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

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

lemma $\mu LTL\text{-}simp[simp]$:
$\varphi \in \mu LTL \longleftrightarrow is\text{-}\mu LTL\ \varphi$
unfolding $\mu LTL\text{-}def$ **by** *simp*

lemma $\nu LTL\text{-}simp[simp]$:
$\varphi \in \nu LTL \longleftrightarrow is\text{-}\nu LTL\ \varphi$
unfolding $\nu LTL\text{-}def$ **by** *simp*

1.1.1 Subformulas in μLTL and νLTL

fun $subformulas_\mu$ :: *'a ltln* ⇒ *'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* ⇒ *'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*:
$subformulas_\mu\ \varphi = \{\psi \in subfrmlsn\ \varphi.\ \exists\psi_1\ \psi_2.\ \psi = \psi_1\ U_n\ \psi_2 \vee \psi = \psi_1\ M_n\ \psi_2\}$
by (*induction* φ) *auto*

lemma *subformulas*$_\nu$*-semantics*:
$subformulas_\nu\ \varphi = \{\psi \in subfrmlsn\ \varphi.\ \exists\psi_1\ \psi_2.\ \psi = \psi_1\ R_n\ \psi_2 \vee \psi = \psi_1\ W_n\ \psi_2\}$
by (*induction* φ) *auto*

lemma *subformulas*$_\mu$*-subfrmlsn*:
$subformulas_\mu\ \varphi \subseteq subfrmlsn\ \varphi$
by (*induction* φ) *auto*

lemma *subformulas*$_\nu$*-subfrmlsn*:
$subformulas_\nu\ \varphi \subseteq subfrmlsn\ \varphi$
by (*induction* φ) *auto*

lemma *subformulas*$_\mu$*-finite*:
finite ($subformulas_\mu\ \varphi$)
by (*induction* φ) *auto*

lemma *subformulas*$_\nu$*-finite*:
finite ($subformulas_\nu\ \varphi$)
by (*induction* φ) *auto*

lemma *subformulas*$_\mu$*-subset*:
$\psi \in subfrmlsn\ \varphi \Longrightarrow subformulas_\mu\ \psi \subseteq subformulas_\mu\ \varphi$
by (*induction* φ) *auto*

lemma *subformulas*$_\nu$*-subset*:
$\psi \in subfrmlsn\ \varphi \Longrightarrow subformulas_\nu\ \psi \subseteq subformulas_\nu\ \varphi$
by (*induction* φ) *auto*

lemma *subfrmlsn-*μ*LTL*:
$\varphi \in \mu LTL \Longrightarrow subfrmlsn\ \varphi \subseteq \mu LTL$
by (*induction* φ) *auto*

lemma *subfrmlsn-*ν*LTL*:
$\varphi \in \nu LTL \Longrightarrow subfrmlsn\ \varphi \subseteq \nu LTL$
by (*induction* φ) *auto*

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

unfolding *subformulas*$_\mu$*-semantics subformulas*$_\nu$*-semantics*
by *fastforce*

lemma *subformulas*$_{\mu\nu}$*-subfrmlsn*:
subformulas$_\mu$ $\varphi \cup$ *subformulas*$_\nu$ $\varphi \subseteq$ *subfrmlsn* φ
using *subformulas*$_\mu$*-subfrmlsn subformulas*$_\nu$*-subfrmlsn* **by** *blast*

lemma *subformulas*$_{\mu\nu}$*-card*:
card (*subformulas*$_\mu$ $\varphi \cup$ *subformulas*$_\nu$ φ) = *card* (*subformulas*$_\mu$ φ) + *card* (*subformulas*$_\nu$ φ)
by (*simp add*: *subformulas*$_{\mu\nu}$*-disjoint subformulas*$_\mu$*-finite subformulas*$_\nu$*-finite card-Un-disjoint*)

1.2 Stability

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

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

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

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

lemma $\mathcal{GF}$*-semantics*:

$\mathcal{GF}\ \varphi\ w = \{\psi.\ \psi \in subformulas_\mu\ \varphi \wedge w \models_n G_n\ (F_n\ \psi)\}$
by (*induction* φ) *force+*

lemma *$\mathcal{F}$-semantics*:
$\mathcal{F}\ \varphi\ w = \{\psi.\ \psi \in subformulas_\mu\ \varphi \wedge w \models_n F_n\ \psi\}$
by (*induction* φ) *force+*

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

lemma *$\mathcal{F}$-semantics'*:
$\mathcal{F}\ \varphi\ w = subformulas_\mu\ \varphi \cap \{\psi.\ w \models_n F_n\ \psi\}$
unfolding *$\mathcal{F}$-semantics* **by** *auto*

lemma *$\mathcal{GF}$-$\mathcal{F}$-subset*:
$\mathcal{GF}\ \varphi\ w \subseteq \mathcal{F}\ \varphi\ w$
unfolding *$\mathcal{GF}$-semantics $\mathcal{F}$-semantics* **by** *force*

lemma *$\mathcal{GF}$-finite*:
finite ($\mathcal{GF}\ \varphi\ w$)
by (*induction* φ) *auto*

lemma *$\mathcal{GF}$-subformulas$_\mu$*:
$\mathcal{GF}\ \varphi\ w \subseteq subformulas_\mu\ \varphi$
unfolding *$\mathcal{GF}$-semantics* **by** *force*

lemma *$\mathcal{GF}$-subfrmlsn*:
$\mathcal{GF}\ \varphi\ w \subseteq subfrmlsn\ \varphi$
using *$\mathcal{GF}$-subformulas$_\mu$ subformulas$_\mu$-subfrmlsn* **by** *blast*

lemma *$\mathcal{GF}$-elim*:
$\psi \in \mathcal{GF}\ \varphi\ w \Longrightarrow w \models_n G_n\ (F_n\ \psi)$
unfolding *$\mathcal{GF}$-semantics* **by** *simp*

lemma *$\mathcal{GF}$-suffix*:
$\mathcal{GF}\ \varphi\ (suffix\ i\ w) = \mathcal{GF}\ \varphi\ w$
proof
show $\mathcal{GF}\ \varphi\ w \subseteq \mathcal{GF}\ \varphi\ (suffix\ i\ w)$
unfolding *$\mathcal{GF}$-semantics* **by** *auto*
next
show $\mathcal{GF}\ \varphi\ (suffix\ i\ w) \subseteq \mathcal{GF}\ \varphi\ w$
unfolding *$\mathcal{GF}$-semantics GF-Inf-many*

proof *auto*
fix ψ
assume $\exists_\infty j.$ *suffix* $(i + j)$ $w \models_n \psi$
then have $\exists_\infty j.$ *suffix* $(j + i)$ $w \models_n \psi$
by (*simp add*: *algebra-simps*)
then show $\exists_\infty j.$ *suffix* j $w \models_n \psi$
using *INFM-nat-add* **by** *blast*
qed
qed

lemma $\mathcal{GF}$*-subset*:
$\psi \in$ *subfrmlsn* $\varphi \Longrightarrow \mathcal{GF}\ \psi\ w \subseteq \mathcal{GF}\ \varphi\ w$
unfolding $\mathcal{GF}$*-semantics* **using** *subformulas*$_\mu$*-subset* **by** *blast*

lemma $\mathcal{F}$*-finite*:
finite $(\mathcal{F}\ \varphi\ w)$
by (*induction* φ) *auto*

lemma $\mathcal{F}$*-subformulas*$_\mu$:
$\mathcal{F}\ \varphi\ w \subseteq$ *subformulas*$_\mu$ φ
unfolding $\mathcal{F}$*-semantics* **by** *force*

lemma $\mathcal{F}$*-subfrmlsn*:
$\mathcal{F}\ \varphi\ w \subseteq$ *subfrmlsn* φ
using $\mathcal{F}$*-subformulas*$_\mu$ *subformulas*$_\mu$*-subfrmlsn* **by** *blast*

lemma $\mathcal{F}$*-elim*:
$\psi \in \mathcal{F}\ \varphi\ w \Longrightarrow w \models_n F_n\ \psi$
unfolding $\mathcal{F}$*-semantics* **by** *simp*

lemma $\mathcal{F}$*-suffix*:
$\mathcal{F}\ \varphi$ (*suffix* i w) $\subseteq \mathcal{F}\ \varphi\ w$
unfolding $\mathcal{F}$*-semantics* **by** *auto*

lemma $\mathcal{F}$*-subset*:
$\psi \in$ *subfrmlsn* $\varphi \Longrightarrow \mathcal{F}\ \psi\ w \subseteq \mathcal{F}\ \varphi\ w$
unfolding $\mathcal{F}$*-semantics* **using** *subformulas*$_\mu$*-subset* **by** *blast*

definition μ*-stable* φ $w \longleftrightarrow \mathcal{GF}\ \varphi\ w = \mathcal{F}\ \varphi\ w$

lemma *suffix*-μ*-stable*:
$\forall_\infty i.$ μ*-stable* φ (*suffix* i w)

proof −
  have $\forall \psi \in$ *subformulas*$_\mu$ φ. $\forall_\infty i$. *suffix i w* $\models_n G_n (F_n \psi) \longleftrightarrow$ *suffix i w* $\models_n F_n \psi$
    using *Alm-all-GF-F* **by** *blast*

  then have $\forall_\infty i$. $\forall \psi \in$ *subformulas*$_\mu$ φ. *suffix i w* $\models_n G_n (F_n \psi) \longleftrightarrow$ *suffix i w* $\models_n F_n \psi$
    using *subformulas*$_\mu$*-finite eventually-ball-finite* **by** *fast*

