A Compositional and Unified Translation of LTL into ω-Automata

Benedikt Seidl and Salomon Sickert

March 17, 2025

Abstract

We present a formalisation of the unified translation approach of linear temporal logic (LTL) into ω-automata from [1]. This approach decomposes LTL formulas into "simple" languages and allows a clear separation of concerns: first, we formalise the purely logical result yielding this decomposition; second, we instantiate this generic theory to obtain a construction for deterministic (state-based) Rabin automata (DRA). We extract from this particular instantiation an executable tool translating LTL to DRAs. To the best of our knowledge this is the first verified translation from LTL to DRAs that is proven to be double exponential in the worst case which asymptotically matches the known lower bound.

Contents

1 Syntactic Fragments and Stability

theory *Syntactic-Fragments-and-Stability*
imports
LTL.LTL HOL−Library.Sublist
begin

— We use prefix and suffix on infinite words.
hide-const *Sublist.prefix Sublist.suffix*

1.1 The fragments μLTL and νLTL

fun *is-μLTL* :: *'a ltln* ⇒ *bool*
where
is-μLTL $true_n$ = *True*
| *is-μLTL* $false_n$ = *True*
| *is-μLTL* $prop_n$(-) = *True*
| *is-μLTL* $nprop_n$(-) = *True*
| *is-μLTL* (φ and_n ψ) = (*is-μLTL* φ ∧ *is-μLTL* ψ)
| *is-μLTL* (φ or_n ψ) = (*is-μLTL* φ ∧ *is-μLTL* ψ)
| *is-μLTL* (X_n φ) = *is-μLTL* φ
| *is-μLTL* (φ U_n ψ) = (*is-μLTL* φ ∧ *is-μLTL* ψ)
| *is-μLTL* (φ M_n ψ) = (*is-μLTL* φ ∧ *is-μLTL* ψ)
| *is-μLTL* - = *False*

fun *is-νLTL* :: *'a ltln* ⇒ *bool*
where
is-νLTL $true_n$ = *True*
| *is-νLTL* $false_n$ = *True*
| *is-νLTL* $prop_n$(-) = *True*
| *is-νLTL* $nprop_n$(-) = *True*
| *is-νLTL* (φ and_n ψ) = (*is-νLTL* φ ∧ *is-νLTL* ψ)
| *is-νLTL* (φ or_n ψ) = (*is-νLTL* φ ∧ *is-νLTL* ψ)
| *is-νLTL* (X_n φ) = *is-νLTL* φ

| $is\text{-}\nu LTL\ (\varphi\ W_n\ \psi) = (is\text{-}\nu LTL\ \varphi \wedge is\text{-}\nu LTL\ \psi)$
| $is\text{-}\nu LTL\ (\varphi\ R_n\ \psi) = (is\text{-}\nu LTL\ \varphi \wedge is\text{-}\nu LTL\ \psi)$
| $is\text{-}\nu LTL$ - $= False$

definition $\mu LTL :: 'a\ ltln\ set$ **where**
$\mu LTL = \{\varphi.\ is\text{-}\mu LTL\ \varphi\}$

definition $\nu LTL :: 'a\ ltln\ set$ **where**
$\nu LTL = \{\varphi.\ is\text{-}\nu LTL\ \varphi\}$

lemma $\mu LTL\text{-}simp[simp]$:
$\varphi \in \mu LTL \longleftrightarrow is\text{-}\mu LTL\ \varphi$
$\langle proof \rangle$

lemma $\nu LTL\text{-}simp[simp]$:
$\varphi \in \nu LTL \longleftrightarrow is\text{-}\nu LTL\ \varphi$
$\langle proof \rangle$

1.1.1 Subformulas in μLTL and νLTL

fun $subformulas_\mu :: 'a\ ltln \Rightarrow 'a\ ltln\ set$
where
$subformulas_\mu\ (\varphi\ and_n\ \psi) = subformulas_\mu\ \varphi \cup subformulas_\mu\ \psi$
| $subformulas_\mu\ (\varphi\ or_n\ \psi) = subformulas_\mu\ \varphi \cup subformulas_\mu\ \psi$
| $subformulas_\mu\ (X_n\ \varphi) = subformulas_\mu\ \varphi$
| $subformulas_\mu\ (\varphi\ U_n\ \psi) = \{\varphi\ U_n\ \psi\} \cup subformulas_\mu\ \varphi \cup subformulas_\mu\ \psi$
| $subformulas_\mu\ (\varphi\ R_n\ \psi) = subformulas_\mu\ \varphi \cup subformulas_\mu\ \psi$
| $subformulas_\mu\ (\varphi\ W_n\ \psi) = subformulas_\mu\ \varphi \cup subformulas_\mu\ \psi$
| $subformulas_\mu\ (\varphi\ M_n\ \psi) = \{\varphi\ M_n\ \psi\} \cup subformulas_\mu\ \varphi \cup subformulas_\mu$
ψ
| $subformulas_\mu$ - $= \{\}$

fun $subformulas_\nu :: 'a\ ltln \Rightarrow 'a\ ltln\ set$
where
$subformulas_\nu\ (\varphi\ and_n\ \psi) = subformulas_\nu\ \varphi \cup subformulas_\nu\ \psi$
| $subformulas_\nu\ (\varphi\ or_n\ \psi) = subformulas_\nu\ \varphi \cup subformulas_\nu\ \psi$
| $subformulas_\nu\ (X_n\ \varphi) = subformulas_\nu\ \varphi$
| $subformulas_\nu\ (\varphi\ U_n\ \psi) = subformulas_\nu\ \varphi \cup subformulas_\nu\ \psi$
| $subformulas_\nu\ (\varphi\ R_n\ \psi) = \{\varphi\ R_n\ \psi\} \cup subformulas_\nu\ \varphi \cup subformulas_\nu\ \psi$
| $subformulas_\nu\ (\varphi\ W_n\ \psi) = \{\varphi\ W_n\ \psi\} \cup subformulas_\nu\ \varphi \cup subformulas_\nu$
ψ
| $subformulas_\nu\ (\varphi\ M_n\ \psi) = subformulas_\nu\ \varphi \cup subformulas_\nu\ \psi$
| $subformulas_\nu$ - $= \{\}$

lemma *subformulas*$_\mu$*-semantics*:
$subformulas_\mu\ \varphi = \{\psi \in subfrmlsn\ \varphi.\ \exists\psi_1\ \psi_2.\ \psi = \psi_1\ U_n\ \psi_2 \vee \psi = \psi_1\ M_n\ \psi_2\}$
⟨*proof*⟩

lemma *subformulas*$_\nu$*-semantics*:
$subformulas_\nu\ \varphi = \{\psi \in subfrmlsn\ \varphi.\ \exists\psi_1\ \psi_2.\ \psi = \psi_1\ R_n\ \psi_2 \vee \psi = \psi_1\ W_n\ \psi_2\}$
⟨*proof*⟩

lemma *subformulas*$_\mu$*-subfrmlsn*:
$subformulas_\mu\ \varphi \subseteq subfrmlsn\ \varphi$
⟨*proof*⟩

lemma *subformulas*$_\nu$*-subfrmlsn*:
$subformulas_\nu\ \varphi \subseteq subfrmlsn\ \varphi$
⟨*proof*⟩

lemma *subformulas*$_\mu$*-finite*:
$finite\ (subformulas_\mu\ \varphi)$
⟨*proof*⟩

lemma *subformulas*$_\nu$*-finite*:
$finite\ (subformulas_\nu\ \varphi)$
⟨*proof*⟩

lemma *subformulas*$_\mu$*-subset*:
$\psi \in subfrmlsn\ \varphi \Longrightarrow subformulas_\mu\ \psi \subseteq subformulas_\mu\ \varphi$
⟨*proof*⟩

lemma *subformulas*$_\nu$*-subset*:
$\psi \in subfrmlsn\ \varphi \Longrightarrow subformulas_\nu\ \psi \subseteq subformulas_\nu\ \varphi$
⟨*proof*⟩

lemma *subfrmlsn-*μ*LTL*:
$\varphi \in \mu LTL \Longrightarrow subfrmlsn\ \varphi \subseteq \mu LTL$
⟨*proof*⟩

lemma *subfrmlsn-*ν*LTL*:
$\varphi \in \nu LTL \Longrightarrow subfrmlsn\ \varphi \subseteq \nu LTL$
⟨*proof*⟩

lemma *subformulas*$_{\mu\nu}$*-disjoint*:
$subformulas_\mu\ \varphi \cap subformulas_\nu\ \varphi = \{\}$

⟨*proof*⟩

lemma *subformulas*$_{\mu\nu}$*-subfrmlsn*:
subformulas$_\mu$ $\varphi \cup$ *subformulas*$_\nu$ $\varphi \subseteq$ *subfrmlsn* φ
⟨*proof*⟩

lemma *subformulas*$_{\mu\nu}$*-card*:
card (*subformulas*$_\mu$ $\varphi \cup$ *subformulas*$_\nu$ φ) = *card* (*subformulas*$_\mu$ φ) + *card* (*subformulas*$_\nu$ φ)
⟨*proof*⟩

1.2 Stability

definition *GF-singleton w* $\varphi \equiv$ *if* $w \models_n G_n$ (F_n φ) *then* $\{\varphi\}$ *else* $\{\}$
definition *F-singleton w* $\varphi \equiv$ *if* $w \models_n F_n$ φ *then* $\{\varphi\}$ *else* $\{\}$

declare *GF-singleton-def* [*simp*] *F-singleton-def* [*simp*]

fun $\mathcal{GF}$ :: *'a ltln* ⇒ *'a set word* ⇒ *'a ltln set*
where
$\mathcal{GF}$ (φ *and*$_n$ ψ) $w = \mathcal{GF}$ φ $w \cup \mathcal{GF}$ ψ w
| $\mathcal{GF}$ (φ *or*$_n$ ψ) $w = \mathcal{GF}$ φ $w \cup \mathcal{GF}$ ψ w
| $\mathcal{GF}$ (X_n φ) $w = \mathcal{GF}$ φ w
| $\mathcal{GF}$ (φ U_n ψ) w = *GF-singleton w* (φ U_n ψ) $\cup \mathcal{GF}$ φ $w \cup \mathcal{GF}$ ψ w
| $\mathcal{GF}$ (φ R_n ψ) $w = \mathcal{GF}$ φ $w \cup \mathcal{GF}$ ψ w
| $\mathcal{GF}$ (φ W_n ψ) $w = \mathcal{GF}$ φ $w \cup \mathcal{GF}$ ψ w
| $\mathcal{GF}$ (φ M_n ψ) w = *GF-singleton w* (φ M_n ψ) $\cup \mathcal{GF}$ φ $w \cup \mathcal{GF}$ ψ w
| $\mathcal{GF}$ - - = {}

fun $\mathcal{F}$ :: *'a ltln* ⇒ *'a set word* ⇒ *'a ltln set*
where
$\mathcal{F}$ (φ *and*$_n$ ψ) $w = \mathcal{F}$ φ $w \cup \mathcal{F}$ ψ w
| $\mathcal{F}$ (φ *or*$_n$ ψ) $w = \mathcal{F}$ φ $w \cup \mathcal{F}$ ψ w
| $\mathcal{F}$ (X_n φ) $w = \mathcal{F}$ φ w
| $\mathcal{F}$ (φ U_n ψ) w = *F-singleton w* (φ U_n ψ) $\cup \mathcal{F}$ φ $w \cup \mathcal{F}$ ψ w
| $\mathcal{F}$ (φ R_n ψ) $w = \mathcal{F}$ φ $w \cup \mathcal{F}$ ψ w
| $\mathcal{F}$ (φ W_n ψ) $w = \mathcal{F}$ φ $w \cup \mathcal{F}$ ψ w
| $\mathcal{F}$ (φ M_n ψ) w = *F-singleton w* (φ M_n ψ) $\cup \mathcal{F}$ φ $w \cup \mathcal{F}$ ψ w
| $\mathcal{F}$ - - = {}

lemma $\mathcal{GF}$*-semantics*:
$\mathcal{GF}$ φ w = $\{\psi.\ \psi \in$ *subformulas*$_\mu$ $\varphi \wedge w \models_n G_n$ (F_n ψ)$\}$
⟨*proof*⟩

lemma $\mathcal{F}$-*semantics*:
$\mathcal{F}\ \varphi\ w = \{\psi.\ \psi \in subformulas_\mu\ \varphi \wedge w \models_n F_n\ \psi\}$
⟨*proof*⟩

lemma $\mathcal{GF}$-*semantics'*:
$\mathcal{GF}\ \varphi\ w = subformulas_\mu\ \varphi \cap \{\psi.\ w \models_n G_n\ (F_n\ \psi)\}$
⟨*proof*⟩

lemma $\mathcal{F}$-*semantics'*:
$\mathcal{F}\ \varphi\ w = subformulas_\mu\ \varphi \cap \{\psi.\ w \models_n F_n\ \psi\}$
⟨*proof*⟩

lemma $\mathcal{GF}$-$\mathcal{F}$-*subset*:
$\mathcal{GF}\ \varphi\ w \subseteq \mathcal{F}\ \varphi\ w$
⟨*proof*⟩

lemma $\mathcal{GF}$-*finite*:
finite $(\mathcal{GF}\ \varphi\ w)$
⟨*proof*⟩

lemma $\mathcal{GF}$-$subformulas_\mu$:
$\mathcal{GF}\ \varphi\ w \subseteq subformulas_\mu\ \varphi$
⟨*proof*⟩

lemma $\mathcal{GF}$-*subfrmlsn*:
$\mathcal{GF}\ \varphi\ w \subseteq subfrmlsn\ \varphi$
⟨*proof*⟩

lemma $\mathcal{GF}$-*elim*:
$\psi \in \mathcal{GF}\ \varphi\ w \Longrightarrow w \models_n G_n\ (F_n\ \psi)$
⟨*proof*⟩

lemma $\mathcal{GF}$-*suffix*:
$\mathcal{GF}\ \varphi\ (suffix\ i\ w) = \mathcal{GF}\ \varphi\ w$
⟨*proof*⟩

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

lemma $\mathcal{F}$-*finite*:

$finite\ (\mathcal{F}\ \varphi\ w)$
⟨*proof*⟩

lemma $\mathcal{F}$*-subformulas*$_\mu$:
$\mathcal{F}\ \varphi\ w \subseteq subformulas_\mu\ \varphi$
⟨*proof*⟩

lemma $\mathcal{F}$*-subfrmlsn*:
$\mathcal{F}\ \varphi\ w \subseteq subfrmlsn\ \varphi$
⟨*proof*⟩

lemma $\mathcal{F}$*-elim*:
$\psi \in \mathcal{F}\ \varphi\ w \Longrightarrow w \models_n F_n\ \psi$
⟨*proof*⟩

lemma $\mathcal{F}$*-suffix*:
$\mathcal{F}\ \varphi\ (suffix\ i\ w) \subseteq \mathcal{F}\ \varphi\ w$
⟨*proof*⟩

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

definition $\mu\text{-}stable\ \varphi\ w \longleftrightarrow \mathcal{GF}\ \varphi\ w = \mathcal{F}\ \varphi\ w$

lemma *suffix-μ-stable*:
$\forall_\infty i.\ \mu\text{-}stable\ \varphi\ (suffix\ i\ w)$
⟨*proof*⟩

lemma *μ-stable-subfrmlsn*:
$\mu\text{-}stable\ \varphi\ w \Longrightarrow \psi \in subfrmlsn\ \varphi \Longrightarrow \mu\text{-}stable\ \psi\ w$
⟨*proof*⟩

lemma *μ-stable-suffix*:
$\mu\text{-}stable\ \varphi\ w \Longrightarrow \mu\text{-}stable\ \varphi\ (suffix\ i\ w)$
⟨*proof*⟩

definition $FG\text{-}singleton\ w\ \varphi \equiv if\ w \models_n F_n\ (G_n\ \varphi)\ then\ \{\varphi\}\ else\ \{\}$
definition $G\text{-}singleton\ w\ \varphi \equiv if\ w \models_n G_n\ \varphi\ then\ \{\varphi\}\ else\ \{\}$

declare *FG-singleton-def* [*simp*] *G-singleton-def* [*simp*]

fun $\mathcal{FG}$ :: $'a\ ltln \Rightarrow 'a\ set\ word \Rightarrow 'a\ ltln\ set$
where
$\mathcal{FG}\ (\varphi\ and_n\ \psi)\ w = \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
$|\ \mathcal{FG}\ (\varphi\ or_n\ \psi)\ w = \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
$|\ \mathcal{FG}\ (X_n\ \varphi)\ w = \mathcal{FG}\ \varphi\ w$
$|\ \mathcal{FG}\ (\varphi\ U_n\ \psi)\ w = \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
$|\ \mathcal{FG}\ (\varphi\ R_n\ \psi)\ w = FG\text{-}singleton\ w\ (\varphi\ R_n\ \psi) \cup \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
$|\ \mathcal{FG}\ (\varphi\ W_n\ \psi)\ w = FG\text{-}singleton\ w\ (\varphi\ W_n\ \psi) \cup \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
$|\ \mathcal{FG}\ (\varphi\ M_n\ \psi)\ w = \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
$|\ \mathcal{FG}\ \text{-}\ \text{-} = \{\}$

fun $\mathcal{G}$ :: $'a\ ltln \Rightarrow 'a\ set\ word \Rightarrow 'a\ ltln\ set$
where
$\mathcal{G}\ (\varphi\ and_n\ \psi)\ w = \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
$|\ \mathcal{G}\ (\varphi\ or_n\ \psi)\ w = \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
$|\ \mathcal{G}\ (X_n\ \varphi)\ w = \mathcal{G}\ \varphi\ w$
$|\ \mathcal{G}\ (\varphi\ U_n\ \psi)\ w = \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
$|\ \mathcal{G}\ (\varphi\ R_n\ \psi)\ w = G\text{-}singleton\ w\ (\varphi\ R_n\ \psi) \cup \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
$|\ \mathcal{G}\ (\varphi\ W_n\ \psi)\ w = G\text{-}singleton\ w\ (\varphi\ W_n\ \psi) \cup \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
$|\ \mathcal{G}\ (\varphi\ M_n\ \psi)\ w = \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
$|\ \mathcal{G}\ \text{-}\ \text{-} = \{\}$

lemma $\mathcal{FG}$*-semantics*:
$\mathcal{FG}\ \varphi\ w = \{\psi \in subformulas_\nu\ \varphi.\ w \models_n F_n\ (G_n\ \psi)\}$
$\langle proof \rangle$

lemma $\mathcal{G}$*-semantics*:
$\mathcal{G}\ \varphi\ w \equiv \{\psi \in subformulas_\nu\ \varphi.\ w \models_n G_n\ \psi\}$
$\langle proof \rangle$

lemma $\mathcal{FG}$*-semantics'*:
$\mathcal{FG}\ \varphi\ w = subformulas_\nu\ \varphi \cap \{\psi.\ w \models_n F_n\ (G_n\ \psi)\}$
$\langle proof \rangle$

lemma $\mathcal{G}$*-semantics'*:
$\mathcal{G}\ \varphi\ w = subformulas_\nu\ \varphi \cap \{\psi.\ w \models_n G_n\ \psi\}$
$\langle proof \rangle$

lemma $\mathcal{G}$-$\mathcal{FG}$*-subset*:
$\mathcal{G}\ \varphi\ w \subseteq \mathcal{FG}\ \varphi\ w$
$\langle proof \rangle$

lemma $\mathcal{FG}$*-finite*:

finite ($\mathcal{FG}$ φ w)
⟨*proof*⟩

lemma $\mathcal{FG}$-*subformulas*$_\nu$:
$\mathcal{FG}$ φ $w \subseteq$ *subformulas*$_\nu$ φ
⟨*proof*⟩

lemma $\mathcal{FG}$-*subfrmlsn*:
$\mathcal{FG}$ φ $w \subseteq$ *subfrmlsn* φ
⟨*proof*⟩

lemma $\mathcal{FG}$-*elim*:
$\psi \in \mathcal{FG}\ \varphi\ w \Longrightarrow w \models_n F_n\ (G_n\ \psi)$
⟨*proof*⟩

lemma $\mathcal{FG}$-*suffix*:
$\mathcal{FG}$ φ (*suffix* i w) = $\mathcal{FG}$ φ w
⟨*proof*⟩

