \leq i. \text{suffix } j \xi \models_n \psi$ 
      using <?rhs> by (cases i) force+
  }

  thus ?lhs
  by auto
qed

lemma ltn-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)
proof
  assume ?lhs
  then obtain i where suffix i  $\xi \models_n \varphi$ 
    and  $\forall j \leq i. \text{suffix } j \xi \models_n \psi$ 
    by auto
  thus ?rhs
    by (cases i) auto
next
  assume ?rhs
  show ?lhs
  proof (cases  $\xi \models_n \varphi$ )
    case True
      thus ?lhs
      using <?rhs> ltn-expand-WeakUntil by fastforce
    next
      case False
        thus ?lhs
        by (metis <?rhs> ltn-expand-WeakUntil not_n.simps(5,6,7,11) not_n-semantics)
  qed
qed

```

```

lemma ltn-Release-alterdef:
   $w \models_n \varphi R_n \psi \longleftrightarrow w \models_n (G_n \psi) \text{ or}_n (\psi U_n (\varphi \text{ and}_n \psi))$ 

```

proof (*cases* $\exists i. \neg \text{suffix } i \ w \models_n \psi$)
case *True*
define *i where* $i \equiv \text{Least } (\lambda i. \neg \text{suffix } i \ w \models_n \psi)$
have $\bigwedge j. j < i \implies \text{suffix } j \ w \models_n \psi$ **and** $\neg \text{suffix } i \ w \models_n \psi$
using *True LeastI not-less-Least unfolding i-def by fast+*
hence $*$: $\forall i. \text{suffix } i \ w \models_n \psi \vee (\exists j < i. \text{suffix } j \ w \models_n \varphi) \implies (\exists i. (\text{suffix } i \ w \models_n \psi \wedge \text{suffix } i \ w \models_n \varphi) \wedge (\forall j < i. \text{suffix } j \ w \models_n \psi))$
by *fastforce*
hence $\exists i. (\text{suffix } i \ w \models_n \psi \wedge \text{suffix } i \ w \models_n \varphi) \wedge (\forall j < i. \text{suffix } j \ w \models_n \psi) \implies (\forall i. \text{suffix } i \ w \models_n \psi \vee (\exists j < i. \text{suffix } j \ w \models_n \varphi))$
using *linorder-cases by blast*
thus *?thesis*
using *True * by auto*
qed *auto*

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* =
 $\text{True-ltlr} \quad (\text{true}_r)$
 $|\ \text{False-ltlr} \quad (\text{false}_r)$
 $|\ \text{Prop-ltlr } 'a \quad (\text{prop}_r '(-))$
 $|\ \text{Nprop-ltlr } 'a \quad (\text{nprop}_r '(-))$
 $|\ \text{And-ltlr } 'a \ \text{ltlr } 'a \ \text{ltlr} \quad (- \ \text{and}_r \ - \ [82,82] \ 81)$
 $|\ \text{Or-ltlr } 'a \ \text{ltlr } 'a \ \text{ltlr} \quad (- \ \text{or}_r \ - \ [84,84] \ 83)$
 $|\ \text{Next-ltlr } 'a \ \text{ltlr} \quad (X_r \ - \ [88] \ 87)$
 $|\ \text{Until-ltlr } 'a \ \text{ltlr } 'a \ \text{ltlr} \quad (- \ U_r \ - \ [84,84] \ 83)$
 $|\ \text{Release-ltlr } 'a \ \text{ltlr } 'a \ \text{ltlr} \quad (- \ R_r \ - \ [84,84] \ 83)$

1.3.2 Semantics

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

where

$\xi \models_r \text{true}_r = \text{True}$
 $|\ \xi \models_r \text{false}_r = \text{False}$
 $|\ \xi \models_r \text{prop}_r(q) = (q \in \xi \ 0)$
 $|\ \xi \models_r \text{nprop}_r(q) = (q \notin \xi \ 0)$
 $|\ \xi \models_r \varphi \ \text{and}_r \ \psi = (\xi \models_r \varphi \wedge \xi \models_r \psi)$
 $|\ \xi \models_r \varphi \ \text{or}_r \ \psi = (\xi \models_r \varphi \vee \xi \models_r \psi)$
 $|\ \xi \models_r X_r \ \varphi = (\text{suffix } 1 \ \xi \models_r \varphi)$

$| \xi \models_r \varphi U_r \psi = (\exists i. \text{suffix } i \xi \models_r \psi \wedge (\forall j < i. \text{suffix } j \xi \models_r \varphi))$
 $| \xi \models_r \varphi R_r \psi = (\forall i. \text{suffix } i \xi \models_r \psi \vee (\exists j < i. \text{suffix } j \xi \models_r \varphi))$

1.3.3 Conversion

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

where

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

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

where

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

lemma *ltn-to-ltlr-semantic*:

$w \models_r \text{ ltn-to-ltlr } \varphi \longleftrightarrow w \models_n \varphi$

by (*induction* φ *arbitrary*: w) (*unfold ltn-WeakUntil-to-Release ltn-StrongRelease-to-Until, simp-all*)

lemma *ltlr-to-ltn-semantic*:

$w \models_n \text{ ltlr-to-ltn } \varphi \longleftrightarrow w \models_r \varphi$

by (*induction* φ *arbitrary*: w) *simp-all*

1.3.4 Negation

fun not_r

where

$not_r\ true_r = false_r$
| $not_r\ false_r = true_r$
| $not_r\ prop_r(a) = nprop_r(a)$
| $not_r\ nprop_r(a) = prop_r(a)$
| $not_r\ (\varphi\ and_r\ \psi) = (not_r\ \varphi)\ or_r\ (not_r\ \psi)$
| $not_r\ (\varphi\ or_r\ \psi) = (not_r\ \varphi)\ and_r\ (not_r\ \psi)$
| $not_r\ (X_r\ \varphi) = X_r\ (not_r\ \varphi)$
| $not_r\ (\varphi\ U_r\ \psi) = (not_r\ \varphi)\ R_r\ (not_r\ \psi)$
| $not_r\ (\varphi\ R_r\ \psi) = (not_r\ \varphi)\ U_r\ (not_r\ \psi)$

lemma not_r -semantics [simp]:

$w \models_r not_r\ \varphi \iff \neg w \models_r \varphi$
by (induction φ arbitrary: w) auto

1.3.5 Subformulas

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

where

$subfrmlsr\ (\varphi\ and_r\ \psi) = \{\varphi\ and_r\ \psi\} \cup subfrmlsr\ \varphi \cup subfrmlsr\ \psi$
| $subfrmlsr\ (\varphi\ or_r\ \psi) = \{\varphi\ or_r\ \psi\} \cup subfrmlsr\ \varphi \cup subfrmlsr\ \psi$
| $subfrmlsr\ (\varphi\ U_r\ \psi) = \{\varphi\ U_r\ \psi\} \cup subfrmlsr\ \varphi \cup subfrmlsr\ \psi$
| $subfrmlsr\ (\varphi\ R_r\ \psi) = \{\varphi\ R_r\ \psi\} \cup subfrmlsr\ \varphi \cup subfrmlsr\ \psi$
| $subfrmlsr\ (X_r\ \varphi) = \{X_r\ \varphi\} \cup subfrmlsr\ \varphi$
| $subfrmlsr\ x = \{x\}$

lemma $subfrmlsr$ -id[simp]:

$\varphi \in subfrmlsr\ \varphi$
by (induction φ) auto

lemma $subfrmlsr$ -finite:

$finite\ (subfrmlsr\ \varphi)$
by (induction φ) auto

lemma $subfrmlsr$ -subset:

$\psi \in subfrmlsr\ \varphi \implies subfrmlsr\ \psi \subseteq subfrmlsr\ \varphi$
by (induction φ) auto

lemma $subfrmlsr$ -size:

$\psi \in subfrmlsr\ \varphi \implies size\ \psi < size\ \varphi \vee \psi = \varphi$
by (induction φ) auto

1.3.6 Expansion lemmas

lemma *ltlr-expand-Until*:

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

by (*metis ltlr-expand-Until ltlr-to-ltln.simps(5-8) ltlr-to-ltln-semantic*)

lemma *ltlr-expand-Release*:

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

by (*metis ltlr-expand-Release ltlr-to-ltln.simps(5-7,9) ltlr-to-ltln-semantic*)

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 'a</i> \Rightarrow <i>bool</i>	$(atom_p'(-))$
<i>Implies-ltlp 'a pttl 'a pttl</i>	$(- \text{ implies}_p - [81,81] 80)$
<i>Next-ltlp 'a pttl</i>	$(X_p - [88] 87)$
<i>Until-ltlp 'a pttl 'a 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

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

definition *True-ltlp* (*true_p*)

where

$$true_p \equiv not_p false_p$$

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

where

$$\varphi \text{ or}_p \psi \equiv (not_p \varphi) \text{ implies}_p \psi$$

definition *And-ltlp* (*- and_p* - [82,82] 81)

where

$$\varphi \text{ and}_p \psi \equiv \text{not}_p ((\text{not}_p \varphi) \text{ or}_p (\text{not}_p \psi))$$

definition *Eventually-ltlp* (F_p - [88] 87)

where

$$F_p \varphi \equiv \text{true}_p U_p \varphi$$

definition *Always-ltlp* (G_p - [88] 87)

where

$$G_p \varphi \equiv \text{not}_p (F_p (\text{not}_p \varphi))$$

definition *Release-ltlp* ($- R_p$ - [84,84] 83)

where

$$\varphi R_p \psi \equiv \text{not}_p ((\text{not}_p \varphi) U_p (\text{not}_p \psi))$$

definition *WeakUntil-ltlp* ($- W_p$ - [84,84] 83)

where

$$\varphi W_p \psi \equiv \psi R_p (\varphi \text{ or}_p \psi)$$

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*]:

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

by (*auto simp: Not-ltlp-def True-ltlp-def Or-ltlp-def And-ltlp-def Eventually-ltlp-def Always-ltlp-def Release-ltlp-def WeakUntil-ltlp-def StrongRe-*

lease-ltlp-def) (*insert le-neq-implies-less, blast*)+

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

ltlc-to-pltl true_c = *true_p*
| *ltlc-to-pltl false_c* = *false_p*
| *ltlc-to-pltl (prop_c(q))* = *atom_p((\in) q)*
| *ltlc-to-pltl (not_c φ)* = *not_p (ltlc-to-pltl φ)*
| *ltlc-to-pltl (φ and_c ψ)* = (*ltlc-to-pltl φ*) *and_p* (*ltlc-to-pltl ψ*)
| *ltlc-to-pltl (φ or_c ψ)* = (*ltlc-to-pltl φ*) *or_p* (*ltlc-to-pltl ψ*)
| *ltlc-to-pltl (φ implies_c ψ)* = (*ltlc-to-pltl φ*) *implies_p* (*ltlc-to-pltl ψ*)
| *ltlc-to-pltl (X_c φ)* = *X_p (ltlc-to-pltl φ)*
| *ltlc-to-pltl (F_c φ)* = *F_p (ltlc-to-pltl φ)*
| *ltlc-to-pltl (G_c φ)* = *G_p (ltlc-to-pltl φ)*
| *ltlc-to-pltl (φ U_c ψ)* = (*ltlc-to-pltl φ*) *U_p* (*ltlc-to-pltl ψ*)
| *ltlc-to-pltl (φ R_c ψ)* = (*ltlc-to-pltl φ*) *R_p* (*ltlc-to-pltl ψ*)
| *ltlc-to-pltl (φ W_c ψ)* = (*ltlc-to-pltl φ*) *W_p* (*ltlc-to-pltl ψ*)
| *ltlc-to-pltl (φ M_c ψ)* = (*ltlc-to-pltl φ*) *M_p* (*ltlc-to-pltl ψ*)

lemma *ltlc-to-pltl-semantic* [*simp*]:

$w \models_p (\text{ltlc-to-pltl } \varphi) \iff w \models_c \varphi$
by (*induction φ arbitrary: w*) *simp-all*

1.4.4 Atoms

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

where

atoms-pltl false_p = {}
| *atoms-pltl atom_p(p)* = {p}
| *atoms-pltl (φ implies_p ψ)* = *atoms-pltl φ* \cup *atoms-pltl ψ*
| *atoms-pltl (X_p φ)* = *atoms-pltl φ*
| *atoms-pltl (φ U_p ψ)* = *atoms-pltl φ* \cup *atoms-pltl ψ*

lemma *atoms-finite* [*iff*]:

finite (atoms-pltl φ)
by (*induct φ*) *auto*

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

atoms-pltl (not_p φ) = *atoms-pltl φ*
atoms-pltl true_p = {}

```

atoms-pltl ( $\varphi$  orp  $\psi$ ) = atoms-pltl  $\varphi$   $\cup$  atoms-pltl  $\psi$ 
atoms-pltl ( $\varphi$  andp  $\psi$ ) = atoms-pltl  $\varphi$   $\cup$  atoms-pltl  $\psi$ 
atoms-pltl ( $F_p$   $\varphi$ ) = atoms-pltl  $\varphi$ 
atoms-pltl ( $G_p$   $\varphi$ ) = atoms-pltl  $\varphi$ 
by (auto simp: Not-ltlp-def True-ltlp-def Or-ltlp-def And-ltlp-def Eventu-
ally-ltlp-def Always-ltlp-def)

```

end

2 Rewrite Rules for LTL Simplification

theory *Rewriting*

imports

LTL HOL-Library.Extended-Nat

begin

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

- Constants Removal
- *Next-ltln*-Normalisation
- Modal Simplification (based on pure eventual, pure universal, or suspendable formulas)
- Syntactic Implication Checking

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

2.1 Constant Eliminating Constructors

definition *mk-and*

where

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

definition *mk-or*

where

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

fun *remove-strong-ops*

where

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

fun *remove-weak-ops*

where

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

definition *mk-finally*

where

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

definition *mk-globally*

where

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

definition *mk-until*

where

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

definition *mk-release*

where

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

definition *mk-weak-until*

where

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

definition *mk-strong-release*

where

mk-strong-release $x y \equiv \text{case } y \text{ of } \text{false}_n \Rightarrow \text{false}_n$
 $| \text{true}_n \Rightarrow \text{mk-finally } x$
 $| - \Rightarrow (\text{case } x \text{ of } \text{true}_n \Rightarrow y | \text{false}_n \Rightarrow \text{false}_n | - \Rightarrow x M_n y)$

definition *mk-next*

where

mk-next $x \equiv \text{case } x \text{ of } \text{true}_n \Rightarrow \text{true}_n | \text{false}_n \Rightarrow \text{false}_n | - \Rightarrow X_n x$

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

where

mk-next-pow $n x \equiv \text{case } x \text{ of } \text{true}_n \Rightarrow \text{true}_n | \text{false}_n \Rightarrow \text{false}_n | - \Rightarrow$
(*Next-ltln* $\widehat{\widehat{n}}$) x

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

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

unfolding *mk-and-def* **by** (*cases x*; *cases y*; *simp*)

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

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

unfolding *mk-or-def* **by** (*cases x*; *cases y*; *simp*)

lemma *remove-strong-ops-sound* [*simp*]:

$w \models_n F_n (\text{remove-strong-ops } y) \longleftrightarrow w \models_n F_n y$

by (*induction y arbitrary: w*) (*auto*; *force*)+

lemma *remove-weak-ops-sound* [*simp*]:

$w \models_n G_n (\text{remove-weak-ops } y) \longleftrightarrow w \models_n G_n y$

by (*induction y arbitrary: w*) (*auto*; *force*)+

lemma *mk-finally-semantic* [*simp*]:

$w \models_n \text{mk-finally } x \longleftrightarrow w \models_n F_n x$

by (*simp add: mk-finally-def del: semantics-ltln.simps(8,9) remove-strong-ops.simps split: ltln.splits*)

lemma *mk-globally-semantic* [*simp*]:

$w \models_n \text{mk-globally } x \longleftrightarrow w \models_n G_n x$

by (*simp add: mk-globally-def del: semantics-ltln.simps(8,9) remove-weak-ops.simps split: ltln.splits*)

lemma *mk-until-semantic* [*simp*]:

$w \models_n \text{mk-until } x y \longleftrightarrow w \models_n x U_n y$

proof (*cases x*)


```

case (True-ltl)
  show ?thesis
    unfolding True-ltl mk-until-def
    by (cases y) auto
next
  case (False-ltl)
    thus ?thesis
    by (force simp: mk-until-def)
qed (cases y; force simp: mk-until-def)+

```

```

lemma mk-release-semantics [simp]:
   $w \models_n \text{mk-release } x \ y \longleftrightarrow w \models_n x \ R_n \ y$ 
proof (cases x)
  case (False-ltl)
    thus ?thesis
    unfolding False-ltl mk-release-def
    by (cases y) auto
next
  case (True-ltl)
    thus ?thesis
    by (force simp: mk-release-def)
qed (cases y; force simp: mk-release-def)+

```