  then have $\forall_\infty i$. $\{\psi \in$ *subformulas*$_\mu$ φ. *suffix i w* $\models_n G_n (F_n \psi)\} = \{\psi \in$ *subformulas*$_\mu$ φ. *suffix i w* $\models_n F_n \psi\}$
    by (*rule MOST-mono*) (*blast intro*: *Collect-cong*)

  then show *?thesis*
    unfolding μ*-stable-def* $\mathcal{GF}$*-semantics* $\mathcal{F}$*-semantics*
    by (*rule MOST-mono*) *simp*
qed

lemma μ*-stable-subfrmlsn*:
  μ*-stable* φ $w \Longrightarrow \psi \in$ *subfrmlsn* $\varphi \Longrightarrow \mu$*-stable* ψ w
proof −
  assume *a1*: $\psi \in$ *subfrmlsn* φ **and** *a2*: μ*-stable* φ w
  have *subformulas*$_\mu$ $\psi \subseteq$ *subformulas*$_\mu$ φ
    using *a1* **by** (*simp add*: *subformulas*$_\mu$*-subset*)
  moreover
  have $\mathcal{GF}$ φ $w = \mathcal{F}$ φ w
    using *a2* **by** (*meson* μ*-stable-def*)
  ultimately show *?thesis*
   by (*metis* (*no-types*) *Un-commute* $\mathcal{F}$*-semantics'* $\mathcal{GF}$*-semantics'* μ*-stable-def inf-left-commute inf-sup-absorb sup.orderE*)
qed

lemma μ*-stable-suffix*:
  μ*-stable* φ $w \Longrightarrow \mu$*-stable* φ (*suffix i w*)
  by (*metis* $\mathcal{F}$*-suffix* $\mathcal{GF}$*-*$\mathcal{F}$*-subset* $\mathcal{GF}$*-suffix* μ*-stable-def subset-antisym*)

definition *FG-singleton* w $\varphi \equiv$ *if* $w \models_n F_n (G_n \varphi)$ *then* $\{\varphi\}$ *else* $\{\}$
definition *G-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-singleton* w $(\varphi\ R_n\ \psi) \cup \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
| $\mathcal{FG}$ $(\varphi\ W_n\ \psi)$ $w =$ *FG-singleton* w $(\varphi\ W_n\ \psi) \cup \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
| $\mathcal{FG}$ $(\varphi\ M_n\ \psi)$ $w = \mathcal{FG}\ \varphi\ w \cup \mathcal{FG}\ \psi\ w$
| $\mathcal{FG}$ - - $= \{\}$

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-singleton* w $(\varphi\ R_n\ \psi) \cup \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
| $\mathcal{G}$ $(\varphi\ W_n\ \psi)$ $w =$ *G-singleton* w $(\varphi\ W_n\ \psi) \cup \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
| $\mathcal{G}$ $(\varphi\ M_n\ \psi)$ $w = \mathcal{G}\ \varphi\ w \cup \mathcal{G}\ \psi\ w$
| $\mathcal{G}$ - - $= \{\}$

lemma $\mathcal{FG}$*-semantics*:
$\mathcal{FG}\ \varphi\ w = \{\psi \in subformulas_\nu\ \varphi.\ w \models_n F_n\ (G_n\ \psi)\}$
by (*induction* φ) *force+*

lemma $\mathcal{G}$*-semantics*:
$\mathcal{G}\ \varphi\ w \equiv \{\psi \in subformulas_\nu\ \varphi.\ w \models_n G_n\ \psi\}$
by (*induction* φ) *force+*

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

lemma $\mathcal{G}$*-semantics'*:
$\mathcal{G}\ \varphi\ w = subformulas_\nu\ \varphi \cap \{\psi.\ w \models_n G_n\ \psi\}$
unfolding $\mathcal{G}$*-semantics* **by** *auto*

lemma $\mathcal{G}$*-*$\mathcal{FG}$*-subset*:
$\mathcal{G}\ \varphi\ w \subseteq \mathcal{FG}\ \varphi\ w$
unfolding $\mathcal{G}$*-semantics* $\mathcal{FG}$*-semantics* **by** *force*

lemma $\mathcal{FG}$*-finite*:
finite $(\mathcal{FG}\ \varphi\ w)$
by (*induction* φ) *auto*

lemma $\mathcal{FG}$-*subformulas*$_\nu$:
  $\mathcal{FG}$ φ w $\subseteq$ *subformulas*$_\nu$ φ
  unfolding $\mathcal{FG}$-*semantics* **by** *force*

lemma $\mathcal{FG}$-*subfrmlsn*:
  $\mathcal{FG}$ φ w $\subseteq$ *subfrmlsn* φ
  using $\mathcal{FG}$-*subformulas*$_\nu$ *subformulas*$_\nu$-*subfrmlsn* **by** *blast*

lemma $\mathcal{FG}$-*elim*:
  $\psi \in \mathcal{FG}\ \varphi\ w \Longrightarrow w \models_n F_n\ (G_n\ \psi)$
  unfolding $\mathcal{FG}$-*semantics* **by** *simp*

lemma $\mathcal{FG}$-*suffix*:
  $\mathcal{FG}\ \varphi$ (*suffix* i w) $= \mathcal{FG}\ \varphi\ w$
proof
  show $\mathcal{FG}\ \varphi$ (*suffix* i w) $\subseteq \mathcal{FG}\ \varphi\ w$
    unfolding $\mathcal{FG}$-*semantics* **by** *auto*
next
  show $\mathcal{FG}\ \varphi\ w \subseteq \mathcal{FG}\ \varphi$ (*suffix* i w)
    unfolding $\mathcal{FG}$-*semantics* *FG-Alm-all*
  proof *auto*
    fix ψ
    assume $\forall_\infty j$. *suffix* j $w \models_n \psi$
    then have $\forall_\infty j$. *suffix* $(j + i)$ $w \models_n \psi$
      using *MOST-nat-add* **by** *meson*
    then show $\forall_\infty j$. *suffix* $(i + j)$ $w \models_n \psi$
      by (*simp add*: *algebra-simps*)
  qed
qed

lemma $\mathcal{FG}$-*subset*:
  $\psi \in$ *subfrmlsn* $\varphi \Longrightarrow \mathcal{FG}\ \psi\ w \subseteq \mathcal{FG}\ \varphi\ w$
  unfolding $\mathcal{FG}$-*semantics* **using** *subformulas*$_\nu$-*subset* **by** *blast*

lemma $\mathcal{G}$-*finite*:
  finite ($\mathcal{G}\ \varphi\ w$)
  by (*induction* φ) *auto*

lemma $\mathcal{G}$-*subformulas*$_\nu$:
  $\mathcal{G}\ \varphi\ w \subseteq$ *subformulas*$_\nu$ φ
  unfolding $\mathcal{G}$-*semantics* **by** *force*

lemma $\mathcal{G}$-*subfrmlsn*:
$\mathcal{G}\ \varphi\ w \subseteq$ *subfrmlsn* φ
using $\mathcal{G}$-*subformulas*$_\nu$ *subformulas*$_\nu$-*subfrmlsn* **by** *blast*

lemma $\mathcal{G}$-*elim*:
$\psi \in \mathcal{G}\ \varphi\ w \Longrightarrow w \models_n G_n\ \psi$
unfolding $\mathcal{G}$-*semantics* **by** *simp*

lemma $\mathcal{G}$-*suffix*:
$\mathcal{G}\ \varphi\ w \subseteq \mathcal{G}\ \varphi$ (*suffix i w*)
unfolding $\mathcal{G}$-*semantics* **by** *auto*

lemma $\mathcal{G}$-*subset*:
$\psi \in$ *subfrmlsn* $\varphi \Longrightarrow \mathcal{G}\ \psi\ w \subseteq \mathcal{G}\ \varphi\ w$
unfolding $\mathcal{G}$-*semantics* **using** *subformulas*$_\nu$-*subset* **by** *blast*

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 −
have $\forall \psi \in$ *subformulas*$_\nu$ $\varphi.$ $\forall_\infty i.$ *suffix i w* $\models_n F_n\ (G_n\ \psi) \longleftrightarrow$ *suffix i w* $\models_n G_n\ \psi$
using *Alm-all-FG-G* **by** *blast*

then have $\forall_\infty i.$ $\forall \psi \in$ *subformulas*$_\nu$ $\varphi.$ *suffix i w* $\models_n F_n\ (G_n\ \psi) \longleftrightarrow$ *suffix i w* $\models_n G_n\ \psi$
using *subformulas*$_\nu$-*finite eventually-ball-finite* **by** *fast*

then have $\forall_\infty i.$ $\{\psi \in$ *subformulas*$_\nu$ $\varphi.$ *suffix i w* $\models_n F_n\ (G_n\ \psi)\} = \{\psi \in$ *subformulas*$_\nu$ $\varphi.$ *suffix i w* $\models_n G_n\ \psi\}$
by (*rule MOST-mono*) (*blast intro*: *Collect-cong*)

then show *?thesis*
unfolding ν-*stable-def* $\mathcal{FG}$-*semantics* $\mathcal{G}$-*semantics*
by (*rule MOST-mono*) *simp*
qed

lemma ν-*stable-subfrmlsn*:
ν-*stable* $\varphi\ w \Longrightarrow \psi \in$ *subfrmlsn* $\varphi \Longrightarrow \nu$-*stable* $\psi\ w$
proof −
assume *a1*: $\psi \in$ *subfrmlsn* φ **and** *a2*: ν-*stable* $\varphi\ w$
have *subformulas*$_\nu$ $\psi \subseteq$ *subformulas*$_\nu$ φ

using *a1* **by** (*simp add*: $subformulas_\nu$*-subset*)
moreover
have $\mathcal{FG}\ \varphi\ w = \mathcal{G}\ \varphi\ w$
using *a2* **by** (*meson* ν*-stable-def*)
ultimately show *?thesis*
by (*metis* (*no-types*) *Un-commute* $\mathcal{G}$*-semantics'* $\mathcal{FG}$*-semantics'* ν*-stable-def* *inf-left-commute inf-sup-absorb sup.orderE*)
qed

lemma ν*-stable-suffix*:
ν*-stable* φ $w \Longrightarrow$ ν*-stable* φ (*suffix i w*)
by (*metis* $\mathcal{FG}$*-suffix* $\mathcal{G}$-$\mathcal{FG}$*-subset* $\mathcal{G}$*-suffix* ν*-stable-def antisym-conv*)