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

lemma $\mathcal{G}$-*finite*:
finite ($\mathcal{G}$ φ w)
⟨*proof*⟩

lemma $\mathcal{G}$-*subformulas*$_\nu$:
$\mathcal{G}$ φ $w \subseteq$ *subformulas*$_\nu$ φ
⟨*proof*⟩

lemma $\mathcal{G}$-*subfrmlsn*:
$\mathcal{G}$ φ $w \subseteq$ *subfrmlsn* φ
⟨*proof*⟩

lemma $\mathcal{G}$-*elim*:
$\psi \in \mathcal{G}\ \varphi\ w \Longrightarrow w \models_n G_n\ \psi$
⟨*proof*⟩

lemma $\mathcal{G}$-*suffix*:
$\mathcal{G}$ φ $w \subseteq \mathcal{G}$ φ (*suffix* i w)
⟨*proof*⟩

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

definition ν-*stable* $\varphi\ w \longleftrightarrow \mathcal{FG}\ \varphi\ w = \mathcal{G}\ \varphi\ w$

lemma *suffix*-ν-*stable*:
$\forall_\infty j.$ ν-*stable* φ (*suffix j w*)
⟨*proof*⟩

lemma ν-*stable-subfrmlsn*:
ν-*stable* $\varphi\ w \Longrightarrow \psi \in subfrmlsn\ \varphi \Longrightarrow$ ν-*stable* $\psi\ w$
⟨*proof*⟩

lemma ν-*stable-suffix*:
ν-*stable* $\varphi\ w \Longrightarrow$ ν-*stable* φ (*suffix i w*)
⟨*proof*⟩

1.3 Definitions with Lists for Code Export

The μ- and ν-subformulas as lists:

fun *subformulas*$_\mu$-*list* :: *'a ltln* ⇒ *'a ltln list*
where
subformulas$_\mu$-*list* (φ *and*$_n$ ψ) = *List.union* (*subformulas*$_\mu$-*list* φ) (*subformulas*$_\mu$-*list* ψ)
| *subformulas*$_\mu$-*list* (φ *or*$_n$ ψ) = *List.union* (*subformulas*$_\mu$-*list* φ) (*subformulas*$_\mu$-*list* ψ)
| *subformulas*$_\mu$-*list* (X_n φ) = *subformulas*$_\mu$-*list* φ
| *subformulas*$_\mu$-*list* (φ U_n ψ) = *List.insert* (φ U_n ψ) (*List.union* (*subformulas*$_\mu$-*list* φ) (*subformulas*$_\mu$-*list* ψ))
| *subformulas*$_\mu$-*list* (φ R_n ψ) = *List.union* (*subformulas*$_\mu$-*list* φ) (*subformulas*$_\mu$-*list* ψ)
| *subformulas*$_\mu$-*list* (φ W_n ψ) = *List.union* (*subformulas*$_\mu$-*list* φ) (*subformulas*$_\mu$-*list* ψ)
| *subformulas*$_\mu$-*list* (φ M_n ψ) = *List.insert* (φ M_n ψ) (*List.union* (*subformulas*$_\mu$-*list* φ) (*subformulas*$_\mu$-*list* ψ))
| *subformulas*$_\mu$-*list* - = []

fun *subformulas*$_\nu$-*list* :: *'a ltln* ⇒ *'a ltln list*
where
subformulas$_\nu$-*list* (φ *and*$_n$ ψ) = *List.union* (*subformulas*$_\nu$-*list* φ) (*subformulas*$_\nu$-*list* ψ)

| *subformulas*$_\nu$*-list* (φ *or*$_n$ ψ) = *List.union* (*subformulas*$_\nu$*-list* φ) (*subformulas*$_\nu$*-list* ψ)
| *subformulas*$_\nu$*-list* (X_n φ) = *subformulas*$_\nu$*-list* φ
| *subformulas*$_\nu$*-list* (φ U_n ψ) = *List.union* (*subformulas*$_\nu$*-list* φ) (*subformulas*$_\nu$*-list* ψ)
| *subformulas*$_\nu$*-list* (φ R_n ψ) = *List.insert* (φ R_n ψ) (*List.union* (*subformulas*$_\nu$*-list* φ) (*subformulas*$_\nu$*-list* ψ))
| *subformulas*$_\nu$*-list* (φ W_n ψ) = *List.insert* (φ W_n ψ) (*List.union* (*subformulas*$_\nu$*-list* φ) (*subformulas*$_\nu$*-list* ψ))
| *subformulas*$_\nu$*-list* (φ M_n ψ) = *List.union* (*subformulas*$_\nu$*-list* φ) (*subformulas*$_\nu$*-list* ψ)
| *subformulas*$_\nu$*-list* - = []

lemma *subformulas*$_\mu$*-list-set*:
set (*subformulas*$_\mu$*-list* φ) = *subformulas*$_\mu$ φ
⟨*proof*⟩

lemma *subformulas*$_\nu$*-list-set*:
set (*subformulas*$_\nu$*-list* φ) = *subformulas*$_\nu$ φ
⟨*proof*⟩

lemma *subformulas*$_\mu$*-list-distinct*:
distinct (*subformulas*$_\mu$*-list* φ)
⟨*proof*⟩

lemma *subformulas*$_\nu$*-list-distinct*:
distinct (*subformulas*$_\nu$*-list* φ)
⟨*proof*⟩

lemma *subformulas*$_\mu$*-list-length*:
length (*subformulas*$_\mu$*-list* φ) = *card* (*subformulas*$_\mu$ φ)
⟨*proof*⟩

lemma *subformulas*$_\nu$*-list-length*:
length (*subformulas*$_\nu$*-list* φ) = *card* (*subformulas*$_\nu$ φ)
⟨*proof*⟩

We define the list of advice sets as the product of all subsequences of the μ- and ν-subformulas of a formula.

definition *advice-sets* :: *'a ltln* ⇒ (*'a ltln list* × *'a ltln list*) *list*
where
advice-sets φ = *List.product* (*subseqs* (*subformulas*$_\mu$*-list* φ)) (*subseqs* (*subformulas*$_\nu$*-list* φ))

lemma *subset-subseq*:
$X \subseteq set\ ys \longleftrightarrow (\exists xs.\ X = set\ xs \wedge subseq\ xs\ ys)$
⟨*proof*⟩

lemma *subseqs-subformulas$_\mu$-list*:
$X \subseteq subformulas_\mu\ \varphi \longleftrightarrow (\exists xs.\ X = set\ xs \wedge xs \in set\ (subseqs\ (subformulas_\mu\text{-}list\ \varphi)))$
⟨*proof*⟩

lemma *subseqs-subformulas$_\nu$-list*:
$Y \subseteq subformulas_\nu\ \varphi \longleftrightarrow (\exists ys.\ Y = set\ ys \wedge ys \in set\ (subseqs\ (subformulas_\nu\text{-}list\ \varphi)))$
⟨*proof*⟩

lemma *advice-sets-subformulas*:
$X \subseteq subformulas_\mu\ \varphi \wedge Y \subseteq subformulas_\nu\ \varphi \longleftrightarrow (\exists xs\ ys.\ X = set\ xs \wedge Y = set\ ys \wedge (xs,\ ys) \in set\ (advice\text{-}sets\ \varphi))$
⟨*proof*⟩

lemma *subseqs-not-empty*:
$subseqs\ xs \neq []$
⟨*proof*⟩

lemma *product-not-empty*:
$xs \neq [] \Longrightarrow ys \neq [] \Longrightarrow List.product\ xs\ ys \neq []$
⟨*proof*⟩

lemma *advice-sets-not-empty*:
$advice\text{-}sets\ \varphi \neq []$
⟨*proof*⟩

lemma *advice-sets-length*:
$length\ (advice\text{-}sets\ \varphi) \leq 2\ \hat{}\ card\ (subfrmlsn\ \varphi)$
⟨*proof*⟩

lemma *advice-sets-element-length*:
$(xs,\ ys) \in set\ (advice\text{-}sets\ \varphi) \Longrightarrow length\ xs \leq card\ (subfrmlsn\ \varphi)$
$(xs,\ ys) \in set\ (advice\text{-}sets\ \varphi) \Longrightarrow length\ ys \leq card\ (subfrmlsn\ \varphi)$
⟨*proof*⟩

lemma *advice-sets-element-subfrmlsn*:
$(xs,\ ys) \in set\ (advice\text{-}sets\ \varphi) \Longrightarrow set\ xs \subseteq subformulas_\mu\ \varphi$

$(xs, ys) \in set\ (advice\text{-}sets\ \varphi) \Longrightarrow set\ ys \subseteq subformulas_\nu\ \varphi$
⟨*proof*⟩

end

2 The "after"-Function

theory *After*
imports
LTL.LTL LTL.Equivalence-Relations Syntactic-Fragments-and-Stability
begin

2.1 Definition of af

primrec *af-letter* :: $'a\ ltln \Rightarrow 'a\ set \Rightarrow 'a\ ltln$
where
$af\text{-}letter\ true_n\ \nu = true_n$
$|\ af\text{-}letter\ false_n\ \nu = false_n$
$|\ af\text{-}letter\ prop_n(a)\ \nu = (if\ a \in \nu\ then\ true_n\ else\ false_n)$
$|\ af\text{-}letter\ nprop_n(a)\ \nu\ = (if\ a \notin \nu\ then\ true_n\ else\ false_n)$
$|\ af\text{-}letter\ (\varphi\ and_n\ \psi)\ \nu = (af\text{-}letter\ \varphi\ \nu)\ and_n\ (af\text{-}letter\ \psi\ \nu)$
$|\ af\text{-}letter\ (\varphi\ or_n\ \psi)\ \nu = (af\text{-}letter\ \varphi\ \nu)\ or_n\ (af\text{-}letter\ \psi\ \nu)$
$|\ af\text{-}letter\ (X_n\ \varphi)\ \nu = \varphi$
$|\ af\text{-}letter\ (\varphi\ U_n\ \psi)\ \nu = (af\text{-}letter\ \psi\ \nu)\ or_n\ ((af\text{-}letter\ \varphi\ \nu)\ and_n\ (\varphi\ U_n\ \psi))$
$|\ af\text{-}letter\ (\varphi\ R_n\ \psi)\ \nu = (af\text{-}letter\ \psi\ \nu)\ and_n\ ((af\text{-}letter\ \varphi\ \nu)\ or_n\ (\varphi\ R_n\ \psi))$
$|\ af\text{-}letter\ (\varphi\ W_n\ \psi)\ \nu = (af\text{-}letter\ \psi\ \nu)\ or_n\ ((af\text{-}letter\ \varphi\ \nu)\ and_n\ (\varphi\ W_n\ \psi))$
$|\ af\text{-}letter\ (\varphi\ M_n\ \psi)\ \nu = (af\text{-}letter\ \psi\ \nu)\ and_n\ ((af\text{-}letter\ \varphi\ \nu)\ or_n\ (\varphi\ M_n\ \psi))$

abbreviation *af* :: $'a\ ltln \Rightarrow 'a\ set\ list \Rightarrow 'a\ ltln$
where
$af\ \varphi\ w \equiv foldl\ af\text{-}letter\ \varphi\ w$

lemma *af-decompose*:
$af\ (\varphi\ and_n\ \psi)\ w = (af\ \varphi\ w)\ and_n\ (af\ \psi\ w)$
$af\ (\varphi\ or_n\ \psi)\ w = (af\ \varphi\ w)\ or_n\ (af\ \psi\ w)$
⟨*proof*⟩

lemma *af-simps*[*simp*]:
$af\ true_n\ w = true_n$
$af\ false_n\ w = false_n$

$af\ (X_n\ \varphi)\ (x\ \#\ xs) = af\ \varphi\ xs$
⟨*proof*⟩

lemma *af-ite-simps*[*simp*]:
$af\ (\textit{if } P \textit{ then } true_n \textit{ else } false_n)\ w = (\textit{if } P \textit{ then } true_n \textit{ else } false_n)$
$af\ (\textit{if } P \textit{ then } false_n \textit{ else } true_n)\ w = (\textit{if } P \textit{ then } false_n \textit{ else } true_n)$
⟨*proof*⟩

lemma *af-subsequence-append*:
$i \leq j \Longrightarrow j \leq k \Longrightarrow af\ (af\ \varphi\ (w\ [i \to j]))\ (w\ [j \to k]) = af\ \varphi\ (w\ [i \to k])$
⟨*proof*⟩

lemma *af-subsequence-U*:
$af\ (\varphi\ U_n\ \psi)\ (w\ [0 \to Suc\ n]) = (af\ \psi\ (w\ [0 \to Suc\ n]))\ or_n\ ((af\ \varphi\ (w\ [0 \to Suc\ n]))\ and_n\ af\ (\varphi\ U_n\ \psi)\ (w\ [1 \to Suc\ n]))$
⟨*proof*⟩

lemma *af-subsequence-U'*:
$af\ (\varphi\ U_n\ \psi)\ (a\ \#\ xs) = (af\ \psi\ (a\ \#\ xs))\ or_n\ ((af\ \varphi\ (a\ \#\ xs))\ and_n\ af\ (\varphi\ U_n\ \psi)\ xs)$
⟨*proof*⟩

lemma *af-subsequence-R*:
$af\ (\varphi\ R_n\ \psi)\ (w\ [0 \to Suc\ n]) = (af\ \psi\ (w\ [0 \to Suc\ n]))\ and_n\ ((af\ \varphi\ (w\ [0 \to Suc\ n]))\ or_n\ af\ (\varphi\ R_n\ \psi)\ (w\ [1 \to Suc\ n]))$
⟨*proof*⟩

lemma *af-subsequence-R'*:
$af\ (\varphi\ R_n\ \psi)\ (a\ \#\ xs) = (af\ \psi\ (a\ \#\ xs))\ and_n\ ((af\ \varphi\ (a\ \#\ xs))\ or_n\ af\ (\varphi\ R_n\ \psi)\ xs)$
⟨*proof*⟩

lemma *af-subsequence-W*:
$af\ (\varphi\ W_n\ \psi)\ (w\ [0 \to Suc\ n]) = (af\ \psi\ (w\ [0 \to Suc\ n]))\ or_n\ ((af\ \varphi\ (w\ [0 \to Suc\ n]))\ and_n\ af\ (\varphi\ W_n\ \psi)\ (w\ [1 \to Suc\ n]))$
⟨*proof*⟩

lemma *af-subsequence-W'*:
$af\ (\varphi\ W_n\ \psi)\ (a\ \#\ xs) = (af\ \psi\ (a\ \#\ xs))\ or_n\ ((af\ \varphi\ (a\ \#\ xs))\ and_n\ af\ (\varphi\ W_n\ \psi)\ xs)$
⟨*proof*⟩

lemma *af-subsequence-M*:
$af\ (\varphi\ M_n\ \psi)\ (w\ [0 \to Suc\ n]) = (af\ \psi\ (w\ [0 \to Suc\ n]))\ and_n\ ((af\ \varphi\ (w$

$[0 \rightarrow Suc\ n]))\ or_n\ af\ (\varphi\ M_n\ \psi)\ (w\ [1 \rightarrow Suc\ n]))$
⟨proof⟩

lemma *af-subsequence-M′*:
$af\ (\varphi\ M_n\ \psi)\ (a\ \#\ xs) = (af\ \psi\ (a\ \#\ xs))\ and_n\ ((af\ \varphi\ (a\ \#\ xs))\ or_n\ af\ (\varphi\ M_n\ \psi)\ xs)$
⟨proof⟩

lemma *suffix-build*[*simp*]:
$suffix\ (Suc\ n)\ (x\ \#\#\ xs) = suffix\ n\ xs$
⟨proof⟩

lemma *af-letter-build*:
$(x\ \#\#\ w) \models_n \varphi \longleftrightarrow w \models_n af\text{-}letter\ \varphi\ x$
⟨proof⟩

lemma *af-ltl-continuation*:
$(w \frown w') \models_n \varphi \longleftrightarrow w' \models_n af\ \varphi\ w$
⟨proof⟩

2.2 Range of the after function

lemma *af-letter-atoms*:
$atoms\text{-}ltln\ (af\text{-}letter\ \varphi\ \nu) \subseteq atoms\text{-}ltln\ \varphi$
⟨proof⟩

lemma *af-atoms*:
$atoms\text{-}ltln\ (af\ \varphi\ w) \subseteq atoms\text{-}ltln\ \varphi$
⟨proof⟩

lemma *af-letter-nested-prop-atoms*:
$nested\text{-}prop\text{-}atoms\ (af\text{-}letter\ \varphi\ \nu) \subseteq nested\text{-}prop\text{-}atoms\ \varphi$
⟨proof⟩

lemma *af-nested-prop-atoms*:
$nested\text{-}prop\text{-}atoms\ (af\ \varphi\ w) \subseteq nested\text{-}prop\text{-}atoms\ \varphi$
⟨proof⟩

lemma *af-letter-range*:
$range\ (af\text{-}letter\ \varphi) \subseteq \{\psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms\ \varphi\}$
⟨proof⟩

lemma *af-range*:
$range\ (af\ \varphi) \subseteq \{\psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms\ \varphi\}$

⟨*proof*⟩

2.3 Subformulas of the after function

lemma *af-letter-subformulas*$_\mu$:
subformulas$_\mu$ (*af-letter* φ ξ) = *subformulas*$_\mu$ φ
⟨*proof*⟩

lemma *af-subformulas*$_\mu$:
subformulas$_\mu$ (*af* φ *w*) = *subformulas*$_\mu$ φ
⟨*proof*⟩

lemma *af-letter-subformulas*$_\nu$:
subformulas$_\nu$ (*af-letter* φ ξ) = *subformulas*$_\nu$ φ
⟨*proof*⟩

lemma *af-subformulas*$_\nu$:
subformulas$_\nu$ (*af* φ *w*) = *subformulas*$_\nu$ φ
⟨*proof*⟩

2.4 Stability and the after function

lemma $\mathcal{GF}$-*af*:
$\mathcal{GF}$ (*af* φ (*prefix i w*)) (*suffix i w*) = $\mathcal{GF}$ φ (*suffix i w*)
⟨*proof*⟩

lemma $\mathcal{F}$-*af*:
$\mathcal{F}$ (*af* φ (*prefix i w*)) (*suffix i w*) = $\mathcal{F}$ φ (*suffix i w*)
⟨*proof*⟩

lemma $\mathcal{FG}$-*af*:
$\mathcal{FG}$ (*af* φ (*prefix i w*)) (*suffix i w*) = $\mathcal{FG}$ φ (*suffix i w*)
⟨*proof*⟩

lemma $\mathcal{G}$-*af*:
$\mathcal{G}$ (*af* φ (*prefix i w*)) (*suffix i w*) = $\mathcal{G}$ φ (*suffix i w*)
⟨*proof*⟩

2.5 Congruence

lemma *af-letter-lang-congruent*:
$\varphi \sim_L \psi \Longrightarrow$ *af-letter* φ ν $\sim_L$ *af-letter* ψ ν
⟨*proof*⟩

lemma *af-lang-congruent*:

$\varphi \sim_L \psi \Longrightarrow af\ \varphi\ w \sim_L af\ \psi\ w$
⟨*proof*⟩

lemma *af-letter-subst*:
$af\text{-}letter\ \varphi\ \nu = subst\ \varphi\ (\lambda\psi.\ Some\ (af\text{-}letter\ \psi\ \nu))$
⟨*proof*⟩

lemma *af-letter-prop-congruent*:
$\varphi \longrightarrow_P \psi \Longrightarrow af\text{-}letter\ \varphi\ \nu \longrightarrow_P af\text{-}letter\ \psi\ \nu$
$\varphi \sim_P \psi \Longrightarrow af\text{-}letter\ \varphi\ \nu \sim_P af\text{-}letter\ \psi\ \nu$
⟨*proof*⟩

lemma *af-prop-congruent*:
$\varphi \longrightarrow_P \psi \Longrightarrow af\ \varphi\ w \longrightarrow_P af\ \psi\ w$
$\varphi \sim_P \psi \Longrightarrow af\ \varphi\ w \sim_P af\ \psi\ w$
⟨*proof*⟩

lemma *af-letter-const-congruent*:
$\varphi \sim_C \psi \Longrightarrow af\text{-}letter\ \varphi\ \nu \sim_C af\text{-}letter\ \psi\ \nu$
⟨*proof*⟩

lemma *af-const-congruent*:
$\varphi \sim_C \psi \Longrightarrow af\ \varphi\ w \sim_C af\ \psi\ w$
⟨*proof*⟩

lemma *af-letter-one-step-back*:
$\{x.\ \mathcal{A} \models_P af\text{-}letter\ x\ \sigma\} \models_P \varphi \longleftrightarrow \mathcal{A} \models_P af\text{-}letter\ \varphi\ \sigma$
⟨*proof*⟩