```

lemma mk-weak-until-semantics [simp]:
   $w \models_n \text{mk-weak-until } x \ y \longleftrightarrow w \models_n x \ W_n \ y$ 
proof (cases y)
  case (False-ltl)
    thus ?thesis
    unfolding False-ltl mk-weak-until-def
    by (cases x) auto
next
  case (True-ltl)
    thus ?thesis
    by (force simp: mk-weak-until-def)
qed (cases x; force simp: mk-weak-until-def)+

```

```

lemma mk-strong-release-semantics [simp]:
   $w \models_n \text{mk-strong-release } x \ y \longleftrightarrow w \models_n x \ M_n \ y$ 
proof (cases y)
  case (True-ltl)
    show ?thesis
    unfolding True-ltl mk-strong-release-def
    by (cases x) auto
next

```

case (*False-ltln*)
thus *?thesis*
by (*force simp: mk-strong-release-def*)
qed (*cases x; force simp: mk-strong-release-def*)+

lemma *mk-next-semantic* [*simp*]:
 $w \models_n \text{mk-next } x \longleftrightarrow w \models_n X_n x$
unfolding *mk-next-def* **by** (*cases x; auto*)

lemma *mk-next-pow-semantic* [*simp*]:
 $w \models_n \text{mk-next-pow } i x \longleftrightarrow \text{suffix } i w \models_n x$
by (*induction i arbitrary: w; cases x*)
(auto simp: mk-next-pow-def)

lemma *mk-next-pow-simp* [*simp, code-unfold*]:
 $\text{mk-next-pow } 0 x = x$
 $\text{mk-next-pow } 1 x = \text{mk-next } x$
by (*cases x; simp add: mk-next-pow-def mk-next-def*)+

2.2 Constant Propagation

fun *is-constant* :: 'a ltln \Rightarrow bool
where
 $\text{is-constant true}_n = \text{True}$
 $\text{is-constant false}_n = \text{True}$
 $\text{is-constant -} = \text{False}$

lemma *is-constant-constructorsI*:
 $\text{is-constant } x \Longrightarrow \text{is-constant } y \Longrightarrow \text{is-constant } (\text{mk-and } x y)$
 $\neg \text{is-constant } x \Longrightarrow \neg \text{is-constant } y \Longrightarrow \neg \text{is-constant } (\text{mk-and } x y)$
 $\text{is-constant } x \Longrightarrow \text{is-constant } y \Longrightarrow \text{is-constant } (\text{mk-or } x y)$
 $\neg \text{is-constant } x \Longrightarrow \neg \text{is-constant } y \Longrightarrow \neg \text{is-constant } (\text{mk-or } x y)$
 $\text{is-constant } x \Longrightarrow \text{is-constant } (\text{mk-finally } x)$
 $\neg \text{is-constant } x \Longrightarrow \neg \text{is-constant } (\text{mk-finally } x)$
 $\text{is-constant } x \Longrightarrow \text{is-constant } (\text{mk-globally } x)$
 $\neg \text{is-constant } x \Longrightarrow \neg \text{is-constant } (\text{mk-globally } x)$
 $\text{is-constant } x \Longrightarrow \text{is-constant } (\text{mk-until } y x)$
 $\neg \text{is-constant } x \Longrightarrow \neg \text{is-constant } (\text{mk-until } y x)$
 $\text{is-constant } x \Longrightarrow \text{is-constant } (\text{mk-release } y x)$
 $\neg \text{is-constant } x \Longrightarrow \neg \text{is-constant } (\text{mk-release } y x)$
 $\text{is-constant } x \Longrightarrow \text{is-constant } y \Longrightarrow \text{is-constant } (\text{mk-weak-until } x y)$
 $\neg \text{is-constant } x \Longrightarrow \neg \text{is-constant } y \Longrightarrow \neg \text{is-constant } (\text{mk-weak-until } x y)$
 $\text{is-constant } x \Longrightarrow \text{is-constant } y \Longrightarrow \text{is-constant } (\text{mk-strong-release } x y)$
 $\neg \text{is-constant } x \Longrightarrow \neg \text{is-constant } y \Longrightarrow \neg \text{is-constant } (\text{mk-strong-release } x y)$

y)

$is_constant\ x \implies is_constant\ (mk_next\ x)$

$\neg is_constant\ x \implies \neg is_constant\ (mk_next\ x)$

$is_constant\ x \implies is_constant\ (mk_next_pow\ n\ x)$

by (cases x; cases y; simp add: mk-and-def mk-or-def mk-finally-def mk-globally-def
mk-until-def mk-release-def mk-weak-until-def mk-strong-release-def mk-next-def
mk-next-pow-def)+

lemma *is-constant-constructors-simps*:

$mk_next_pow\ n\ x = false_n \iff x = false_n$

$mk_next_pow\ n\ x = true_n \iff x = true_n$

$is_constant\ (mk_next_pow\ n\ x) \iff is_constant\ x$

by (induction n) (cases x; simp add: mk-next-pow-def)+

lemma *is-constant-constructors-simps2*:

$is_constant\ (mk_and\ x\ y) \iff (x = true_n \wedge y = true_n \vee x = false_n \vee y = false_n)$

$is_constant\ (mk_or\ x\ y) \iff (x = false_n \wedge y = false_n \vee x = true_n \vee y = true_n)$

$is_constant\ (mk_finally\ x) \iff is_constant\ x$

$is_constant\ (mk_globally\ x) \iff is_constant\ x$

$is_constant\ (mk_until\ y\ x) \iff is_constant\ x$

$is_constant\ (mk_release\ y\ x) \iff is_constant\ x$

$is_constant\ (mk_next\ x) \iff is_constant\ x$

by ((cases x; cases y; simp add: mk-and-def),
(cases x; cases y; simp add: mk-or-def),
(meson is-constant-constructorsI)+)

lemma *is-constant-constructors-simps3*:

$is_constant\ (mk_weak_until\ x\ y) \iff (x = false_n \wedge y = false_n \vee x = true_n \vee y = true_n)$

$is_constant\ (mk_strong_release\ x\ y) \iff (x = true_n \wedge y = true_n \vee x = false_n \vee y = false_n)$

by (cases x; cases y; simp add: mk-weak-until-def mk-strong-release-def
is-constant-constructors-simps2)+

lemma *is-constant-semantics*:

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

by (cases φ) auto

lemma *until-constant-simp*:

$is_constant\ \psi \implies w \models_n \varphi \ U_n\ \psi \iff w \models_n \psi$

by (cases ψ) auto

lemma *release-constant-simp*:

is-constant $\psi \implies w \models_n \varphi R_n \psi \longleftrightarrow w \models_n \psi$
by (*cases* ψ) *auto*

lemma *mk-next-pow-dist*:

mk-next-pow $(i + j) \varphi = \text{mk-next-pow } i \ (\text{mk-next-pow } j \ \varphi)$
by (*cases* j ; *simp*) (*cases* φ ; *simp* *add*: *mk-next-pow-def funpow-add*; *simp* *add*: *funpow-swap1*)

lemma *mk-next-pow-until*:

suffix $(\min i j) w \models_n (\text{mk-next-pow } (i - j) \ \varphi) U_n (\text{mk-next-pow } (j - i) \ \psi) \longleftrightarrow w \models_n (\text{mk-next-pow } i \ \varphi) U_n (\text{mk-next-pow } j \ \psi)$
by (*simp* *add*: *mk-next-pow-dist min-def add commute*)

lemma *mk-next-pow-release*:

suffix $(\min i j) w \models_n (\text{mk-next-pow } (i - j) \ \varphi) R_n (\text{mk-next-pow } (j - i) \ \psi) \longleftrightarrow w \models_n (\text{mk-next-pow } i \ \varphi) R_n (\text{mk-next-pow } j \ \psi)$
by (*simp* *add*: *mk-next-pow-dist min-def add commute*)

2.3 X-Normalisation

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

fun *the-enat-0* :: *enat* \Rightarrow *nat*

where

the-enat-0 $i = i$
| *the-enat-0* $\infty = 0$

lemma *the-enat-0-simp* [*simp*]:

the-enat-0 $0 = 0$
the-enat-0 $1 = 1$
by (*simp* *add*: *zero-enat-def one-enat-def*) $+$

fun *combine* :: $('a \ \text{ltln} \Rightarrow 'a \ \text{ltln} \Rightarrow 'a \ \text{ltln}) \Rightarrow ('a \ \text{ltln} * \text{enat}) \Rightarrow ('a \ \text{ltln} * \text{enat}) \Rightarrow ('a \ \text{ltln} * \text{enat})$

where

combine binop $(\varphi, i) (\psi, j) = (
\text{let } \chi = \text{binop } (\text{mk-next-pow } (\text{the-enat-0 } (i - j)) \ \varphi) (\text{mk-next-pow } (\text{the-enat-0 } (j - i)) \ \psi)
\text{in } (\chi, \text{if } \text{is-constant } \chi \text{ then } \infty \text{ else } \min i j))$

lemma *fst-combine*:

$\text{fst } (\text{combine binop } (\varphi, i) (\psi, j)) = \text{binop } (\text{mk-next-pow } (\text{the-enat-0 } (i - j)) \varphi) (\text{mk-next-pow } (\text{the-enat-0 } (j - i)) \psi)$
unfolding *combine.simps* **by** (*meson fstI*)

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

where

$\text{to-ltln } x \equiv \text{mk-next-pow } (\text{the-enat-0 } (\text{snd } x)) (\text{fst } x)$

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

where

$\text{rewrite-X-enat true}_n = (\text{true}_n, \infty)$
 $|\ \text{rewrite-X-enat false}_n = (\text{false}_n, \infty)$
 $|\ \text{rewrite-X-enat prop}_n(a) = (\text{prop}_n(a), 0)$
 $|\ \text{rewrite-X-enat nprop}_n(a) = (\text{nprop}_n(a), 0)$
 $|\ \text{rewrite-X-enat } (\varphi \text{ and}_n \psi) = \text{combine mk-and } (\text{rewrite-X-enat } \varphi) (\text{rewrite-X-enat } \psi)$
 $|\ \text{rewrite-X-enat } (\varphi \text{ or}_n \psi) = \text{combine mk-or } (\text{rewrite-X-enat } \varphi) (\text{rewrite-X-enat } \psi)$
 $|\ \text{rewrite-X-enat } (\varphi \text{ U}_n \psi) = \text{combine mk-until } (\text{rewrite-X-enat } \varphi) (\text{rewrite-X-enat } \psi)$
 $|\ \text{rewrite-X-enat } (\varphi \text{ R}_n \psi) = \text{combine mk-release } (\text{rewrite-X-enat } \varphi) (\text{rewrite-X-enat } \psi)$
 $|\ \text{rewrite-X-enat } (\varphi \text{ W}_n \psi) = \text{combine mk-weak-until } (\text{rewrite-X-enat } \varphi) (\text{rewrite-X-enat } \psi)$
 $|\ \text{rewrite-X-enat } (\varphi \text{ M}_n \psi) = \text{combine mk-strong-release } (\text{rewrite-X-enat } \varphi) (\text{rewrite-X-enat } \psi)$
 $|\ \text{rewrite-X-enat } (X_n \varphi) = (\lambda(\varphi, n). (\varphi, \text{eSuc } n)) (\text{rewrite-X-enat } \varphi)$

definition

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

lemma *combine-infinity-invariant*:

assumes $i = \infty \longleftrightarrow \text{is-constant } x$

assumes $j = \infty \longleftrightarrow \text{is-constant } y$

shows $\text{combine mk-and } (x, i) (y, j) = (z, k) \Longrightarrow (k = \infty \longleftrightarrow \text{is-constant } z)$

and $\text{combine mk-or } (x, i) (y, j) = (z, k) \Longrightarrow (k = \infty \longleftrightarrow \text{is-constant } z)$

and $\text{combine mk-until } (x, i) (y, j) = (z, k) \Longrightarrow (k = \infty \longleftrightarrow \text{is-constant } z)$

and $\text{combine mk-release } (x, i) (y, j) = (z, k) \Longrightarrow (k = \infty \longleftrightarrow \text{is-constant } z)$

and combine *mk-weak-until* $(x, i) (y, j) = (z, k) \implies (k = \infty \longleftrightarrow$
is-constant z)
and combine *mk-strong-release* $(x, i) (y, j) = (z, k) \implies (k = \infty \longleftrightarrow$
is-constant z)
by (*cases i; cases j; simp add: assms Let-def; force intro: is-constant-constructorsI*)+

lemma *combine-and-or-semantic:*

assumes $i = \infty \longleftrightarrow$ *is-constant φ*
assumes $j = \infty \longleftrightarrow$ *is-constant ψ*
shows $w \models_n \text{to-ltln} (\text{combine mk-and } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n \text{to-ltln } (\varphi,$
i) and_n to-ltln (ψ, j)
and $w \models_n \text{to-ltln} (\text{combine mk-or } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n \text{to-ltln } (\varphi, i)$
or_n to-ltln (ψ, j)
by ((*cases i; cases j; simp add: min-def is-constant-constructors-simps*
is-constant-constructors-simps2 assms),
(cases ψ ; insert assms; auto),
(cases φ ; insert assms; auto),
(blast elim!: is-constant.elims))+

lemma *combine-until-release-semantic:*

assumes $i = \infty \longleftrightarrow$ *is-constant φ*
assumes $j = \infty \longleftrightarrow$ *is-constant ψ*
shows $w \models_n \text{to-ltln} (\text{combine mk-until } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n \text{to-ltln}$
(φ, i) U_n to-ltln (ψ, j)
and $w \models_n \text{to-ltln} (\text{combine mk-release } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n \text{to-ltln}$
(φ, i) R_n to-ltln (ψ, j)
by ((*cases i; cases j; simp add: is-constant-constructors-simps is-constant-constructors-simps2*
until-constant-simp release-constant-simp mk-next-pow-until mk-next-pow-release
del: semantics-ltln.simps),
(blast dest: is-constant-semantic),
(cases ψ ; simp add: assms),
(cases φ ; insert assms; auto simp: add.commute))+

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

assumes $i = \infty \longleftrightarrow$ *is-constant φ*
assumes $j = \infty \longleftrightarrow$ *is-constant ψ*
shows $w \models_n \text{to-ltln} (\text{combine mk-weak-until } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n$
to-ltln (φ, i) W_n to-ltln (ψ, j)
and $w \models_n \text{to-ltln} (\text{combine mk-strong-release } (\varphi, i) (\psi, j)) \longleftrightarrow w \models_n$
to-ltln (φ, i) M_n to-ltln (ψ, j)
by ((*cases i; cases j; simp add: min-def is-constant-constructors-simps*
is-constant-constructors-simps3 del: semantics-ltln.simps),
(cases φ ; simp add: assms),

(cases ψ ; insert assms; auto simp: add.commute))+

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

snd (rewrite-X-enat φ) = $\infty \longleftrightarrow is_constant$ (fst (rewrite-X-enat φ))

proof (induction φ)

case (And-ltln $\varphi \psi$)

thus ?case

by (simp add: combine-infinity-invariant[OF And-ltln(1,2), unfolded prod.collapse])

next

case (Or-ltln $\varphi \psi$)

thus ?case

by (simp add: combine-infinity-invariant[OF Or-ltln(1,2), unfolded prod.collapse])

next

case (Until-ltln $\varphi \psi$)

thus ?case

by (simp add: combine-infinity-invariant[OF Until-ltln(1,2), unfolded prod.collapse])

next

case (Release-ltln $\varphi \psi$)

thus ?case

by (simp add: combine-infinity-invariant[OF Release-ltln(1,2), unfolded prod.collapse])

next

case (WeakUntil-ltln $\varphi \psi$)

thus ?case

by (simp add: combine-infinity-invariant[OF WeakUntil-ltln(1,2), unfolded prod.collapse])

next

case (StrongRelease-ltln $\varphi \psi$)

thus ?case

by (simp add: combine-infinity-invariant[OF StrongRelease-ltln(1,2), unfolded prod.collapse])

next

case (Next-ltln φ)

thus ?case

by (simp add: split-def) (metis eSuc-infinity eSuc-inject)

qed auto

lemma *rewrite-X-enat-correct*:

$w \models_n \varphi \longleftrightarrow w \models_n to_ltln$ (rewrite-X-enat φ)