1.3 Definitions with Lists for Code Export

The μ- and ν-subformulas as lists:

fun $subformulas_\mu$*-list* :: *'a ltln* $\Rightarrow$ *'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* $\Rightarrow$ *'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
by (*induction* φ) *auto*

lemma $subformulas_\nu$-*list-set*:
set ($subformulas_\nu$-*list* φ) = $subformulas_\nu$ φ
by (*induction* φ) *auto*

lemma $subformulas_\mu$-*list-distinct*:
distinct ($subformulas_\mu$-*list* φ)
by (*induction* φ) *auto*

lemma $subformulas_\nu$-*list-distinct*:
distinct ($subformulas_\nu$-*list* φ)
by (*induction* φ) *auto*

lemma $subformulas_\mu$-*list-length*:
length ($subformulas_\mu$-*list* φ) = *card* ($subformulas_\mu$ φ)
by (*metis* $subformulas_\mu$-*list-set* $subformulas_\mu$-*list-distinct* *distinct-card*)

lemma $subformulas_\nu$-*list-length*:
length ($subformulas_\nu$-*list* φ) = *card* ($subformulas_\nu$ φ)
by (*metis* $subformulas_\nu$-*list-set* $subformulas_\nu$-*list-distinct* *distinct-card*)

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* ⟷ (∃ *xs*. X = *set xs* ∧ *subseq xs ys*)
by (*metis* (*no-types*, *lifting*) *Pow-iff image-iff in-set-subseqs subseqs-powset*)

lemma *subseqs*-$subformulas_\mu$-*list*:
$X \subseteq subformulas_\mu$ φ ⟷ (∃ *xs*. X = *set xs* ∧ *xs* ∈ *set* (*subseqs* ($subformulas_\mu$-*list*

φ)))
  by (*metis subset-subseq subformulas*$_\mu$*-list-set in-set-subseqs*)

lemma *subseqs-subformulas*$_\nu$*-list*:
  $Y \subseteq$ *subformulas*$_\nu$ $\varphi \longleftrightarrow (\exists$ *ys*. $Y =$ *set ys* $\wedge$ *ys* $\in$ *set* (*subseqs* (*subformulas*$_\nu$*-list*
φ)))
  by (*metis subset-subseq subformulas*$_\nu$*-list-set in-set-subseqs*)

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-sets* φ))
 unfolding *advice-sets-def set-product subseqs-subformulas*$_\mu$*-list subseqs-subformulas*$_\nu$*-list*
by *blast*

lemma *subseqs-not-empty*:
  subseqs xs $\neq$ []
  by (*metis empty-iff list.set*(*1*) *subseqs-refl*)

lemma *product-not-empty*:
  xs $\neq [] \Longrightarrow$ *ys* $\neq [] \Longrightarrow$ *List.product xs ys* $\neq$ []
  by (*induction xs*) *simp-all*

lemma *advice-sets-not-empty*:
  advice-sets $\varphi \neq$ []
 unfolding *advice-sets-def* **using** *subseqs-not-empty product-not-empty* **by**
blast

lemma *advice-sets-length*:
  length (*advice-sets* φ) $\leq$ *2* ^ *card* (*subfrmlsn* φ)
 unfolding *advice-sets-def length-product length-subseqs subformulas*$_\mu$*-list-length*
subformulas$_\nu$*-list-length power-add*[*symmetric*]
  by (*metis Suc-1 card-mono lessI power-increasing-iff subformulas*$_{\mu\nu}$*-card*
subformulas$_{\mu\nu}$*-subfrmlsn subfrmlsn-finite*)

lemma *advice-sets-element-length*:
  (*xs*, *ys*) $\in$ *set* (*advice-sets* φ) $\Longrightarrow$ *length xs* $\leq$ *card* (*subfrmlsn* φ)
  (*xs*, *ys*) $\in$ *set* (*advice-sets* φ) $\Longrightarrow$ *length ys* $\leq$ *card* (*subfrmlsn* φ)
  unfolding *advice-sets-def set-product*
  by (*metis SigmaD1 card-mono in-set-subseqs list-emb-length order-trans*
subformulas$_\mu$*-list-length subformulas*$_\mu$*-subfrmlsn subfrmlsn-finite*)
    (*metis SigmaD2 card-mono in-set-subseqs list-emb-length order-trans*
subformulas$_\nu$*-list-length subformulas*$_\nu$*-subfrmlsn subfrmlsn-finite*)

lemma *advice-sets-element-subfrmlsn*:
$(xs,\ ys) \in set\ (advice\text{-}sets\ \varphi) \Longrightarrow set\ xs \subseteq subformulas_\mu\ \varphi$
$(xs,\ ys) \in set\ (advice\text{-}sets\ \varphi) \Longrightarrow set\ ys \subseteq subformulas_\nu\ \varphi$
unfolding *advice-sets-def set-product*
by (*meson SigmaD1 subseqs-subformulas*$_\mu$*-list*)
(*meson SigmaD2 subseqs-subformulas*$_\nu$*-list*)

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*:
$af\ (\varphi\ and_n\ \psi)\ w = (af\ \varphi\ w)\ and_n\ (af\ \psi\ w)$
$af\ (\varphi\ or_n\ \psi)\ w = (af\ \varphi\ w)\ or_n\ (af\ \psi\ w)$

by (*induction w rule*: *rev-induct*) *simp-all*

lemma *af-simps*[*simp*]:
af $true_n$ *w* = $true_n$
af $false_n$ *w* = $false_n$
af (X_n φ) (*x* # *xs*) = *af* φ *xs*
by (*induction w*) *simp-all*

lemma *af-ite-simps*[*simp*]:
af (*if P then* $true_n$ *else* $false_n$) *w* = (*if P then* $true_n$ *else* $false_n$)
af (*if P then* $false_n$ *else* $true_n$) *w* = (*if P then* $false_n$ *else* $true_n$)
by *simp-all*

lemma *af-subsequence-append*:
$i \leq j \Longrightarrow j \leq k \Longrightarrow$ *af* (*af* φ (*w* [*i* → *j*])) (*w* [*j* → *k*]) = *af* φ (*w* [*i* → *k*])
by (*metis foldl-append le-Suc-ex map-append subsequence-def upt-add-eq-append*)

lemma *af-subsequence-U*:
af (φ U_n ψ) (*w* [*0* → *Suc n*]) = (*af* ψ (*w* [*0* → *Suc n*])) or_n ((*af* φ (*w* [*0* → *Suc n*])) and_n *af* (φ U_n ψ) (*w* [*1* → *Suc n*]))
by (*induction n*) *fastforce+*

lemma *af-subsequence-U'*:
af (φ U_n ψ) (*a* # *xs*) = (*af* ψ (*a* # *xs*)) or_n ((*af* φ (*a* # *xs*)) and_n *af* (φ U_n ψ) *xs*)
by (*simp add*: *af-decompose*)

lemma *af-subsequence-R*:
af (φ R_n ψ) (*w* [*0* → *Suc n*]) = (*af* ψ (*w* [*0* → *Suc n*])) and_n ((*af* φ (*w* [*0* → *Suc n*])) or_n *af* (φ R_n ψ) (*w* [*1* → *Suc n*]))
by (*induction n*) *fastforce+*

lemma *af-subsequence-R'*:
af (φ R_n ψ) (*a* # *xs*) = (*af* ψ (*a* # *xs*)) and_n ((*af* φ (*a* # *xs*)) or_n *af* (φ R_n ψ) *xs*)
by (*simp add*: *af-decompose*)

lemma *af-subsequence-W*:
af (φ W_n ψ) (*w* [*0* → *Suc n*]) = (*af* ψ (*w* [*0* → *Suc n*])) or_n ((*af* φ (*w* [*0* → *Suc n*])) and_n *af* (φ W_n ψ) (*w* [*1* → *Suc n*]))
by (*induction n*) *fastforce+*

lemma *af-subsequence-W'*:
af (φ W_n ψ) (*a* # *xs*) = (*af* ψ (*a* # *xs*)) or_n ((*af* φ (*a* # *xs*)) and_n *af*