2.6 Implications

lemma *af-F-prefix-prop*:
$af\ (F_n\ \varphi)\ w \longrightarrow_P af\ (F_n\ \varphi)\ (w'\ @\ w)$
⟨*proof*⟩

lemma *af-G-prefix-prop*:
$af\ (G_n\ \varphi)\ (w'\ @\ w) \longrightarrow_P af\ (G_n\ \varphi)\ w$
⟨*proof*⟩

lemma *af-F-prefix-lang*:
$w \models_n af\ (F_n\ \varphi)\ ys \Longrightarrow w \models_n af\ (F_n\ \varphi)\ (xs\ @\ ys)$
⟨*proof*⟩

lemma *af-G-prefix-lang*:
$w \models_n af\ (G_n\ \varphi)\ (xs\ @\ ys) \Longrightarrow w \models_n af\ (G_n\ \varphi)\ ys$
⟨*proof*⟩

lemma *af-F-prefix-const-equiv-true*:
$af\ (F_n\ \varphi)\ w \sim_C true_n \Longrightarrow af\ (F_n\ \varphi)\ (w'\ @\ w) \sim_C true_n$
⟨*proof*⟩

lemma *af-G-prefix-const-equiv-false*:
$af\ (G_n\ \varphi)\ w \sim_C false_n \Longrightarrow af\ (G_n\ \varphi)\ (w'\ @\ w) \sim_C false_n$
⟨*proof*⟩

lemma *af-F-prefix-lang-equiv-true*:
$af\ (F_n\ \varphi)\ w \sim_L true_n \Longrightarrow af\ (F_n\ \varphi)\ (w'\ @\ w) \sim_L true_n$
⟨*proof*⟩

lemma *af-G-prefix-lang-equiv-false*:
$af\ (G_n\ \varphi)\ w \sim_L false_n \Longrightarrow af\ (G_n\ \varphi)\ (w'\ @\ w) \sim_L false_n$
⟨*proof*⟩

locale *af-congruent* = *ltl-equivalence* +
assumes
af-letter-congruent: $\varphi \sim \psi \Longrightarrow af\text{-}letter\ \varphi\ \nu \sim af\text{-}letter\ \psi\ \nu$
begin

lemma *af-congruentness*:
$\varphi \sim \psi \Longrightarrow af\ \varphi\ xs \sim af\ \psi\ xs$
⟨*proof*⟩

lemma *af-append-congruent*:
$af\ \varphi\ w \sim af\ \psi\ w \Longrightarrow af\ \varphi\ (w\ @\ w') \sim af\ \psi\ (w\ @\ w')$
⟨*proof*⟩

lemma *af-append-true*:
$af\ \varphi\ w \sim true_n \Longrightarrow af\ \varphi\ (w\ @\ w') \sim true_n$
⟨*proof*⟩

lemma *af-append-false*:
$af\ \varphi\ w \sim false_n \Longrightarrow af\ \varphi\ (w\ @\ w') \sim false_n$
⟨*proof*⟩

lemma *prefix-append-subsequence*:
$i \leq j \Longrightarrow (prefix\ i\ w)\ @\ (w\ [i \rightarrow j]) = prefix\ j\ w$
⟨*proof*⟩

lemma *af-prefix-congruent*:
$i \leq j \Longrightarrow af\ \varphi\ (prefix\ i\ w) \sim af\ \psi\ (prefix\ i\ w) \Longrightarrow af\ \varphi\ (prefix\ j\ w) \sim af\ \psi\ (prefix\ j\ w)$
⟨*proof*⟩

lemma *af-prefix-true*:
$i \leq j \Longrightarrow af\ \varphi\ (prefix\ i\ w) \sim true_n \Longrightarrow af\ \varphi\ (prefix\ j\ w) \sim true_n$
⟨*proof*⟩

lemma *af-prefix-false*:
$i \leq j \Longrightarrow af\ \varphi\ (prefix\ i\ w) \sim false_n \Longrightarrow af\ \varphi\ (prefix\ j\ w) \sim false_n$
⟨*proof*⟩

end

interpretation *lang-af-congruent*: *af-congruent* $(\sim_L)$
⟨*proof*⟩

interpretation *prop-af-congruent*: *af-congruent* $(\sim_P)$
⟨*proof*⟩

interpretation *const-af-congruent*: *af-congruent* $(\sim_C)$
⟨*proof*⟩

2.7 After in μLTL and νLTL

lemma *valid-prefix-implies-ltl*:
$af\ \varphi\ (prefix\ i\ w) \sim_L true_n \Longrightarrow w \models_n \varphi$
⟨*proof*⟩

lemma *ltl-implies-satisfiable-prefix*:
$w \models_n \varphi \Longrightarrow \neg\ (af\ \varphi\ (prefix\ i\ w) \sim_L false_n)$

$\langle proof \rangle$

lemma *μLTL-implies-valid-prefix*:
$\varphi \in \mu LTL \Longrightarrow w \models_n \varphi \Longrightarrow \exists\, i.\ af\ \varphi\ (prefix\ i\ w) \sim_C true_n$
$\langle proof \rangle$

lemma *satisfiable-prefix-implies-νLTL*:
$\varphi \in \nu LTL \Longrightarrow \nexists\, i.\ af\ \varphi\ (prefix\ i\ w) \sim_C false_n \Longrightarrow w \models_n \varphi$
$\langle proof \rangle$

context *ltl-equivalence*
begin

lemma *valid-prefix-implies-ltl*:
$af\ \varphi\ (prefix\ i\ w) \sim true_n \Longrightarrow w \models_n \varphi$
$\langle proof \rangle$

lemma *ltl-implies-satisfiable-prefix*:
$w \models_n \varphi \Longrightarrow \neg\ (af\ \varphi\ (prefix\ i\ w) \sim false_n)$
$\langle proof \rangle$

lemma *μLTL-implies-valid-prefix*:
$\varphi \in \mu LTL \Longrightarrow w \models_n \varphi \Longrightarrow \exists\, i.\ af\ \varphi\ (prefix\ i\ w) \sim true_n$
$\langle proof \rangle$

lemma *satisfiable-prefix-implies-νLTL*:
$\varphi \in \nu LTL \Longrightarrow \nexists\, i.\ af\ \varphi\ (prefix\ i\ w) \sim false_n \Longrightarrow w \models_n \varphi$
$\langle proof \rangle$

lemma *af-μLTL*:
$\varphi \in \mu LTL \Longrightarrow w \models_n \varphi \longleftrightarrow (\exists\, i.\ af\ \varphi\ (prefix\ i\ w) \sim true_n)$
$\langle proof \rangle$

lemma *af-νLTL*:
$\varphi \in \nu LTL \Longrightarrow w \models_n \varphi \longleftrightarrow (\forall\, i.\ \neg\ (af\ \varphi\ (prefix\ i\ w) \sim false_n))$
$\langle proof \rangle$

lemma *af-μLTL-GF*:
$\varphi \in \mu LTL \Longrightarrow w \models_n G_n\ (F_n\ \varphi) \longleftrightarrow (\forall\, i.\ \exists\, j.\ af\ (F_n\ \varphi)\ (w[i \rightarrow j]) \sim true_n)$

$\langle proof \rangle$

lemma *af-νLTL-FG*:
$\varphi \in \nu LTL \Longrightarrow w \models_n F_n\ (G_n\ \varphi) \longleftrightarrow (\exists\, i.\ \forall\, j.\ \neg\ (af\ \ (G_n\ \varphi)\ (w[i \to j])$
$\sim false_n))$
$\langle proof \rangle$

end

Bring Propositional Equivalence into scope

interpretation *af-congruent* $(\sim_P)$
$\langle proof \rangle$

end

3 Advice functions

theory *Advice*
imports
LTL.LTL LTL.Equivalence-Relations
Syntactic-Fragments-and-Stability After
begin

3.1 The GF and FG Advice Functions

fun *GF-advice* :: $'a\ ltln \Rightarrow\ 'a\ ltln\ set \Rightarrow\ 'a\ ltln$ (‹-[-]$_\nu$› [90,60] 89)
where
$(X_n\ \psi)[X]_\nu = X_n\ (\psi[X]_\nu)$
$|\ (\psi_1\ and_n\ \psi_2)[X]_\nu = (\psi_1[X]_\nu)\ and_n\ (\psi_2[X]_\nu)$
$|\ (\psi_1\ or_n\ \psi_2)[X]_\nu = (\psi_1[X]_\nu)\ or_n\ (\psi_2[X]_\nu)$
$|\ (\psi_1\ W_n\ \psi_2)[X]_\nu = (\psi_1[X]_\nu)\ W_n\ (\psi_2[X]_\nu)$
$|\ (\psi_1\ R_n\ \psi_2)[X]_\nu = (\psi_1[X]_\nu)\ R_n\ (\psi_2[X]_\nu)$
$|\ (\psi_1\ U_n\ \psi_2)[X]_\nu = (if\ (\psi_1\ U_n\ \psi_2) \in X\ then\ (\psi_1[X]_\nu)\ W_n\ (\psi_2[X]_\nu)\ else$
$false_n)$
$|\ (\psi_1\ M_n\ \psi_2)[X]_\nu = (if\ (\psi_1\ M_n\ \psi_2) \in X\ then\ (\psi_1[X]_\nu)\ R_n\ (\psi_2[X]_\nu)\ else$
$false_n)$
$|\ \varphi[-]_\nu = \varphi$

fun *FG-advice* :: $'a\ ltln \Rightarrow\ 'a\ ltln\ set \Rightarrow\ 'a\ ltln$ (‹-[-]$_\mu$› [90,60] 89)
where
$(X_n\ \psi)[Y]_\mu = X_n\ (\psi[Y]_\mu)$
$|\ (\psi_1\ and_n\ \psi_2)[Y]_\mu = (\psi_1[Y]_\mu)\ and_n\ (\psi_2[Y]_\mu)$
$|\ (\psi_1\ or_n\ \psi_2)[Y]_\mu = (\psi_1[Y]_\mu)\ or_n\ (\psi_2[Y]_\mu)$
$|\ (\psi_1\ U_n\ \psi_2)[Y]_\mu = (\psi_1[Y]_\mu)\ U_n\ (\psi_2[Y]_\mu)$

| $(\psi_1\ M_n\ \psi_2)[Y]_\mu = (\psi_1[Y]_\mu)\ M_n\ (\psi_2[Y]_\mu)$
| $(\psi_1\ W_n\ \psi_2)[Y]_\mu = (\mathit{if}\ (\psi_1\ W_n\ \psi_2) \in Y\ \mathit{then}\ \mathit{true}_n\ \mathit{else}\ (\psi_1[Y]_\mu)\ U_n\ (\psi_2[Y]_\mu))$
| $(\psi_1\ R_n\ \psi_2)[Y]_\mu = (\mathit{if}\ (\psi_1\ R_n\ \psi_2) \in Y\ \mathit{then}\ \mathit{true}_n\ \mathit{else}\ (\psi_1[Y]_\mu)\ M_n\ (\psi_2[Y]_\mu))$
| $\varphi[\text{-}]_\mu = \varphi$

lemma *GF-advice-νLTL*:
$\varphi[X]_\nu \in \nu LTL$
$\varphi \in \nu LTL \Longrightarrow \varphi[X]_\nu = \varphi$
⟨*proof*⟩

lemma *FG-advice-μLTL*:
$\varphi[X]_\mu \in \mu LTL$
$\varphi \in \mu LTL \Longrightarrow \varphi[X]_\mu = \varphi$
⟨*proof*⟩

lemma *GF-advice-subfrmlsn*:
$\mathit{subfrmlsn}\ (\varphi[X]_\nu) \subseteq \{\psi[X]_\nu \mid \psi.\ \psi \in \mathit{subfrmlsn}\ \varphi\}$
⟨*proof*⟩

lemma *FG-advice-subfrmlsn*:
$\mathit{subfrmlsn}\ (\varphi[Y]_\mu) \subseteq \{\psi[Y]_\mu \mid \psi.\ \psi \in \mathit{subfrmlsn}\ \varphi\}$
⟨*proof*⟩

lemma *GF-advice-subfrmlsn-card*:
$\mathit{card}\ (\mathit{subfrmlsn}\ (\varphi[X]_\nu)) \leq \mathit{card}\ (\mathit{subfrmlsn}\ \varphi)$
⟨*proof*⟩

lemma *FG-advice-subfrmlsn-card*:
$\mathit{card}\ (\mathit{subfrmlsn}\ (\varphi[Y]_\mu)) \leq \mathit{card}\ (\mathit{subfrmlsn}\ \varphi)$
⟨*proof*⟩

lemma *GF-advice-monotone*:
$X \subseteq Y \Longrightarrow w \models_n \varphi[X]_\nu \Longrightarrow w \models_n \varphi[Y]_\nu$
⟨*proof*⟩

lemma *FG-advice-monotone*:
$X \subseteq Y \Longrightarrow w \models_n \varphi[X]_\mu \Longrightarrow w \models_n \varphi[Y]_\mu$
⟨*proof*⟩

lemma *GF-advice-ite-simps*[*simp*]:
$(\mathit{if}\ P\ \mathit{then}\ \mathit{true}_n\ \mathit{else}\ \mathit{false}_n)[X]_\nu = (\mathit{if}\ P\ \mathit{then}\ \mathit{true}_n\ \mathit{else}\ \mathit{false}_n)$

$(\textit{if } P \textit{ then } \textit{false}_n \textit{ else } \textit{true}_n)[X]_\nu = (\textit{if } P \textit{ then } \textit{false}_n \textit{ else } \textit{true}_n)$
⟨*proof*⟩

lemma *FG-advice-ite-simps*[*simp*]:
$(\textit{if } P \textit{ then } \textit{true}_n \textit{ else } \textit{false}_n)[Y]_\mu = (\textit{if } P \textit{ then } \textit{true}_n \textit{ else } \textit{false}_n)$
$(\textit{if } P \textit{ then } \textit{false}_n \textit{ else } \textit{true}_n)[Y]_\mu = (\textit{if } P \textit{ then } \textit{false}_n \textit{ else } \textit{true}_n)$
⟨*proof*⟩

3.2 Advice Functions on Nested Propositions

definition $\textit{nested-prop-atoms}_\nu$ :: $'a\ \textit{ltln} \Rightarrow 'a\ \textit{ltln set} \Rightarrow 'a\ \textit{ltln set}$
where
$\textit{nested-prop-atoms}_\nu\ \varphi\ X = \{\psi[X]_\nu \mid \psi.\ \psi \in \textit{nested-prop-atoms}\ \varphi\}$

definition $\textit{nested-prop-atoms}_\mu$ :: $'a\ \textit{ltln} \Rightarrow 'a\ \textit{ltln set} \Rightarrow 'a\ \textit{ltln set}$
where
$\textit{nested-prop-atoms}_\mu\ \varphi\ X = \{\psi[X]_\mu \mid \psi.\ \psi \in \textit{nested-prop-atoms}\ \varphi\}$

lemma $\textit{nested-prop-atoms}_\nu$*-finite*:
$\textit{finite}\ (\textit{nested-prop-atoms}_\nu\ \varphi\ X)$
⟨*proof*⟩

lemma $\textit{nested-prop-atoms}_\mu$*-finite*:
$\textit{finite}\ (\textit{nested-prop-atoms}_\mu\ \varphi\ X)$
⟨*proof*⟩

lemma $\textit{nested-prop-atoms}_\nu$*-card*:
$\textit{card}\ (\textit{nested-prop-atoms}_\nu\ \varphi\ X) \leq \textit{card}\ (\textit{nested-prop-atoms}\ \varphi)$
⟨*proof*⟩

lemma $\textit{nested-prop-atoms}_\mu$*-card*:
$\textit{card}\ (\textit{nested-prop-atoms}_\mu\ \varphi\ X) \leq \textit{card}\ (\textit{nested-prop-atoms}\ \varphi)$
⟨*proof*⟩

lemma *GF-advice-nested-prop-atoms*$_\nu$:
$\textit{nested-prop-atoms}\ (\varphi[X]_\nu) \subseteq \textit{nested-prop-atoms}_\nu\ \varphi\ X$
⟨*proof*⟩

lemma *FG-advice-nested-prop-atoms*$_\mu$:
$\textit{nested-prop-atoms}\ (\varphi[Y]_\mu) \subseteq \textit{nested-prop-atoms}_\mu\ \varphi\ Y$
⟨*proof*⟩

lemma $\textit{nested-prop-atoms}_\nu$*-subset*:
$\textit{nested-prop-atoms}\ \varphi \subseteq \textit{nested-prop-atoms}\ \psi \Longrightarrow \textit{nested-prop-atoms}_\nu\ \varphi\ X$

$\subseteq$ *nested-prop-atoms*$_\nu$ ψ X
⟨*proof*⟩

lemma *nested-prop-atoms*$_\mu$*-subset*:
nested-prop-atoms $\varphi \subseteq$ *nested-prop-atoms* $\psi \Longrightarrow$ *nested-prop-atoms*$_\mu$ φ Y $\subseteq$ *nested-prop-atoms*$_\mu$ ψ Y
⟨*proof*⟩

lemma *GF-advice-nested-prop-atoms-card*:
card (*nested-prop-atoms* ($\varphi[X]_\nu$)) $\leq$ *card* (*nested-prop-atoms* φ)
⟨*proof*⟩

lemma *FG-advice-nested-prop-atoms-card*:
card (*nested-prop-atoms* ($\varphi[Y]_\mu$)) $\leq$ *card* (*nested-prop-atoms* φ)
⟨*proof*⟩

3.3 Intersecting the Advice Set

lemma *GF-advice-inter*:
$X \cap$ *subformulas*$_\mu$ $\varphi \subseteq S \Longrightarrow \varphi[X \cap S]_\nu = \varphi[X]_\nu$
⟨*proof*⟩

lemma *GF-advice-inter-subformulas*:
$\varphi[X \cap$ *subformulas*$_\mu$ $\varphi]_\nu = \varphi[X]_\nu$
⟨*proof*⟩

lemma *GF-advice-minus-subformulas*:
$\psi \notin$ *subformulas*$_\mu$ $\varphi \Longrightarrow \varphi[X - \{\psi\}]_\nu = \varphi[X]_\nu$
⟨*proof*⟩

lemma *GF-advice-minus-size*:
$[\![$*size* $\varphi \leq$ *size* ψ; $\varphi \neq \psi]\!] \Longrightarrow \varphi[X - \{\psi\}]_\nu = \varphi[X]_\nu$
⟨*proof*⟩

lemma *FG-advice-inter*:
$Y \cap$ *subformulas*$_\nu$ $\varphi \subseteq S \Longrightarrow \varphi[Y \cap S]_\mu = \varphi[Y]_\mu$
⟨*proof*⟩

lemma *FG-advice-inter-subformulas*:
$\varphi[Y \cap$ *subformulas*$_\nu$ $\varphi]_\mu = \varphi[Y]_\mu$
⟨*proof*⟩

lemma *FG-advice-minus-subformulas*:

$\psi \notin subformulas_\nu\ \varphi \Longrightarrow \varphi[Y - \{\psi\}]_\mu = \varphi[Y]_\mu$
⟨proof⟩

lemma *FG-advice-minus-size*:
$[\![size\ \varphi \leq size\ \psi;\ \varphi \neq \psi]\!] \Longrightarrow \varphi[Y - \{\psi\}]_\mu = \varphi[Y]_\mu$
⟨proof⟩

lemma *FG-advice-insert*:
$[\![\psi \notin Y;\ size\ \varphi < size\ \psi]\!] \Longrightarrow \varphi[insert\ \psi\ Y]_\mu = \varphi[Y]_\mu$
⟨proof⟩

3.4 Correctness GF-advice function

lemma *GF-advice-a1*:
$[\![\mathcal{F}\ \varphi\ w \subseteq X;\ w \models_n \varphi]\!] \Longrightarrow w \models_n \varphi[X]_\nu$
⟨proof⟩

lemma *GF-advice-a2-helper*:
$[\![\forall \psi \in X.\ w \models_n G_n\ (F_n\ \psi);\ w \models_n \varphi[X]_\nu]\!] \Longrightarrow w \models_n \varphi$
⟨proof⟩

lemma *GF-advice-a2*:
$[\![X \subseteq \mathcal{GF}\ \varphi\ w;\ w \models_n \varphi[X]_\nu]\!] \Longrightarrow w \models_n \varphi$
⟨proof⟩

lemma *GF-advice-a3*:
$[\![X = \mathcal{F}\ \varphi\ w;\ X = \mathcal{GF}\ \varphi\ w]\!] \Longrightarrow w \models_n \varphi \longleftrightarrow w \models_n \varphi[X]_\nu$
⟨proof⟩