```

proof (induction  $\varphi$  arbitrary:  $w$ )
  case (And-ltln  $\varphi \psi$ )
    thus ?case
    using combine-and-or-semantic[OF rewrite-X-enat-infinity-invariant
rewrite-X-enat-infinity-invariant] by fastforce
  next
    case (Or-ltln  $\varphi \psi$ )
      thus ?case
      using combine-and-or-semantic[OF rewrite-X-enat-infinity-invariant
rewrite-X-enat-infinity-invariant] by fastforce
    next
      case (Until-ltln  $\varphi \psi$ )
        thus ?case
        unfolding rewrite-X-enat.simps combine-until-release-semantic[OF
rewrite-X-enat-infinity-invariant rewrite-X-enat-infinity-invariant, unfolded
prod.collapse] by fastforce
      next
        case (Release-ltln  $\varphi \psi$ )
          thus ?case
          unfolding rewrite-X-enat.simps combine-until-release-semantic[OF
rewrite-X-enat-infinity-invariant rewrite-X-enat-infinity-invariant, unfolded
prod.collapse] by fastforce
        next
          case (WeakUntil-ltln  $\varphi \psi$ )
            thus ?case
            unfolding rewrite-X-enat.simps combine-weak-until-strong-release-semantic[OF
rewrite-X-enat-infinity-invariant rewrite-X-enat-infinity-invariant, unfolded
prod.collapse] by fastforce
          next
            case (StrongRelease-ltln  $\varphi \psi$ )
              thus ?case
              unfolding rewrite-X-enat.simps combine-weak-until-strong-release-semantic[OF
rewrite-X-enat-infinity-invariant rewrite-X-enat-infinity-invariant, unfolded
prod.collapse] by fastforce
            next
              case (Next-ltln  $\varphi$ )
                moreover
                  have  $w \models_n \text{to-ltln} (\text{rewrite-X-enat} (X_n \varphi)) \longleftrightarrow \text{suffix } 1 w \models_n \text{to-ltln}$ 
( $\text{rewrite-X-enat } \varphi$ )
                  by (simp add: split-def; cases snd ( $\text{rewrite-X-enat } \varphi$ )  $\neq \infty$ )
                    (auto simp: eSuc-def, auto simp: rewrite-X-enat-infinity-invariant
eSuc-def dest: is-constant-semantic)
                ultimately
                  show ?case

```


using *semantics-ltln.simps*(γ) **by** *blast*
qed *auto*

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

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

using *rewrite-X-enat-correct* **unfolding** *rewrite-X-def* *Let-def* **by** *auto*

2.4 Pure Eventual, Pure Universal, and Suspendable Formulas

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

where

pure-eventual true_n = True
| *pure-eventual false_n* = True
| *pure-eventual* (μ *and_n* μ') = (*pure-eventual* μ \wedge *pure-eventual* μ')
| *pure-eventual* (μ *or_n* μ') = (*pure-eventual* μ \wedge *pure-eventual* μ')
| *pure-eventual* (μ *U_n* μ') = ($\mu = \text{true}_n \vee$ *pure-eventual* μ')
| *pure-eventual* (μ *R_n* μ') = (*pure-eventual* μ \wedge *pure-eventual* μ')
| *pure-eventual* (μ *W_n* μ') = (*pure-eventual* μ \wedge *pure-eventual* μ')
| *pure-eventual* (μ *M_n* μ') = (*pure-eventual* μ \wedge *pure-eventual* $\mu' \vee \mu' = \text{true}_n$)
| *pure-eventual* (*X_n* μ) = *pure-eventual* μ
| *pure-eventual* - = False

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

where

pure-universal true_n = True
| *pure-universal false_n* = True
| *pure-universal* (ν *and_n* ν') = (*pure-universal* ν \wedge *pure-universal* ν')
| *pure-universal* (ν *or_n* ν') = (*pure-universal* ν \wedge *pure-universal* ν')
| *pure-universal* (ν *U_n* ν') = (*pure-universal* ν \wedge *pure-universal* ν')
| *pure-universal* (ν *R_n* ν') = ($\nu = \text{false}_n \vee$ *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* ($\varphi U_n \xi$) = ($\varphi = \text{true}_n \wedge \text{pure-universal } \xi \vee \text{suspendable } \xi$)
 | *suspendable* ($\varphi R_n \xi$) = ($\varphi = \text{false}_n \wedge \text{pure-eventual } \xi \vee \text{suspendable } \xi$)
 | *suspendable* ($\varphi W_n \xi$) = (*suspendable* $\varphi \wedge \text{suspendable } \xi \vee \text{pure-eventual } \varphi \wedge \xi = \text{false}_n$)
 | *suspendable* ($\varphi M_n \xi$) = (*suspendable* $\varphi \wedge \text{suspendable } \xi \vee \text{pure-universal } \varphi \wedge \xi = \text{true}_n$)
 | *suspendable* ($X_n \xi$) = *suspendable* ξ
 | *suspendable* - = *False*

lemma *pure-eventual-left-append*:

pure-eventual $\mu \implies w \models_n \mu \implies (u \frown w) \models_n \mu$

proof (*induction* μ *arbitrary*: $u w$)

case (*Until-ltln* $\mu \mu'$)

moreover

then obtain i **where** *suffix* $i w \models_n \mu'$

by *auto*

hence $\mu = \text{true}_n \implies ?\text{case}$

by *simp* (*metis suffix-conc-length suffix-suffix*)

moreover

have *pure-eventual* $\mu' \implies (u \frown w) \models_n \mu'$

by (*metis* $\langle \text{suffix } i w \models_n \mu' \rangle$ *Until-ltln*(2) *prefix-suffix*)

hence *pure-eventual* $\mu' \implies ?\text{case}$

by *force*

ultimately

show *?case*

by *fastforce*

next

case (*Release-ltln* $\mu \mu'$)

thus *?case*

by (*cases* $\forall i. \text{suffix } i w \models_n \mu'$; *simp-all*)

(*metis linear suffix-conc-snd grOI not-less0 prefix-suffix suffix-0*)+

next

case (*WeakUntil-ltln* $\mu \mu'$)

thus *?case*

by (*cases* $\forall i. \text{suffix } i w \models_n \mu'$; *simp-all*)

(*metis zero-le le0 nat-le-linear prefix-suffix suffix-0 suffix-conc-length suffix-conc-snd suffix-subseq-join*)+

next

case (*StrongRelease-ltln* $\mu \mu'$)

moreover

then obtain i **where** *suffix* $i w \models_n \mu$ *and* _{n} μ'

by *auto*

hence $\mu' = \text{true}_n \implies ?\text{case}$

by *simp* (*metis suffix-conc-length suffix-suffix*)

moreover
have $\text{pure-eventual } \mu \implies \text{pure-eventual } \mu' \implies (u \frown w) \models_n \mu \text{ and}_n \mu'$
by ($\text{metis } \langle \text{suffix } i \ w \models_n \mu \text{ and}_n \mu' \rangle$ $\text{calculation}(1)$ $\text{calculation}(2)$
 $\text{prefix-suffix semantics-ltln.simps}(5)$)
hence $\text{pure-eventual } \mu \implies \text{pure-eventual } \mu' \implies ?\text{case}$
by force
ultimately
show $?\text{case}$
by fastforce
qed ($\text{auto, metis diff-zero le-0-eq not-less-eq-eq suffix-conc-length suffix-conc-snd}$
 word-split)

lemma $\text{pure-universal-suffix-closed}$:

$\text{pure-universal } \nu \implies (u \frown w) \models_n \nu \implies w \models_n \nu$

proof ($\text{induction } \nu$ $\text{arbitrary: } u \ w$)

case ($\text{Until-ltln } \nu \ \nu'$)

hence $\exists i. \text{suffix } i \ (u \frown w) \models_n \nu' \wedge (\forall j < i. \text{suffix } j \ (u \frown w) \models_n \nu)$

using $\text{semantics-ltln.simps}(8)$ **by blast**

thus $?\text{case}$

by simp ($\text{metis Until-ltln}(1-3)$ $\text{le-0-eq le-eq-less-or-eq le-less-linear}$
 $\text{prefix-suffix pure-universal.simps}(5)$ $\text{suffix-conc-fst suffix-conc-snd}$)

next

case ($\text{Release-ltln } \nu \ \nu'$)

moreover

hence $\forall i. \text{suffix } i \ (u \frown w) \models_n \nu' \vee (\exists j < i. \text{suffix } j \ (u \frown w) \models_n \nu)$

by simp

ultimately

show $?\text{case}$

by simp ($\text{metis semantics-ltln.simps}(2)$ $\text{not-less0 prefix-suffix suffix-0}$
 $\text{suffix-conc-length suffix-suffix}$)

next

case ($\text{WeakUntil-ltln } \nu \ \nu'$)

moreover

hence $\forall i. \text{suffix } i \ (u \frown w) \models_n \nu \vee (\exists j \leq i. \text{suffix } j \ (u \frown w) \models_n \nu')$

by simp

ultimately

show $?\text{case}$

by simp ($\text{metis (full-types) le-antisym prefix-suffix semantics-ltln.simps}(2)$
 $\text{suffix-0 suffix-conc-length suffix-suffix zero-le}$)

next

case ($\text{StrongRelease-ltln } \nu \ \nu'$)

hence $\exists i. \text{suffix } i \ (u \frown w) \models_n \nu \wedge (\forall j \leq i. \text{suffix } j \ (u \frown w) \models_n \nu')$

using $\text{semantics-ltln.simps}(11)$ **by blast**

thus $?\text{case}$

by *simp* (*metis StrongRelease-ltln*(1-3) *diff-is-0-eq nat-le-linear prefix-conc-length prefix-suffix pure-universal.simps*(8) *subsequence-length suffix-conc-snd suffix-subseq-join*)

next

case (*Next-ltln* μ)

thus ?*case*

by (*metis prefix-suffix pure-universal.simps*(9) *semantics-ltln.simps*(7) *semiring-normalization-rules*(24) *suffix-conc-length suffix-suffix*)

qed *auto*

lemma *suspendable-prefix-invariant*:

suspendable $\xi \implies (u \frown w) \models_n \xi \longleftrightarrow w \models_n \xi$

proof (*induction* ξ *arbitrary*: u w)

case (*Until-ltln* ξ ξ')

show ?*case*

proof (*cases suspendable* ξ')

case *False*

hence $\xi = \text{true}_n$ and *pure-universal* ξ'

using *Until-ltln* by *simp+*

thus ?*thesis*

by (*simp*; *metis* (*no-types*) *linear pure-universal-suffix-closed suffix-conc-fst suffix-conc-length suffix-conc-snd suffix-suffix*)

qed (*simp*; *metis* *Until-ltln*(2) *not-less0 prefix-suffix*)

next

case (*Release-ltln* ξ ξ')

show ?*case*

proof (*cases suspendable* ξ')

case *False*

hence $\xi = \text{false}_n$ and *pure-eventual* ξ'

using *Release-ltln* by *simp+*

thus ?*thesis*

by (*simp*; *metis* (*no-types*) *le-iff-add add-diff-cancel-left' linear pure-eventual-left-append suffix-0 suffix-conc-fst suffix-conc-snd*)

qed (*simp*; *metis* *Release-ltln*(2) *not-less0 prefix-suffix*)

next

case (*WeakUntil-ltln* ξ ξ')

show ?*case*

proof (*cases suspendable* $\xi \wedge \text{suspendable } \xi'$)

case *False*

hence $\xi' = \text{false}_n$ and *pure-eventual* ξ

using *WeakUntil-ltln* by *simp+*

thus ?*thesis*

by (*simp*; *metis* (*no-types*) *le-iff-add add-diff-cancel-left' linear pure-eventual-left-append suffix-0 suffix-conc-fst suffix-conc-snd*)

qed (*simp*; *metis* (*full-types*) *WeakUntil-ltn.IH prefix-suffix*)
next
case (*StrongRelease-ltn* ξ ξ')
show *?case*
proof (*cases suspendable* $\xi \wedge$ *suspendable* ξ')
case *False*
hence $\xi' = \text{true}_n$ **and** *pure-universal* ξ
using *StrongRelease-ltn* **by** *simp+*
thus *?thesis*
by (*simp*; *metis* (*no-types*) *linear pure-universal-suffix-closed suffix-conc-fst suffix-conc-length suffix-conc-snd suffix-suffix*)
qed (*simp*; *metis* (*full-types*) *StrongRelease-ltn.IH(1) StrongRelease-ltn.IH(2) prefix-suffix*)
qed (*simp-all*, *metis prefix-suffix*)

theorem *pure-eventual-until-simp*:

assumes *pure-eventual* μ
shows $w \models_n \varphi \ U_n \ \mu \longleftrightarrow w \models_n \mu$
proof –
have $\bigwedge i. \text{suffix } i \ w \models_n \mu \implies w \models_n \mu$
using *pure-eventual-left-append[OF assms] prefix-suffix* **by** *metis*
thus *?thesis*
by *force*
qed

theorem *pure-universal-release-simp*:

assumes *pure-universal* ν
shows $w \models_n \varphi \ R_n \ \nu \longleftrightarrow w \models_n \nu$
proof –
have $\bigwedge i. w \models_n \nu \implies \text{suffix } i \ w \models_n \nu$
using *pure-universal-suffix-closed[OF assms] prefix-suffix* **by** *metis*
thus *?thesis*
by *force*
qed

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 –
have $\bigwedge i. w \models_n \varphi \implies \text{suffix } i \ w \models_n \varphi$ **and** $\bigwedge i. w \models_n \psi \implies \text{suffix } i \ w \models_n \psi$
using *assms pure-universal-suffix-closed prefix-suffix* **by** *metis+*
thus *?thesis*
by *force*

qed

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 –

have $\bigwedge i. \text{suffix } i \ w \models_n \varphi \implies w \models_n \varphi$ **and** $\bigwedge i. \text{suffix } i \ w \models_n \psi \implies w \models_n \psi$

using *assms pure-eventual-left-append prefix-suffix* **by** *metis+*

thus *?thesis*

by *force*

qed

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 –

have $\bigwedge i. \text{suffix } i \ w \models_n \xi \longleftrightarrow w \models_n \xi$

using *suspendable-prefix-invariant[OF assms] prefix-suffix* **by** *metis*

thus *?t1 ?t2 ?t3*

by *force+*

qed

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 –

have $\bigwedge i. \text{suffix } i \ w \models_n \varphi \longleftrightarrow w \models_n \varphi$ **and** $\bigwedge i. \text{suffix } i \ w \models_n \psi \longleftrightarrow w \models_n \psi$

using *assms suspendable-prefix-invariant prefix-suffix* **by** *metis+*

thus *?t1 ?t2*

by *force+*

qed

fun *rewrite-modal* :: $'a \ \text{ltln} \Rightarrow 'a \ \text{ltln}$

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 $\psi \vee$ suspendable ψ then
rewrite-modal ψ else (*rewrite-modal* φU_n *rewrite-modal* ψ))
 | *rewrite-modal* ($\varphi R_n \psi$) = (if pure-universal $\psi \vee$ suspendable ψ then
rewrite-modal ψ else (*rewrite-modal* φR_n *rewrite-modal* ψ))
 | *rewrite-modal* ($\varphi W_n \psi$) = (if pure-universal $\varphi \wedge$ pure-universal $\psi \vee$
 suspendable $\varphi \wedge$ suspendable ψ then (*rewrite-modal* φ or _{n} *rewrite-modal* ψ)
 else (*rewrite-modal* φW_n *rewrite-modal* ψ))
 | *rewrite-modal* ($\varphi M_n \psi$) = (if pure-eventual $\varphi \wedge$ pure-eventual $\psi \vee$ sus-
 pendable $\varphi \wedge$ suspendable ψ then (*rewrite-modal* φ and _{n} *rewrite-modal* ψ)
 else (*rewrite-modal* φM_n *rewrite-modal* ψ))
 | *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*]:

$w \models_n \text{rewrite-modal } \varphi \longleftrightarrow w \models_n \varphi$

proof (*induction* φ *arbitrary*: w)

case (*Until-ltl* $\varphi \psi$)

thus ?*case*

apply (*cases* pure-eventual $\psi \vee$ suspendable ψ)

apply (*insert* pure-eventual-until-*simp*[of ψ] suspendable-formula-*simp*[of
 ψ])

apply *fastforce+*

done

next

case (*Release-ltl* $\varphi \psi$)

thus ?*case*

apply (*cases* pure-universal $\psi \vee$ suspendable ψ)

apply (*insert* pure-universal-release-*simp*[of ψ] suspendable-formula-*simp*[of
 ψ])

apply *fastforce+*

done

next

case (*WeakUntil-ltl* $\varphi \psi$)

thus ?*case*

apply (*cases* pure-universal $\varphi \wedge$ pure-universal $\psi \vee$ suspendable $\varphi \wedge$
 suspendable ψ)

apply (*insert* pure-universal-weak-until-*simp*[of $\varphi \psi$] suspendable-formula-*simp2*[of
 $\varphi \psi$])

apply *fastforce+*

done

next

case (*StrongRelease-ltl* $\varphi \psi$)

thus ?*case*

```

      apply (cases pure-eventual  $\varphi \wedge$  pure-eventual  $\psi \vee$  suspendable  $\varphi \wedge$ 
suspendable  $\psi$ )
      apply (insert pure-eventual-strong-release-simp[of  $\varphi \psi$ ] suspendable-formula-simp2[of
 $\varphi \psi$ ])
      apply fastforce+
      done
next
  case (Next-ltln  $\varphi$ )
  thus ?case
  apply (cases suspendable  $\varphi$ )
  apply (insert suspendable-formula-simp[of  $\varphi$ ])
  apply fastforce+
  done
qed auto

```

lemma *rewrite-modal-size*:
 $size (rewrite-modal \varphi) \leq size \varphi$
by (*induction φ*) *auto*

2.5 Syntactical Implication Based Simplification

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

where