(φ W_n ψ) *xs*)
  by (*simp add*: *af-decompose*)

lemma *af-subsequence-M*:
  af (φ M_n ψ) (*w* [*0* $\rightarrow$ *Suc n*]) = (*af* ψ (*w* [*0* $\rightarrow$ *Suc n*])) and_n ((*af* φ (*w* [*0* $\rightarrow$ *Suc n*])) or_n *af* (φ M_n ψ) (*w* [*1* $\rightarrow$ *Suc n*]))
  by (*induction n*) *fastforce+*

lemma *af-subsequence-M'*:
  af (φ M_n ψ) (*a* # *xs*) = (*af* ψ (*a* # *xs*)) and_n ((*af* φ (*a* # *xs*)) or_n *af* (φ M_n ψ) *xs*)
  by (*simp add*: *af-decompose*)

lemma *suffix-build*[*simp*]:
  suffix (*Suc n*) (*x* ## *xs*) = *suffix n xs*
  by *fastforce*

lemma *af-letter-build*:
  (*x* ## *w*) $\models_n$ φ $\longleftrightarrow$ *w* $\models_n$ *af-letter* φ *x*
proof (*induction* φ *arbitrary*: *x w*)
  case (*Until-ltln* φ ψ)
  then show *?case*
    unfolding *ltln-expand-Until* **by** *force*
next
  case (*Release-ltln* φ ψ)
  then show *?case*
    unfolding *ltln-expand-Release* **by** *force*
next
  case (*WeakUntil-ltln* φ ψ)
  then show *?case*
    unfolding *ltln-expand-WeakUntil* **by** *force*
next
  case (*StrongRelease-ltln* φ ψ)
  then show *?case*
    unfolding *ltln-expand-StrongRelease* **by** *force*
qed *simp+*

lemma *af-ltl-continuation*:
  (*w* $\frown$ *w'*) $\models_n$ φ $\longleftrightarrow$ *w'* $\models_n$ *af* φ *w*
proof (*induction w arbitrary*: φ *w'*)
  case (*Cons x xs*)
  then show *?case*
    using *af-letter-build* **by** *fastforce*
qed *simp*

2.2 Range of the after function

lemma *af-letter-atoms*:
  atoms-ltln (*af-letter* φ ν) $\subseteq$ *atoms-ltln* φ
  by (*induction* φ) *auto*

lemma *af-atoms*:
  atoms-ltln (*af* φ *w*) $\subseteq$ *atoms-ltln* φ
  by (*induction w rule*: *rev-induct*) (*simp*, *insert af-letter-atoms*, *fastforce*)

lemma *af-letter-nested-prop-atoms*:
  nested-prop-atoms (*af-letter* φ ν) $\subseteq$ *nested-prop-atoms* φ
  by (*induction* φ) *auto*

lemma *af-nested-prop-atoms*:
  nested-prop-atoms (*af* φ *w*) $\subseteq$ *nested-prop-atoms* φ
  by (*induction w rule*: *rev-induct*) (*auto*, *insert af-letter-nested-prop-atoms*, *blast*)

lemma *af-letter-range*:
  range (*af-letter* φ) $\subseteq$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $\varphi\}$
  using *af-letter-nested-prop-atoms* **by** *blast*

lemma *af-range*:
  range (*af* φ) $\subseteq$ $\{\psi.$ *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $\varphi\}$
  using *af-nested-prop-atoms* **by** *blast*

2.3 Subformulas of the after function

lemma *af-letter-subformulas*$_\mu$:
  subformulas$_\mu$ (*af-letter* φ ξ) = *subformulas*$_\mu$ φ
  by (*induction* φ) *auto*

lemma *af-subformulas*$_\mu$:
  subformulas$_\mu$ (*af* φ *w*) = *subformulas*$_\mu$ φ
  using *af-letter-subformulas*$_\mu$
  by (*induction w arbitrary*: φ *rule*: *rev-induct*) (*simp*, *fastforce*)

lemma *af-letter-subformulas*$_\nu$:
  subformulas$_\nu$ (*af-letter* φ ξ) = *subformulas*$_\nu$ φ
  by (*induction* φ) *auto*

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

using *af-letter-subformulas*$_\nu$
by (*induction w arbitrary*: φ *rule*: *rev-induct*) (*simp*, *fastforce*)

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*)
unfolding $\mathcal{GF}$*-semantics′ af-subformulas*$_\mu$ **by** *blast*

lemma $\mathcal{F}$*-af*:
$\mathcal{F}$ (*af* φ (*prefix i w*)) (*suffix i w*) = $\mathcal{F}$ φ (*suffix i w*)
unfolding $\mathcal{F}$*-semantics′ af-subformulas*$_\mu$ **by** *blast*

lemma $\mathcal{FG}$*-af*:
$\mathcal{FG}$ (*af* φ (*prefix i w*)) (*suffix i w*) = $\mathcal{FG}$ φ (*suffix i w*)
unfolding $\mathcal{FG}$*-semantics′ af-subformulas*$_\nu$ **by** *blast*

lemma $\mathcal{G}$*-af*:
$\mathcal{G}$ (*af* φ (*prefix i w*)) (*suffix i w*) = $\mathcal{G}$ φ (*suffix i w*)
unfolding $\mathcal{G}$*-semantics′ af-subformulas*$_\nu$ **by** *blast*

2.5 Congruence

lemma *af-letter-lang-congruent*:
$\varphi \sim_L \psi \Longrightarrow$ *af-letter* φ ν $\sim_L$ *af-letter* ψ ν
unfolding *ltl-lang-equiv-def*
using *af-letter-build* **by** *blast*

lemma *af-lang-congruent*:
$\varphi \sim_L \psi \Longrightarrow$ *af* φ *w* $\sim_L$ *af* ψ *w*
unfolding *ltl-lang-equiv-def* **using** *af-ltl-continuation*
by (*induction* φ) *blast*+

lemma *af-letter-subst*:
af-letter φ ν = *subst* φ ($\lambda\psi$. *Some* (*af-letter* ψ ν))
by (*induction* φ) *auto*

lemma *af-letter-prop-congruent*:
$\varphi \longrightarrow_P \psi \Longrightarrow$ *af-letter* φ ν $\longrightarrow_P$ *af-letter* ψ ν
$\varphi \sim_P \psi \Longrightarrow$ *af-letter* φ ν $\sim_P$ *af-letter* ψ ν
by (*metis af-letter-subst subst-respects-ltl-prop-entailment*)+

lemma *af-prop-congruent*:
$\varphi \longrightarrow_P \psi \Longrightarrow af\ \varphi\ w \longrightarrow_P af\ \psi\ w$
$\varphi \sim_P \psi \Longrightarrow af\ \varphi\ w \sim_P af\ \psi\ w$
by (*induction w arbitrary*: φ ψ) (*insert af-letter-prop-congruent*, *fast-force*+)

lemma *af-letter-const-congruent*:
$\varphi \sim_C \psi \Longrightarrow$ *af-letter* $\varphi\ \nu \sim_C$ *af-letter* $\psi\ \nu$
by (*metis af-letter-subst subst-respects-ltl-const-entailment*)

lemma *af-const-congruent*:
$\varphi \sim_C \psi \Longrightarrow af\ \varphi\ w \sim_C af\ \psi\ w$
by (*induction w arbitrary*: φ ψ) (*insert af-letter-const-congruent*, *fast-force*+)

lemma *af-letter-one-step-back*:
$\{x.\ \mathcal{A} \models_P$ *af-letter* $x\ \sigma\} \models_P \varphi \longleftrightarrow \mathcal{A} \models_P$ *af-letter* $\varphi\ \sigma$
by (*induction* φ) *simp-all*

2.6 Implications

lemma *af-F-prefix-prop*:
$af\ (F_n\ \varphi)\ w \longrightarrow_P af\ (F_n\ \varphi)\ (w' @ w)$
by (*induction* w') (*simp add*: *ltl-prop-implies-def af-decompose*(*1*,*2*))+

lemma *af-G-prefix-prop*:
$af\ (G_n\ \varphi)\ (w' @ w) \longrightarrow_P af\ (G_n\ \varphi)\ w$
by (*induction* w') (*simp add*: *ltl-prop-implies-def af-decompose*(*1*,*2*))+

lemma *af-F-prefix-lang*:
$w \models_n af\ (F_n\ \varphi)\ ys \Longrightarrow w \models_n af\ (F_n\ \varphi)\ (xs @ ys)$
using *af-F-prefix-prop ltl-prop-implication-implies-ltl-implication* **by** *blast*

lemma *af-G-prefix-lang*:
$w \models_n af\ (G_n\ \varphi)\ (xs @ ys) \Longrightarrow w \models_n af\ (G_n\ \varphi)\ ys$
using *af-G-prefix-prop ltl-prop-implication-implies-ltl-implication* **by** *blast*

lemma *af-F-prefix-const-equiv-true*:
$af\ (F_n\ \varphi)\ w \sim_C true_n \Longrightarrow af\ (F_n\ \varphi)\ (w' @ w) \sim_C true_n$
using *af-F-prefix-prop ltl-const-equiv-implies-prop-equiv*(*1*) *ltl-prop-equiv-true-implies-true*

by *blast*

lemma *af-G-prefix-const-equiv-false*:
$af\ (G_n\ \varphi)\ w \sim_C false_n \Longrightarrow af\ (G_n\ \varphi)\ (w' @ w) \sim_C false_n$
using *af-G-prefix-prop ltl-const-equiv-implies-prop-equiv(2) ltl-prop-equiv-false-implied-by-false*
by *blast*

lemma *af-F-prefix-lang-equiv-true*:
$af\ (F_n\ \varphi)\ w \sim_L true_n \Longrightarrow af\ (F_n\ \varphi)\ (w' @ w) \sim_L true_n$
unfolding *ltl-lang-equiv-def*
using *af-F-prefix-lang* **by** *fastforce*

lemma *af-G-prefix-lang-equiv-false*:
$af\ (G_n\ \varphi)\ w \sim_L false_n \Longrightarrow af\ (G_n\ \varphi)\ (w' @ w) \sim_L false_n$
unfolding *ltl-lang-equiv-def*
using *af-G-prefix-lang* **by** *fastforce*