3.5 Correctness FG-advice function

lemma *FG-advice-b1*:
$[\![\mathcal{FG}\ \varphi\ w \subseteq Y;\ w \models_n \varphi]\!] \Longrightarrow w \models_n \varphi[Y]_\mu$
⟨proof⟩

lemma *FG-advice-b2-helper*:
$[\![\forall \psi \in Y.\ w \models_n G_n\ \psi;\ w \models_n \varphi[Y]_\mu]\!] \Longrightarrow w \models_n \varphi$
⟨proof⟩

lemma *FG-advice-b2*:
$[\![Y \subseteq \mathcal{G}\ \varphi\ w;\ w \models_n \varphi[Y]_\mu]\!] \Longrightarrow w \models_n \varphi$
⟨proof⟩

lemma *FG-advice-b3*:

$[\![Y = \mathcal{F}\mathcal{G}\ \varphi\ w;\ Y = \mathcal{G}\ \varphi\ w]\!] \Longrightarrow w \models_n \varphi \longleftrightarrow w \models_n \varphi[Y]_\mu$
⟨*proof*⟩

3.6 Advice Functions and the "after" Function

lemma *GF-advice-af-letter*:
$(x\ \#\#\ w) \models_n \varphi[X]_\nu \Longrightarrow w \models_n (\textit{af-letter}\ \varphi\ x)[X]_\nu$
⟨*proof*⟩

lemma *FG-advice-af-letter*:
$w \models_n (\textit{af-letter}\ \varphi\ x)[Y]_\mu \Longrightarrow (x\ \#\#\ w) \models_n \varphi[Y]_\mu$
⟨*proof*⟩

lemma *GF-advice-af*:
$(w \frown w') \models_n \varphi[X]_\nu \Longrightarrow w' \models_n (\textit{af}\ \varphi\ w)[X]_\nu$
⟨*proof*⟩

lemma *FG-advice-af*:
$w' \models_n (\textit{af}\ \varphi\ w)[X]_\mu \Longrightarrow (w \frown w') \models_n \varphi[X]_\mu$
⟨*proof*⟩

lemma *GF-advice-af-2*:
$w \models_n \varphi[X]_\nu \Longrightarrow \textit{suffix}\ i\ w \models_n (\textit{af}\ \varphi\ (\textit{prefix}\ i\ w))[X]_\nu$
⟨*proof*⟩

lemma *FG-advice-af-2*:
$\textit{suffix}\ i\ w \models_n (\textit{af}\ \varphi\ (\textit{prefix}\ i\ w))[X]_\mu \Longrightarrow w \models_n \varphi[X]_\mu$
⟨*proof*⟩

lemma *prefix-suffix-subsequence*: $\textit{prefix}\ i\ (\textit{suffix}\ j\ w) = (w\ [j \rightarrow i + j])$
⟨*proof*⟩

We show this generic lemma to prove the following theorems:

lemma *GF-advice-sync*:
fixes $\textit{index} :: \textit{nat} \Rightarrow \textit{nat}$
fixes $\textit{formula} :: \textit{nat} \Rightarrow {'a}\ \textit{ltln}$
assumes $\bigwedge i.\ i < n \Longrightarrow \exists j.\ \textit{suffix}\ ((\textit{index}\ i) + j)\ w \models_n \textit{af}\ (\textit{formula}\ i)\ (w\ [\textit{index}\ i \rightarrow (\textit{index}\ i) + j])[X]_\nu$
shows $\exists k.\ (\forall i < n.\ k \geq \textit{index}\ i \wedge \textit{suffix}\ k\ w \models_n \textit{af}\ (\textit{formula}\ i)\ (w\ [\textit{index}\ i \rightarrow k])[X]_\nu)$
⟨*proof*⟩

lemma *GF-advice-sync-and*:

assumes $\exists\, i.\ suffix\ i\ w \models_n af\ \varphi\ (prefix\ i\ w)[X]_\nu$
assumes $\exists\, i.\ suffix\ i\ w \models_n af\ \psi\ (prefix\ i\ w)[X]_\nu$
shows $\exists\, i.\ suffix\ i\ w \models_n af\ \varphi\ (prefix\ i\ w)[X]_\nu \wedge suffix\ i\ w \models_n af\ \psi\ (prefix\ i\ w)[X]_\nu$
$\langle proof \rangle$

lemma *GF-advice-sync-less*:
assumes $\bigwedge i.\ i < n \Longrightarrow \exists\, j.\ suffix\ (i + j)\ w \models_n af\ \varphi\ (w\ [i \to j + i])[X]_\nu$
assumes $\exists\, j.\ suffix\ (n + j)\ w \models_n af\ \psi\ (w\ [n \to j + n])[X]_\nu$
shows $\exists\, k \geq n.\ (\forall\, j < n.\ suffix\ k\ w \models_n af\ \varphi\ (w\ [j \to k])[X]_\nu) \wedge suffix\ k\ w \models_n af\ \psi\ (w\ [n \to k])[X]_\nu$
$\langle proof \rangle$

lemma *GF-advice-sync-lesseq*:
assumes $\bigwedge i.\ i \leq n \Longrightarrow \exists\, j.\ suffix\ (i + j)\ w \models_n af\ \varphi\ (w\ [i \to j + i])[X]_\nu$
assumes $\exists\, j.\ suffix\ (n + j)\ w \models_n af\ \psi\ (w\ [n \to j + n])[X]_\nu$
shows $\exists\, k \geq n.\ (\forall\, j \leq n.\ suffix\ k\ w \models_n af\ \varphi\ (w\ [j \to k])[X]_\nu) \wedge suffix\ k\ w \models_n af\ \psi\ (w\ [n \to k])[X]_\nu$
$\langle proof \rangle$

lemma *af-subsequence-U-GF-advice*:
assumes $i \leq n$
assumes $suffix\ n\ w \models_n ((af\ \psi\ (w\ [i \to n]))[X]_\nu)$
assumes $\bigwedge j.\ j < i \Longrightarrow suffix\ n\ w \models_n ((af\ \varphi\ (w\ [j \to n]))[X]_\nu)$
shows $suffix\ (Suc\ n)\ w \models_n (af\ (\varphi\ U_n\ \psi)\ (prefix\ (Suc\ n)\ w))[X]_\nu$
$\langle proof \rangle$

lemma *af-subsequence-M-GF-advice*:
assumes $i \leq n$
assumes $suffix\ n\ w \models_n ((af\ \varphi\ (w\ [i \to n]))[X]_\nu)$
assumes $\bigwedge j.\ j \leq i \Longrightarrow suffix\ n\ w \models_n ((af\ \psi\ (w\ [j \to n]))[X]_\nu)$
shows $suffix\ (Suc\ n)\ w \models_n (af\ (\varphi\ M_n\ \psi)\ (prefix\ (Suc\ n)\ w))[X]_\nu$
$\langle proof \rangle$

lemma *af-subsequence-R-GF-advice*:
assumes $i \leq n$
assumes $suffix\ n\ w \models_n ((af\ \varphi\ (w\ [i \to n]))[X]_\nu)$
assumes $\bigwedge j.\ j \leq i \Longrightarrow suffix\ n\ w \models_n ((af\ \psi\ (w\ [j \to n]))[X]_\nu)$
shows $suffix\ (Suc\ n)\ w \models_n (af\ (\varphi\ R_n\ \psi)\ (prefix\ (Suc\ n)\ w))[X]_\nu$
$\langle proof \rangle$

lemma *af-subsequence-W-GF-advice*:
assumes $i \leq n$
assumes $suffix\ n\ w \models_n ((af\ \psi\ (w\ [i \to n]))[X]_\nu)$

assumes $\bigwedge j.\ j < i \Longrightarrow suffix\ n\ w \models_n ((af\ \varphi\ (w\ [j \to n]))[X]_\nu)$
shows $suffix\ (Suc\ n)\ w \models_n (af\ (\varphi\ W_n\ \psi)\ (prefix\ (Suc\ n)\ w))[X]_\nu$
$\langle proof \rangle$

lemma *af-subsequence-R-GF-advice-connect*:
assumes $i \leq n$
assumes $suffix\ n\ w \models_n af\ (\varphi\ R_n\ \psi)\ (w\ [i \to n])[X]_\nu$
assumes $\bigwedge j.\ j \leq i \Longrightarrow suffix\ n\ w \models_n ((af\ \psi\ (w\ [j \to n]))[X]_\nu)$
shows $suffix\ (Suc\ n)\ w \models_n (af\ (\varphi\ R_n\ \psi)\ (prefix\ (Suc\ n)\ w))[X]_\nu$
$\langle proof \rangle$

lemma *af-subsequence-W-GF-advice-connect*:
assumes $i \leq n$
assumes $suffix\ n\ w \models_n af\ (\varphi\ W_n\ \psi)\ (w\ [i \to n])[X]_\nu$
assumes $\bigwedge j.\ j < i \Longrightarrow suffix\ n\ w \models_n ((af\ \varphi\ (w\ [j \to n]))[X]_\nu)$
shows $suffix\ (Suc\ n)\ w \models_n (af\ (\varphi\ W_n\ \psi)\ (prefix\ (Suc\ n)\ w))[X]_\nu$
$\langle proof \rangle$

3.7 Advice Functions and Propositional Entailment

lemma *GF-advice-prop-entailment*:
$\mathcal{A} \models_P \varphi[X]_\nu \Longrightarrow \{\psi.\ \psi[X]_\nu \in \mathcal{A}\} \models_P \varphi$
$false_n \notin \mathcal{A} \Longrightarrow \{\psi.\ \psi[X]_\nu \in \mathcal{A}\} \models_P \varphi \Longrightarrow \mathcal{A} \models_P \varphi[X]_\nu$
$\langle proof \rangle$

lemma *GF-advice-iff-prop-entailment*:
$false_n \notin \mathcal{A} \Longrightarrow \mathcal{A} \models_P \varphi[X]_\nu \longleftrightarrow \{\psi.\ \psi[X]_\nu \in \mathcal{A}\} \models_P \varphi$
$\langle proof \rangle$

lemma *FG-advice-prop-entailment*:
$true_n \in \mathcal{A} \Longrightarrow \mathcal{A} \models_P \varphi[Y]_\mu \Longrightarrow \{\psi.\ \psi[Y]_\mu \in \mathcal{A}\} \models_P \varphi$
$\{\psi.\ \psi[Y]_\mu \in \mathcal{A}\} \models_P \varphi \Longrightarrow \mathcal{A} \models_P \varphi[Y]_\mu$
$\langle proof \rangle$

lemma *FG-advice-iff-prop-entailment*:
$true_n \in \mathcal{A} \Longrightarrow \mathcal{A} \models_P \varphi[X]_\mu \longleftrightarrow \{\psi.\ \psi[X]_\mu \in \mathcal{A}\} \models_P \varphi$
$\langle proof \rangle$

lemma *GF-advice-subst*:
$\varphi[X]_\nu = subst\ \varphi\ (\lambda\psi.\ Some\ (\psi[X]_\nu))$
$\langle proof \rangle$

lemma *FG-advice-subst*:
$\varphi[X]_\mu = subst\ \varphi\ (\lambda\psi.\ Some\ (\psi[X]_\mu))$

$\langle proof \rangle$

lemma *GF-advice-prop-congruent*:
$\varphi \longrightarrow_P \psi \Longrightarrow \varphi[X]_\nu \longrightarrow_P \psi[X]_\nu$
$\varphi \sim_P \psi \Longrightarrow \varphi[X]_\nu \sim_P \psi[X]_\nu$
$\langle proof \rangle$

lemma *FG-advice-prop-congruent*:
$\varphi \longrightarrow_P \psi \Longrightarrow \varphi[X]_\mu \longrightarrow_P \psi[X]_\mu$
$\varphi \sim_P \psi \Longrightarrow \varphi[X]_\mu \sim_P \psi[X]_\mu$
$\langle proof \rangle$

3.8 GF-advice with Equivalence Relations

locale *GF-advice-congruent* = *ltl-equivalence* +
fixes
normalise :: *'a ltln* ⇒ *'a ltln*
assumes
normalise-eq: $\varphi \sim$ *normalise* φ
assumes
normalise-monotonic: $w \models_n \varphi[X]_\nu \Longrightarrow w \models_n (normalise\ \varphi)[X]_\nu$
assumes
normalise-eventually-equivalent:
$w \models_n (normalise\ \varphi)[X]_\nu \Longrightarrow (\exists\, i.\ suffix\ i\ w \models_n (af\ \varphi\ (prefix\ i\ w))[X]_\nu)$
assumes
GF-advice-congruent: $\varphi \sim \psi \Longrightarrow (normalise\ \varphi)[X]_\nu \sim (normalise\ \psi)[X]_\nu$
begin

lemma *normalise-language-equivalent*[*simp*]:
$w \models_n normalise\ \varphi \longleftrightarrow w \models_n \varphi$
$\langle proof \rangle$

end

interpretation *prop-GF-advice-compatible*: *GF-advice-congruent* $(\sim_P)$ *id*
$\langle proof \rangle$

end

4 The Master Theorem

theory *Master-Theorem*
imports
Advice After

begin

4.1 Checking $X \subseteq \mathcal{GF}\ \varphi\ w$ and $Y \subseteq \mathcal{FG}\ \varphi\ w$

lemma $X\text{-}\mathcal{GF}\text{-}Y\text{-}\mathcal{FG}$:
assumes
$X\text{-}\mu$: $X \subseteq subformulas_\mu\ \varphi$
and
$Y\text{-}\nu$: $Y \subseteq subformulas_\nu\ \varphi$
and
$X\text{-}GF$: $\forall \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)$
and
$Y\text{-}FG$: $\forall \psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu)$
shows
$X \subseteq \mathcal{GF}\ \varphi\ w \wedge Y \subseteq \mathcal{FG}\ \varphi\ w$
⟨*proof*⟩

lemma $\mathcal{GF}\text{-}implies\text{-}GF$:
$\forall \psi \in \mathcal{GF}\ \varphi\ w.\ w \models_n G_n\ (F_n\ \psi[\mathcal{FG}\ \varphi\ w]_\mu)$
⟨*proof*⟩

lemma $\mathcal{FG}\text{-}implies\text{-}FG$:
$\forall \psi \in \mathcal{FG}\ \varphi\ w.\ w \models_n F_n\ (G_n\ \psi[\mathcal{GF}\ \varphi\ w]_\nu)$
⟨*proof*⟩

4.2 Putting the pieces together: The Master Theorem

theorem *master-theorem-ltr*:
assumes
$w \models_n \varphi$
obtains X **and** Y **where**
$X \subseteq subformulas_\mu\ \varphi$
and
$Y \subseteq subformulas_\nu\ \varphi$
and
$\exists i.\ suffix\ i\ w \models_n af\ \varphi\ (prefix\ i\ w)[X]_\nu$
and
$\forall \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)$
and
$\forall \psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu)$
⟨*proof*⟩

theorem *master-theorem-rtl*:
  assumes
    $X \subseteq subformulas_\mu\ \varphi$
  and
    $Y \subseteq subformulas_\nu\ \varphi$
  and
    1: $\exists i.\ suffix\ i\ w \models_n af\ \varphi\ (prefix\ i\ w)[X]_\nu$
  and
    2: $\forall \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)$
  and
    3: $\forall \psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu)$
  shows
    $w \models_n \varphi$
⟨*proof*⟩

theorem *master-theorem*:
  $w \models_n \varphi \longleftrightarrow$
    $(\exists X \subseteq subformulas_\mu\ \varphi.$
      $(\exists Y \subseteq subformulas_\nu\ \varphi.$
        $(\exists i.\ suffix\ i\ w \models_n af\ \varphi\ (prefix\ i\ w)[X]_\nu)$
        $\wedge (\forall \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu))$
        $\wedge (\forall \psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu))))$
  ⟨*proof*⟩

4.3 The Master Theorem on Languages

definition $L_1\ \varphi\ X = \{w.\ \exists i.\ suffix\ i\ w \models_n af\ \varphi\ (prefix\ i\ w)[X]_\nu\}$

definition $L_2\ X\ Y = \{w.\ \forall \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)\}$

definition $L_3\ X\ Y = \{w.\ \forall \psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu)\}$

corollary *master-theorem-language*:
  $language\text{-}ltln\ \varphi = \bigcup\ \{L_1\ \varphi\ X \cap L_2\ X\ Y \cap L_3\ X\ Y \mid X\ Y.\ X \subseteq subformulas_\mu\ \varphi \wedge Y \subseteq subformulas_\nu\ \varphi\}$
⟨*proof*⟩

end

5 Asymmetric Variant of the Master Theorem

theory *Asymmetric-Master-Theorem*
imports
  Advice After

begin

This variant of the Master Theorem fixes only a subset Y of νLTL subformulas and all conditions depend on the index i. While this does not lead to a simple DRA construction, but can be used to build NBAs and LDBAs.

lemma *FG-advice-b1-helper*:
$\psi \in \mathit{subfrmlsn}\ \varphi \Longrightarrow \mathit{suffix}\ i\ w \models_n \psi \Longrightarrow \mathit{suffix}\ i\ w \models_n \psi[\mathcal{FG}\ \varphi\ w]_\mu$
⟨*proof*⟩

lemma *FG-advice-b2-helper*:
$S \subseteq \mathcal{G}\ \varphi\ (\mathit{suffix}\ i\ w) \Longrightarrow i \le j \Longrightarrow \mathit{suffix}\ j\ w \models_n \psi[S]_\mu \Longrightarrow \mathit{suffix}\ j\ w \models_n \psi$
⟨*proof*⟩

lemma *Y-$\mathcal{G}$*:
assumes
Y-ν: $Y \subseteq \mathit{subformulas}_\nu\ \varphi$
and
Y-G-1: $\forall \psi_1\ \psi_2.\ \psi_1\ R_n\ \psi_2 \in Y \longrightarrow \mathit{suffix}\ i\ w \models_n G_n\ (\psi_2[Y]_\mu)$
and
Y-G-2: $\forall \psi_1\ \psi_2.\ \psi_1\ W_n\ \psi_2 \in Y \longrightarrow \mathit{suffix}\ i\ w \models_n G_n\ (\psi_1[Y]_\mu\ or_n\ \psi_2[Y]_\mu)$
shows
$Y \subseteq \mathcal{G}\ \varphi\ (\mathit{suffix}\ i\ w)$
⟨*proof*⟩

theorem *asymmetric-master-theorem-ltr*:
assumes
$w \models_n \varphi$
obtains Y **and** i **where**
$Y \subseteq \mathit{subformulas}_\nu\ \varphi$
and
$\mathit{suffix}\ i\ w \models_n \mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w)[Y]_\mu$
and
$\forall \psi_1\ \psi_2.\ \psi_1\ R_n\ \psi_2 \in Y \longrightarrow \mathit{suffix}\ i\ w \models_n G_n\ (\psi_2[Y]_\mu)$
and
$\forall \psi_1\ \psi_2.\ \psi_1\ W_n\ \psi_2 \in Y \longrightarrow \mathit{suffix}\ i\ w \models_n G_n\ (\psi_1[Y]_\mu\ or_n\ \psi_2[Y]_\mu)$
⟨*proof*⟩

theorem *asymmetric-master-theorem-rtl*:
assumes
1: $Y \subseteq \mathit{subformulas}_\nu\ \varphi$
and
2: $\mathit{suffix}\ i\ w \models_n \mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w)[Y]_\mu$

and
3: $\forall \psi_1\ \psi_2.\ \psi_1\ R_n\ \psi_2 \in Y \longrightarrow$ *suffix i w* $\models_n G_n\ (\psi_2[Y]_\mu)$
and
4: $\forall \psi_1\ \psi_2.\ \psi_1\ W_n\ \psi_2 \in Y \longrightarrow$ *suffix i w* $\models_n G_n\ (\psi_1[Y]_\mu\ or_n\ \psi_2[Y]_\mu)$
shows
$w \models_n \varphi$
⟨*proof*⟩

theorem *asymmetric-master-theorem*:
$w \models_n \varphi \longleftrightarrow$
$(\exists\, i.\ \exists\, Y \subseteq subformulas_\nu\ \varphi.$
suffix i w $\models_n$ *af* φ (*prefix i w*)$[Y]_\mu$
$\wedge$ ($\forall \psi_1\ \psi_2.\ \psi_1\ R_n\ \psi_2 \in Y \longrightarrow$ *suffix i w* $\models_n G_n\ (\psi_2[Y]_\mu)$)
$\wedge$ ($\forall \psi_1\ \psi_2.\ \psi_1\ W_n\ \psi_2 \in Y \longrightarrow$ *suffix i w* $\models_n G_n\ (\psi_1[Y]_\mu\ or_n\ \psi_2[Y]_\mu)$))
⟨*proof*⟩