```

-  $\vdash_s true_n$ 
|  $false_n \vdash_s -$ 
|  $\varphi = \varphi \Rightarrow \varphi \vdash_s \varphi$ 

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

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

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

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

|  $(false_n R_n \varphi) \vdash_s \psi \Rightarrow (false_n R_n \varphi) \vdash_s X_n \psi$ 

```


| $\varphi \vdash_s (\text{true}_n U_n \psi) \implies (X_n \varphi) \vdash_s (\text{true}_n U_n \psi)$
| $\varphi \vdash_s \psi \implies (X_n \varphi) \vdash_s (X_n \psi)$

lemma *syntactical-implies-correct*:

$\varphi \vdash_s \psi \implies w \models_n \varphi \implies w \models_n \psi$

by (*induction arbitrary*: *w rule: syntactical-implies.induct; auto; force*)

fun *rewrite-syn-imp*

where

rewrite-syn-imp ($\varphi \text{ and}_n \psi$) = (
 if $\varphi \vdash_s \psi$ then
 rewrite-syn-imp φ
 else if $\psi \vdash_s \varphi$ then
 rewrite-syn-imp ψ
 else if $\varphi \vdash_s (\text{not}_n \psi) \vee \psi \vdash_s (\text{not}_n \varphi)$ then
 false_n
 else
 mk-and (*rewrite-syn-imp* φ) (*rewrite-syn-imp* ψ)
| *rewrite-syn-imp* ($\varphi \text{ or}_n \psi$) = (
 if $\varphi \vdash_s \psi$ then
 rewrite-syn-imp ψ
 else if $\psi \vdash_s \varphi$ then
 rewrite-syn-imp φ
 else if $(\text{not}_n \varphi) \vdash_s \psi \vee (\text{not}_n \psi) \vdash_s \varphi$ then
 true_n
 else
 mk-or (*rewrite-syn-imp* φ) (*rewrite-syn-imp* ψ)
| *rewrite-syn-imp* ($\varphi U_n \psi$) = (
 if $\varphi \vdash_s \psi$ then
 rewrite-syn-imp ψ
 else if $(\text{not}_n \varphi) \vdash_s \psi$ then
 mk-finally (*rewrite-syn-imp* ψ)
 else
 mk-until (*rewrite-syn-imp* φ) (*rewrite-syn-imp* ψ)
| *rewrite-syn-imp* ($\varphi R_n \psi$) = (
 if $\psi \vdash_s \varphi$ then
 rewrite-syn-imp ψ
 else if $\psi \vdash_s (\text{not}_n \varphi)$ then
 mk-globally (*rewrite-syn-imp* ψ)
 else
 mk-release (*rewrite-syn-imp* φ) (*rewrite-syn-imp* ψ)
| *rewrite-syn-imp* ($X_n \varphi$) = *mk-next* (*rewrite-syn-imp* φ)
| *rewrite-syn-imp* φ = φ

lemma *rewrite-syn-imp-sound*:

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

proof (*induction φ arbitrary: w*)

case *And-ltln*

thus *?case*

by (*simp add: Let-def; metis syntactical-implies-correct not_n-semantics*)

next

case (*Or-ltln $\varphi \psi$*)

moreover

have (*not_n φ*) $\vdash_s \psi \implies \forall w. w \models_n \varphi \text{ or}_n \psi$

by (*auto intro: syntactical-implies-correct[of not_n φ]*)

moreover

have (*not_n ψ*) $\vdash_s \varphi \implies \forall w. w \models_n \varphi \text{ or}_n \psi$

by (*auto intro: syntactical-implies-correct[of not_n ψ]*)

ultimately

show *?case*

by (*auto intro: syntactical-implies-correct*)

next

case (*Until-ltln $\varphi \psi$*)

moreover

have $\varphi \vdash_s \psi \implies ?case$

by (*force simp add: Until-ltln dest: syntactical-implies-correct*)

moreover

{

assume *A: (not_n φ) $\vdash_s \psi$ and B: $\neg \varphi \vdash_s \psi$*

hence [*simp*]: *rewrite-syn-imp ($\varphi U_n \psi$) = mk-finally (rewrite-syn-imp*

ψ)

by *simp*

{

assume $\exists i. \text{suffix } i \ w \models_n \psi$

moreover

define *i where $i \equiv \text{LEAST } i. \text{suffix } i \ w \models_n \psi$*

ultimately

have $\forall j < i. \neg \text{suffix } j \ w \models_n \psi$ **and** $\text{suffix } i \ w \models_n \psi$

by (*blast dest: not-less-Least, metis LeastI $\langle \exists i. \text{suffix } i \ w \models_n \psi \rangle$*

i-def)

hence $\forall j < i. \text{suffix } j \ w \models_n \varphi$ **and** $\text{suffix } i \ w \models_n \psi$

using *syntactical-implies-correct[OF A]* **by** *auto*

}

hence *?case*

by (*simp del: rewrite-syn-imp.simps; unfold Until-ltln(2)*) *blast*

}

ultimately

show *?case*

```

    by fastforce
next
case (Release-ltln  $\varphi$   $\psi$ )
  moreover
  have  $\psi \vdash_s \varphi \implies ?case$ 
    by (force simp add: Release-ltln dest: syntactical-implies-correct)
  moreover
  {
    assume  $A: \psi \vdash_s (\text{not}_n \varphi)$  and  $B: \neg \psi \vdash_s \varphi$ 
    hence [simp]:  $\text{rewrite-syn-imp} (\varphi R_n \psi) = \text{mk-globally} (\text{rewrite-syn-imp}$ 
 $\psi)$ 
    by simp
    {
      assume  $\exists i. \neg \text{suffix } i \ w \models_n \psi$ 
      moreover
      define  $i$  where  $i \equiv \text{LEAST } i. \neg \text{suffix } i \ w \models_n \psi$ 
      ultimately
      have  $\forall j < i. \text{suffix } j \ w \models_n \psi$  and  $\neg \text{suffix } i \ w \models_n \psi$ 
        by (blast dest: not-less-Least , metis LeastI < $\exists i. \neg \text{suffix } i \ w \models_n \psi$ >
 $i\text{-def}$ )
      hence  $\forall j < i. \neg \text{suffix } j \ w \models_n \varphi$  and  $\neg \text{suffix } i \ w \models_n \psi$ 
        using syntactical-implies-correct[OF  $A$ ] by auto
    }
    hence ?case
      by (simp del: rewrite-syn-imp.simps; unfold Release-ltln(2)) blast
  }
  ultimately
  show ?case
    by fastforce
qed auto

```

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

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

definition

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

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 then 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 $w \models_n \text{iterate } f \varphi n \longleftrightarrow w \models_n \varphi$
by (*induction n arbitrary: φ ; simp add: assms Let-def*)

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

$w \models_n \text{rewrite-iter-fast } \varphi \longleftrightarrow w \models_n \varphi$
using *iterate-sound*[*of - rewrite-modal o rewrite-X*]
unfolding *comp-def rewrite-modal-sound rewrite-X-sound rewrite-iter-fast-def*
by *blast*

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

$w \models_n \text{rewrite-iter-slow } \varphi \longleftrightarrow w \models_n \varphi$
using *iterate-sound*[*of - rewrite-syn-imp o rewrite-modal o rewrite-X*]
unfolding *comp-def rewrite-modal-sound rewrite-X-sound rewrite-syn-imp-sound rewrite-iter-slow-def*
by *blast*

2.7 Preservation of atoms

lemma *iterate-atoms*:

assumes
 $\bigwedge \varphi. \text{atoms-ltln } (f \varphi) \subseteq \text{atoms-ltln } \varphi$
shows
 $\text{atoms-ltln } (\text{iterate } f \varphi n) \subseteq \text{atoms-ltln } \varphi$
by (*induction n arbitrary: φ) (auto, metis (mono-tags, lifting) assms in-mono)*)

lemma *rewrite-modal-atoms*:

$\text{atoms-ltln } (\text{rewrite-modal } \varphi) \subseteq \text{atoms-ltln } \varphi$
by (*induction φ) auto*)

lemma *mk-and-atoms*:

$\text{atoms-ltln } (\text{mk-and } \varphi \psi) \subseteq \text{atoms-ltln } \varphi \cup \text{atoms-ltln } \psi$
by (*auto simp: mk-and-def split: ltln.splits*)

lemma *mk-or-atoms*:

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

by (*auto simp: mk-or-def split: ltn.splits*)

lemma *remove-strong-ops-atoms:*

$atoms_ltn (remove_strong_ops \ \varphi) \subseteq atoms_ltn \ \varphi$

by (*induction \varphi auto*)

lemma *remove-weak-ops-atoms:*

$atoms_ltn (remove_weak_ops \ \varphi) \subseteq atoms_ltn \ \varphi$

by (*induction \varphi auto*)

lemma *mk-finally-atoms:*

$atoms_ltn (mk_finally \ \varphi) \subseteq atoms_ltn \ \varphi$

by (*auto simp: mk-finally-def split: ltn.splits*) (*insert remove-strong-ops-atoms, fast+*)

lemma *mk-globally-atoms:*

$atoms_ltn (mk_globally \ \varphi) \subseteq atoms_ltn \ \varphi$

by (*auto simp: mk-globally-def split: ltn.splits*) (*insert remove-weak-ops-atoms, fast+*)

lemma *mk-until-atoms:*

$atoms_ltn (mk_until \ \varphi \ \psi) \subseteq atoms_ltn \ \varphi \cup atoms_ltn \ \psi$

by (*auto simp: mk-until-def split: ltn.splits*) (*insert mk-finally-atoms, fastforce+*)

lemma *mk-release-atoms:*

$atoms_ltn (mk_release \ \varphi \ \psi) \subseteq atoms_ltn \ \varphi \cup atoms_ltn \ \psi$

by (*auto simp: mk-release-def split: ltn.splits*) (*insert mk-globally-atoms, fastforce+*)

lemma *mk-weak-until-atoms:*

$atoms_ltn (mk_weak_until \ \varphi \ \psi) \subseteq atoms_ltn \ \varphi \cup atoms_ltn \ \psi$

by (*auto simp: mk-weak-until-def split: ltn.splits*) (*insert mk-globally-atoms, fastforce+*)

lemma *mk-strong-release-atoms:*

$atoms_ltn (mk_strong_release \ \varphi \ \psi) \subseteq atoms_ltn \ \varphi \cup atoms_ltn \ \psi$

by (*auto simp: mk-strong-release-def split: ltn.splits*) (*insert mk-finally-atoms, fastforce+*)

lemma *mk-next-atoms:*

$atoms_ltn (mk_next \ \varphi) = atoms_ltn \ \varphi$

by (*auto simp: mk-next-def split: ltn.splits*)

lemma *mk-next-pow-atoms*:

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

by (*induction* n) (*auto simp: mk-next-pow-def split: ltn.splits*)

lemma *combine-atoms*:

assumes

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

shows

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

by (*metis assms fst-combine mk-next-pow-atoms prod.collapse*)

lemmas *combine-mk-atoms* =

combine-atoms[*OF mk-and-atoms*]

combine-atoms[*OF mk-or-atoms*]

combine-atoms[*OF mk-until-atoms*]

combine-atoms[*OF mk-release-atoms*]

combine-atoms[*OF mk-weak-until-atoms*]

combine-atoms[*OF mk-strong-release-atoms*]

lemma *rewrite-X-enat-atoms*:

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

by (*induction* φ) (*simp-all add: case-prod-beta, insert combine-mk-atoms, fast+*)

lemma *rewrite-X-atoms*:

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

by (*induction* φ) (*simp-all add: rewrite-X-def mk-next-pow-atoms case-prod-beta, insert combine-mk-atoms, fast+*)

lemma *rewrite-syn-imp-atoms*:

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

proof (*induction* φ)

case (*And-ltn* $\varphi1\ \varphi2$)

then show *?case*

using *mk-and-atoms* **by** *simp fast*

next

case (*Or-ltn* $\varphi1\ \varphi2$)

then show *?case*

using *mk-or-atoms* **by** *simp fast*

next

case (*Next-ltn* φ)

then show *?case*

using *mk-next-atoms* **by** *simp fast*

next

```

case (Until-ltln  $\varphi1$   $\varphi2$ )
then show ?case
  using mk-finally-atoms mk-until-atoms by simp fast
next
  case (Release-ltln  $\varphi1$   $\varphi2$ )
  then show ?case
    using mk-globally-atoms mk-release-atoms by simp fast
qed simp-all

```

lemma *rewrite-iter-fast-atoms:*

atoms-ltln (*rewrite-iter-fast* φ) \subseteq *atoms-ltln* φ

proof –

have 1: $\bigwedge\varphi.$ *atoms-ltln* (*rewrite-modal* (*rewrite-X* φ)) \subseteq *atoms-ltln* φ

using *rewrite-modal-atoms rewrite-X-atoms* **by** *force*

show ?*thesis*

by (*simp add: rewrite-iter-fast-def 1 iterate-atoms*)

qed

lemma *rewrite-iter-slow-atoms:*

atoms-ltln (*rewrite-iter-slow* φ) \subseteq *atoms-ltln* φ

proof –

have 1: $\bigwedge\varphi.$ *atoms-ltln* (*rewrite-syn-imp* (*rewrite-modal* (*rewrite-X* φ)))
 \subseteq *atoms-ltln* φ

using *rewrite-syn-imp-atoms rewrite-modal-atoms rewrite-X-atoms* **by**
force

show ?*thesis*

by (*simp add: rewrite-iter-slow-def 1 iterate-atoms*)

qed

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:*

$w \models_n \text{simplify } m \ \varphi \longleftrightarrow w \models_n \varphi$
by (cases m) simp+

lemma *simplify-atoms*:

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

by (cases m) (insert rewrite-iter-fast-atoms rewrite-iter-slow-atoms, fast-force+)

2.9 Code Generation

code-pred *syntactical-implies* .

export-code *simplify* **checking**

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

by *eval*

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

by *eval*

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

by *eval*

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)

unfolding *ltl-lang-equiv-def*

by (*simp add: equivpI reflp-def symp-def transp-def*)

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

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

unfolding *ltl-lang-equiv-def* **by** *auto*

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

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

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

unfolding *ltl-lang-equiv-def* **by** *auto*

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

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

unfolding *ltl-lang-equiv-def* **by** *auto*

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

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

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

unfolding *ltl-lang-equiv-def* **by** *auto*

3.2 Propositional Equivalence

fun *ltl-prop-entailment* :: 'a *ltln* \Rightarrow 'a *ltln* \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]:

$$S \models_P \varphi \implies S \subseteq S' \implies S' \models_P \varphi$$

by (*induction* φ) *auto*

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

$$w \models_n \varphi \iff \{\psi. w \models_n \psi\} \models_P \varphi$$

by (*induction* φ) *auto*

definition *ltl-prop-equiv* :: 'a *ltln* \Rightarrow 'a *ltln* \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 *ltln* \Rightarrow 'a *ltln* \Rightarrow *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*:

$$\varphi \sim_P \psi \iff (\varphi \longrightarrow_P \psi \wedge \psi \longrightarrow_P \varphi)$$

unfolding *ltl-prop-equiv-def ltl-prop-implies-def* **by** *meson*

lemma *ltl-prop-equiv-equivp*:

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

by (*simp add: ltl-prop-equiv-def equivpI reflp-def symp-def transp-def*)

lemma *ltl-prop-equiv-trans[trans]*:

$$\varphi \sim_P \psi \implies \psi \sim_P \chi \implies \varphi \sim_P \chi$$

by (*simp add: ltl-prop-equiv-def*)

lemma *ltl-prop-equiv-true*:

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

using *bot.extremum ltl-prop-entailment.simps(1) ltl-prop-equiv-def* **by** *blast*

lemma *ltl-prop-equiv-false*:

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

by (*meson ltl-prop-entailment.simps(2) ltl-prop-entailment-monotonI ltl-prop-equiv-def top-greatest*)

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

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

by (*simp add: ltl-prop-equiv-def ltl-prop-implies-def*)

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

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

by (*simp add: ltl-prop-equiv-def ltl-prop-implies-def*)

lemma *ltl-prop-implication-implies-ltl-implication*:

$$w \models_n \varphi \implies \varphi \longrightarrow_P \psi \implies w \models_n \psi$$

using *ltl-models-equiv-prop-entailment ltl-prop-implies-def* **by** *blast*

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

$$\varphi \sim_P \psi \implies \varphi \sim_L \psi$$

using *ltl-lang-equiv-def ltl-prop-implication-implies-ltl-implication ltl-prop-implies-equiv*
by *blast*

lemma *ltl-prop-equiv-lt-ltl-lang-equiv[simp]*:

$$(\sim_P) \leq (\sim_L)$$

using *ltl-prop-equiv-implies-ltl-lang-equiv* **by** *blast*

3.3 Constants Equivalence

datatype $tlv = Yes \mid No \mid Maybe$

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

where

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

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

where

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

fun $eval :: 'a \text{ ltl}n \Rightarrow tlv$

where

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

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

$eval\text{-}and \ \varphi \ \psi = No \iff \varphi = No \vee \psi = No$
 $eval\text{-}and \ \varphi \ \psi = Yes \iff \varphi = Yes \wedge \psi = Yes$

unfolding $eval\text{-}and\text{-}def$

by (cases φ ; cases ψ , auto)+

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

$eval\text{-}or \ \varphi \ \psi = Yes \iff \varphi = Yes \vee \psi = Yes$
 $eval\text{-}or \ \varphi \ \psi = No \iff \varphi = No \wedge \psi = No$

unfolding $eval\text{-}or\text{-}def$

by (cases φ ; cases ψ , auto)+

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

$eval \ \varphi = Yes \iff \{\} \models_P \varphi$

$eval \varphi = No \iff \neg UNIV \models_P \varphi$
by (*induction* φ) *auto*

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

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

lemma *ltl-const-equiv-equiv*:

equiv (\sim_C)

unfolding *ltl-const-equiv-def*

by (*intro equivI reflI sympI transpI*) *auto*

lemma *ltl-const-equiv-const*:

$\varphi \sim_C true_n \iff eval \varphi = Yes$

$\varphi \sim_C false_n \iff eval \varphi = No$

unfolding *ltl-const-equiv-def* **by** *force+*

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

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

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

unfolding *ltl-const-equiv-const* **by** *force+*

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

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

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

unfolding *ltl-const-equiv-const* **by** *force+*

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

$\varphi \sim_C prop_n(a) \iff \varphi = prop_n(a)$

$\varphi \sim_C nprop_n(a) \iff \varphi = nprop_n(a)$

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

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

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

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

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

using *ltl-const-equiv-def* **by** *fastforce+*

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

$eval \psi = Maybe \implies \varphi \sim_C \psi \implies \varphi = \psi$

unfolding *ltl-const-equiv-def* **by** *fastforce*

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

$\varphi \sim_C true_n \iff \varphi \sim_P true_n$

$\varphi \sim_C false_n \iff \varphi \sim_P false_n$

unfolding *ltl-const-equiv-const eval-prop-entailment ltl-prop-equiv-def*
by *auto*

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

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

using *ltl-const-equiv-no-const-singleton equivp-reflp[OF ltl-prop-equiv-equivp]*

by *blast*

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

$\varphi \sim_C \psi \implies \varphi \sim_P \psi$

proof (*induction ψ*)

case (*And-ltln $\psi_1 \psi_2$*)

show *?case*

proof (*cases eval (ψ_1 and_n ψ_2)*)

case *Yes*

then have *$\varphi \sim_C \text{true}_n$*

by (*meson And-ltln.prem1 equivp-transp ltl-const-equiv-const(1) ltl-const-equiv-equivp*)

then show *?thesis*

by (*metis (full-types) Yes ltl-const-equiv-const(1) ltl-const-equiv-implies-prop-equiv(1)*)

ltl-prop-equiv-trans ltl-prop-implies-equiv)

next

case *No*

then have *$\varphi \sim_C \text{false}_n$*

by (*meson And-ltln.prem2 equivp-transp ltl-const-equiv-const(2) ltl-const-equiv-equivp*)

then show *?thesis*

by (*metis (full-types) No ltl-const-equiv-const(2) ltl-const-equiv-implies-prop-equiv(2)*)

ltl-prop-equiv-trans ltl-prop-implies-equiv)

next

case *Maybe*

then show *?thesis*

using *And-ltln.prem3 ltl-const-equiv-no-const-prop-equiv* **by** *force*

qed

next

case (*Or-ltln $\psi_1 \psi_2$*)

then show *?case*

proof (*cases eval (ψ_1 or_n ψ_2)*)

case *Yes*

then have *$\varphi \sim_C \text{true}_n$*