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*:
$\varphi \sim \psi \Longrightarrow af\ \varphi\ xs \sim af\ \psi\ xs$
by (*induction xs arbitrary*: $\varphi\ \psi$) (*insert af-letter-congruent*, *fastforce+*)

lemma *af-append-congruent*:
$af\ \varphi\ w \sim af\ \psi\ w \Longrightarrow af\ \varphi\ (w @ w') \sim af\ \psi\ (w @ w')$
by (*simp add*: *af-congruentness*)

lemma *af-append-true*:
$af\ \varphi\ w \sim true_n \Longrightarrow af\ \varphi\ (w @ w') \sim true_n$
using *af-congruentness* **by** *fastforce*

lemma *af-append-false*:
$af\ \varphi\ w \sim false_n \Longrightarrow af\ \varphi\ (w @ w') \sim false_n$
using *af-congruentness* **by** *fastforce*

lemma *prefix-append-subsequence*:
$i \leq j \Longrightarrow (prefix\ i\ w)\ @\ (w\ [i \rightarrow j]) = prefix\ j\ w$
by (*metis le-add-diff-inverse subsequence-append*)

lemma *af-prefix-congruent*:
$i \leq j \Longrightarrow af\ \varphi\ (prefix\ i\ w) \sim af\ \psi\ (prefix\ i\ w) \Longrightarrow af\ \varphi\ (prefix\ j\ w) \sim af\ \psi\ (prefix\ j\ w)$
by (*metis af-congruentness foldl-append prefix-append-subsequence*)+

lemma *af-prefix-true*:
$i \leq j \Longrightarrow af\ \varphi\ (prefix\ i\ w) \sim true_n \Longrightarrow af\ \varphi\ (prefix\ j\ w) \sim true_n$
by (*metis af-append-true prefix-append-subsequence*)

lemma *af-prefix-false*:
$i \leq j \Longrightarrow af\ \varphi\ (prefix\ i\ w) \sim false_n \Longrightarrow af\ \varphi\ (prefix\ j\ w) \sim false_n$
by (*metis af-append-false prefix-append-subsequence*)

end

interpretation *lang-af-congruent*: *af-congruent* $(\sim_L)$
by *unfold-locales* (*rule af-letter-lang-congruent*)

interpretation *prop-af-congruent*: *af-congruent* $(\sim_P)$
by *unfold-locales* (*rule af-letter-prop-congruent*)

interpretation *const-af-congruent*: *af-congruent* $(\sim_C)$
by *unfold-locales* (*rule af-letter-const-congruent*)

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 −
assume $af\ \varphi\ (prefix\ i\ w) \sim_L true_n$

then have $suffix\ i\ w \models_n af\ \varphi\ (prefix\ i\ w)$
unfolding *ltl-lang-equiv-def* **using** *semantics-ltln.simps*(*1*) **by** *blast*

then show $w \models_n \varphi$
using *af-ltl-continuation* **by** *force*
qed

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

proof −
  assume $w \models_n \varphi$

  then have *suffix i w* $\models_n$ *af* φ (*prefix i w*)
    using *af-ltl-continuation* **by** *fastforce*

  then show $\neg$ (*af* φ (*prefix i w*) $\sim_L$ *false*$_n$)
    unfolding *ltl-lang-equiv-def* **using** *semantics-ltln.simps*(*2*) **by** *blast*
qed

lemma *μLTL-implies-valid-prefix*:
  $\varphi \in \mu LTL \Longrightarrow w \models_n \varphi \Longrightarrow \exists\, i.$ *af* φ (*prefix i w*) $\sim_C$ *true*$_n$
proof (*induction* φ *arbitrary*: *w*)
  case *True-ltln*
  then show *?case*
    using *ltl-const-equiv-equivp equivp-reflp* **by** *fastforce*
next
  case (*Prop-ltln x*)
  then show *?case*
    by (*metis af-letter.simps*(*3*) *foldl-Cons foldl-Nil ltl-const-equiv-equivp equivp-reflp semantics-ltln.simps*(*3*) *subsequence-singleton*)
next
  case (*Nprop-ltln x*)
  then show *?case*
    by (*metis af-letter.simps*(*4*) *foldl-Cons foldl-Nil ltl-const-equiv-equivp equivp-reflp semantics-ltln.simps*(*4*) *subsequence-singleton*)
next
  case (*And-ltln* $\varphi 1$ $\varphi 2$)

  then obtain *i1 i2* **where** *af* $\varphi 1$ (*prefix i1 w*) $\sim_C$ *true*$_n$ **and** *af* $\varphi 2$ (*prefix i2 w*) $\sim_C$ *true*$_n$
    by *fastforce*

  then have *af* $\varphi 1$ (*prefix* (*i1* + *i2*) *w*) $\sim_C$ *true*$_n$ **and** *af* $\varphi 2$ (*prefix* (*i2* + *i1*) *w*) $\sim_C$ *true*$_n$
    using *const-af-congruent.af-prefix-true le-add1* **by** *blast+*

  then have *af* ($\varphi 1$ *and*$_n$ $\varphi 2$) (*prefix* (*i1* + *i2*) *w*) $\sim_C$ *true*$_n$
    unfolding *af-decompose* **by** (*simp add*: *add.commute*)

  then show *?case*
    by *blast*
next
  case (*Or-ltln* $\varphi 1$ $\varphi 2$)

then obtain *i* **where** *af* $\varphi 1$ (*prefix i w*) $\sim_C$ $true_n$ $\vee$ *af* $\varphi 2$ (*prefix i w*) $\sim_C$ $true_n$
by *auto*

then show *?case*
unfolding *af-decompose* **by** *auto*
next
case (*Next-ltln* φ)

then obtain *i* **where** *af* φ (*prefix i* (*suffix 1 w*)) $\sim_C$ $true_n$
by *fastforce*

then show *?case*
by (*metis* (*no-types*, *lifting*) *One-nat-def add.right-neutral af-simps*(*3*) *foldl-Nil foldl-append subsequence-append subsequence-shift subsequence-singleton*)
next
case (*Until-ltln* $\varphi 1$ $\varphi 2$)

then obtain *k* **where** *suffix k w* $\models_n$ $\varphi 2$ **and** $\forall j<k.$ *suffix j w* $\models_n$ $\varphi 1$
by *fastforce*

then show *?case*
proof (*induction k arbitrary*: *w*)
case *0*

then obtain *i* **where** *af* $\varphi 2$ (*prefix i w*) $\sim_C$ $true_n$
using *Until-ltln* **by** *fastforce*

then have *af* $\varphi 2$ (*prefix* (*Suc i*) *w*) $\sim_C$ $true_n$
using *const-af-congruent.af-prefix-true le-SucI* **by** *blast*

then have *af* ($\varphi 1$ U_n $\varphi 2$) (*prefix* (*Suc i*) *w*) $\sim_C$ $true_n$
unfolding *af-subsequence-U* **by** *simp*

then show *?case*
by *blast*
next
case (*Suc k*)

then have *suffix k* (*suffix 1 w*) $\models_n$ $\varphi 2$ **and** $\forall j<k.$ *suffix j* (*suffix 1 w*) $\models_n$ $\varphi 1$
by (*simp add*: *Suc.prems*)+

then obtain *i* **where** *i-def*: *af* (φ*1* U_n φ*2*) (*prefix i* (*suffix 1 w*)) $\sim_C$ *true*$_n$
using *Suc.IH* **by** *blast*

obtain *j* **where** *af* φ*1* (*prefix j w*) $\sim_C$ *true*$_n$
using *Until-ltln Suc* **by** *fastforce*

then have *af* φ*1* (*prefix* (*Suc* (*j* + *i*)) *w*) $\sim_C$ *true*$_n$
using *const-af-congruent.af-prefix-true le-SucI le-add1* **by** *blast*

moreover

from *i-def* **have** *af* (φ*1* U_n φ*2*) (*w* [*1* $\rightarrow$ *Suc* (*j* + *i*)]) $\sim_C$ *true*$_n$
by (*metis One-nat-def const-af-congruent.af-prefix-true le-add2 plus-1-eq-Suc subsequence-shift*)

ultimately

have *af* (φ*1* U_n φ*2*) (*prefix* (*Suc* (*j* + *i*)) *w*) $\sim_C$ *true*$_n$
unfolding *af-subsequence-U* **by** *simp*

then show *?case*
by *blast*
qed
next
case (*StrongRelease-ltln* φ*1* φ*2*)

then obtain *k* **where** *suffix k w* $\models_n$ φ*1* **and** $\forall$ *j*$\leq$*k*. *suffix j w* $\models_n$ φ*2*
by *fastforce*

then show *?case*
proof (*induction k arbitrary*: *w*)
case *0*

then obtain *i1 i2* **where** *af* φ*1* (*prefix i1 w*) $\sim_C$ *true*$_n$ **and** *af* φ*2* (*prefix i2 w*) $\sim_C$ *true*$_n$
using *StrongRelease-ltln* **by** *fastforce*