end

6 Master Theorem with Reduced Subformulas

theory *Restricted-Master-Theorem*
imports
Master-Theorem
begin

6.1 Restricted Set of Subformulas

fun *restricted-subformulas-inner* :: *'a ltln* ⇒ *'a ltln set*
where
restricted-subformulas-inner (φ and_n ψ) = *restricted-subformulas-inner* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas-inner* (φ or_n ψ) = *restricted-subformulas-inner* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas-inner* (X_n φ) = *restricted-subformulas-inner* φ
| *restricted-subformulas-inner* (φ U_n ψ) = $subformulas_\nu$ (φ U_n ψ) ∪ $subformulas_\mu$ (φ U_n ψ)
| *restricted-subformulas-inner* (φ R_n ψ) = *restricted-subformulas-inner* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas-inner* (φ W_n ψ) = *restricted-subformulas-inner* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas-inner* (φ M_n ψ) = $subformulas_\nu$ (φ M_n ψ) ∪ $subformulas_\mu$ (φ M_n ψ)
| *restricted-subformulas-inner* - = {}

fun *restricted-subformulas* :: *'a ltln* ⇒ *'a ltln set*
where
restricted-subformulas (φ *and*$_n$ ψ) = *restricted-subformulas* φ ∪ *restricted-subformulas* ψ
| *restricted-subformulas* (φ *or*$_n$ ψ) = *restricted-subformulas* φ ∪ *restricted-subformulas* ψ
| *restricted-subformulas* (X_n φ) = *restricted-subformulas* φ
| *restricted-subformulas* (φ U_n ψ) = *restricted-subformulas* φ ∪ *restricted-subformulas* ψ
| *restricted-subformulas* (φ R_n ψ) = *restricted-subformulas* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas* (φ W_n ψ) = *restricted-subformulas-inner* φ ∪ *restricted-subformulas* ψ
| *restricted-subformulas* (φ M_n ψ) = *restricted-subformulas* φ ∪ *restricted-subformulas* ψ
| *restricted-subformulas* - = {}

lemma *GF-advice-restricted-subformulas-inner*:
restricted-subformulas-inner ($\varphi[X]_\nu$) = {}
⟨*proof*⟩

lemma *GF-advice-restricted-subformulas*:
restricted-subformulas ($\varphi[X]_\nu$) = {}
⟨*proof*⟩

lemma *restricted-subformulas-inner-subset*:
restricted-subformulas-inner φ ⊆ *subformulas*$_\nu$ φ ∪ *subformulas*$_\mu$ φ
⟨*proof*⟩

lemma *restricted-subformulas-subset'*:
restricted-subformulas φ ⊆ *restricted-subformulas-inner* φ
⟨*proof*⟩

lemma *restricted-subformulas-subset*:
restricted-subformulas φ ⊆ *subformulas*$_\nu$ φ ∪ *subformulas*$_\mu$ φ
⟨*proof*⟩

lemma *restricted-subformulas-size*:
ψ ∈ *restricted-subformulas* φ ⟹ *size* ψ < *size* φ
⟨*proof*⟩

lemma *restricted-subformulas-notin*:
φ ∉ *restricted-subformulas* φ
⟨*proof*⟩

lemma *restricted-subformulas-superset*:
$\psi \in$ *restricted-subformulas* $\varphi \Longrightarrow$ *subformulas*$_\nu$ $\psi \cup$ *subformulas*$_\mu$ $\psi \subseteq$ *restricted-subformulas* φ
⟨*proof*⟩

lemma *restricted-subformulas-W-μ*:
subformulas$_\mu$ $\varphi \subseteq$ *restricted-subformulas* $(\varphi\ W_n\ \psi)$
⟨*proof*⟩

lemma *restricted-subformulas-R-μ*:
subformulas$_\mu$ $\psi \subseteq$ *restricted-subformulas* $(\varphi\ R_n\ \psi)$
⟨*proof*⟩

lemma *restrict-af-letter*:
restricted-subformulas (*af-letter* φ σ) = *restricted-subformulas* φ
⟨*proof*⟩

lemma *restrict-af*:
restricted-subformulas (*af* φ *w*) = *restricted-subformulas* φ
⟨*proof*⟩

6.2 Restricted Master Theorem / Lemmas

lemma *delay-2*:
assumes *μ-stable* φ *w*
assumes $w \models_n \varphi$
shows $\exists\, i.$ *suffix i w* $\models_n$ *af* φ (*prefix i w*)$[\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi]_\nu$
⟨*proof*⟩

theorem *master-theorem-restricted*:
$w \models_n \varphi \longleftrightarrow$
($\exists\, X \subseteq$ *subformulas*$_\mu$ $\varphi \cap$ *restricted-subformulas* φ.
($\exists\, Y \subseteq$ *subformulas*$_\nu$ $\varphi \cap$ *restricted-subformulas* φ.
($\exists\, i.$ (*suffix i w* $\models_n$ *af* φ (*prefix i w*)$[X]_\nu$)
$\wedge$ ($\forall\, \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)$)
$\wedge$ ($\forall\, \psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu)$)))))
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

corollary *master-theorem-restricted-language*:
language-ltln $\varphi = \bigcup\ \{L_1\ \varphi\ X \cap L_2\ X\ Y \cap L_3\ X\ Y \mid X\ Y.\ X \subseteq$

$subformulas_\mu\ \varphi \cap restricted\text{-}subformulas\ \varphi \wedge Y \subseteq subformulas_\nu\ \varphi \cap restricted\text{-}subformulas\ \varphi\}$
⟨*proof*⟩

6.3 Definitions with Lists for Code Export

definition *restricted-advice-sets* :: $'a\ ltln \Rightarrow ('a\ ltln\ list \times 'a\ ltln\ list)\ list$
where
$restricted\text{-}advice\text{-}sets\ \varphi = List.product\ (subseqs\ (List.filter\ (\lambda x.\ x \in restricted\text{-}subformulas\ \varphi)\ (subformulas_\mu\text{-}list\ \varphi)))\ (subseqs\ (List.filter\ (\lambda x.\ x \in restricted\text{-}subformulas\ \varphi)\ (subformulas_\nu\text{-}list\ \varphi)))$

lemma *subseqs-subformulas*$_\mu$*-restricted-list*:
$X \subseteq subformulas_\mu\ \varphi \cap restricted\text{-}subformulas\ \varphi \longleftrightarrow (\exists\, xs.\ X = set\ xs \wedge xs \in set\ (subseqs\ (List.filter\ (\lambda x.\ x \in restricted\text{-}subformulas\ \varphi)\ (subformulas_\mu\text{-}list\ \varphi))))$
⟨*proof*⟩

lemma *subseqs-subformulas*$_\nu$*-restricted-list*:
$Y \subseteq subformulas_\nu\ \varphi \cap restricted\text{-}subformulas\ \varphi \longleftrightarrow (\exists\, ys.\ Y = set\ ys \wedge ys \in set\ (subseqs\ (List.filter\ (\lambda x.\ x \in restricted\text{-}subformulas\ \varphi)\ (subformulas_\nu\text{-}list\ \varphi))))$
⟨*proof*⟩

lemma *restricted-advice-sets-subformulas*:
$X \subseteq subformulas_\mu\ \varphi \cap restricted\text{-}subformulas\ \varphi \wedge Y \subseteq subformulas_\nu\ \varphi \cap restricted\text{-}subformulas\ \varphi \longleftrightarrow (\exists\, xs\ ys.\ X = set\ xs \wedge Y = set\ ys \wedge (xs, ys) \in set\ (restricted\text{-}advice\text{-}sets\ \varphi))$
⟨*proof*⟩

lemma *restricted-advice-sets-not-empty*:
$restricted\text{-}advice\text{-}sets\ \varphi \neq []$
⟨*proof*⟩

end

7 Transition Functions for Deterministic Automata

theory *Transition-Functions*
imports
../Logical-Characterization/After
../Logical-Characterization/Advice
begin

This theory defines three functions based on the "after"-function which we use as transition functions for deterministic automata.

locale *transition-functions* =
  af-congruent + *GF-advice-congruent*
begin

7.1 After Functions with Resets for *GF* μLTL and *FG* νLTL

definition $\mathit{af\text{-}letter}_F$ :: $'a\ ltln \Rightarrow 'a\ ltln \Rightarrow 'a\ set \Rightarrow 'a\ ltln$
where
  $\mathit{af\text{-}letter}_F\ \varphi\ \psi\ \nu = (\mathit{if}\ \psi \sim \mathit{true}_n\ \mathit{then}\ F_n\ \varphi\ \mathit{else}\ \mathit{af\text{-}letter}\ \psi\ \nu)$

definition $\mathit{af\text{-}letter}_G$ :: $'a\ ltln \Rightarrow 'a\ ltln \Rightarrow 'a\ set \Rightarrow 'a\ ltln$
where
  $\mathit{af\text{-}letter}_G\ \varphi\ \psi\ \nu = (\mathit{if}\ \psi \sim \mathit{false}_n\ \mathit{then}\ G_n\ \varphi\ \mathit{else}\ \mathit{af\text{-}letter}\ \psi\ \nu)$

abbreviation af_F :: $'a\ ltln \Rightarrow 'a\ ltln \Rightarrow 'a\ set\ list \Rightarrow 'a\ ltln$
where
  $\mathit{af}_F\ \varphi\ \psi\ w \equiv \mathit{foldl}\ (\mathit{af\text{-}letter}_F\ \varphi)\ \psi\ w$

abbreviation af_G :: $'a\ ltln \Rightarrow 'a\ ltln \Rightarrow 'a\ set\ list \Rightarrow 'a\ ltln$
where
  $\mathit{af}_G\ \varphi\ \psi\ w \equiv \mathit{foldl}\ (\mathit{af\text{-}letter}_G\ \varphi)\ \psi\ w$

lemma af_F*-step*:
  $\mathit{af}_F\ \varphi\ \psi\ w \sim \mathit{true}_n \Longrightarrow \mathit{af}_F\ \varphi\ \psi\ (w\ @\ [\nu]) = F_n\ \varphi$
  ⟨*proof*⟩

lemma af_G*-step*:
  $\mathit{af}_G\ \varphi\ \psi\ w \sim \mathit{false}_n \Longrightarrow \mathit{af}_G\ \varphi\ \psi\ (w\ @\ [\nu]) = G_n\ \varphi$
  ⟨*proof*⟩

lemma af_F*-segments*:
  $\mathit{af}_F\ \varphi\ \psi\ w = F_n\ \varphi \Longrightarrow \mathit{af}_F\ \varphi\ \psi\ (w\ @\ w') = \mathit{af}_F\ \varphi\ (F_n\ \varphi)\ w'$
  ⟨*proof*⟩

lemma af_G*-segments*:
  $\mathit{af}_G\ \varphi\ \psi\ w = G_n\ \varphi \Longrightarrow \mathit{af}_G\ \varphi\ \psi\ (w\ @\ w') = \mathit{af}_G\ \varphi\ (G_n\ \varphi)\ w'$
  ⟨*proof*⟩

lemma *af-not-true-implies-af-equals-af*$_F$:
  $(\bigwedge xs\ ys.\ w = xs\ @\ ys \Longrightarrow \neg\ \mathit{af}\ \psi\ xs \sim \mathit{true}_n) \Longrightarrow \mathit{af}_F\ \varphi\ \psi\ w = \mathit{af}\ \psi\ w$

⟨*proof*⟩

lemma *af-not-false-implies-af-equals-af*$_G$:
$(\bigwedge xs\ ys.\ w = xs\ @\ ys \Longrightarrow \neg\ af\ \psi\ xs \sim false_n) \Longrightarrow af_G\ \varphi\ \psi\ w = af\ \psi\ w$
⟨*proof*⟩

lemma *af*$_F$*-not-true-implies-af-equals-af*$_F$:
$(\bigwedge xs\ ys.\ w = xs\ @\ ys \Longrightarrow \neg\ af_F\ \varphi\ \psi\ xs \sim true_n) \Longrightarrow af_F\ \varphi\ \psi\ w = af\ \psi\ w$
⟨*proof*⟩

lemma *af*$_G$*-not-false-implies-af-equals-af*$_G$:
$(\bigwedge xs\ ys.\ w = xs\ @\ ys \Longrightarrow \neg\ af_G\ \varphi\ \psi\ xs \sim false_n) \Longrightarrow af_G\ \varphi\ \psi\ w = af\ \psi\ w$
⟨*proof*⟩

lemma *af*$_F$*-true-implies-af-true*:
$af_F\ \varphi\ \psi\ w \sim true_n \Longrightarrow af\ \psi\ w \sim true_n$
⟨*proof*⟩

lemma *af*$_G$*-false-implies-af-false*:
$af_G\ \varphi\ \psi\ w \sim false_n \Longrightarrow af\ \psi\ w \sim false_n$
⟨*proof*⟩

lemma *af-equiv-true-af*$_F$*-prefix-true*:
$af\ \psi\ w \sim true_n \Longrightarrow \exists\, xs\ ys.\ w = xs\ @\ ys \wedge af_F\ \varphi\ \psi\ xs \sim true_n$
⟨*proof*⟩

lemma *af-equiv-false-af*$_G$*-prefix-false*:
$af\ \psi\ w \sim false_n \Longrightarrow \exists\, xs\ ys.\ w = xs\ @\ ys \wedge af_G\ \varphi\ \psi\ xs \sim false_n$
⟨*proof*⟩

lemma *append-take-drop*:
$w = xs\ @\ ys \longleftrightarrow xs = take\ (length\ xs)\ w \wedge ys = drop\ (length\ xs)\ w$
⟨*proof*⟩

lemma *subsequence-split*:
$(w\ [i \rightarrow j]) = xs\ @\ ys \Longrightarrow xs = (w\ [i \rightarrow i + length\ xs])$
⟨*proof*⟩

lemma *subsequence-append-general*:
$i \le k \Longrightarrow k \le j \Longrightarrow (w\ [i \to j]) = (w\ [i \to k])\ @\ (w\ [k \to j])$
⟨*proof*⟩

lemma af_F*-semantics-rtl*:
assumes
$\forall\, i.\ \exists\, j{>}i.\ af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \to j]) \sim true_n$
shows
$\forall\, i.\ \exists\, j.\ af\ (F_n\ \varphi)\ (w\ [i \to j]) \sim_L true_n$
⟨*proof*⟩

lemma af_F*-semantics-ltr*:
assumes
$\forall\, i.\ \exists\, j.\ af\ (F_n\ \varphi)\ (w\ [i \to j]) \sim true_n$
shows
$\forall\, i.\ \exists\, j{>}i.\ af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \to j]) \sim true_n$
⟨*proof*⟩

lemma af_G*-semantics-rtl*:
assumes
$\exists\, i.\ \forall\, j{>}i.\ \neg\ af_G\ \varphi\ (G_n\ \varphi)\ (w\ [0 \to j]) \sim false_n$
shows
$\exists\, i.\ \forall\, j.\ \neg\ af\ (G_n\ \varphi)\ (w\ [i \to j]) \sim false_n$
⟨*proof*⟩

lemma af_G*-semantics-ltr*:
assumes
$\exists\, i.\ \forall\, j.\ \neg\ af\ (G_n\ \varphi)\ (w\ [i \to j]) \sim_L false_n$
shows
$\exists\, i.\ \forall\, j{>}i.\ \neg\ af_G\ \varphi\ (G_n\ \varphi)\ (w\ [0 \to j]) \sim false_n$
⟨*proof*⟩

7.2 After Function using GF-advice

definition $af\text{-}letter_\nu$ :: $'a\ ltln\ set \Rightarrow {'a}\ ltln \times {'a}\ ltln \Rightarrow {'a}\ set \Rightarrow {'a}\ ltln \times {'a}\ ltln$
where
$af\text{-}letter_\nu\ X\ p\ \nu = (if\ snd\ p \sim false_n$
$then\ (af\text{-}letter\ (fst\ p)\ \nu,\ (normalise\ (af\text{-}letter\ (fst\ p)\ \nu))[X]_\nu)$
$else\ (af\text{-}letter\ (fst\ p)\ \nu,\ af\text{-}letter\ (snd\ p)\ \nu))$

abbreviation af_ν :: $'a\ ltln\ set \Rightarrow {'a}\ ltln \times {'a}\ ltln \Rightarrow {'a}\ set\ list \Rightarrow {'a}\ ltln \times$

$'a$ *ltln*
where
$af_\nu\ X\ p\ w \equiv foldl\ (af\text{-}letter_\nu\ X)\ p\ w$

lemma *af-letter*$_\nu$*-fst*[*simp*]:
$fst\ (af\text{-}letter_\nu\ X\ p\ \nu) = af\text{-}letter\ (fst\ p)\ \nu$
⟨*proof*⟩

lemma *af-letter*$_\nu$*-snd*[*simp*]:
$snd\ p \sim false_n \Longrightarrow snd\ (af\text{-}letter_\nu\ X\ p\ \nu) = (normalise\ (af\text{-}letter\ (fst\ p)\ \nu))[X]_\nu$
$\neg\ (snd\ p) \sim false_n \Longrightarrow snd\ (af\text{-}letter_\nu\ X\ p\ \nu) = af\text{-}letter\ (snd\ p)\ \nu$
⟨*proof*⟩

lemma *af*$_\nu$*-fst*:
$fst\ (af_\nu\ X\ p\ w) = af\ (fst\ p)\ w$
⟨*proof*⟩

lemma *af*$_\nu$*-snd*:
$\neg\ af\ (snd\ p)\ w \sim false_n \Longrightarrow snd\ (af_\nu\ X\ p\ w) = af\ (snd\ p)\ w$
⟨*proof*⟩

lemma *af*$_\nu$*-snd'*:
$\forall\, i.\ \neg\ snd\ (af_\nu\ X\ p\ (take\ i\ w)) \sim false_n \Longrightarrow snd\ (af_\nu\ X\ p\ w) = af\ (snd\ p)\ w$
⟨*proof*⟩

lemma *af*$_\nu$*-step*:
$snd\ (af_\nu\ X\ (\xi,\ \zeta)\ w) \sim false_n \Longrightarrow snd\ (af_\nu\ X\ (\xi,\ \zeta)\ (w\ @\ [\nu])) = (normalise\ (af\ \xi\ (w\ @\ [\nu])))[X]_\nu$
⟨*proof*⟩

lemma *af*$_\nu$*-segments*:
$af_\nu\ X\ (\xi,\ \zeta)\ w = (af\ \xi\ w,\ (af\ \xi\ w)[X]_\nu) \Longrightarrow af_\nu\ X\ (\xi,\ \zeta)\ (w\ @\ w') = af_\nu\ X\ (af\ \xi\ w,\ (af\ \xi\ w)[X]_\nu)\ w'$
⟨*proof*⟩

lemma *af*$_\nu$*-semantics-ltr*:
assumes
$\exists\, i.\ suffix\ i\ w \models_n (af\ \varphi\ (prefix\ i\ w))[X]_\nu$
shows
$\exists\, m.\ \forall\, k{\geq}m.\ \neg\ snd\ (af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ (prefix\ (Suc\ k)\ w)) \sim false_n$

⟨*proof*⟩

lemma af_ν*-semantics-rtl*:
assumes
$\exists\, n.\ \forall\, k{\geq}n.\ \neg\ snd\ (af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ (prefix\ (Suc\ k)\ w)) \sim false_n$
shows
$\exists\, i.\ suffix\ i\ w \models_n af\ \varphi\ (prefix\ i\ w)[X]_\nu$
⟨*proof*⟩

end

7.3 Reachability Bounds

We show that the reach of each after-function is bounded by the atomic propositions of the input formula.