```

    by (meson Or-ltln.premis equivp-transp ltl-const-equiv-const(1) ltl-const-equiv-equivp)
  then show ?thesis
    by (metis (full-types) Yes ltl-const-equiv-const(1) ltl-const-equiv-implies-prop-equiv(1)
ltl-prop-equiv-trans ltl-prop-implies-equiv)
  next
    case No

    then have  $\varphi \sim_C \text{false}_n$ 
    by (meson Or-ltln.premis equivp-transp ltl-const-equiv-const(2) ltl-const-equiv-equivp)
    then show ?thesis
    by (metis (full-types) No ltl-const-equiv-const(2) ltl-const-equiv-implies-prop-equiv(2)
ltl-prop-equiv-trans ltl-prop-implies-equiv)
  next
    case Maybe

    then show ?thesis
      using Or-ltln.premis ltl-const-equiv-no-const-prop-equiv by force
    qed
  qed (simp-all add: ltl-const-equiv-implies-prop-equiv equivp-reflp[OF ltl-prop-equiv-equivp])

```

```

lemma ltl-const-equiv-lt-ltl-prop-equiv[simp]:
  ( $\sim_C$ )  $\leq$  ( $\sim_P$ )
  using ltl-const-equiv-implies-ltl-prop-equiv by blast

```

3.4 Quotient types

```

quotient-type 'a ltlL = 'a ltlL / ( $\sim_L$ )
  by (rule ltl-lang-equiv-equivp)

```

```

instantiation ltlL :: (type) equal
begin

```

```

lift-definition ltlL-eq-test :: 'a ltlL  $\Rightarrow$  'a ltlL  $\Rightarrow$  bool is  $\lambda x y. x \sim_L y$ 
  by (metis ltlL.abs-eq-iff)

```

```

definition
  eqL: equal-class.equal  $\equiv$  ltlL-eq-test

```

```

instance
  by (standard; simp add: eqL ltlL-eq-test.rep-eq, metis Quotient-ltlL Quo-
tient-rel-rep)

```