then have *af* φ*1* (*prefix* (*Suc* (*i1* + *i2*)) *w*) $\sim_C$ *true*$_n$ **and** *af* φ*2* (*prefix* (*Suc* (*i2* + *i1*)) *w*) $\sim_C$ *true*$_n$
using *const-af-congruent.af-prefix-true le-SucI le-add1* **by** *blast+*

then have *af* (φ*1* M_n φ*2*) (*prefix* (*Suc* (*i1* + *i2*)) *w*) $\sim_C$ *true*$_n$
unfolding *af-subsequence-M* **by** (*simp add*: *add.commute*)

then show *?case*
by *blast*
next
case (*Suc k*)

then have *suffix k* (*suffix 1 w*) $\models_n$ *φ1* **and** ∀ *j*≤*k*. *suffix j* (*suffix 1 w*) $\models_n$ *φ2*
by (*simp add*: *Suc.prems*)+

then obtain *i* **where** *i-def*: *af* (*φ1* M_n *φ2*) (*prefix i* (*suffix 1 w*)) $\sim_C$ $true_n$
using *Suc.IH* **by** *blast*

obtain *j* **where** *af φ2* (*prefix j w*) $\sim_C$ $true_n$
using *StrongRelease-ltln Suc* **by** *fastforce*

then have *af φ2* (*prefix* (*Suc* (*j* + *i*)) *w*) $\sim_C$ $true_n$
using *const-af-congruent.af-prefix-true le-SucI le-add1* **by** *blast*

moreover

from *i-def* **have** *af* (*φ1* M_n *φ2*) (*w* [*1* → *Suc* (*j* + *i*)]) $\sim_C$ $true_n$
by (*metis One-nat-def const-af-congruent.af-prefix-true le-add2 plus-1-eq-Suc subsequence-shift*)

ultimately

have *af* (*φ1* M_n *φ2*) (*prefix* (*Suc* (*j* + *i*)) *w*) $\sim_C$ $true_n$
unfolding *af-subsequence-M* **by** *simp*

then show *?case*
by *blast*
qed
qed *force+*

lemma *satisfiable-prefix-implies-νLTL*:
φ ∈ *νLTL* ⟹ ∄ *i*. *af φ* (*prefix i w*) $\sim_C$ $false_n$ ⟹ *w* $\models_n$ *φ*
proof (*erule contrapos-np*, *induction φ arbitrary*: *w*)
case *False-ltln*
then show *?case*
using *ltl-const-equiv-equivp equivp-reflp* **by** *fastforce*
next
case (*Prop-ltln x*)

then show *?case*
by (*metis af-letter.simps*(*3*) *foldl-Cons foldl-Nil ltl-const-equiv-equivp equivp-reflp semantics-ltln.simps*(*3*) *subsequence-singleton*)
next
case (*Nprop-ltln x*)
then show *?case*
by (*metis af-letter.simps*(*4*) *foldl-Cons foldl-Nil ltl-const-equiv-equivp equivp-reflp semantics-ltln.simps*(*4*) *subsequence-singleton*)
next
case (*And-ltln φ1 φ2*)

then obtain *i* **where** *af φ1* (*prefix i w*) $\sim_C$ $false_n$ ∨ *af φ2* (*prefix i w*) $\sim_C$ $false_n$
by *auto*

then show *?case*
unfolding *af-decompose* **by** *auto*
next
case (*Or-ltln φ1 φ2*)

then obtain *i1 i2* **where** *af φ1* (*prefix i1 w*) $\sim_C$ $false_n$ **and** *af φ2* (*prefix i2 w*) $\sim_C$ $false_n$
by *fastforce*

then have *af φ1* (*prefix* (*i1* + *i2*) *w*) $\sim_C$ $false_n$ **and** *af φ2* (*prefix* (*i2* + *i1*) *w*) $\sim_C$ $false_n$
using *const-af-congruent.af-prefix-false le-add1* **by** *blast+*

then have *af* (*φ1* or_n *φ2*) (*prefix* (*i1* + *i2*) *w*) $\sim_C$ $false_n$
unfolding *af-decompose* **by** (*simp add*: *add.commute*)

then show *?case*
by *blast*
next
case (*Next-ltln φ*)

then obtain *i* **where** *af φ* (*prefix i* (*suffix 1 w*)) $\sim_C$ $false_n$
by *fastforce*

then show *?case*
by (*metis* (*no-types*, *lifting*) *One-nat-def add.right-neutral af-simps*(*3*) *foldl-Nil foldl-append subsequence-append subsequence-shift subsequence-singleton*)
next
case (*Release-ltln φ1 φ2*)

then obtain k **where** $\neg$ *suffix k w* $\models_n$ $\varphi 2$ **and** $\forall j<k.$ $\neg$ *suffix j w* $\models_n$ $\varphi 1$
by *fastforce*

then show *?case*
proof (*induction k arbitrary*: *w*)
case *0*

then obtain i **where** *af* $\varphi 2$ (*prefix i w*) $\sim_C$ $false_n$
using *Release-ltln* **by** *fastforce*

then have *af* $\varphi 2$ (*prefix* (*Suc i*) *w*) $\sim_C$ $false_n$
using *const-af-congruent.af-prefix-false le-SucI* **by** *blast*

then have *af* ($\varphi 1$ R_n $\varphi 2$) (*prefix* (*Suc i*) *w*) $\sim_C$ $false_n$
unfolding *af-subsequence-R* **by** *simp*

then show *?case*
by *blast*
next
case (*Suc k*)

then have $\neg$ *suffix k* (*suffix 1 w*) $\models_n$ $\varphi 2$ **and** $\forall j<k.$ $\neg$ *suffix j* (*suffix 1 w*) $\models_n$ $\varphi 1$
by (*simp add*: *Suc.prems*)+

then obtain i **where** *i-def*: *af* ($\varphi 1$ R_n $\varphi 2$) (*prefix i* (*suffix 1 w*)) $\sim_C$ $false_n$
using *Suc.IH* **by** *blast*

obtain j **where** *af* $\varphi 1$ (*prefix j w*) $\sim_C$ $false_n$
using *Release-ltln Suc* **by** *fastforce*

then have *af* $\varphi 1$ (*prefix* (*Suc* (j + i)) *w*) $\sim_C$ $false_n$
using *const-af-congruent.af-prefix-false le-SucI le-add1* **by** *blast*

moreover

from *i-def* **have** *af* ($\varphi 1$ R_n $\varphi 2$) (*w* [$1 \rightarrow$ *Suc* (j + i)]) $\sim_C$ $false_n$
by (*metis One-nat-def const-af-congruent.af-prefix-false le-add2 plus-1-eq-Suc subsequence-shift*)

ultimately

have *af* (φ*1* R_n φ*2*) (*prefix* (*Suc* (*j* + *i*)) *w*) $\sim_C$ $false_n$
unfolding *af-subsequence-R* **by** *auto*

then show *?case*
by *blast*
qed
next
case (*WeakUntil-ltln* φ*1* φ*2*)

then obtain *k* **where** ¬ *suffix k w* $\models_n$ φ*1* **and** $\forall j \leq k$. ¬ *suffix j w* $\models_n$ φ*2*
by *fastforce*

then show *?case*
proof (*induction k arbitrary*: *w*)
case *0*

then obtain *i1 i2* **where** *af* φ*1* (*prefix i1 w*) $\sim_C$ $false_n$ **and** *af* φ*2* (*prefix i2 w*) $\sim_C$ $false_n$
using *WeakUntil-ltln* **by** *fastforce*

then have *af* φ*1* (*prefix* (*Suc* (*i1* + *i2*)) *w*) $\sim_C$ $false_n$ **and** *af* φ*2* (*prefix* (*Suc* (*i2* + *i1*)) *w*) $\sim_C$ $false_n$
using *const-af-congruent.af-prefix-false le-SucI le-add1* **by** *blast+*

then have *af* (φ*1* W_n φ*2*) (*prefix* (*Suc* (*i1* + *i2*)) *w*) $\sim_C$ $false_n$
unfolding *af-subsequence-W* **by** (*simp add*: *add.commute*)

then show *?case*
by *blast*
next
case (*Suc k*)

then have ¬ *suffix k* (*suffix 1 w*) $\models_n$ φ*1* **and** $\forall j \leq k$. ¬ *suffix j* (*suffix 1 w*) $\models_n$ φ*2*
by (*simp add*: *Suc.prems*)+

then obtain *i* **where** *i-def*: *af* (φ*1* W_n φ*2*) (*prefix i* (*suffix 1 w*)) $\sim_C$ $false_n$
using *Suc.IH* **by** *blast*

obtain *j* **where** *af* φ*2* (*prefix j w*) $\sim_C$ $false_n$
using *WeakUntil-ltln Suc* **by** *fastforce*

then have *af* $\varphi 2$ (*prefix* (*Suc* (*j* + *i*)) *w*) $\sim_C$ $false_n$
  using *const-af-congruent.af-prefix-false le-SucI le-add1* **by** *blast*

moreover

from *i-def* **have** *af* ($\varphi 1$ W_n $\varphi 2$) (*w* [*1* → *Suc* (*j* + *i*)]) $\sim_C$ $false_n$
by (*metis One-nat-def const-af-congruent.af-prefix-false le-add2 plus-1-eq-Suc subsequence-shift*)

ultimately

have *af* ($\varphi 1$ W_n $\varphi 2$) (*prefix* (*Suc* (*j* + *i*)) *w*) $\sim_C$ $false_n$
  unfolding *af-subsequence-W* **by** *simp*

then show *?case*
  by *blast*
qed
qed *force+*

context *ltl-equivalence*
begin

lemma *valid-prefix-implies-ltl*:
af φ (*prefix i w*) $\sim$ $true_n$ $\Longrightarrow$ $w \models_n \varphi$
using *valid-prefix-implies-ltl eq-implies-lang* **by** *blast*

lemma *ltl-implies-satisfiable-prefix*:
$w \models_n \varphi \Longrightarrow \neg$ (*af* φ (*prefix i w*) $\sim$ $false_n$)
using *ltl-implies-satisfiable-prefix eq-implies-lang* **by** *blast*

lemma *μLTL-implies-valid-prefix*:
$\varphi \in \mu LTL \Longrightarrow w \models_n \varphi \Longrightarrow \exists\, i.$ *af* φ (*prefix i w*) $\sim$ $true_n$
using *μLTL-implies-valid-prefix const-implies-eq* **by** *blast*

lemma *satisfiable-prefix-implies-νLTL*:
$\varphi \in \nu LTL \Longrightarrow \nexists\, i.$ *af* φ (*prefix i w*) $\sim$ $false_n \Longrightarrow w \models_n \varphi$
using *satisfiable-prefix-implies-νLTL const-implies-eq* **by** *blast*