locale *transition-functions-size* = *transition-functions* +
assumes
normalise-nested-propos: *nested-prop-atoms* $\varphi \supseteq$ *nested-prop-atoms* (*normalise* φ)
begin

lemma *af-letter*$_F$*-nested-prop-atoms*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi) \Longrightarrow$ *nested-prop-atoms* (*af-letter*$_F$ $\varphi\ \psi\ \nu$) $\subseteq$ *nested-prop-atoms* $(F_n\ \varphi)$
⟨*proof*⟩

lemma af_F*-nested-prop-atoms*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi) \Longrightarrow$ *nested-prop-atoms* $(af_F\ \varphi\ \psi\ w) \subseteq$ *nested-prop-atoms* $(F_n\ \varphi)$
⟨*proof*⟩

lemma *af-letter*$_F$*-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi) \Longrightarrow$ *range* (*af-letter*$_F$ $\varphi\ \psi$) $\subseteq$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi)\}$
⟨*proof*⟩

lemma af_F*-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi) \Longrightarrow$ *range* $(af_F\ \varphi\ \psi) \subseteq$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi)\}$
⟨*proof*⟩

lemma *af-letter*$_G$*-nested-prop-atoms*:

nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(G_n\ \varphi) \Longrightarrow$ *nested-prop-atoms* (*af-letter*$_G$ $\varphi\ \psi\ \nu$) $\subseteq$ *nested-prop-atoms* $(G_n\ \varphi)$
⟨*proof*⟩

lemma *af*$_G$*-nested-prop-atoms*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(G_n\ \varphi) \Longrightarrow$ *nested-prop-atoms* (*af*$_G$ $\varphi\ \psi\ w$) $\subseteq$ *nested-prop-atoms* $(G_n\ \varphi)$
⟨*proof*⟩

lemma *af-letter*$_G$*-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(G_n\ \varphi) \Longrightarrow$ *range* (*af-letter*$_G$ φ ψ) $\subseteq$ $\{\psi$. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $(G_n\ \varphi)\}$
⟨*proof*⟩

lemma *af*$_G$*-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(G_n\ \varphi) \Longrightarrow$ *range* (*af*$_G$ $\varphi\ \psi$) $\subseteq$ $\{\psi$. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $(G_n\ \varphi)\}$
⟨*proof*⟩

lemma *af-letter*$_\nu$*-snd-nested-prop-atoms-helper*:
snd p $\sim$ *false*$_n$ $\Longrightarrow$ *nested-prop-atoms* (*snd* (*af-letter*$_\nu$ *X p* ν)) $\subseteq$ *nested-prop-atoms*$_\nu$ (*fst p*) *X*
$\neg$ *snd p* $\sim$ *false*$_n$ $\Longrightarrow$ *nested-prop-atoms* (*snd* (*af-letter*$_\nu$ *X p* ν)) $\subseteq$ *nested-prop-atoms* (*snd p*)
⟨*proof*⟩

lemma *af-letter*$_\nu$*-fst-nested-prop-atoms*:
nested-prop-atoms (*fst* (*af-letter*$_\nu$ *X p* ν)) $\subseteq$ *nested-prop-atoms* (*fst p*)
⟨*proof*⟩

lemma *af-letter*$_\nu$*-snd-nested-prop-atoms*:
nested-prop-atoms (*snd* (*af-letter*$_\nu$ *X p* ν)) $\subseteq$ (*nested-prop-atoms*$_\nu$ (*fst p*) *X*) $\cup$ (*nested-prop-atoms* (*snd p*))
⟨*proof*⟩

lemma *af-letter*$_\nu$*-fst-range*:
range (*fst* $\circ$ *af-letter*$_\nu$ *X p*) $\subseteq$ $\{\psi$. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (*fst p*)$\}$
⟨*proof*⟩

lemma *af-letter*$_\nu$*-snd-range*:
range (*snd* $\circ$ *af-letter*$_\nu$ *X p*) $\subseteq$ $\{\psi$. *nested-prop-atoms* $\psi \subseteq$ (*nested-prop-atoms*$_\nu$ (*fst p*) *X*) $\cup$ *nested-prop-atoms* (*snd p*)$\}$
⟨*proof*⟩

lemma *af-letter*$_\nu$*-range*:
range (*af-letter*$_\nu$ X p) $\subseteq$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (*fst* p)$\}$ $\times$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ (*nested-prop-atoms*$_\nu$ (*fst* p) X) $\cup$ *nested-prop-atoms* (*snd* p)$\}$
⟨*proof*⟩

lemma *af*$_\nu$*-fst-nested-prop-atoms*:
nested-prop-atoms (*fst* (*af*$_\nu$ X p w)) $\subseteq$ *nested-prop-atoms* (*fst* p)
⟨*proof*⟩

lemma *af-letter-nested-prop-atoms*$_\nu$:
nested-prop-atoms$_\nu$ (*af-letter* φ ν) $X \subseteq$ *nested-prop-atoms*$_\nu$ φ X
⟨*proof*⟩

lemma *af*$_\nu$*-fst-nested-prop-atoms*$_\nu$:
nested-prop-atoms$_\nu$ (*fst* (*af*$_\nu$ X p w)) $X \subseteq$ *nested-prop-atoms*$_\nu$ (*fst* p) X
⟨*proof*⟩

lemma *af*$_\nu$*-fst-range*:
range (*fst* $\circ$ *af*$_\nu$ X p) $\subseteq$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (*fst* p)$\}$
⟨*proof*⟩

lemma *af*$_\nu$*-snd-nested-prop-atoms*:
nested-prop-atoms (*snd* (*af*$_\nu$ X p w)) $\subseteq$ (*nested-prop-atoms*$_\nu$ (*fst* p) X) $\cup$ (*nested-prop-atoms* (*snd* p))
⟨*proof*⟩

lemma *af*$_\nu$*-snd-range*:
range (*snd* $\circ$ *af*$_\nu$ X p) $\subseteq$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ (*nested-prop-atoms*$_\nu$ (*fst* p) X) $\cup$ *nested-prop-atoms* (*snd* p)$\}$
⟨*proof*⟩

lemma *af*$_\nu$*-range*:
range (*af*$_\nu$ X p) $\subseteq$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (*fst* p)$\}$ $\times$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ (*nested-prop-atoms*$_\nu$ (*fst* p) X) $\cup$ *nested-prop-atoms* (*snd* p)$\}$
⟨*proof*⟩

end

end

8 Quotient Type Emulation for Locales

theory *Quotient-Type*
imports
  Main
begin

locale *quotient* =
  fixes
    eq :: $'a \Rightarrow 'a \Rightarrow bool$
  and
    Rep :: $'b \Rightarrow 'a$
  and
    Abs :: $'a \Rightarrow 'b$
  assumes
    Rep-inverse: $Abs\ (Rep\ a) = a$
  and
    Abs-eq: $Abs\ x = Abs\ y \longleftrightarrow eq\ x\ y$
begin

lemma *Rep-inject*:
  $Rep\ x = Rep\ y \longleftrightarrow x = y$
  ⟨*proof*⟩

lemma *Rep-Abs-eq*:
  $eq\ x\ (Rep\ (Abs\ x))$
  ⟨*proof*⟩

end

end

9 Convert between ω-Words and Streams

theory *Omega-Words-Fun-Stream*
imports
  HOL−Library.Omega-Words-Fun HOL−Library.Stream
begin

definition *to-omega* :: $'a\ stream \Rightarrow 'a\ word$ **where**
  to-omega ≡ *snth*

definition *to-stream* :: $'a$ *word* $\Rightarrow$ $'a$ *stream* **where**
to-stream $w \equiv$ *smap* w *nats*

lemma *to-omega-to-stream*[*simp*]:
to-omega (*to-stream* w) $= w$
⟨*proof*⟩

lemma *to-stream-to-omega*[*simp*]:
to-stream (*to-omega* s) $= s$
⟨*proof*⟩

lemma *bij-to-omega*:
bij to-omega
⟨*proof*⟩

lemma *bij-to-stream*:
bij to-stream
⟨*proof*⟩

lemma *image-intersection*[*simp*]:
to-omega ‘ $(A \cap B) =$ *to-omega* ‘ $A \cap$ *to-omega* ‘ B
to-stream ‘ $(C \cap D) =$ *to-stream* ‘ $C \cap$ *to-stream* ‘ D
⟨*proof*⟩

lemma *to-stream-snth*[*simp*]:
(*to-stream* w) !! $k = w\ k$
⟨*proof*⟩

lemma *to-omega-index*[*simp*]:
(*to-omega* s) $k = s$!! k
⟨*proof*⟩

lemma *to-stream-stake*[*simp*]:
stake k (*to-stream* w) $=$ *prefix* $k\ w$
⟨*proof*⟩

lemma *to-omega-prefix*[*simp*]:
prefix k (*to-omega* s) $=$ *stake* $k\ s$
⟨*proof*⟩

lemma *in-image*[*simp*]:
$x \in$ *to-omega* ‘ $X \longleftrightarrow$ *to-stream* $x \in X$

y ∈ *to-stream* ‘ *Y* ⟷ *to-omega y* ∈ *Y*
⟨*proof*⟩

end

10 Constructing DRAs for LTL Formulas

theory *DRA-Construction*
imports
Transition-Functions

../Quotient-Type
../Omega-Words-Fun-Stream

HOL−Library.Log-Nat

../Logical-Characterization/Master-Theorem
../Logical-Characterization/Restricted-Master-Theorem

Transition-Systems-and-Automata.DBA-Combine
Transition-Systems-and-Automata.DCA-Combine
Transition-Systems-and-Automata.DRA-Combine
begin

— We use prefix and suffix on infinite words.
hide-const *Sublist.prefix Sublist.suffix*

locale *dra-construction* = *transition-functions eq normalise* + *quotient eq Rep Abs*
for
eq :: *'a ltln* ⇒ *'a ltln* ⇒ *bool* (**infix** ‹∼› *75*)
and
normalise :: *'a ltln* ⇒ *'a ltln*
and
Rep :: *'ltlq* ⇒ *'a ltln*
and
Abs :: *'a ltln* ⇒ *'ltlq*
begin

10.1 Lifting Setup

abbreviation $true_n$*-lifted* :: *'ltlq* (‹↑$true_n$›) **where**
↑$true_n$ ≡ *Abs* $true_n$

abbreviation *false*$_n$*-lifted* :: *'ltlq* (‹↑*false*$_n$›) **where**
↑*false*$_n$ ≡ *Abs false*$_n$

abbreviation *af-letter-lifted* :: *'a set* ⇒ *'ltlq* ⇒ *'ltlq* (‹↑*afletter*›) **where**
↑*afletter* ν φ ≡ *Abs* (*af-letter* (*Rep* φ) ν)

abbreviation *af-lifted* :: *'ltlq* ⇒ *'a set list* ⇒ *'ltlq* (‹↑*af*›) **where**
↑*af* φ *w* ≡ *fold* ↑*afletter w* φ

abbreviation *GF-advice-lifted* :: *'ltlq* ⇒ *'a ltln set* ⇒ *'ltlq* (‹-↑[-]$_\nu$› [*90*,*60*] *89*) **where**
φ↑[*X*]$_\nu$ ≡ *Abs* ((*Rep* φ)[*X*]$_\nu$)

lemma *af-letter-lifted-semantics*:
↑*afletter* ν (*Abs* φ) = *Abs* (*af-letter* φ ν)
⟨*proof*⟩

lemma *af-lifted-semantics*:
↑*af* (*Abs* φ) *w* = *Abs* (*af* φ *w*)
⟨*proof*⟩

lemma *af-lifted-range*:
range (↑*af* (*Abs* φ)) ⊆ {*Abs* ψ | ψ. *nested-prop-atoms* ψ ⊆ *nested-prop-atoms* φ}
⟨*proof*⟩

definition *af-letter*$_F$*-lifted* :: *'a ltln* ⇒ *'a set* ⇒ *'ltlq* ⇒ *'ltlq* (‹↑*afletter*$_F$›) **where**
↑*afletter*$_F$ φ ν ψ ≡ *Abs* (*af-letter*$_F$ φ (*Rep* ψ) ν)

definition *af-letter*$_G$*-lifted* :: *'a ltln* ⇒ *'a set* ⇒ *'ltlq* ⇒ *'ltlq* (‹↑*afletter*$_G$›) **where**
↑*afletter*$_G$ φ ν ψ ≡ *Abs* (*af-letter*$_G$ φ (*Rep* ψ) ν)

lemma *af-letter*$_F$*-lifted-semantics*:
↑*afletter*$_F$ φ ν (*Abs* ψ) = *Abs* (*af-letter*$_F$ φ ψ ν)
⟨*proof*⟩

lemma *af-letter*$_G$*-lifted-semantics*:
↑*afletter*$_G$ φ ν (*Abs* ψ) = *Abs* (*af-letter*$_G$ φ ψ ν)
⟨*proof*⟩

abbreviation af_F*-lifted* :: $'a\ ltln \Rightarrow 'ltlq \Rightarrow 'a\ set\ list \Rightarrow 'ltlq$ ($\langle\uparrow af_F\rangle$)
where
$\uparrow af_F\ \varphi\ \psi\ w \equiv fold\ (\uparrow afletter_F\ \varphi)\ w\ \psi$

abbreviation af_G*-lifted* :: $'a\ ltln \Rightarrow 'ltlq \Rightarrow 'a\ set\ list \Rightarrow 'ltlq$ ($\langle\uparrow af_G\rangle$)
where
$\uparrow af_G\ \varphi\ \psi\ w \equiv fold\ (\uparrow afletter_G\ \varphi)\ w\ \psi$

lemma af_F*-lifted-semantics*:
$\uparrow af_F\ \varphi\ (Abs\ \psi)\ w = Abs\ (af_F\ \varphi\ \psi\ w)$
$\langle proof\rangle$

lemma af_G*-lifted-semantics*:
$\uparrow af_G\ \varphi\ (Abs\ \psi)\ w = Abs\ (af_G\ \varphi\ \psi\ w)$
$\langle proof\rangle$

definition *af-letter*$_\nu$*-lifted* :: $'a\ ltln\ set \Rightarrow 'a\ set \Rightarrow 'ltlq \times 'ltlq \Rightarrow 'ltlq \times 'ltlq$ ($\langle\uparrow afletter_\nu\rangle$)
where
$\uparrow afletter_\nu\ X\ \nu\ p \equiv$
$(Abs\ (fst\ (af\text{-}letter_\nu\ X\ (Rep\ (fst\ p),\ Rep\ (snd\ p))\ \nu)),$
$Abs\ (snd\ (af\text{-}letter_\nu\ X\ (Rep\ (fst\ p),\ Rep\ (snd\ p))\ \nu)))$

abbreviation af_ν*-lifted* :: $'a\ ltln\ set \Rightarrow 'ltlq \times 'ltlq \Rightarrow 'a\ set\ list \Rightarrow 'ltlq \times 'ltlq$ ($\langle\uparrow af_\nu\rangle$)
where
$\uparrow af_\nu\ X\ p\ w \equiv fold\ (\uparrow afletter_\nu\ X)\ w\ p$

lemma *af-letter*$_\nu$*-lifted-semantics*:
$\uparrow afletter_\nu\ X\ \nu\ (Abs\ x,\ Abs\ y) = (Abs\ (fst\ (af\text{-}letter_\nu\ X\ (x,\ y)\ \nu)),\ Abs\ (snd\ (af\text{-}letter_\nu\ X\ (x,\ y)\ \nu)))$
$\langle proof\rangle$

lemma af_ν*-lifted-semantics*:
$\uparrow af_\nu\ X\ (Abs\ \xi,\ Abs\ \zeta)\ w = (Abs\ (fst\ (af_\nu\ X\ (\xi,\ \zeta)\ w)),\ Abs\ (snd\ (af_\nu\ X\ (\xi,\ \zeta)\ w)))$
$\langle proof\rangle$

10.2 Büchi automata for basic languages

definition $\mathfrak{A}_\mu$ :: $'a\ ltln \Rightarrow ('a\ set,\ 'ltlq)\ dba$ **where**
$\mathfrak{A}_\mu\ \varphi = dba\ UNIV\ (Abs\ \varphi)\ \uparrow afletter\ (\lambda\psi.\ \psi = \uparrow true_n)$

definition $\mathfrak{A}_\mu$*-GF* :: *'a ltln* ⇒ (*'a set*, *'ltlq*) *dba* **where**
$\mathfrak{A}_\mu\text{-}GF\ \varphi = dba\ UNIV\ (Abs\ (F_n\ \varphi))\ (\uparrow afletter_F\ \varphi)\ (\lambda\psi.\ \psi = \uparrow true_n)$

definition $\mathfrak{A}_\nu$ :: *'a ltln* ⇒ (*'a set*, *'ltlq*) *dca* **where**
$\mathfrak{A}_\nu\ \varphi = dca\ UNIV\ (Abs\ \varphi)\ \uparrow afletter\ (\lambda\psi.\ \psi = \uparrow false_n)$

definition $\mathfrak{A}_\nu$*-FG* :: *'a ltln* ⇒ (*'a set*, *'ltlq*) *dca* **where**
$\mathfrak{A}_\nu\text{-}FG\ \varphi = dca\ UNIV\ (Abs\ (G_n\ \varphi))\ (\uparrow afletter_G\ \varphi)\ (\lambda\psi.\ \psi = \uparrow false_n)$

lemma *dba-run*:
DBA.run (*dba UNIV p* δ α) (*to-stream w*) *p* ⟨*proof*⟩

lemma *dca-run*:
DCA.run (*dca UNIV p* δ α) (*to-stream w*) *p* ⟨*proof*⟩

lemma $\mathfrak{A}_\mu$*-language*:
$\varphi \in \mu LTL \Longrightarrow$ *to-stream w* $\in$ *DBA.language* ($\mathfrak{A}_\mu$ φ) $\longleftrightarrow$ $w \models_n \varphi$
⟨*proof*⟩

lemma $\mathfrak{A}_\mu$*-GF-language*:
$\varphi \in \mu LTL \Longrightarrow$ *to-stream w* $\in$ *DBA.language* ($\mathfrak{A}_\mu$*-GF* φ) $\longleftrightarrow$ $w \models_n G_n$ $(F_n\ \varphi)$
⟨*proof*⟩

lemma $\mathfrak{A}_\nu$*-language*:
$\varphi \in \nu LTL \Longrightarrow$ *to-stream w* $\in$ *DCA.language* ($\mathfrak{A}_\nu$ φ) $\longleftrightarrow$ $w \models_n \varphi$
⟨*proof*⟩

lemma $\mathfrak{A}_\nu$*-FG-language*:
$\varphi \in \nu LTL \Longrightarrow$ *to-stream w* $\in$ *DCA.language* ($\mathfrak{A}_\nu$*-FG* φ) $\longleftrightarrow$ $w \models_n F_n$ $(G_n\ \varphi)$
⟨*proof*⟩

10.3 A DCA checking the GF-advice Function

definition $\mathfrak{C}$ :: *'a ltln* ⇒ *'a ltln set* ⇒ (*'a set*, *'ltlq* × *'ltlq*) *dca* **where**
$\mathfrak{C}\ \varphi\ X = dca\ UNIV\ (Abs\ \varphi,\ Abs\ ((normalise\ \varphi)[X]_\nu))\ (\uparrow afletter_\nu\ X)\ (\lambda p.\ snd\ p = \uparrow false_n)$

lemma $\mathfrak{C}$*-language*:
to-stream w $\in$ *DCA.language* ($\mathfrak{C}$ φ X) $\longleftrightarrow$ ($\exists\, i.$ *suffix i w* $\models_n$ *af* φ (*prefix* *i w*)$[X]_\nu$)

$\langle proof \rangle$

10.4 A DRA for each combination of sets X and Y

lemma *dba-language*:
$(\bigwedge w.\ \textit{to-stream}\ w \in \textit{DBA.language}\ \mathfrak{A} \longleftrightarrow w \models_n \varphi) \Longrightarrow \textit{DBA.language}\ \mathfrak{A} = \{w.\ \textit{to-omega}\ w \models_n \varphi\}$
$\langle proof \rangle$

lemma *dca-language*:
$(\bigwedge w.\ \textit{to-stream}\ w \in \textit{DCA.language}\ \mathfrak{A} \longleftrightarrow w \models_n \varphi) \Longrightarrow \textit{DCA.language}\ \mathfrak{A} = \{w.\ \textit{to-omega}\ w \models_n \varphi\}$
$\langle proof \rangle$

definition $\mathfrak{A}_1$ :: $'a\ ltln \Rightarrow {'a}\ ltln\ list \Rightarrow ({'a}\ set, {'ltlq} \times {'ltlq})\ dca$ **where**
$\mathfrak{A}_1\ \varphi\ xs = \mathfrak{C}\ \varphi\ (set\ xs)$

lemma $\mathfrak{A}_1$-*language*:
$\textit{to-omega}\ `\ \textit{DCA.language}\ (\mathfrak{A}_1\ \varphi\ xs) = L_1\ \varphi\ (set\ xs)$
$\langle proof \rangle$

lemma $\mathfrak{A}_1$-*alphabet*:
$\textit{DCA.alphabet}\ (\mathfrak{A}_1\ \varphi\ xs) = \textit{UNIV}$
$\langle proof \rangle$

definition $\mathfrak{A}_2$ :: $'a\ ltln\ list \Rightarrow {'a}\ ltln\ list \Rightarrow ({'a}\ set, {'ltlq}\ list\ degen)\ dba$ **where**
$\mathfrak{A}_2\ xs\ ys = \textit{DBA-Combine.intersect-list}\ (map\ (\lambda\psi.\ \mathfrak{A}_\mu\text{-}GF\ (\psi[set\ ys]_\mu))\ xs)$

lemma $\mathfrak{A}_2$-*language*:
$\textit{to-omega}\ `\ \textit{DBA.language}\ (\mathfrak{A}_2\ xs\ ys) = L_2\ (set\ xs)\ (set\ ys)$
$\langle proof \rangle$

lemma $\mathfrak{A}_2$-*alphabet*:
$\textit{DBA.alphabet}\ (\mathfrak{A}_2\ xs\ ys) = \textit{UNIV}$
$\langle proof \rangle$

definition $\mathfrak{A}_3$ :: $'a\ ltln\ list \Rightarrow {'a}\ ltln\ list \Rightarrow ({'a}\ set, {'ltlq}\ list)\ dca$ **where**
$\mathfrak{A}_3\ xs\ ys = \textit{DCA-Combine.intersect-list}\ (map\ (\lambda\psi.\ \mathfrak{A}_\nu\text{-}FG\ (\psi[set\ xs]_\nu))\ ys)$

lemma $\mathfrak{A}_3$-*language*:
to-omega ' *DCA.language* ($\mathfrak{A}_3$ *xs ys*) = L_3 (*set xs*) (*set ys*)
⟨*proof*⟩

lemma $\mathfrak{A}_3$-*alphabet*:
DCA.alphabet ($\mathfrak{A}_3$ *xs ys*) = *UNIV*
⟨*proof*⟩

definition $\mathfrak{A}'$ φ *xs ys* = *intersect-bc* ($\mathfrak{A}_2$ *xs ys*) (*DCA-Combine.intersect* ($\mathfrak{A}_1$ φ *xs*) ($\mathfrak{A}_3$ *xs ys*))

lemma $\mathfrak{A}'$-*language*:
to-omega ' *DRA.language* ($\mathfrak{A}'$ φ *xs ys*) = (L_1 φ (*set xs*) $\cap$ L_2 (*set xs*) (*set ys*) $\cap$ L_3 (*set xs*) (*set ys*))
⟨*proof*⟩

lemma $\mathfrak{A}'$-*alphabet*:
DRA.alphabet ($\mathfrak{A}'$ φ *xs ys*) = *UNIV*
⟨*proof*⟩