```

end

```

quotient-type $'a \text{ ltl}_P = 'a \text{ ltl} / (\sim_P)$
by (*rule ltl-prop-equiv-equivp*)

instantiation $\text{ltl}_P :: (\text{type}) \text{ equal}$
begin

lift-definition $\text{ltl}_P\text{-eq-test} :: 'a \text{ ltl}_P \Rightarrow 'a \text{ ltl}_P \Rightarrow \text{bool}$ **is** $\lambda x y. x \sim_P y$
by (*metis ltl_P.abs-eq-iff*)

definition
 $\text{eq}_P: \text{equal-class.equal} \equiv \text{ltl}_P\text{-eq-test}$

instance
by (*standard; simp add: eq_P ltl_P-eq-test.rep-eq, metis Quotient-ltl_P Quotient-rel-rep*)

end

quotient-type $'a \text{ ltl}_C = 'a \text{ ltl} / (\sim_C)$
by (*rule ltl-const-equiv-equivp*)

instantiation $\text{ltl}_C :: (\text{type}) \text{ equal}$
begin

lift-definition $\text{ltl}_C\text{-eq-test} :: 'a \text{ ltl}_C \Rightarrow 'a \text{ ltl}_C \Rightarrow \text{bool}$ **is** $\lambda x y. x \sim_C y$
by (*metis ltl_C.abs-eq-iff*)

definition
 $\text{eq}_C: \text{equal-class.equal} \equiv \text{ltl}_C\text{-eq-test}$

instance
by (*standard; simp add: eq_C ltl_C-eq-test.rep-eq, metis Quotient-ltl_C Quotient-rel-rep*)

end

3.5 Cardinality of propositional quotient sets

definition $\text{sat-models} :: 'a \text{ ltl}_P \Rightarrow 'a \text{ ltl} \text{ set set}$
where

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

lemma *Rep-Abs-prop-entailment*[simp]:
 $\mathcal{A} \models_P \text{rep-ltln}_P (\text{abs-ltln}_P \varphi) = \mathcal{A} \models_P \varphi$
by (*metis Quotient3-ltln_P Quotient3-rep-abs ltl-prop-equiv-def*)

lemma *sat-models-Abs*:
 $\mathcal{A} \in \text{sat-models} (\text{abs-ltln}_P \varphi) = \mathcal{A} \models_P \varphi$
by (*simp add: sat-models-def*)

lemma *sat-models-inj*:
inj sat-models

proof (*rule injI*)
fix $\varphi \psi :: 'a \text{ ltln}_P$
assume *sat-models* $\varphi = \text{sat-models } \psi$

then have $\text{rep-ltln}_P \varphi \sim_P \text{rep-ltln}_P \psi$
unfolding *sat-models-def ltl-prop-equiv-def* **by force**

then show $\varphi = \psi$
by (*meson Quotient3-ltln_P Quotient3-rel-rep*)

qed

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

prop-atoms true_n = $\{\}$
| *prop-atoms false_n* = $\{\}$
| *prop-atoms* ($\varphi \text{ and}_n \psi$) = *prop-atoms* $\varphi \cup \text{prop-atoms } \psi$
| *prop-atoms* ($\varphi \text{ or}_n \psi$) = *prop-atoms* $\varphi \cup \text{prop-atoms } \psi$
| *prop-atoms* φ = $\{\varphi\}$

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

nested-prop-atoms true_n = $\{\}$
| *nested-prop-atoms false_n* = $\{\}$
| *nested-prop-atoms* ($\varphi \text{ and}_n \psi$) = *nested-prop-atoms* $\varphi \cup \text{nested-prop-atoms } \psi$
| *nested-prop-atoms* ($\varphi \text{ or}_n \psi$) = *nested-prop-atoms* $\varphi \cup \text{nested-prop-atoms } \psi$
| *nested-prop-atoms* ($X_n \varphi$) = $\{X_n \varphi\} \cup \text{nested-prop-atoms } \varphi$
| *nested-prop-atoms* ($\varphi U_n \psi$) = $\{\varphi U_n \psi\} \cup \text{nested-prop-atoms } \varphi \cup \text{nested-prop-atoms } \psi$
| *nested-prop-atoms* ($\varphi R_n \psi$) = $\{\varphi R_n \psi\} \cup \text{nested-prop-atoms } \varphi \cup \text{nested-prop-atoms } \psi$
| *nested-prop-atoms* ($\varphi W_n \psi$) = $\{\varphi W_n \psi\} \cup \text{nested-prop-atoms } \varphi \cup$

nested-prop-atoms ψ
 | *nested-prop-atoms* $(\varphi M_n \psi) = \{\varphi M_n \psi\} \cup \textit{nested-prop-atoms } \varphi \cup$
nested-prop-atoms ψ
 | *nested-prop-atoms* $\varphi = \{\varphi\}$

lemma *prop-atoms-nested-prop-atoms*:
prop-atoms $\varphi \subseteq \textit{nested-prop-atoms } \varphi$
by (*induction* φ) *auto*

lemma *prop-atoms-subfrmlsn*:
prop-atoms $\varphi \subseteq \textit{subfrmlsn } \varphi$
by (*induction* φ) *auto*

lemma *nested-prop-atoms-subfrmlsn*:
nested-prop-atoms $\varphi \subseteq \textit{subfrmlsn } \varphi$
by (*induction* φ) *auto*

lemma *prop-atoms-notin[simp]*:
 $\textit{true}_n \notin \textit{prop-atoms } \varphi$
 $\textit{false}_n \notin \textit{prop-atoms } \varphi$
 $\varphi_1 \textit{ and}_n \varphi_2 \notin \textit{prop-atoms } \varphi$
 $\varphi_1 \textit{ or}_n \varphi_2 \notin \textit{prop-atoms } \varphi$
by (*induction* φ) *auto*

lemma *nested-prop-atoms-notin[simp]*:
 $\textit{true}_n \notin \textit{nested-prop-atoms } \varphi$
 $\textit{false}_n \notin \textit{nested-prop-atoms } \varphi$
 $\varphi_1 \textit{ and}_n \varphi_2 \notin \textit{nested-prop-atoms } \varphi$
 $\varphi_1 \textit{ or}_n \varphi_2 \notin \textit{nested-prop-atoms } \varphi$
by (*induction* φ) *auto*

lemma *prop-atoms-finite*:
finite (*prop-atoms* φ)
by (*induction* φ) *auto*

lemma *nested-prop-atoms-finite*:
finite (*nested-prop-atoms* φ)
by (*induction* φ) *auto*

lemma *prop-atoms-entailment-iff*:
 $\varphi \in \textit{prop-atoms } \psi \implies \mathcal{A} \models_P \varphi \longleftrightarrow \varphi \in \mathcal{A}$
by (*induction* φ) *auto*

lemma *prop-atoms-entailment-inter*:

prop-atoms $\varphi \subseteq P \implies (\mathcal{A} \cap P) \models_P \varphi = \mathcal{A} \models_P \varphi$
by (*induction* φ) *auto*

lemma *nested-prop-atoms-entailment-inter*:
nested-prop-atoms $\varphi \subseteq P \implies (\mathcal{A} \cap P) \models_P \varphi = \mathcal{A} \models_P \varphi$
by (*induction* φ) *auto*

lemma *sat-models-inter-inj-helper*:
assumes
prop-atoms $\varphi \subseteq P$
and
prop-atoms $\psi \subseteq P$
and
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 –
from *assms* **have** $\forall \mathcal{A}. (\mathcal{A} \cap P) \models_P \varphi \longleftrightarrow (\mathcal{A} \cap P) \models_P \psi$
by (*auto simp: sat-models-Abs*)

with *assms* **show** $\varphi \sim_P \psi$
by (*simp add: prop-atoms-entailment-inter ltl-prop-equiv-def*)
qed

lemma *sat-models-inter-inj*:
inj-on $(\lambda \varphi. \text{sat-models } \varphi \cap \text{Pow } P) \{ \text{abs-ltln}_P \varphi \mid \varphi. \text{prop-atoms } \varphi \subseteq P \}$
by (*auto simp: inj-on-def sat-models-inter-inj-helper ltln_P.abs-eq-iff*)

lemma *sat-models-pow-pow*:
 $\{ \text{sat-models } (\text{abs-ltln}_P \varphi) \cap \text{Pow } P \mid \varphi. \text{prop-atoms } \varphi \subseteq P \} \subseteq \text{Pow } (\text{Pow } P)$
by (*auto simp: sat-models-def*)

lemma *sat-models-finite*:
finite $P \implies \text{finite } \{ \text{sat-models } (\text{abs-ltln}_P \varphi) \cap \text{Pow } P \mid \varphi. \text{prop-atoms } \varphi \subseteq P \}$
using *sat-models-pow-pow finite-subset* **by** *fastforce*

lemma *sat-models-card*:
finite $P \implies \text{card } (\{ \text{sat-models } (\text{abs-ltln}_P \varphi) \cap \text{Pow } P \mid \varphi. \text{prop-atoms } \varphi \subseteq P \}) \leq 2 \wedge 2 \wedge \text{card } P$
by (*metis (mono-tags, lifting) sat-models-pow-pow Pow-def card-Pow card-mono finite-Collect-subsets*)

lemma *image-filter*:

$f \text{ ' } \{g \ a \mid a. P \ a\} = \{f \ (g \ a) \mid a. P \ a\}$
by *blast*

lemma *prop-equiv-finite*:

$finite \ P \implies finite \ \{abs\text{-}ltn_P \ \psi \mid \psi. \text{prop-atoms } \psi \subseteq P\}$
by (*auto simp: image-filter sat-models-finite finite-imageD[OF - sat-models-inter-inj]*)

lemma *prop-equiv-card*:

$finite \ P \implies card \ \{abs\text{-}ltn_P \ \psi \mid \psi. \text{prop-atoms } \psi \subseteq P\} \leq 2 \wedge 2 \wedge card \ P$
by (*auto simp: image-filter sat-models-card card-image[OF sat-models-inter-inj, symmetric]*)

lemma *prop-equiv-subset*:

$\{abs\text{-}ltn_P \ \psi \mid \psi. \text{nested-prop-atoms } \psi \subseteq P\} \subseteq \{abs\text{-}ltn_P \ \psi \mid \psi. \text{prop-atoms } \psi \subseteq P\}$
using *prop-atoms-nested-prop-atoms* **by** *blast*

lemma *prop-equiv-finite'*:

$finite \ P \implies finite \ \{abs\text{-}ltn_P \ \psi \mid \psi. \text{nested-prop-atoms } \psi \subseteq P\}$
using *prop-equiv-finite prop-equiv-subset finite-subset* **by** *fast*

lemma *prop-equiv-card'*:

$finite \ P \implies card \ \{abs\text{-}ltn_P \ \psi \mid \psi. \text{nested-prop-atoms } \psi \subseteq P\} \leq 2 \wedge 2 \wedge card \ P$
by (*metis (mono-tags, lifting) prop-equiv-card prop-equiv-subset prop-equiv-finite card-mono le-trans*)

3.6 Substitution

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

where

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

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

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

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

by (induction φ) (auto split: option.split)

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

$\varphi \longrightarrow_P \psi \implies \text{subst } \varphi \ m \longrightarrow_P \text{subst } \psi \ m$

$\varphi \sim_P \psi \implies \text{subst } \varphi \ m \sim_P \text{subst } \psi \ m$

unfolding *ltl-prop-equiv-def ltl-prop-implies-def ltl-prop-equiv-subst-S* by *blast+*

lemma *eval-subst*:

$\text{eval } \varphi = \text{Yes} \implies \text{eval } (\text{subst } \varphi \ m) = \text{Yes}$

$\text{eval } \varphi = \text{No} \implies \text{eval } (\text{subst } \varphi \ m) = \text{No}$

by (*meson empty-subsetI eval-prop-entailment ltl-prop-entailment-monotonI ltl-prop-equiv-subst-S subset-UNIV*) $+$

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

$\varphi \sim_C \psi \implies \text{subst } \varphi \ m \sim_C \text{subst } \psi \ m$

unfolding *ltl-const-equiv-def*

by (*cases eval ψ*) (*metis eval-subst(1), metis eval-subst(2), blast*)

3.7 Order of Equivalence Relations

locale *ltl-equivalence* =

fixes

$\text{eq} :: 'a \ \text{ltln} \Rightarrow 'a \ \text{ltln} \Rightarrow \text{bool}$ (**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*:

$\varphi \sim \psi \implies w \models_n \varphi = w \models_n \psi$

using *le-lang-equiv ltl-lang-equiv-def* **by** *blast*

lemma *const-implies-ge*:

$\varphi \sim_C \psi \implies \varphi \sim \psi$

using *ge-const-equiv* **by** *blast*

lemma *eq-implies-lang*:

$\varphi \sim \psi \implies \varphi \sim_L \psi$

using *le-lang-equiv* **by** *blast*

lemma *eq-refl*[*simp*]:
 $\varphi \sim \varphi$
by (*meson eq-equivp equivp-reflp*)

lemma *eq-sym*[*sym*]:
 $\varphi \sim \psi \implies \psi \sim \varphi$
by (*meson eq-equivp equivp-symp*)

lemma *eq-trans*[*trans*]:
 $\varphi \sim \psi \implies \psi \sim \chi \implies \varphi \sim \chi$
by (*meson eq-equivp equivp-transp*)

end

interpretation *ltl-lang-equivalence*: *ltl-equivalence* (\sim_L)
using *ltl-lang-equiv-equivp ltl-const-equiv-lt-ltl-prop-equiv ltl-prop-equiv-lt-ltl-lang-equiv*
by *unfold-locales blast+*

interpretation *ltl-prop-equivalence*: *ltl-equivalence* (\sim_P)
using *ltl-prop-equiv-equivp ltl-const-equiv-lt-ltl-prop-equiv ltl-prop-equiv-lt-ltl-lang-equiv*
by *unfold-locales blast+*

interpretation *ltl-const-equivalence*: *ltl-equivalence* (\sim_C)
using *ltl-const-equiv-equivp ltl-const-equiv-lt-ltl-prop-equiv ltl-prop-equiv-lt-ltl-lang-equiv*
by *unfold-locales blast+*

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*:

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

unfolding *min-set-def* **by** *blast*

lemma *min-set-subset*:

$$\text{min-set } X \subseteq X$$

by (*auto simp: min-set-def*)

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

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

by (*auto simp: min-set-def*)

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

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

using *min-set-subset* **by** *blast*

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

$$\text{min-set } \{x\} = \{x\}$$

by (*auto simp: min-set-def*)

lemma *min-set-finite*:

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

by (*simp add: min-set-def*)

lemma *min-set-obtains-helper*:

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

proof (*induction fcard A arbitrary: A rule: less-induct*)

case *less*

then have $(\forall A'. A' \notin B \vee \neg A' \sqsubseteq A \vee A' = A) \vee (\exists A'. A' \sqsubseteq A \wedge A' \in \text{min-set } B)$

by (*metis (no-types) dual-order.trans order.not-eq-order-implies-strict pfssubset-fcard-mono*)

then show *?case*

using *less.premis min-set-def* **by** *auto*

qed

lemma *min-set-obtains*:

assumes $A \in B$
obtains C where $C \sqsubseteq A$ and $C \in \text{min-set } B$
using *min-set-obtains-helper* *assms* **by** *metis*

4.2 Minimal operators on sets

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

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]*:
 $A \otimes_m A = \text{min-set } A$
by (*auto simp: min-product-def product-def min-set-def*) *fastforce*

lemma *min-union-idem[simp]*:
 $A \cup_m A = \text{min-set } A$
by (*simp add: min-union-def*)

lemma *product-empty[simp]*:
 $A \otimes \{\} = \{\}$
 $\{\} \otimes A = \{\}$
by (*simp-all add: product-def*)

lemma *min-product-empty[simp]*:
 $A \otimes_m \{\} = \{\}$
 $\{\} \otimes_m A = \{\}$
by (*simp-all add: min-product-def*)

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

$A \cup_m \{\} = \text{min-set } A$
 $\{\} \cup_m A = \text{min-set } A$
by (*simp-all add: min-union-def*)

lemma *product-empty-singleton[simp]*:
 $A \otimes \{\{\|\}\} = A$
 $\{\{\|\}\} \otimes A = A$
by (*simp-all add: product-def*)

lemma *min-product-empty-singleton[simp]*:
 $A \otimes_m \{\{\|\}\} = \text{min-set } A$
 $\{\{\|\}\} \otimes_m A = \text{min-set } A$
by (*simp-all add: min-product-def*)

lemma *product-singleton-singleton*:
 $A \otimes \{\{|x|\}\} = \text{finsert } x \text{ ' } A$
 $\{\{|x|\}\} \otimes A = \text{finsert } x \text{ ' } A$
unfolding *product-def* **by** *blast+*

lemma *product-mono*:
 $A \subseteq B \implies A \otimes C \subseteq B \otimes C$
 $B \subseteq C \implies A \otimes B \subseteq A \otimes C$
unfolding *product-def* **by** *auto*

lemma *product-finite*:
 $\text{finite } A \implies \text{finite } B \implies \text{finite } (A \otimes B)$
by (*simp add: product-def finite-image-set2*)

lemma *min-product-finite*:
 $\text{finite } A \implies \text{finite } B \implies \text{finite } (A \otimes_m B)$
by (*metis min-product-def product-finite min-set-finite*)

lemma *min-union-finite*:
 $\text{finite } A \implies \text{finite } B \implies \text{finite } (A \cup_m B)$
by (*simp add: min-union-def min-set-finite*)

lemma *product-set-infinite[simp]*:
 $\text{infinite } X \implies \bigotimes X = \{\{\|\}\}$
by (*simp add: product-set-def*)

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

infinite $X \implies \bigotimes_m X = \{\{\|\}\}$
by (*simp add: min-product-set-def*)

lemma *product-comm*:
 $A \otimes B = B \otimes A$
unfolding *product-def* **by** *blast*

lemma *min-product-comm*:
 $A \otimes_m B = B \otimes_m A$
unfolding *min-product-def*
by (*simp add: product-comm*)

lemma *min-union-comm*:
 $A \cup_m B = B \cup_m A$
unfolding *min-union-def*
by (*simp add: sup commute*)

lemma *product-iff*:
 $x \in A \otimes B \longleftrightarrow (\exists a \in A. \exists b \in B. x = a \mid\cup\mid b)$
unfolding *product-def* **by** *blast*

lemma *min-product-iff*:
 $x \in A \otimes_m B \longleftrightarrow (\exists a \in A. \exists b \in B. x = a \mid\cup\mid b) \wedge (\forall a \in A. \forall b \in B. a \mid\cup\mid b \mid\subseteq\mid x \longrightarrow a \mid\cup\mid b = x)$
unfolding *min-product-def min-set-iff product-iff product-def* **by** *blast*

lemma *min-union-iff*:
 $x \in A \cup_m B \longleftrightarrow x \in A \cup B \wedge (\forall a \in A. a \mid\subseteq\mid x \longrightarrow a = x) \wedge (\forall b \in B. b \mid\subseteq\mid x \longrightarrow b = x)$
unfolding *min-union-def min-set-iff* **by** *blast*

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

proof
fix x
assume $x \in (\text{min-set } A) \otimes_m B$

then obtain a b **where** $a \in \text{min-set } A$ **and** $b \in B$ **and** $x = a \mid\cup\mid b$ **and**
 $1: \forall a \in \text{min-set } A. \forall b \in B. a \mid\cup\mid b \mid\subseteq\mid x \longrightarrow a \mid\cup\mid b = x$
unfolding *min-product-iff* **by** *blast*

moreover

{

fix $a' b'$

assume $a' \in A$ **and** $b' \in B$ **and** $a' \sqcup b' \sqsubseteq x$

then obtain a'' **where** $a'' \sqsubseteq a'$ **and** $a'' \in \text{min-set } A$

using *min-set-obtains* **by** *metis*

then have $a'' \sqcup b' = x$

by (*metis (full-types) 1 <b' ∈ B> <a' ∪ b' ⊆ x> dual-order.trans le-sup-iff*)

then have $a' \sqcup b' = x$

using $\langle a' \sqcup b' \sqsubseteq x \rangle \langle a'' \sqsubseteq a' \rangle$ **by** *blast*

}

ultimately show $x \in A \otimes_m B$

by (*metis min-product-iff min-set-iff*)

next

fix x

assume $x \in A \otimes_m B$

then have $1: x \in A \otimes B$ **and** $\forall y \in A \otimes B. y \sqsubseteq x \longrightarrow y = x$

unfolding *min-product-def min-set-iff* **by** *simp+*

then have $2: \forall y \in \text{min-set } A \otimes B. y \sqsubseteq x \longrightarrow y = x$

by (*metis product-iff min-set-iff*)

then have $x \in \text{min-set } A \otimes B$

by (*metis 1 union-mono min-set-obtains order-refl product-iff*)

then show $x \in \text{min-set } A \otimes_m B$

by (*simp add: 2 min-product-def min-set-iff*)

qed

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

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

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

using *min-product-comm min-set-min-product-helper* **by** *blast+*

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 (*unfold min-union-def min-set-def, safe*)
show $\bigwedge x xa xb. \llbracket \forall xa \in \{y \in A. \forall x \in A. x \sqsubseteq y \rightarrow x = y\} \cup B. xa \sqsubseteq x \rightarrow xa = x; x \in B; xa \sqsubseteq x; xb \in x; xa \in A \rrbracket \Longrightarrow xb \in xa$
by (*metis (mono-tags) UnCI dual-order.trans fequalityI min-set-def min-set-obtains*)
next
show $\bigwedge x xa xb. \llbracket \forall xa \in A \cup \{y \in B. \forall x \in B. x \sqsubseteq y \rightarrow x = y\}. xa \sqsubseteq x \rightarrow xa = x; x \in A; xa \sqsubseteq x; xb \in x; xa \in B \rrbracket \Longrightarrow xb \in xa$
by (*metis (mono-tags) UnCI dual-order.trans fequalityI min-set-def min-set-obtains*)
qed *blast+*

lemma *product-assoc[simp]*:
 $(A \otimes B) \otimes C = A \otimes (B \otimes C)$
proof (*unfold product-def, safe*)
fix *a b c*
assume *a* $\in A$ **and** *c* $\in C$ **and** *b* $\in B$
then have *b* $\cup c \in \{b \cup c \mid b \in B \wedge c \in C\}$
by *blast*
then show $\exists a' bc. a \cup b \cup c = a' \cup bc \wedge a' \in A \wedge bc \in \{b \cup c \mid b \in B \wedge c \in C\}$
using $\langle a \in A \rangle$ **by** (*metis (no-types) inf-sup-aci(5) sup-left-commute*)
qed (*metis (mono-tags, lifting) mem-Collect-eq sup-assoc*)

lemma *min-product-assoc[simp]*:
 $(A \otimes_m B) \otimes_m C = A \otimes_m (B \otimes_m C)$
unfolding *min-product-def[of A B] min-product-def[of B C]*
by *simp (simp add: min-product-def)*

lemma *min-union-assoc[simp]*:
 $(A \cup_m B) \cup_m C = A \cup_m (B \cup_m C)$
unfolding *min-union-def[of A B] min-union-def[of B C]*
by *simp (simp add: min-union-def sup-assoc)*

lemma *min-product-comp*:
 $a \in A \Longrightarrow b \in B \Longrightarrow \exists c. c \sqsubseteq (a \cup b) \wedge c \in A \otimes_m B$
by (*metis (mono-tags, lifting) mem-Collect-eq min-product-def product-def min-set-obtains*)

lemma *min-union-comp*:
 $a \in A \Longrightarrow \exists c. c \sqsubseteq a \wedge c \in A \cup_m B$
by (*metis Un-iff min-set-obtains min-union-def*)

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

proof *unfold-locales*

have $\bigwedge x y z. x \otimes (y \otimes z) = y \otimes (x \otimes z)$

by (*simp only: product-assoc[symmetric]*) (*simp only: product-comm*)

then show $\bigwedge x y. (\otimes) y \circ (\otimes) x = (\otimes) x \circ (\otimes) y$

by *fastforce*

qed

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

proof *unfold-locales*

have $\bigwedge x y z. x \otimes_m (y \otimes_m z) = y \otimes_m (x \otimes_m z)$

by (*simp only: min-product-assoc[symmetric]*) (*simp only: min-product-comm*)

then show $\bigwedge x y. (\otimes_m) y \circ (\otimes_m) x = (\otimes_m) x \circ (\otimes_m) y$

by *fastforce*

next

have $\bigwedge x y. x \otimes_m (x \otimes_m y) = x \otimes_m y$

by (*simp add: min-product-assoc[symmetric]*)

then show $\bigwedge x. (\otimes_m) x \circ (\otimes_m) x = (\otimes_m) x$

by *fastforce*

qed

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

proof *unfold-locales*

have $\bigwedge x y z. x \cup_m (y \cup_m z) = y \cup_m (x \cup_m z)$

by (*simp only: min-union-assoc[symmetric]*) (*simp only: min-union-comm*)

then show $\bigwedge x y. (\cup_m) y \circ (\cup_m) x = (\cup_m) x \circ (\cup_m) y$

by *fastforce*

next

have $\bigwedge x y. x \cup_m (x \cup_m y) = x \cup_m y$

by (*simp add: min-union-assoc[symmetric]*)

then show $\bigwedge x. (\cup_m) x \circ (\cup_m) x = (\cup_m) x$

by *fastforce*

qed

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

$\otimes \{\} = \{\{\|\}\}$
 $\otimes \{\{\}\} = \{\}$
 $\otimes \{\{\{\|\}\}\} = \{\{\|\}\}$
by (*simp-all add: product-set-def*)

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

$\otimes_m \{\} = \{\{\|\}\}$
 $\otimes_m \{\{\}\} = \{\}$
 $\otimes_m \{\{\{\|\}\}\} = \{\{\|\}\}$
by (*simp-all add: min-product-set-def*)

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

$\otimes (\text{set } xs) = \text{fold product (remdups } xs) \{\{\|\}\}$
by (*simp add: product-set-def product-set-thms.fold-set-fold-remdups*)

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

$\otimes_m (\text{set } xs) = \text{fold min-product (remdups } xs) \{\{\|\}\}$
by (*simp add: min-product-set-def min-product-set-thms.fold-set-fold-remdups*)

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

$\text{finite } X \implies \otimes (\text{insert } x X) = x \otimes (\otimes (X - \{x\}))$
unfolding *product-set-def product-set-thms.fold-insert-remove ..*

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

$\text{finite } X \implies \otimes_m (\text{insert } x X) = x \otimes_m (\otimes_m X)$
unfolding *min-product-set-def min-product-set-thms.fold-insert-idem ..*

lemma *min-product-subseteq*:

$x \in A \otimes_m B \implies \exists a. a \sqsubseteq x \wedge a \in A$
by (*metis union-upper1 min-product-iff*)

lemma *min-product-set-subseteq*:

$\text{finite } X \implies x \in \otimes_m X \implies A \in X \implies \exists a \in A. a \sqsubseteq x$
by (*induction X rule: finite-induct*) (*blast, metis finite-insert insert-absorb min-product-set-insert min-product-subseteq*)

lemma *min-set-product-set*:

$\otimes_m A = \text{min-set } (\otimes A)$
by (*cases finite A, induction A rule: finite-induct*) (*simp-all add: min-product-set-def product-set-def, metis min-product-def*)

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

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

by (simp add: min-product-def)

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

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

by (simp add: min-union-def)

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

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

by (induction X rule: finite-induct, auto simp add: min-product-set-def min-set-iff)

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

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

by (induction X rule: finite-induct) simp-all

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

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

by (induction X rule: finite-induct) simp-all

lemma product-set-finite:

$(\bigwedge x. x \in X \implies finite\ x) \implies finite\ (\bigotimes X)$

by (cases finite X, rotate-tac, induction X rule: finite-induct) (simp-all add: product-set-def, insert product-finite, blast)

lemma min-product-set-finite:

$(\bigwedge x. x \in X \implies finite\ x) \implies finite\ (\bigotimes_m X)$

by (cases finite X, rotate-tac, induction X rule: finite-induct) (simp-all add: min-product-set-def, insert min-product-finite, blast)

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 = {|| φ ||}

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

where

min-dnf true_n = {||}

| min-dnf false_n = {}

| $\text{min-dnf } (\varphi \text{ and}_n \psi) = (\text{min-dnf } \varphi) \otimes_m (\text{min-dnf } \psi)$
| $\text{min-dnf } (\varphi \text{ or}_n \psi) = (\text{min-dnf } \varphi) \cup_m (\text{min-dnf } \psi)$
| $\text{min-dnf } \varphi = \{\{|\varphi|\}\}$

lemma *dnf-min-set*:

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

by (*induction* φ) (*simp-all, simp-all only: min-product-def min-union-def*)

lemma *dnf-finite*:

$\text{finite } (\text{dnf } \varphi)$

by (*induction* φ) (*auto simp: product-finite*)

lemma *min-dnf-finite*:

$\text{finite } (\text{min-dnf } \varphi)$

by (*induction* φ) (*auto simp: min-product-finite min-union-finite*)

lemma *dnf-Abs-fset[simp]*:

$\text{fset } (\text{Abs-fset } (\text{dnf } \varphi)) = \text{dnf } \varphi$

by (*simp add: dnf-finite Abs-fset-inverse*)

lemma *min-dnf-Abs-fset[simp]*:

$\text{fset } (\text{Abs-fset } (\text{min-dnf } \varphi)) = \text{min-dnf } \varphi$

by (*simp add: min-dnf-finite Abs-fset-inverse*)

lemma *dnf-prop-atoms*:

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

by (*induction* φ *arbitrary:* Φ) (*auto simp: product-def*)

lemma *min-dnf-prop-atoms*:

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

using *dnf-min-set dnf-prop-atoms min-set-subset* **by** *blast*

lemma *min-dnf-atoms-dnf*:

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

proof (*induction* φ)

case *True-ltln*

then show *?case*

using *min-dnf-prop-atoms prop-atoms-notin(1)* **by** *blast*

next

case *False-ltln*

then show *?case*

using *min-dnf-prop-atoms prop-atoms-notin(2)* **by** *blast*

next

case (*And-ltln* φ_1 φ_2)

then show *?case*
using *min-dnf-prop-atoms prop-atoms-notin(3)* **by force**
next
case (*Or-ltln* φ_1 φ_2)
then show *?case*
using *min-dnf-prop-atoms prop-atoms-notin(4)* **by force**
qed *auto*

lemma *min-dnf-min-set[simp]*:
 $\text{min-set } (\text{min-dnf } \varphi) = \text{min-dnf } \varphi$
by (*induction* φ) (*simp-all add: min-set-def min-product-def min-union-def, blast+*)

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

then show $\exists B. \text{fset } B \subseteq \mathcal{A} \wedge B \in \text{min-dnf } \varphi$

proof (*induction* φ *arbitrary: A*)

case (*And-ltln* φ_1 φ_2)

then obtain B_1 B_2 **where** $1: \text{fset } B_1 \subseteq \mathcal{A} \wedge B_1 \in \text{min-dnf } \varphi_1$ **and** $2: \text{fset } B_2 \subseteq \mathcal{A} \wedge B_2 \in \text{min-dnf } \varphi_2$
by *fastforce*

then obtain C **where** $C \sqsubseteq | B_1 \cup | B_2$ **and** $C \in \text{min-dnf } \varphi_1 \otimes_m \text{min-dnf } \varphi_2$
using *min-product-comp* **by** *metis*

then show *?case*

by (*metis 1 2 le-sup-iff min-dnf.simps(3) sup.absorb-iff1 sup-fset.rep-eq*)

next

case (*Or-ltln* φ_1 φ_2)

{
assume $\mathcal{A} \models_P \varphi_1$

then obtain B **where** $1: \text{fset } B \subseteq \mathcal{A} \wedge B \in \text{min-dnf } \varphi_1$
using *Or-ltln* **by** *fastforce*

then obtain C **where** $C \sqsubseteq | B$ **and** $C \in \text{min-dnf } \varphi_1 \cup_m \text{min-dnf } \varphi_2$
using *min-union-comp* **by** *metis*