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

using *valid-prefix-implies-ltl* μ*LTL-implies-valid-prefix* **by** *blast*

lemma *af-*ν*LTL*:
$\varphi \in \nu LTL \Longrightarrow w \models_n \varphi \longleftrightarrow (\forall\, i.\ \neg\ (af\ \varphi\ (prefix\ i\ w) \sim false_n))$
using *ltl-implies-satisfiable-prefix satisfiable-prefix-implies-*ν*LTL* **by** *blast*

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

then have $F_n\ \varphi \in \mu LTL$
by *simp*

have $w \models_n G_n\ (F_n\ \varphi) \longleftrightarrow (\forall\, i.\ suffix\ i\ w \models_n F_n\ \varphi)$
by *simp*
also have $\ldots \longleftrightarrow (\forall\, i.\ \exists\, j.\ af\ (F_n\ \varphi)\ (prefix\ j\ (suffix\ i\ w)) \sim true_n)$
using *af-*μ*LTL*[*OF* ‹$F_n\ \varphi \in \mu LTL$›] **by** *blast*
also have $\ldots \longleftrightarrow (\forall\, i.\ \exists\, j.\ af\ (F_n\ \varphi)\ (prefix\ (j - i)\ (suffix\ i\ w)) \sim true_n)$
by (*metis diff-add-inverse*)
also have $\ldots \longleftrightarrow (\forall\, i.\ \exists\, j.\ af\ (F_n\ \varphi)\ (w[i \to j]) \sim true_n)$
unfolding *subsequence-prefix-suffix* **..**
finally show *?thesis*
by *blast*
qed

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

then have $G_n\ \varphi \in \nu LTL$
by *simp*

have $w \models_n F_n\ (G_n\ \varphi) \longleftrightarrow (\exists\, i.\ suffix\ i\ w \models_n G_n\ \varphi)$
by *force*
also have $\ldots \longleftrightarrow (\exists\, i.\ \forall\, j.\ \neg\ (af\ (G_n\ \varphi)\ (prefix\ j\ (suffix\ i\ w)) \sim false_n))$
using *af-*ν*LTL*[*OF* ‹$G_n\ \varphi \in \nu LTL$›] **by** *blast*
also have $\ldots \longleftrightarrow (\exists\, i.\ \forall\, j.\ \neg\ (af\ (G_n\ \varphi)\ (prefix\ (j - i)\ (suffix\ i\ w)) \sim false_n))$
by (*metis diff-add-inverse*)

also have ... $\longleftrightarrow$ $(\exists\, i.\ \forall\, j.\ \neg\ (af\ (G_n\ \varphi)\ (w[i \to j]) \sim false_n))$
unfolding *subsequence-prefix-suffix* **..**
finally show *?thesis*
by *blast*
qed

end

Bring Propositional Equivalence into scope

interpretation *af-congruent* $(\sim_P)$
by *unfold-locales* (*rule af-letter-prop-congruent*)

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
$(X_n\ \psi)[X]_\nu = X_n\ (\psi[X]_\nu)$
$|\ (\psi_1\ and_n\ \psi_2)[X]_\nu = (\psi_1[X]_\nu)\ and_n\ (\psi_2[X]_\nu)$
$|\ (\psi_1\ or_n\ \psi_2)[X]_\nu = (\psi_1[X]_\nu)\ or_n\ (\psi_2[X]_\nu)$
$|\ (\psi_1\ W_n\ \psi_2)[X]_\nu = (\psi_1[X]_\nu)\ W_n\ (\psi_2[X]_\nu)$
$|\ (\psi_1\ R_n\ \psi_2)[X]_\nu = (\psi_1[X]_\nu)\ R_n\ (\psi_2[X]_\nu)$
$|\ (\psi_1\ U_n\ \psi_2)[X]_\nu = (if\ (\psi_1\ U_n\ \psi_2) \in X\ then\ (\psi_1[X]_\nu)\ W_n\ (\psi_2[X]_\nu)\ else\ false_n)$
$|\ (\psi_1\ M_n\ \psi_2)[X]_\nu = (if\ (\psi_1\ M_n\ \psi_2) \in X\ then\ (\psi_1[X]_\nu)\ R_n\ (\psi_2[X]_\nu)\ else\ false_n)$
$|\ \varphi[\text{-}]_\nu = \varphi$

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*:
$\varphi[X]_\nu \in \nu LTL$
$\varphi \in \nu LTL \Longrightarrow \varphi[X]_\nu = \varphi$
by (*induction* φ) *auto*

lemma *FG-advice-μLTL*:
$\varphi[X]_\mu \in \mu LTL$
$\varphi \in \mu LTL \Longrightarrow \varphi[X]_\mu = \varphi$
by (*induction* φ) *auto*

lemma *GF-advice-subfrmlsn*:
$\mathit{subfrmlsn}\ (\varphi[X]_\nu) \subseteq \{\psi[X]_\nu \mid \psi.\ \psi \in \mathit{subfrmlsn}\ \varphi\}$
by (*induction* φ) *force+*

lemma *FG-advice-subfrmlsn*:
$\mathit{subfrmlsn}\ (\varphi[Y]_\mu) \subseteq \{\psi[Y]_\mu \mid \psi.\ \psi \in \mathit{subfrmlsn}\ \varphi\}$
by (*induction* φ) *force+*

lemma *GF-advice-subfrmlsn-card*:
$\mathit{card}\ (\mathit{subfrmlsn}\ (\varphi[X]_\nu)) \leq \mathit{card}\ (\mathit{subfrmlsn}\ \varphi)$
proof −
have $\mathit{card}\ (\mathit{subfrmlsn}\ (\varphi[X]_\nu)) \leq \mathit{card}\ \{\psi[X]_\nu \mid \psi.\ \psi \in \mathit{subfrmlsn}\ \varphi\}$
by (*simp add*: *subfrmlsn-finite GF-advice-subfrmlsn card-mono*)

also have $\ldots \leq \mathit{card}\ (\mathit{subfrmlsn}\ \varphi)$
by (*metis Collect-mem-eq card-image-le image-Collect subfrmlsn-finite*)

finally show *?thesis* **.**
qed

lemma *FG-advice-subfrmlsn-card*:
$\mathit{card}\ (\mathit{subfrmlsn}\ (\varphi[Y]_\mu)) \leq \mathit{card}\ (\mathit{subfrmlsn}\ \varphi)$
proof −
have $\mathit{card}\ (\mathit{subfrmlsn}\ (\varphi[Y]_\mu)) \leq \mathit{card}\ \{\psi[Y]_\mu \mid \psi.\ \psi \in \mathit{subfrmlsn}\ \varphi\}$
by (*simp add*: *subfrmlsn-finite FG-advice-subfrmlsn card-mono*)

also have $\ldots \leq \mathit{card}\ (\mathit{subfrmlsn}\ \varphi)$

by (*metis Collect-mem-eq card-image-le image-Collect subfrmlsn-finite*)

finally show *?thesis* **.**
qed

lemma *GF-advice-monotone*:
$X \subseteq Y \Longrightarrow w \models_n \varphi[X]_\nu \Longrightarrow w \models_n \varphi[Y]_\nu$
proof (*induction* φ *arbitrary*: *w*)
case (*Until-ltln* φ ψ)
then show *?case*
by (*cases* φ U_n $\psi \in X$) (*simp-all*, *blast*)
next
case (*Release-ltln* φ ψ)
then show *?case* **by** (*simp*, *blast*)
next
case (*WeakUntil-ltln* φ ψ)
then show *?case* **by** (*simp*, *blast*)
next
case (*StrongRelease-ltln* φ ψ)
then show *?case*
by (*cases* φ M_n $\psi \in X$) (*simp-all*, *blast*)
qed *auto*

lemma *FG-advice-monotone*:
$X \subseteq Y \Longrightarrow w \models_n \varphi[X]_\mu \Longrightarrow w \models_n \varphi[Y]_\mu$
proof (*induction* φ *arbitrary*: *w*)
case (*Until-ltln* φ ψ)
then show *?case* **by** (*simp*, *blast*)
next
case (*Release-ltln* φ ψ)
then show *?case*
by (*cases* φ R_n $\psi \in X$) (*auto*, *blast*)
next
case (*WeakUntil-ltln* φ ψ)
then show *?case*
by (*cases* φ W_n $\psi \in X$) (*auto*, *blast*)
next
case (*StrongRelease-ltln* φ ψ)
then show *?case* **by** (*simp*, *blast*)
qed *auto*