10.5 A DRA for $L\ \varphi$

This is the final constant constructing a deterministic Rabin automaton using the pure version of the $?w \models_n ?\varphi = (\exists X \subseteq subformulas_\mu\ ?\varphi.\ \exists Y \subseteq subformulas_\nu\ ?\varphi.\ (\exists i.\ suffix\ i\ ?w \models_n af\ ?\varphi\ (prefix\ i\ ?w)[X]_\nu) \wedge (\forall \psi \in X.\ ?w \models_n G_n\ (F_n\ \psi[Y]_\mu)) \wedge (\forall \psi \in Y.\ ?w \models_n F_n\ (G_n\ \psi[X]_\nu)))$.

definition *ltl-to-dra* φ = *DRA-Combine.union-list* (*map* (λ(*xs*, *ys*). $\mathfrak{A}'$ φ *xs ys*) (*advice-sets* φ))

lemma *ltl-to-dra-language*:
to-omega ' *DRA.language* (*ltl-to-dra* φ) = *language-ltln* φ
⟨*proof*⟩

lemma *ltl-to-dra-alphabet*:
alphabet (*ltl-to-dra* φ) = *UNIV*
⟨*proof*⟩

10.6 A DRA for $L\ \varphi$ with Restricted Advice Sets

The following constant uses the $?w \models_n ?\varphi = (\exists X \subseteq subformulas_\mu\ ?\varphi \cap restricted\text{-}subformulas\ ?\varphi.\ \exists Y \subseteq subformulas_\nu\ ?\varphi \cap restricted\text{-}subformulas\ ?\varphi.\ \exists i.\ suffix\ i\ ?w \models_n af\ ?\varphi\ (prefix\ i\ ?w)[X]_\nu \wedge (\forall \psi \in X.\ ?w \models_n G_n\ (F_n\ \psi[Y]_\mu))$

$\wedge$ ($\forall\,\psi\in Y.$ *?w* $\models_n$ F_n (G_n $\psi[X]_\nu$))) to reduce the size of the resulting automaton.

definition *ltl-to-dra-restricted* φ = *DRA-Combine.union-list* (*map* (λ(*xs*, *ys*). $\mathfrak{A}'$ φ *xs ys*) (*restricted-advice-sets* φ))

lemma *ltl-to-dra-restricted-language*:
to-omega ‘ *DRA.language* (*ltl-to-dra-restricted* φ) = *language-ltln* φ
⟨*proof*⟩

lemma *ltl-to-dra-restricted-alphabet*:
alphabet (*ltl-to-dra-restricted* φ) = *UNIV*
⟨*proof*⟩

10.7 A DRA for $L\ \varphi$ with a finite alphabet

Until this point, we use *UNIV* as the alphabet in all places. To explore the automaton, however, we need a way to fix the alphabet to some finite set.

definition *dra-set-alphabet* :: (*'a set*, *'b*) *dra* ⇒ *'a set set* ⇒ (*'a set*, *'b*) *dra* **where**
dra-set-alphabet $\mathfrak{A}$ Σ = *dra* Σ (*initial* $\mathfrak{A}$) (*transition* $\mathfrak{A}$) (*condition* $\mathfrak{A}$)

lemma *dra-set-alphabet-language*:
$\Sigma \subseteq$ *alphabet* $\mathfrak{A}$ $\Longrightarrow$ *language* (*dra-set-alphabet* $\mathfrak{A}$ Σ) = *language* $\mathfrak{A}$ $\cap$ {*s*. *sset s* $\subseteq \Sigma$}
⟨*proof*⟩

lemma *dra-set-alphabet-alphabet*[*simp*]:
alphabet (*dra-set-alphabet* $\mathfrak{A}$ Σ) = Σ
⟨*proof*⟩

lemma *dra-set-alphabet-nodes*:
$\Sigma \subseteq$ *alphabet* $\mathfrak{A}$ $\Longrightarrow$ *DRA.nodes* (*dra-set-alphabet* $\mathfrak{A}$ Σ) $\subseteq$ *DRA.nodes* $\mathfrak{A}$
⟨*proof*⟩

definition *ltl-to-dra-alphabet* φ *Ap* = *dra-set-alphabet* (*ltl-to-dra-restricted* φ) (*Pow Ap*)

lemma *ltl-to-dra-alphabet-language*:
assumes
atoms-ltln $\varphi \subseteq$ *Ap*
shows
to-omega ‘ *language* (*ltl-to-dra-alphabet* φ *Ap*) = *language-ltln* φ $\cap$ {*w*.

range w $\subseteq$ *Pow Ap*}
⟨*proof*⟩

lemma *ltl-to-dra-alphabet-alphabet*[*simp*]:
alphabet (*ltl-to-dra-alphabet* φ *Ap*) = *Pow Ap*
⟨*proof*⟩

lemma *ltl-to-dra-alphabet-nodes*:
DRA.nodes (*ltl-to-dra-alphabet* φ *Ap*) $\subseteq$ *DRA.nodes* (*ltl-to-dra-restricted* φ)
⟨*proof*⟩

end

10.8 Verified Bounds for Number of Nodes

Using two additional assumptions, we can show a double-exponential size bound for the constructed automaton.

lemma *list-prod-mono*:
$f \leq g \Longrightarrow (\prod x \leftarrow xs.\ f\ x) \leq (\prod x \leftarrow xs.\ g\ x)$ **for** $f\ g$ :: $'a \Rightarrow nat$
⟨*proof*⟩

lemma *list-prod-const*:
$(\bigwedge x.\ x \in set\ xs \Longrightarrow f\ x \leq c) \Longrightarrow (\prod x \leftarrow xs.\ f\ x) \leq c\ \hat{}\ length\ xs$ **for** f :: $'a \Rightarrow nat$
⟨*proof*⟩

lemma *card-insert-Suc*:
card (*insert x S*) $\leq$ *Suc* (*card S*)
⟨*proof*⟩

lemma *nat-power-le-imp-le*:
$0 < a \Longrightarrow a \leq b \Longrightarrow x\ \hat{}\ a \leq x\ \hat{}\ b$ **for** x :: *nat*
⟨*proof*⟩

lemma *const-less-power*:
$n < x\ \hat{}\ n$ **if** $x > 1$
⟨*proof*⟩

lemma *floorlog-le-const*:
floorlog x n $\leq$ *n*
$\langle$*proof*$\rangle$

locale *dra-construction-size* = *dra-construction* + *transition-functions-size* +
assumes
equiv-finite: *finite P* $\Longrightarrow$ *finite* {*Abs* ψ | ψ. *prop-atoms* $\psi \subseteq$ *P*}
assumes
equiv-card: *finite P* $\Longrightarrow$ *card* {*Abs* ψ | ψ. *prop-atoms* $\psi \subseteq$ *P*} $\leq$ *2* ^ *2* ^ *card P*
begin

lemma af_F*-lifted-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* (F_n φ) $\Longrightarrow$ *range* ($\uparrow af_F$ φ (*Abs* ψ)) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (F_n φ)}
$\langle$*proof*$\rangle$

lemma af_G*-lifted-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* (G_n φ) $\Longrightarrow$ *range* ($\uparrow af_G$ φ (*Abs* ψ)) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (G_n φ)}
$\langle$*proof*$\rangle$

lemma $\mathfrak{A}_\mu$*-nodes*:
DBA.nodes ($\mathfrak{A}_\mu$ φ) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* φ}
$\langle$*proof*$\rangle$

lemma $\mathfrak{A}_\mu$*-GF-nodes*:
DBA.nodes ($\mathfrak{A}_\mu$*-GF* φ) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (F_n φ)}
$\langle$*proof*$\rangle$

lemma $\mathfrak{A}_\nu$*-nodes*:
DCA.nodes ($\mathfrak{A}_\nu$ φ) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* φ}
$\langle$*proof*$\rangle$

lemma $\mathfrak{A}_\nu$*-FG-nodes*:
DCA.nodes ($\mathfrak{A}_\nu$*-FG* φ) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (G_n φ)}

⟨*proof*⟩

lemma $\mathfrak{C}$-*nodes-normalise*:
$DCA.nodes\ (\mathfrak{C}\ \varphi\ X) \subseteq \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms\ \varphi\} \times \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms_\nu\ (normalise\ \varphi)\ X\}$
⟨*proof*⟩

lemma $\mathfrak{C}$-*nodes*:
$DCA.nodes\ (\mathfrak{C}\ \varphi\ X) \subseteq \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms\ \varphi\} \times \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms_\nu\ \varphi\ X\}$
⟨*proof*⟩

lemma *equiv-subset*:
$\{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq P\} \subseteq \{Abs\ \psi \mid \psi.\ prop\text{-}atoms\ \psi \subseteq P\}$
⟨*proof*⟩

lemma *equiv-finite'*:
$finite\ P \Longrightarrow finite\ \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq P\}$
⟨*proof*⟩

lemma *equiv-card'*:
$finite\ P \Longrightarrow card\ \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq P\} \leq 2\ \hat{}\ 2\ \hat{}\ card\ P$
⟨*proof*⟩

lemma *nested-prop-atoms-finite*:
$finite\ \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms\ \varphi\}$
⟨*proof*⟩

lemma *nested-prop-atoms-card*:
$card\ \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms\ \varphi\} \leq 2\ \hat{}\ 2\ \hat{}\ card\ (nested\text{-}prop\text{-}atoms\ \varphi)$
⟨*proof*⟩

lemma *nested-prop-atoms*$_\nu$*-finite*:
$finite\ \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms_\nu\ \varphi\ X\}$
⟨*proof*⟩

lemma *nested-prop-atoms*$_\nu$*-card*:
$card\ \{Abs\ \psi \mid \psi.\ nested\text{-}prop\text{-}atoms\ \psi \subseteq nested\text{-}prop\text{-}atoms_\nu\ \varphi\ X\} \leq 2\ \hat{}$

$2 \,\hat{}\, card\ (nested\text{-}prop\text{-}atoms\ \varphi)$ (**is** $?lhs \leq ?rhs$)
⟨*proof*⟩

lemma $\mathfrak{A}_\mu$-*GF-nodes-finite*:
finite (*DBA.nodes* ($\mathfrak{A}_\mu$-*GF* φ))
⟨*proof*⟩

lemma $\mathfrak{A}_\nu$-*FG-nodes-finite*:
finite (*DCA.nodes* ($\mathfrak{A}_\nu$-*FG* φ))
⟨*proof*⟩

lemma $\mathfrak{A}_\mu$-*GF-nodes-card*:
card (*DBA.nodes* ($\mathfrak{A}_\mu$-*GF* φ)) $\leq 2 \,\hat{}\, 2 \,\hat{}\,$ *card* (*nested-prop-atoms* ($F_n\ \varphi$))
⟨*proof*⟩

lemma $\mathfrak{A}_\nu$-*FG-nodes-card*:
card (*DCA.nodes* ($\mathfrak{A}_\nu$-*FG* φ)) $\leq 2 \,\hat{}\, 2 \,\hat{}\,$ *card* (*nested-prop-atoms* ($G_n\ \varphi$))
⟨*proof*⟩

lemma $\mathfrak{A}_2$-*nodes-finite-helper*:
list-all (*finite* $\circ$ *DBA.nodes*) (*map* ($\lambda\psi$. $\mathfrak{A}_\mu$-*GF* ($\psi[set\ ys]_\mu$)) *xs*)
⟨*proof*⟩

lemma $\mathfrak{A}_2$-*nodes-finite*:
finite (*DBA.nodes* ($\mathfrak{A}_2$ *xs ys*))
⟨*proof*⟩

lemma $\mathfrak{A}_3$-*nodes-finite-helper*:
list-all (*finite* $\circ$ *DCA.nodes*) (*map* ($\lambda\psi$. $\mathfrak{A}_\nu$-*FG* ($\psi[set\ xs]_\nu$)) *ys*)
⟨*proof*⟩

lemma $\mathfrak{A}_3$-*nodes-finite*:
finite (*DCA.nodes* ($\mathfrak{A}_3$ *xs ys*))
⟨*proof*⟩

lemma $\mathfrak{A}_2$-*nodes-card*:
assumes
length xs $\leq n$
and
$\bigwedge\psi.\ \psi \in set\ xs \Longrightarrow card\ (nested\text{-}prop\text{-}atoms\ \psi) \leq n$
shows
card (*DBA.nodes* ($\mathfrak{A}_2$ *xs ys*)) $\leq 2 \,\hat{}\, 2 \,\hat{}\, (n + floorlog\ 2\ n + 2)$

⟨*proof*⟩

lemma $\mathfrak{A}_3$-*nodes-card*:
assumes
length ys $\leq$ *n*
and
$\bigwedge\psi.\ \psi \in$ *set ys* $\Longrightarrow$ *card* (*nested-prop-atoms* ψ) $\leq$ *n*
shows
card (*DCA.nodes* ($\mathfrak{A}_3$ *xs ys*)) $\leq$ *2* ^ *2* ^ (*n* + *floorlog 2 n* + *1*)
⟨*proof*⟩

lemma $\mathfrak{A}_1$-*nodes-finite*:
finite (*DCA.nodes* ($\mathfrak{A}_1$ φ *xs*))
⟨*proof*⟩

lemma $\mathfrak{A}_1$-*nodes-card*:
assumes
card (*subfrmlsn* φ) $\leq$ *n*
shows
card (*DCA.nodes* ($\mathfrak{A}_1$ φ *xs*)) $\leq$ *2* ^ *2* ^ (*n* + *1*)
⟨*proof*⟩

lemma $\mathfrak{A}'$-*nodes-finite*:
finite (*DRA.nodes* ($\mathfrak{A}'$ φ *xs ys*))
⟨*proof*⟩

lemma $\mathfrak{A}'$-*nodes-card*:
assumes
length xs $\leq$ *n*
and
$\bigwedge\psi.\ \psi \in$ *set xs* $\Longrightarrow$ *card* (*nested-prop-atoms* ψ) $\leq$ *n*
and
length ys $\leq$ *n*
and
$\bigwedge\psi.\ \psi \in$ *set ys* $\Longrightarrow$ *card* (*nested-prop-atoms* ψ) $\leq$ *n*
and
card (*subfrmlsn* φ) $\leq$ *n*
shows
card (*DRA.nodes* ($\mathfrak{A}'$ φ *xs ys*)) $\leq$ *2* ^ *2* ^ (*n* + *floorlog 2 n* + *4*)
⟨*proof*⟩

lemma *subformula-nested-prop-atoms-subfrmlsn*:
$\psi \in subfrmlsn\ \varphi \Longrightarrow nested\text{-}prop\text{-}atoms\ \psi \subseteq subfrmlsn\ \varphi$
⟨*proof*⟩

lemma *ltl-to-dra-nodes-finite*:
finite (*DRA.nodes* (*ltl-to-dra* φ))
⟨*proof*⟩

lemma *ltl-to-dra-restricted-nodes-finite*:
finite (*DRA.nodes* (*ltl-to-dra-restricted* φ))
⟨*proof*⟩

lemma *ltl-to-dra-alphabet-nodes-finite*:
finite (*DRA.nodes* (*ltl-to-dra-alphabet* φ *AP*))
⟨*proof*⟩

lemma *ltl-to-dra-nodes-card*:
assumes
card (*subfrmlsn* φ) $\leq n$
shows
card (*DRA.nodes* (*ltl-to-dra* φ)) $\leq 2\ \hat{}\ 2\ \hat{}\ (2 * n + floorlog\ 2\ n + 4)$
⟨*proof*⟩

We verify the size bound of the automaton to be double exponential.

theorem *ltl-to-dra-size*:
card (*DRA.nodes* (*ltl-to-dra* φ)) $\leq 2\ \hat{}\ 2\ \hat{}\ (2 * size\ \varphi + floorlog\ 2\ (size\ \varphi) + 4)$
⟨*proof*⟩

end

end

11 Implementation of the DRA Construction

theory *DRA-Implementation*
imports
DRA-Construction
LTL.Rewriting
Transition-Systems-and-Automata.DRA-Translate
begin

11.1 Generating the Explicit Automaton

We convert the implicit automaton to its explicit representation and afterwards proof the final correctness theorem and the overall size bound.

definition *dra-to-drai* :: $('a, 'b)$ *dra* $\Rightarrow$ *'a list* $\Rightarrow$ $('a, 'b)$ *drai*
where
dra-to-drai $\mathfrak{A}$ Σ = *drai* Σ (*initial* $\mathfrak{A}$) (*transition* $\mathfrak{A}$) (*condition* $\mathfrak{A}$)

lemma *dra-to-drai-language*:
set Σ = *alphabet* $\mathfrak{A} \Longrightarrow$ *language* (*drai-dra* (*dra-to-drai* $\mathfrak{A}$ Σ)) = *language* $\mathfrak{A}$
⟨*proof*⟩

definition *drai-to-draei* :: *nat* $\Rightarrow$ $('a, 'b$ :: *hashable*) *drai* $\Rightarrow$ $('a,$ *nat*) *draei*
where
drai-to-draei hms = *to-draei-impl* (=) *bounded-hashcode-nat hms*

lemma *dra-to-drai-rel*:
assumes
(Σ, *alphabet A*) $\in$ ⟨*Id*⟩ *list-set-rel*
shows
(*dra-to-drai A* Σ, *A*) $\in$ ⟨*Id*, *Id*⟩*drai-dra-rel*
⟨*proof*⟩

lemma *draei-language-rel*:
fixes
A :: $('label, 'state$ :: *hashable*) *dra*
assumes
(Σ, *alphabet A*) $\in$ ⟨*Id*⟩ *list-set-rel*
and
finite (*DRA.nodes A*)
and
is-valid-def-hm-size TYPE$('state)$ *hms*
shows
DRA.language (*drae-dra* (*draei-drae* (*drai-to-draei hms* (*dra-to-drai A* Σ)))) = *DRA.language A*
⟨*proof*⟩