```

    then have ?case
      by (metis 1 dual-order.trans less-eq-fset.rep-eq min-dnf.simps(4))
    }

  moreover

  {
    assume  $\mathcal{A} \models_P \varphi_2$ 

    then obtain  $B$  where  $2: \text{fset } B \subseteq \mathcal{A} \wedge B \in \text{min-dnf } \varphi_2$ 
      using Or-ltln by fastforce

    then obtain  $C$  where  $C \sqsubseteq B$  and  $C \in \text{min-dnf } \varphi_1 \cup_m \text{min-dnf } \varphi_2$ 
      using min-union-comp min-union-comm by metis

    then have ?case
      by (metis 2 dual-order.trans less-eq-fset.rep-eq min-dnf.simps(4))
    }

  ultimately show ?case
    using Or-ltln.prem by auto
  qed simp-all
next
assume  $\exists B. \text{fset } B \subseteq \mathcal{A} \wedge B \in \text{min-dnf } \varphi$ 

then obtain  $B$  where  $\text{fset } B \subseteq \mathcal{A}$  and  $B \in \text{min-dnf } \varphi$ 
  by auto

then have  $\text{fset } B \models_P \varphi$ 
  by (induction  $\varphi$  arbitrary:  $B$ ) (auto simp: min-set-def min-product-def
product-def min-union-def)

then show  $\mathcal{A} \models_P \varphi$ 
  using  $\langle \text{fset } B \subseteq \mathcal{A} \rangle$  by blast
qed

lemma ltl-prop-implies-min-dnf:
 $\varphi \longrightarrow_P \psi = (\forall A \in \text{min-dnf } \varphi. \exists B \in \text{min-dnf } \psi. B \sqsubseteq A)$ 
  by (meson less-eq-fset.rep-eq ltl-prop-implies-def min-dnf-iff-prop-assignment-subset
order-refl dual-order.trans)

lemma ltl-prop-equiv-min-dnf:

```

$\varphi \sim_P \psi = (\text{min-dnf } \varphi = \text{min-dnf } \psi)$

proof

assume $\varphi \sim_P \psi$

then have $\bigwedge x. x \in \text{min-set } (\text{min-dnf } \varphi) \longleftrightarrow x \in \text{min-set } (\text{min-dnf } \psi)$

unfolding *ltl-prop-implies-equiv ltl-prop-implies-min-dnf min-set-iff*

by *fastforce*

then show $\text{min-dnf } \varphi = \text{min-dnf } \psi$

by *auto*

qed (*simp add: ltl-prop-equiv-def min-dnf-iff-prop-assignment-subset*)

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

$\text{min-dnf } (\text{rep-ltln}_P (\text{abs-ltln}_P \varphi)) = \text{min-dnf } \varphi$

by (*simp add: ltl-prop-equiv-min-dnf[symmetric] Quotient3-ltln_P rep-abs-rsp-left*)

4.4 Folding of and_n and or_n over Finite Sets

definition $\text{And}_n :: 'a \text{ ltln set} \Rightarrow 'a \text{ ltln}$

where

$\text{And}_n \Phi \equiv \text{SOME } \varphi. \text{fold-graph And-ltln True-ltln } \Phi \varphi$

definition $\text{Or}_n :: 'a \text{ ltln set} \Rightarrow 'a \text{ ltln}$

where

$\text{Or}_n \Phi \equiv \text{SOME } \varphi. \text{fold-graph Or-ltln False-ltln } \Phi \varphi$

lemma *fold-graph-And_n*:

$\text{finite } \Phi \Longrightarrow \text{fold-graph And-ltln True-ltln } \Phi (\text{And}_n \Phi)$

unfolding *And_n-def* **by** (*rule someI2-ex[OF finite-imp-fold-graph]*)

lemma *fold-graph-Or_n*:

$\text{finite } \Phi \Longrightarrow \text{fold-graph Or-ltln False-ltln } \Phi (\text{Or}_n \Phi)$

unfolding *Or_n-def* **by** (*rule someI2-ex[OF finite-imp-fold-graph]*)

lemma *Or_n-empty[simp]*:

$\text{Or}_n \{\} = \text{False-ltln}$

by (*metis empty-fold-graphE finite.emptyI fold-graph-Or_n*)

lemma *And_n-empty[simp]*:

$\text{And}_n \{\} = \text{True-ltln}$

by (*metis empty-fold-graphE finite.emptyI fold-graph-And_n*)

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

by *unfold-locales fastforce*

interpretation *dnf-product-thms: Finite-Set.comp-fun-commute* $\lambda\varphi. (\otimes) (f \varphi)$

by *unfold-locales (simp add: product-set-thms.comp-fun-commute)*

— Copied from locale *comp-fun-commute*

lemma *fold-graph-finite:*

assumes *fold-graph f z A y*

shows *finite A*

using *assms* by *induct simp-all*

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

lemma *And_n-dnf:*

finite $\Phi \implies \text{dnf } (And_n \Phi) = \text{Finite-Set.fold } (\lambda\varphi. (\otimes) (\text{dnf } \varphi)) \{\{\}\} \Phi$

proof (*drule fold-graph-And_n, induction rule: fold-graph.induct*)

case (*insertI x A y*)

then have *finite A*

using *fold-graph-finite* by *fast*

then show *?case*

using *insertI* by *auto*

qed *simp*

lemma *Or_n-dnf:*

finite $\Phi \implies \text{dnf } (Or_n \Phi) = \text{Finite-Set.fold } (\lambda\varphi. (\cup) (\text{dnf } \varphi)) \{\} \Phi$

proof (*drule fold-graph-Or_n, induction rule: fold-graph.induct*)

case (*insertI x A y*)

then have *finite A*

using *fold-graph-finite* by *fast*

then show *?case*

using *insertI* by *auto*

qed *simp*

And_n and Or_n are injective on finite sets.

lemma *And_n-inj:*

inj-on $And_n \{s. \text{finite } s\}$

proof (*standard, simp*)

fix $x y :: 'a \text{ tln set}$

assume *finite x* and *finite y*

then have 1: *fold-graph And-ltln True-ltln x (And_n x)* **and** 2: *fold-graph And-ltln True-ltln y (And_n y)*
using *fold-graph-And_n* **by** *blast+*

assume $And_n x = And_n y$

with 1 **show** $x = y$

proof (*induction rule: fold-graph.induct*)

case *emptyI*

then show *?case*

using 2 *fold-graph.cases* **by** *force*

next

case (*insertI x A y*)

with 2 **show** *?case*

proof (*induction arbitrary: x A y rule: fold-graph.induct*)

case (*insertI x A y*)

then show *?case*

by (*metis fold-graph.cases insertI1 ltln.distinct(7) ltln.inject(3)*)

qed *blast*

qed

qed

lemma *Or_n-inj*:

inj-on Or_n {s. finite s}

proof (*standard, simp*)

fix $x y :: 'a\ ltln\ set$

assume *finite x and finite y*

then have 1: *fold-graph Or-ltln False-ltln x (Or_n x)* **and** 2: *fold-graph Or-ltln False-ltln y (Or_n y)*

using *fold-graph-Or_n* **by** *blast+*

assume $Or_n x = Or_n y$

with 1 **show** $x = y$

proof (*induction rule: fold-graph.induct*)

case *emptyI*

then show *?case*

using 2 *fold-graph.cases* **by** *force*

next

case (*insertI x A y*)

with 2 **show** *?case*

proof (*induction arbitrary: x A y rule: fold-graph.induct*)

```

    case (insertI x A y)
  then show ?case
    by (metis fold-graph.cases insertI1 ltln.distinct(27) ltln.inject(4))
  qed blast
qed
qed

```

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

lemma *And_n-semantics:*

$finite\ \Phi \implies w \models_n And_n\ \Phi \iff (\forall \varphi \in \Phi. w \models_n \varphi)$

proof –

assume $finite\ \Phi$

have $\bigwedge \psi. fold-graph\ And-ltln\ True-ltln\ \Phi\ \psi \implies w \models_n \psi \iff (\forall \varphi \in \Phi. w \models_n \varphi)$

by (rule *fold-graph.induct*) *auto*

then show ?thesis

using *fold-graph-And_n[OF ⟨finite Φ⟩]* by *simp*

qed

lemma *Or_n-semantics:*

$finite\ \Phi \implies w \models_n Or_n\ \Phi \iff (\exists \varphi \in \Phi. w \models_n \varphi)$

proof –

assume $finite\ \Phi$

have $\bigwedge \psi. fold-graph\ Or-ltln\ False-ltln\ \Phi\ \psi \implies w \models_n \psi \iff (\exists \varphi \in \Phi. w \models_n \varphi)$

by (rule *fold-graph.induct*) *auto*

then show ?thesis

using *fold-graph-Or_n[OF ⟨finite Φ⟩]* by *simp*

qed

lemma *And_n-prop-semantics:*

$finite\ \Phi \implies \mathcal{A} \models_P And_n\ \Phi \iff (\forall \varphi \in \Phi. \mathcal{A} \models_P \varphi)$

proof –

assume $finite\ \Phi$

have $\bigwedge \psi. fold-graph\ And-ltln\ True-ltln\ \Phi\ \psi \implies \mathcal{A} \models_P \psi \iff (\forall \varphi \in \Phi. \mathcal{A} \models_P \varphi)$

by (rule *fold-graph.induct*) *auto*

then show ?thesis

using *fold-graph-And_n[OF ⟨finite Φ⟩]* by *simp*

qed

lemma *Or_n-prop-semantics:*

$finite\ \Phi \implies \mathcal{A} \models_P Or_n\ \Phi \iff (\exists \varphi \in \Phi. \mathcal{A} \models_P \varphi)$

proof –

assume *finite* Φ
have $\bigwedge \psi. \text{fold-graph } Or\text{-ltn } False\text{-ltn } \Phi \psi \implies \mathcal{A} \models_P \psi \iff (\exists \varphi \in \Phi. \mathcal{A} \models_P \varphi)$
by (*rule fold-graph.induct*) *auto*
then show *?thesis*
using *fold-graph-Or_n[OF ⟨finite Φ⟩]* **by** *simp*
qed

lemma *Or_n-And_n-image-antics:*
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 –
have $w \models_n Or_n (And_n \text{ ' } \mathcal{A}) \iff (\exists \Phi \in \mathcal{A}. w \models_n And_n \Phi)$
using *Or_n-antics assms* **by** *auto*
then show *?thesis*
using *And_n-antics assms* **by** *fast*
qed

lemma *Or_n-And_n-image-prop-antics:*
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 –
have $\mathcal{I} \models_P Or_n (And_n \text{ ' } \mathcal{A}) \iff (\exists \Phi \in \mathcal{A}. \mathcal{I} \models_P And_n \Phi)$
using *Or_n-prop-antics assms* **by** *blast*
then show *?thesis*
using *And_n-prop-antics assms* **by** *metis*
qed

4.5 DNF to LTL conversion

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

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

lemma *ltn-of-dnf-antics:*
assumes *finite* \mathcal{A}
shows $w \models_n ltn\text{-of-dnf } \mathcal{A} \iff (\exists \Phi \in \mathcal{A}. \forall \varphi. \varphi \in \Phi \longrightarrow w \models_n \varphi)$
proof –
have $w \models_n ltn\text{-of-dnf } \mathcal{A} \iff (\exists \Phi \in fset \text{ ' } \mathcal{A}. \forall \varphi \in \Phi. w \models_n \varphi)$
unfolding *ltn-of-dnf-def*
proof (*rule Or_n-And_n-image-antics*)
show *finite (fset ' A)*
using *assms* **by** *blast*
next

show $\bigwedge \Phi. \Phi \in \text{fset } \mathcal{A} \implies \text{finite } \Phi$
by *auto*
qed

then show *?thesis*
by (*metis image-iff*)
qed

lemma *ltln-of-dnf-prop-semantics*:
assumes *finite* \mathcal{A}
shows $\mathcal{I} \models_P \text{ltln-of-dnf } \mathcal{A} \iff (\exists \Phi \in \mathcal{A}. \forall \varphi. \varphi \in \Phi \implies \mathcal{I} \models_P \varphi)$
proof –
have $\mathcal{I} \models_P \text{ltln-of-dnf } \mathcal{A} \iff (\exists \Phi \in \text{fset } \mathcal{A}. \forall \varphi \in \Phi. \mathcal{I} \models_P \varphi)$
unfolding *ltln-of-dnf-def*
proof (*rule Or_n-And_n-image-prop-semantics*)
show *finite* (*fset* \mathcal{A})
using *assms* **by** *blast*
next
show $\bigwedge \Phi. \Phi \in \text{fset } \mathcal{A} \implies \text{finite } \Phi$
by *auto*
qed

then show *?thesis*
by (*metis image-iff*)
qed

lemma *ltln-of-dnf-prop-equiv*:
 $\text{ltln-of-dnf } (\text{min-dnf } \varphi) \sim_P \varphi$
unfolding *ltl-prop-equiv-def*
proof
fix \mathcal{A}
have $\mathcal{A} \models_P \text{ltln-of-dnf } (\text{min-dnf } \varphi) \iff (\exists \Phi \in \text{min-dnf } \varphi. \forall \varphi. \varphi \in \Phi \implies \mathcal{A} \models_P \varphi)$
using *ltln-of-dnf-prop-semantics min-dnf-finite* **by** *metis*
also have $\dots \iff (\exists \Phi \in \text{min-dnf } \varphi. \text{fset } \Phi \subseteq \mathcal{A})$
by (*metis min-dnf-prop-atoms prop-atoms-entailment-iff subset-eq*)
also have $\dots \iff \mathcal{A} \models_P \varphi$
using *min-dnf-iff-prop-assignment-subset* **by** *blast*
finally show $\mathcal{A} \models_P \text{ltln-of-dnf } (\text{min-dnf } \varphi) = \mathcal{A} \models_P \varphi$.
qed

lemma *min-dnf-ltln-of-dnf[simp]*:
 $\text{min-dnf } (\text{ltln-of-dnf } (\text{min-dnf } \varphi)) = \text{min-dnf } \varphi$
using *ltl-prop-equiv-min-dnf ltln-of-dnf-prop-equiv* **by** *blast*

4.6 Substitution in DNF formulas

definition *subst-clause* :: 'a ltl n fset \Rightarrow ('a ltl n \rightarrow 'a ltl n) \Rightarrow 'a ltl n 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 ltl n fset set \Rightarrow ('a ltl n \rightarrow 'a ltl n) \Rightarrow 'a ltl n fset set
where

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

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

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

by (*simp add: subst-clause-def*)

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

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

by (*simp add: subst-dnf-def*)

lemma *subst-clause-inner-finite*:

finite {*min-dnf* (*subst* φ *m*) \mid $\varphi. \varphi \in \Phi$ } **if** *finite* Φ
using *that by simp*

lemma *subst-clause-finite*:

finite (*subst-clause* Φ *m*)

unfolding *subst-clause-def*

by (*auto intro: min-dnf-finite min-product-set-finite*)

lemma *subst-dnf-finite*:

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

unfolding *subst-dnf-def using subst-clause-finite by blast*

lemma *subst-dnf-mono*:

$\mathcal{A} \subseteq \mathcal{B} \Longrightarrow$ *subst-dnf* \mathcal{A} *m* \subseteq *subst-dnf* \mathcal{B} *m*

unfolding *subst-dnf-def by blast*

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

min-set (*subst-clause* Φ *m*) = *subst-clause* Φ *m*

unfolding *subst-clause-def by simp*

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

subst-clause (*finsert* φ Φ) *m* = (*min-dnf* (*subst* φ *m*)) \otimes_m (*subst-clause* Φ *m*)

proof –

have {*min-dnf* (*subst* ψ *m*) \mid $\psi. \psi \in \text{fset } (\text{finsert } \varphi \ \Phi)$ }

$= \text{insert } (\text{min-dnf } (\text{subst } \varphi \ m)) \ \{\text{min-dnf } (\text{subst } \psi \ m) \mid \psi. \psi \in \text{fset } \Phi\}$
by *auto*

then show *?thesis*
by (*simp add: subst-clause-def*)

qed

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

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

proof (*induction* Ψ)

case (*insert* $x \ F$)

then show *?case*

using *min-product-set-thms.fun-left-comm* **by** *fastforce*

qed *simp*

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

definition *list-product* :: $'a \ \text{list set} \Rightarrow 'a \ \text{list set} \Rightarrow 'a \ \text{list set}$ (**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]*:

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

unfolding *list-product-def product-def image-def* **by** *fastforce*

lemma *list-product-finite*:

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

unfolding *list-product-def* **by** (*simp add: finite-image-set2*)

lemma *list-product-iff*:

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

unfolding *list-product-def* **by** *blast*

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

$A \otimes_l (B \otimes_l C) = A \otimes_l B \otimes_l C$

unfolding *set-eq-iff list-product-iff* **by** *fastforce*

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

fun *list-dnf* :: $'a \ \text{ltln} \Rightarrow 'a \ \text{ltln list set}$

where

$\text{list-dnf } \text{true}_n = \{\{\}\}$

$\mid \text{list-dnf } \text{false}_n = \{\}$

$\mid \text{list-dnf } (\varphi \ \text{and}_n \ \psi) = (\text{list-dnf } \varphi) \otimes_l (\text{list-dnf } \psi)$

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

$\mid \text{list-dnf } \varphi = \{[\varphi]\}$

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 φ) = dnf φ

by (induction φ) (simp-all add: list-dnf-to-dnf-def image-Un)

lemma *list-dnf-finite*:

finite (list-dnf φ)

by (induction φ) (simp-all add: list-product-finite)

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' Φ m = fold ($\lambda\varphi$ acc. acc \otimes_l list-dnf (subst φ m)) Φ {[]}

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'* Φ m)

lemma *subst-clause'-finite*:

finite (*subst-clause'* Φ m)

by (induction Φ rule: rev-induct) (simp-all add: subst-clause'-def list-dnf-finite list-product-finite)

lemma *subst-clause'-nil[simp]*:

subst-clause' [] m = {[]}

by (simp add: subst-clause'-def)

lemma *subst-clause'-cons[simp]*:

subst-clause' (xs @ [x]) m = *subst-clause'* xs m \otimes_l list-dnf (subst x m)

by (simp add: subst-clause'-def)

lemma *subst-clause'-append[simp]*:

subst-clause' (A @ B) m = *subst-clause'* A m \otimes_l *subst-clause'* B m

proof (induction B rule: rev-induct)

case (snoc x xs)

then show ?case

by simp (metis append-assoc subst-clause'-cons)

qed(simp add: list-product-def)

lemma *subst-dnf'-iff*:

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

by (*simp add: subst-dnf'-def*)

lemma *subst-dnf'-product*:

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

proof (*unfold set-eq-iff, safe*)

fix x

assume $x \in ?lhs$

then obtain Φ **where** $\Phi \in A \otimes_l B$ **and** $x \in \text{subst-clause}' \Phi m$
unfolding *subst-dnf'-iff* **by** *blast*

then obtain $a b$ **where** $a \in A$ **and** $b \in B$ **and** $\Phi = a @ b$
unfolding *list-product-def* **by** *blast*

then have $x \in (\text{subst-clause}' a m) \otimes_l (\text{subst-clause}' b m)$
using $\langle x \in \text{subst-clause}' \Phi m \rangle$ **by** *simp*

then obtain $a' b'$ **where** $a' \in \text{subst-clause}' a m$ **and** $b' \in \text{subst-clause}' b m$
and $x = a' @ b'$
unfolding *list-product-iff* **by** *blast*

then have $a' \in \text{subst-dnf}' A m$ **and** $b' \in \text{subst-dnf}' B m$
unfolding *subst-dnf'-iff* **using** $\langle a \in A \rangle \langle b \in B \rangle$ **by** *auto*

then have $\exists a \in \text{subst-dnf}' A m. \exists b \in \text{subst-dnf}' B m. x = a @ b$
using $\langle x = a' @ b' \rangle$ **by** *blast*

then show $x \in ?rhs$

unfolding *list-product-iff* **by** *blast*

next

fix x

assume $x \in ?rhs$

then obtain $a b$ **where** $a \in \text{subst-dnf}' A m$ **and** $b \in \text{subst-dnf}' B m$ **and**
 $x = a @ b$

unfolding *list-product-iff* **by** *blast*

then obtain $a' b'$ **where** $a' \in A$ **and** $b' \in B$ **and** $a: a \in \text{subst-clause}' a'$
 m **and** $b: b \in \text{subst-clause}' b' m$

unfolding *subst-dnf'-iff* **by** *blast*

then have $x \in (\text{subst-clause}' a' m) \otimes_l (\text{subst-clause}' b' m)$
unfolding *list-product-iff* **using** $\langle x = a @ b \rangle$ **by** *blast*

moreover

have $a' @ b' \in A \otimes_l B$
unfolding *list-product-iff* **using** $\langle a' \in A \rangle \langle b' \in B \rangle$ **by** *blast*

ultimately show $x \in ?lhs$
unfolding *subst-dnf'-iff* **by** *force*

qed

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

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

proof (*induction* φ)

case (*And-ltln* $\varphi_1 \varphi_2$)

then show *?case*

by (*simp add: subst-dnf'-product*)

qed (*simp-all add: subst-dnf'-def subst-clause'-def list-product-def*)

lemma *min-set-Union*:

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

by (*induction X rule: finite-induct*) (*force,metis Sup-insert image-insert min-set-min-union min-union-def*)

lemma *min-set-Union-image*:

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

proof –

assume *finite X*

then have $*$: *finite (f ' X)* **by** *auto*

with *min-set-Union* **show** *?thesis*

unfolding *image-image* **by** *fastforce*

qed

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

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

unfolding *list-dnf-to-dnf-def subst-clause'-def*

proof (*induction* Φ *rule: rev-induct*)

case (*snoc* x xs)
then show *?case*
by *simp* (*metis* (*no-types*, *lifting*) *dnf-min-set list-dnf-to-dnf-def list-dnf-to-dnf-list-dnf*
min-product-comm min-product-def min-set-min-product(1))
qed *simp*

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

$finite\ X \implies min_set\ (list_dnf_to_dnf\ (subst_dnf'\ X\ m)) = min_set\ (subst_dnf\ (list_dnf_to_dnf\ X)\ m)$

by (*simp* *add: subst-dnf'-def subst-dnf-def subst-clause-fset-of-list list-dnf-to-dnf-def min-set-Union-image image-Union*)

lemma *subst-dnf-dnf*:

$min_set\ (subst_dnf\ (dnf\ \varphi)\ m) = min_dnf\ (subst\ \varphi\ m)$

unfolding *dnf-min-set*

unfolding *list-dnf-to-dnf-list-dnf[symmetric]*

unfolding *subst-dnf'-list-dnf[symmetric]*

unfolding *min-set-list-dnf-to-dnf-subst-dnf'[OF list-dnf-finite]*

by *simp*

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

lemma *fold-product*:

$Finite_Set.fold\ (\lambda x. (\otimes)\ \{\{|x|\}\})\ \{\{|\}\}\ (fset\ x) = \{x\}$

by (*induction* x) (*simp-all*, *simp* *add: product-singleton-singleton*)

lemma *fold-union*:

$Finite_Set.fold\ (\lambda x. (\cup)\ \{x\})\ \{\}\ (fset\ x) = fset\ x$

by (*induction* x) (*simp-all* *add: comp-fun-idem-on.fold-insert-idem[OF comp-fun-idem-insert[unfolded comp-fun-idem-def']*])

lemma *fold-union-fold-product*:

assumes *finite* X **and** $\bigwedge \Psi\ \psi. \Psi \in X \implies \psi \in fset\ \Psi \implies dnf\ \psi = \{\{|\psi|\}\}$

shows $Finite_Set.fold\ (\lambda x. (\cup)\ (Finite_Set.fold\ (\lambda \varphi. (\otimes)\ (dnf\ \varphi))\ \{\{|\}\}\ (fset\ x)))\ \{\}\ X = X$ (**is** *?lhs = X*)

proof –

from *assms* **have** *?lhs = Finite-Set.fold* ($\lambda x. (\cup)\ (Finite_Set.fold\ (\lambda \varphi. (\otimes)\ \{\{|\varphi|\}\})\ \{\{|\}\}\ (fset\ x)))\ \{\}\ X$

proof (*induction* X *rule: finite-induct*)

case (*insert* $\Phi\ X$)

from *insert.prem*s **have** $1: \bigwedge \Psi\ \psi. [\Psi \in X; \psi \in fset\ \Psi] \implies dnf\ \psi = \{\{|\psi|\}\}$

by *force*

from *insert.prem*s **have** *Finite-Set.fold* ($\lambda\varphi. (\otimes) (\text{dnf } \varphi)$) $\{\{\|\|\}\}$ (*fset* Φ) = *Finite-Set.fold* ($\lambda\varphi. (\otimes) \{\{|\varphi|\}\}$) $\{\{\|\|\}\}$ (*fset* Φ)
by (*induction* Φ) *force+*

with *insert 1* **show** *?case*
by *simp*
qed *simp*

with $\langle \text{finite } X \rangle$ **show** *?thesis*
unfolding *fold-product* **by** (*metis fset-to-fset fold-union*)
qed

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

dnf (ltln-of-dnf (min-dnf φ)) = min-dnf φ

proof –

have *1*: *finite* (*And*_{*n*} ‘*fset* ‘*min-dnf* φ)
using *min-dnf-finite* **by** *blast*

have *2*: *inj-on* *And*_{*n*} (*fset* ‘*min-dnf* φ)
by (*metis (mono-tags, lifting) And_n-inj f-inv-into-f fset inj-onI inj-on-contrad*)

have *3*: *inj-on* *fset* (*min-dnf* φ)
by (*meson fset-inject inj-onI*)

show *?thesis*

unfolding *ltln-of-dnf-def*

unfolding *Or_n-dnf*[*OF 1*]

unfolding *fold-image*[*OF 2*]

unfolding *fold-image*[*OF 3*]

unfolding *comp-def*

unfolding *And_n-dnf*[*OF finite-fset*]

by (*metis fold-union-fold-product min-dnf-finite min-dnf-atoms-dnf*)

qed

lemma *min-dnf-subst*:

*min-set (subst-dnf (min-dnf φ) *m*) = min-dnf (subst φ *m*)* (**is** *?lhs = ?rhs*)

proof –

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

have *?lhs = min-set (subst-dnf (dnf $?\varphi'$) *m*)*

unfolding *dnf-ltln-of-dnf-min-dnf ..*

```

also have ... = min-dnf (subst ? $\varphi'$  m)
  unfolding subst-dnf-dnf ..

also have ... = min-dnf (subst  $\varphi$  m)
  using ltl-prop-equiv-min-dnf ltl-of-dnf-prop-equiv subst-respects-ltl-prop-entailment(2)
by blast

  finally show ?thesis .
qed

end

```

5 Code lemmas for abstract definitions

```

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

```

5.1 Propositional Equivalence

```

fun ifex-of-ltl :: 'a ltln  $\Rightarrow$  'a ltln ifex
where
  ifex-of-ltl truen = Trueif
| ifex-of-ltl falsen = Falseif
| ifex-of-ltl ( $\varphi$  andn  $\psi$ ) = normif Mapping.empty (ifex-of-ltl  $\varphi$ ) (ifex-of-ltl
 $\psi$ ) Falseif
| ifex-of-ltl ( $\varphi$  orn  $\psi$ ) = normif Mapping.empty (ifex-of-ltl  $\varphi$ ) Trueif (ifex-of-ltl
 $\psi$ )
| ifex-of-ltl  $\varphi$  = IF  $\varphi$  Trueif Falseif

```

```

lemma val-ifex:
  val-ifex (ifex-of-ltl b) s = {x. s x}  $\models_P$  b
by (induction b) (simp add: agree-Nil val-normif)+

```

```

lemma reduced-ifex:
  reduced (ifex-of-ltl b) {}
by (induction b) (simp; metis keys-empty reduced-normif)+

```

```

lemma ifex-of-ltl-reduced-bdt-checker:
  reduced-bdt-checkers ifex-of-ltl ( $\lambda y s. \{x. s x\} \models_P y$ )
unfolding reduced-bdt-checkers-def

```

```

using val-ifex reduced-ifex by blast

lemma ltl-prop-equiv-impl[code]:
  ( $\varphi \sim_P \psi$ ) = equiv-test ifex-of-ltl  $\varphi \psi$ 
  by (simp add: ltl-prop-equiv-def reduced-bdt-checkers.equiv-test[OF ifex-of-ltl-reduced-bdt-checker];
fastforce)

lemma ltl-prop-implies-impl[code]:
  ( $\varphi \longrightarrow_P \psi$ ) = impl-test ifex-of-ltl  $\varphi \psi$ 
  by (simp add: ltl-prop-implies-def reduced-bdt-checkers.impl-test[OF ifex-of-ltl-reduced-bdt-checker];
force)

— Check code export
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  $\equiv$  ltn-to-ltlc o rewrite-iter-slow o ltlc-to-ltn o (map-ltlc ( $\lambda s.$  String.implode
"prop" + s + String.implode ""))

— Export rewriting engine (and also constructors)

export-code truec Iff-ltlc rewrite in SML file-prefix  $\langle$ rewrite-example $\rangle$ 

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.