11.2 Defining the Alphabet

fun *atoms-ltlc-list* :: *'a ltlc* $\Rightarrow$ *'a list*
where
atoms-ltlc-list $true_c$ = []

| *atoms-ltlc-list* $false_c$ = []
| *atoms-ltlc-list* $prop_c(q)$ = [*q*]
| *atoms-ltlc-list* (not_c φ) = *atoms-ltlc-list* φ
| *atoms-ltlc-list* (φ and_c ψ) = *List.union* (*atoms-ltlc-list* φ) (*atoms-ltlc-list* ψ)
| *atoms-ltlc-list* (φ or_c ψ) = *List.union* (*atoms-ltlc-list* φ) (*atoms-ltlc-list* ψ)
| *atoms-ltlc-list* (φ $implies_c$ ψ) = *List.union* (*atoms-ltlc-list* φ) (*atoms-ltlc-list* ψ)
| *atoms-ltlc-list* (X_c φ) = *atoms-ltlc-list* φ
| *atoms-ltlc-list* (F_c φ) = *atoms-ltlc-list* φ
| *atoms-ltlc-list* (G_c φ) = *atoms-ltlc-list* φ
| *atoms-ltlc-list* (φ U_c ψ) = *List.union* (*atoms-ltlc-list* φ) (*atoms-ltlc-list* ψ)
| *atoms-ltlc-list* (φ R_c ψ) = *List.union* (*atoms-ltlc-list* φ) (*atoms-ltlc-list* ψ)
| *atoms-ltlc-list* (φ W_c ψ) = *List.union* (*atoms-ltlc-list* φ) (*atoms-ltlc-list* ψ)
| *atoms-ltlc-list* (φ M_c ψ) = *List.union* (*atoms-ltlc-list* φ) (*atoms-ltlc-list* ψ)

lemma *atoms-ltlc-list-set*:
set (*atoms-ltlc-list* φ) = *atoms-ltlc* φ
⟨*proof*⟩

lemma *atoms-ltlc-list-distinct*:
distinct (*atoms-ltlc-list* φ)
⟨*proof*⟩

definition *ltl-alphabet* :: *'a list* ⇒ *'a set list*
where
ltl-alphabet AP = *map set* (*subseqs AP*)

11.3 The Final Constant

We require the quotient type to be hashable in order to efficiently explore the automaton.

locale *dra-implementation* = *dra-construction-size* - - - *Abs*
for
Abs :: *'a ltln* ⇒ *'ltlq* :: *hashable*
begin

definition *ltln-to-draei* :: *'a list* ⇒ *'a ltln* ⇒ (*'a set*, *nat*) *draei*
where
ltln-to-draei AP φ = *drai-to-draei* (*Suc* (*size* φ)) (*dra-to-drai* (*ltl-to-dra-alphabet*

φ (*set AP*)) (*ltl-alphabet AP*))

definition *ltlc-to-draei* :: *'a ltlc* ⇒ (*'a set*, *nat*) *draei*
where
ltlc-to-draei φ = *ltln-to-draei* (*atoms-ltlc-list* φ) (*simplify Slow* (*ltlc-to-ltln* φ))

lemma *ltl-to-dra-alphabet-rel*:
distinct AP ⟹ (*ltl-alphabet AP*, *alphabet* (*ltl-to-dra-alphabet* ψ (*set AP*))) ∈ ⟨*Id*⟩ *list-set-rel*
⟨*proof*⟩

lemma *ltlc-to-ltln-simplify-atoms*:
atoms-ltln (*simplify Slow* (*ltlc-to-ltln* φ)) ⊆ *atoms-ltlc* φ
⟨*proof*⟩

lemma *valid-def-hm-size*:
is-valid-def-hm-size TYPE(*'state*) (*Suc* (*size* φ)) **for** φ :: *'a ltln*
⟨*proof*⟩

theorem *final-correctness*:
to-omega ‘ *language* (*drae-dra* (*draei-drae* (*ltlc-to-draei* φ)))
= *language-ltlc* φ ∩ {*w*. *range w* ⊆ *Pow* (*atoms-ltlc* φ)}
⟨*proof*⟩

end

end

12 Additional Equivalence Relations

theory *Extra-Equivalence-Relations*
imports
LTL.LTL LTL.Equivalence-Relations After Advice
begin

12.1 Propositional Equivalence with Implicit LTL Unfolding

fun *Unf* :: *'a ltln* ⇒ *'a ltln*
where
Unf (φ U_n ψ) = ((φ U_n ψ) and_n *Unf* φ) or_n *Unf* ψ
| *Unf* (φ W_n ψ) = ((φ W_n ψ) and_n *Unf* φ) or_n *Unf* ψ
| *Unf* (φ M_n ψ) = ((φ M_n ψ) or_n *Unf* φ) and_n *Unf* ψ

| $Unf\ (\varphi\ R_n\ \psi) = ((\varphi\ R_n\ \psi)\ or_n\ Unf\ \varphi)\ and_n\ Unf\ \psi$
| $Unf\ (\varphi\ and_n\ \psi) = Unf\ \varphi\ and_n\ Unf\ \psi$
| $Unf\ (\varphi\ or_n\ \psi) = Unf\ \varphi\ or_n\ Unf\ \psi$
| $Unf\ \varphi = \varphi$

lemma *Unf-sound*:
$w \models_n Unf\ \varphi \longleftrightarrow w \models_n \varphi$
⟨*proof*⟩

lemma *Unf-lang-equiv*:
$\varphi \sim_L Unf\ \varphi$
⟨*proof*⟩

lemma *Unf-idem*:
$Unf\ (Unf\ \varphi) \sim_P Unf\ \varphi$
⟨*proof*⟩

definition *ltl-prop-unfold-equiv* :: $'a\ ltln \Rightarrow 'a\ ltln \Rightarrow bool$ (**infix** ‹$\sim_Q$› *75*)
where
$\varphi \sim_Q \psi \equiv (Unf\ \varphi) \sim_P (Unf\ \psi)$

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

lemma *unfolding-prop-unfold-idem*:
$Unf\ \varphi \sim_Q \varphi$
⟨*proof*⟩

lemma *unfolding-is-subst*: $Unf\ \varphi = subst\ \varphi\ (\lambda\psi.\ Some\ (Unf\ \psi))$
⟨*proof*⟩

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

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

lemma *ltl-prop-unfold-equiv-gt-and-lt*:
$(\sim_C) \leq (\sim_Q)\ (\sim_P) \leq (\sim_Q)\ (\sim_Q) \leq (\sim_L)$
⟨*proof*⟩

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

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

lift-definition $ltln_Q$*-eq-test* :: *'a* $ltln_Q$ ⇒ *'a* $ltln_Q$ ⇒ *bool* **is** $\lambda x\ y.\ x \sim_Q y$
⟨*proof*⟩

definition
eq_Q: *equal-class.equal* ≡ $ltln_Q$*-eq-test*

instance
⟨*proof*⟩

end

lemma *af-letter-unfolding*:
af-letter (*Unf* φ) $\nu \sim_P$ *af-letter* φ ν
⟨*proof*⟩

lemma *af-letter-prop-unfold-congruent*:
assumes $\varphi \sim_Q \psi$
shows *af-letter* φ $\nu \sim_Q$ *af-letter* ψ ν
⟨*proof*⟩

lemma *GF-advice-prop-unfold-congruent*:
assumes $\varphi \sim_Q \psi$
shows (*Unf* φ)$[X]_\nu \sim_Q$ (*Unf* ψ)$[X]_\nu$
⟨*proof*⟩

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

interpretation *af-congruent* ($\sim_Q$)
⟨*proof*⟩

lemma *unfolding-monotonic*:
$w \models_n \varphi[X]_\nu \Longrightarrow w \models_n$ (*Unf* φ)$[X]_\nu$
⟨*proof*⟩

lemma *unfolding-next-step-equivalent*:
$w \models_n$ (*Unf* φ)$[X]_\nu \Longrightarrow$ *suffix 1 w* $\models_n$ (*af-letter* φ (*w 0*))$[X]_\nu$
⟨*proof*⟩

lemma *nested-prop-atoms-Unf*:
  nested-prop-atoms (*Unf* φ) $\subseteq$ *nested-prop-atoms* φ
  ⟨*proof*⟩

lemma *refine-image*:
  assumes $\bigwedge x\ y.\ f\ x = f\ y \longrightarrow g\ x = g\ y$
  assumes *finite* (f ‘ X)
  shows *finite* (g ‘ X)
  and *card* (f ‘ X) $\geq$ *card* (g ‘ X)
⟨*proof*⟩

lemma *abs-ltln*$_P$*-implies-abs-ltln*$_Q$:
  abs-ltln$_P$ φ = *abs-ltln*$_P$ ψ $\longrightarrow$ *abs-ltln*$_Q$ φ = *abs-ltln*$_Q$ ψ
  ⟨*proof*⟩

lemmas *prop-unfold-equiv-helper* = *refine-image*[*of abs-ltln*$_P$ *abs-ltln*$_Q$, *OF abs-ltln*$_P$*-implies-abs-ltln*$_Q$]

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

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

lemma *Unf-eventually-equivalent*:
  $w \models_n$ *Unf* $\varphi[X]_\nu \Longrightarrow \exists\, i.$ *suffix i w* $\models_n$ *af* φ (*prefix i w*)$[X]_\nu$
  ⟨*proof*⟩

interpretation *prop-unfold-GF-advice-compatible*: *GF-advice-congruent* ($\sim_Q$) *Unf*
  ⟨*proof*⟩

end

13 Instantiation of the LTL to DRA construction

theory *DRA-Instantiation*
imports
  DRA-Implementation

LTL.Equivalence-Relations
LTL.Disjunctive-Normal-Form
../Logical-Characterization/Extra-Equivalence-Relations
HOL−Library.Log-Nat
Deriving.Derive
begin

13.1 Hash Functions for Quotient Types

derive *hashable ltln*

definition *cube a* = *a* ∗ *a* ∗ *a*

instantiation *set* :: (*hashable*) *hashable*
begin

definition [*simp*]: *hashcode* (*x* :: ′*a set*) = *Finite-Set.fold* (*plus o cube o hashcode*) (*uint32-of-nat* (*card x*)) *x*
definition *def-hashmap-size* = (λ- :: ′*a set itself*. *2* ∗ *def-hashmap-size TYPE*(′*a*))

instance
⟨*proof*⟩

end

instantiation *fset* :: (*hashable*) *hashable*
begin

definition [*simp*]: *hashcode* (*x* :: ′*a fset*) = *hashcode* (*fset x*)
definition *def-hashmap-size* = (λ- :: ′*a fset itself*. *2* ∗ *def-hashmap-size TYPE*(′*a*))

instance
⟨*proof*⟩

end

instantiation $ltln_P$:: (*hashable*) *hashable*
begin

definition [*simp*]: *hashcode* (φ :: *'a ltln*$_P$) = *hashcode* (*min-dnf* (*rep-ltln*$_P$ φ))
definition *def-hashmap-size* = (λ- :: *'a ltln*$_P$ *itself*. *def-hashmap-size* *TYPE*(*'a ltln*))

instance
⟨*proof*⟩

end

instantiation *ltln*$_Q$ :: (*hashable*) *hashable*
begin

definition [*simp*]: *hashcode* (φ :: *'a ltln*$_Q$) = *hashcode* (*min-dnf* (*Unf* (*rep-ltln*$_Q$ φ)))
definition *def-hashmap-size* = (λ- :: *'a ltln*$_Q$ *itself*. *def-hashmap-size* *TYPE*(*'a ltln*))

instance
⟨*proof*⟩

end

13.2 Interpretations with Equivalence Relations

We instantiate the construction locale with propositional equivalence and obtain a function converting a formula into an abstract automaton.

global-interpretation *ltl-to-dra*$_P$: *dra-implementation* ($\sim_P$) *id* *rep-ltln*$_P$ *abs-ltln*$_P$
  defines *ltl-to-dra*$_P$ = *ltl-to-dra*$_P$.*ltl-to-dra*
    and *ltl-to-dra-restricted*$_P$ = *ltl-to-dra*$_P$.*ltl-to-dra-restricted*
    and *ltl-to-dra-alphabet*$_P$ = *ltl-to-dra*$_P$.*ltl-to-dra-alphabet*
    and $\mathfrak{A}'_P$ = *ltl-to-dra*$_P$.$\mathfrak{A}'$
    and $\mathfrak{A}_{1P}$ = *ltl-to-dra*$_P$.$\mathfrak{A}_1$
    and $\mathfrak{A}_{2P}$ = *ltl-to-dra*$_P$.$\mathfrak{A}_2$
    and $\mathfrak{A}_{3P}$ = *ltl-to-dra*$_P$.$\mathfrak{A}_3$
    and $\mathfrak{A}_\nu$-*FG*$_P$ = *ltl-to-dra*$_P$.$\mathfrak{A}_\nu$-*FG*
    and $\mathfrak{A}_\mu$-*GF*$_P$ = *ltl-to-dra*$_P$.$\mathfrak{A}_\mu$-*GF*
    and *af-letter*$_{GP}$ = *ltl-to-dra*$_P$.*af-letter*$_G$
    and *af-letter*$_{FP}$ = *ltl-to-dra*$_P$.*af-letter*$_F$
    and *af-letter*$_G$-*lifted*$_P$ = *ltl-to-dra*$_P$.*af-letter*$_G$-*lifted*
    and *af-letter*$_F$-*lifted*$_P$ = *ltl-to-dra*$_P$.*af-letter*$_F$-*lifted*

and $af\text{-}letter_\nu\text{-}lifted_P$ = $ltl\text{-}to\text{-}dra_P.af\text{-}letter_\nu\text{-}lifted$
and $\mathfrak{C}_P$ = $ltl\text{-}to\text{-}dra_P.\mathfrak{C}$
and $af\text{-}letter_{\nu P}$ = $ltl\text{-}to\text{-}dra_P.af\text{-}letter_\nu$
and $ltln\text{-}to\text{-}draei_P$ = $ltl\text{-}to\text{-}dra_P.ltln\text{-}to\text{-}draei$
and $ltlc\text{-}to\text{-}draei_P$ = $ltl\text{-}to\text{-}dra_P.ltlc\text{-}to\text{-}draei$
⟨*proof*⟩

thm $ltl\text{-}to\text{-}dra_P.ltl\text{-}to\text{-}dra\text{-}language$
thm $ltl\text{-}to\text{-}dra_P.ltl\text{-}to\text{-}dra\text{-}size$
thm $ltl\text{-}to\text{-}dra_P.final\text{-}correctness$

Similarly, we instantiate the locale with a different equivalence relation and obtain another constant for translation of LTL to deterministic Rabin automata.

global-interpretation $ltl\text{-}to\text{-}dra_Q$: *dra-implementation* $(\sim_Q)$ *Unf* $rep\text{-}ltln_Q$ $abs\text{-}ltln_Q$
defines $ltl\text{-}to\text{-}dra_Q$ = $ltl\text{-}to\text{-}dra_Q.ltl\text{-}to\text{-}dra$
and $ltl\text{-}to\text{-}dra\text{-}restricted_Q$ = $ltl\text{-}to\text{-}dra_Q.ltl\text{-}to\text{-}dra\text{-}restricted$
and $ltl\text{-}to\text{-}dra\text{-}alphabet_Q$ = $ltl\text{-}to\text{-}dra_Q.ltl\text{-}to\text{-}dra\text{-}alphabet$
and $\mathfrak{A}'_Q$ = $ltl\text{-}to\text{-}dra_Q.\mathfrak{A}'$
and $\mathfrak{A}_{1Q}$ = $ltl\text{-}to\text{-}dra_Q.\mathfrak{A}_1$
and $\mathfrak{A}_{2Q}$ = $ltl\text{-}to\text{-}dra_Q.\mathfrak{A}_2$
and $\mathfrak{A}_{3Q}$ = $ltl\text{-}to\text{-}dra_Q.\mathfrak{A}_3$
and $\mathfrak{A}_\nu\text{-}FG_Q$ = $ltl\text{-}to\text{-}dra_Q.\mathfrak{A}_\nu\text{-}FG$
and $\mathfrak{A}_\mu\text{-}GF_Q$ = $ltl\text{-}to\text{-}dra_Q.\mathfrak{A}_\mu\text{-}GF$
and $af\text{-}letter_{GQ}$ = $ltl\text{-}to\text{-}dra_Q.af\text{-}letter_G$
and $af\text{-}letter_{FQ}$ = $ltl\text{-}to\text{-}dra_Q.af\text{-}letter_F$
and $af\text{-}letter_G\text{-}lifted_Q$ = $ltl\text{-}to\text{-}dra_Q.af\text{-}letter_G\text{-}lifted$
and $af\text{-}letter_F\text{-}lifted_Q$ = $ltl\text{-}to\text{-}dra_Q.af\text{-}letter_F\text{-}lifted$
and $af\text{-}letter_\nu\text{-}lifted_Q$ = $ltl\text{-}to\text{-}dra_Q.af\text{-}letter_\nu\text{-}lifted$
and $\mathfrak{C}_Q$ = $ltl\text{-}to\text{-}dra_Q.\mathfrak{C}$
and $af\text{-}letter_{\nu Q}$ = $ltl\text{-}to\text{-}dra_Q.af\text{-}letter_\nu$
and $ltln\text{-}to\text{-}draei_Q$ = $ltl\text{-}to\text{-}dra_Q.ltln\text{-}to\text{-}draei$
and $ltlc\text{-}to\text{-}draei_Q$ = $ltl\text{-}to\text{-}dra_Q.ltlc\text{-}to\text{-}draei$
⟨*proof*⟩

thm $ltl\text{-}to\text{-}dra_Q.ltl\text{-}to\text{-}dra\text{-}language$
thm $ltl\text{-}to\text{-}dra_Q.ltl\text{-}to\text{-}dra\text{-}size$
thm $ltl\text{-}to\text{-}dra_Q.final\text{-}correctness$

We allow the user to choose one of the two equivalence relations.

datatype *equiv* = *Prop* | *PropUnfold*

fun *ltlc-to-draei* :: *equiv* ⇒ (*'a* :: *hashable*) *ltlc* ⇒ (*'a set*, *nat*) *draei*

where
ltlc-to-draei Prop = $\textit{ltlc-to-draei}_P$
| *ltlc-to-draei PropUnfold* = $\textit{ltlc-to-draei}_Q$

end

14 Code export to Standard ML

theory *Code-Export*
imports
LTL-to-DRA/DRA-Instantiation
LTL.Code-Equations
HOL−Library.Code-Target-Numeral
begin

14.1 Hashing Sets

global-interpretation *comp-fun-commute plus o cube o hashcode* :: ('*a* :: *hashable*) ⇒ *hashcode* ⇒ *hashcode*
⟨*proof*⟩

lemma [*code*]:
hashcode (*set xs*) = *fold* (*plus o cube o hashcode*) (*remdups xs*) (*uint32-of-nat* (*length* (*remdups xs*)))
⟨*proof*⟩

lemma [*code*]:
hashcode ($\textit{abs-ltln}_P$ φ) = *hashcode* (*min-dnf* φ)
⟨*proof*⟩

lemma *min-dnf-rep-abs*[*simp*]:
min-dnf (*Unf* ($\textit{rep-ltln}_Q$ ($\textit{abs-ltln}_Q$ φ))) = *min-dnf* (*Unf* φ)
⟨*proof*⟩

lemma [*code*]:
hashcode ($\textit{abs-ltln}_Q$ φ) = *hashcode* (*min-dnf* (*Unf* φ))
⟨*proof*⟩

14.2 LTL to DRA

declare $\textit{ltl-to-dra}_P$.$\textit{af-letter}_F$*-lifted-semantics* [*code*]
declare $\textit{ltl-to-dra}_P$.$\textit{af-letter}_G$*-lifted-semantics* [*code*]
declare $\textit{ltl-to-dra}_P$.$\textit{af-letter}_\nu$*-lifted-semantics* [*code*]

declare *ltl-to-dra*$_Q$*.af-letter*$_F$*-lifted-semantics* [*code*]
declare *ltl-to-dra*$_Q$*.af-letter*$_G$*-lifted-semantics* [*code*]
declare *ltl-to-dra*$_Q$*.af-letter*$_\nu$*-lifted-semantics* [*code*]

definition *atoms-ltlc-list-literals* :: *String.literal ltlc* ⇒ *String.literal list*
where
atoms-ltlc-list-literals = *atoms-ltlc-list*

definition *ltlc-to-draei-literals* :: *equiv* ⇒ *String.literal ltlc* ⇒ (*String.literal set*, *nat*) *draei*
where
ltlc-to-draei-literals = *ltlc-to-draei*

definition *sort-transitions* :: (*nat* × *String.literal set* × *nat*) *list* ⇒ (*nat* × *String.literal set* × *nat*) *list*
where
sort-transitions = *sort-key fst*

export-code *True-ltlc Iff-ltlc ltlc-to-draei-literals Prop PropUnfold*
alphabetei initialei transitionei conditionei
integer-of-nat atoms-ltlc-list-literals sort-transitions set
in *SML* **module-name** *LTL* **file-prefix** *LTL-to-DRA*

14.3 LTL to NBA

14.4 LTL to LDBA

end

References

[1] J. Esparza, J. Kretínský, and S. Sickert. One theorem to rule them all: A unified translation of LTL into ω-automata. In A. Dawar and E. Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 384–393. ACM, 2018.