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

by *simp-all*

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

3.2 Advice Functions on Nested Propositions

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

lemma $\textit{nested-prop-atoms}_\nu\textit{-finite}$:
finite $(\textit{nested-prop-atoms}_\nu\ \varphi\ X)$
by (*simp add*: $\textit{nested-prop-atoms}_\nu\textit{-def}$ *nested-prop-atoms-finite*)

lemma $\textit{nested-prop-atoms}_\mu\textit{-finite}$:
finite $(\textit{nested-prop-atoms}_\mu\ \varphi\ X)$
by (*simp add*: $\textit{nested-prop-atoms}_\mu\textit{-def}$ *nested-prop-atoms-finite*)

lemma $\textit{nested-prop-atoms}_\nu\textit{-card}$:
$\textit{card}\ (\textit{nested-prop-atoms}_\nu\ \varphi\ X) \le \textit{card}\ (\textit{nested-prop-atoms}\ \varphi)$
unfolding $\textit{nested-prop-atoms}_\nu\textit{-def}$
by (*metis Collect-mem-eq card-image-le image-Collect nested-prop-atoms-finite*)

lemma $\textit{nested-prop-atoms}_\mu\textit{-card}$:
$\textit{card}\ (\textit{nested-prop-atoms}_\mu\ \varphi\ X) \le \textit{card}\ (\textit{nested-prop-atoms}\ \varphi)$
unfolding $\textit{nested-prop-atoms}_\mu\textit{-def}$
by (*metis Collect-mem-eq card-image-le image-Collect nested-prop-atoms-finite*)

lemma $\textit{GF-advice-nested-prop-atoms}_\nu$:
$\textit{nested-prop-atoms}\ (\varphi[X]_\nu) \subseteq \textit{nested-prop-atoms}_\nu\ \varphi\ X$
by (*induction* φ) (*unfold* $\textit{nested-prop-atoms}_\nu\textit{-def}$, *force*+)

lemma $\textit{FG-advice-nested-prop-atoms}_\mu$:
$\textit{nested-prop-atoms}\ (\varphi[Y]_\mu) \subseteq \textit{nested-prop-atoms}_\mu\ \varphi\ Y$
by (*induction* φ) (*unfold* $\textit{nested-prop-atoms}_\mu\textit{-def}$, *force*+)

lemma $\textit{nested-prop-atoms}_\nu\textit{-subset}$:

nested-prop-atoms $\varphi \subseteq$ *nested-prop-atoms* $\psi \Longrightarrow$ *nested-prop-atoms*$_\nu$ φ X $\subseteq$ *nested-prop-atoms*$_\nu$ ψ X
unfolding *nested-prop-atoms*$_\nu$*-def* **by** *blast*

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
unfolding *nested-prop-atoms*$_\mu$*-def* **by** *blast*

lemma *GF-advice-nested-prop-atoms-card*:
card (*nested-prop-atoms* ($\varphi[X]_\nu$)) $\leq$ *card* (*nested-prop-atoms* φ)
proof −
have *card* (*nested-prop-atoms* ($\varphi[X]_\nu$)) $\leq$ *card* (*nested-prop-atoms*$_\nu$ φ X)
by (*simp add*: *nested-prop-atoms*$_\nu$*-finite GF-advice-nested-prop-atoms*$_\nu$ *card-mono*)

then show *?thesis*
using *nested-prop-atoms*$_\nu$*-card le-trans* **by** *blast*
qed

lemma *FG-advice-nested-prop-atoms-card*:
card (*nested-prop-atoms* ($\varphi[Y]_\mu$)) $\leq$ *card* (*nested-prop-atoms* φ)
proof −
have *card* (*nested-prop-atoms* ($\varphi[Y]_\mu$)) $\leq$ *card* (*nested-prop-atoms*$_\mu$ φ Y)
by (*simp add*: *nested-prop-atoms*$_\mu$*-finite FG-advice-nested-prop-atoms*$_\mu$ *card-mono*)

then show *?thesis*
using *nested-prop-atoms*$_\mu$*-card le-trans* **by** *blast*
qed

3.3 Intersecting the Advice Set

lemma *GF-advice-inter*:
$X \cap$ *subformulas*$_\mu$ $\varphi \subseteq S \Longrightarrow \varphi[X \cap S]_\nu = \varphi[X]_\nu$
by (*induction* φ) *auto*

lemma *GF-advice-inter-subformulas*:
$\varphi[X \cap$ *subformulas*$_\mu$ $\varphi]_\nu = \varphi[X]_\nu$
using *GF-advice-inter* **by** *blast*

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

assume $\psi \notin \mathit{subformulas}_\mu\ \varphi$
then have $\mathit{subformulas}_\mu\ \varphi \cap X \subseteq X - \{\psi\}$
by *blast*
then show $\varphi[X - \{\psi\}]_\nu = \varphi[X]_\nu$
by (*metis GF-advice-inter Diff-subset Int-absorb1 inf.commute*)
qed

lemma *GF-advice-minus-size*:
$[\![\mathit{size}\ \varphi \leq \mathit{size}\ \psi;\ \varphi \neq \psi]\!] \Longrightarrow \varphi[X - \{\psi\}]_\nu = \varphi[X]_\nu$
using *subfrmlsn-size* $\mathit{subformulas}_\mu$*-subfrmlsn GF-advice-minus-subformulas*
by *fastforce*

lemma *FG-advice-inter*:
$Y \cap \mathit{subformulas}_\nu\ \varphi \subseteq S \Longrightarrow \varphi[Y \cap S]_\mu = \varphi[Y]_\mu$
by (*induction* φ) *auto*

lemma *FG-advice-inter-subformulas*:
$\varphi[Y \cap \mathit{subformulas}_\nu\ \varphi]_\mu = \varphi[Y]_\mu$
using *FG-advice-inter* **by** *blast*

lemma *FG-advice-minus-subformulas*:
$\psi \notin \mathit{subformulas}_\nu\ \varphi \Longrightarrow \varphi[Y - \{\psi\}]_\mu = \varphi[Y]_\mu$
proof −
assume $\psi \notin \mathit{subformulas}_\nu\ \varphi$
then have $\mathit{subformulas}_\nu\ \varphi \cap Y \subseteq Y - \{\psi\}$
by *blast*
then show $\varphi[Y - \{\psi\}]_\mu = \varphi[Y]_\mu$
by (*metis FG-advice-inter Diff-subset Int-absorb1 inf.commute*)
qed

lemma *FG-advice-minus-size*:
$[\![\mathit{size}\ \varphi \leq \mathit{size}\ \psi;\ \varphi \neq \psi]\!] \Longrightarrow \varphi[Y - \{\psi\}]_\mu = \varphi[Y]_\mu$
using *subfrmlsn-size* $\mathit{subformulas}_\nu$*-subfrmlsn FG-advice-minus-subformulas*
by *fastforce*

lemma *FG-advice-insert*:
$[\![\psi \notin Y;\ \mathit{size}\ \varphi < \mathit{size}\ \psi]\!] \Longrightarrow \varphi[\mathit{insert}\ \psi\ Y]_\mu = \varphi[Y]_\mu$
by (*metis FG-advice-minus-size Diff-insert-absorb less-imp-le neq-iff*)

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 (*induction* φ *arbitrary*: w)
  case (*Next-ltln* φ)
  then show *?case*
    using $\mathcal{F}$*-suffix* **by** *simp blast*
next
  case (*Until-ltln* $\varphi 1$ $\varphi 2$)

  have $\mathcal{F}$ ($\varphi 1$ W_n $\varphi 2$) $w \subseteq \mathcal{F}$ ($\varphi 1$ U_n $\varphi 2$) w
    by *fastforce*
  then have $\mathcal{F}$ ($\varphi 1$ W_n $\varphi 2$) $w \subseteq X$ **and** $w \models_n \varphi 1$ W_n $\varphi 2$
    using *Until-ltln.prems ltln-strong-to-weak* **by** *blast+*
  then have $w \models_n \varphi 1[X]_\nu$ W_n $\varphi 2[X]_\nu$
    using *Until-ltln.IH*
    by *simp* (*meson* $\mathcal{F}$*-suffix subset-trans sup.boundedE*)

  moreover

  have $w \models_n \varphi 1$ U_n $\varphi 2$
    using *Until-ltln.prems* **by** *simp*
  then have $\varphi 1$ U_n $\varphi 2 \in \mathcal{F}$ ($\varphi 1$ U_n $\varphi 2$) w
    by *force*
  then have $\varphi 1$ U_n $\varphi 2 \in X$
    using *Until-ltln.prems* **by** *fast*

  ultimately show *?case*
    by *auto*
next
  case (*Release-ltln* $\varphi 1$ $\varphi 2$)
  then show *?case*
    by *simp* (*meson* $\mathcal{F}$*-suffix subset-trans sup.boundedE*)
next
  case (*WeakUntil-ltln* $\varphi 1$ $\varphi 2$)
  then show *?case*
    by *simp* (*meson* $\mathcal{F}$*-suffix subset-trans sup.boundedE*)
next
  case (*StrongRelease-ltln* $\varphi 1$ $\varphi 2$)

  have $\mathcal{F}$ ($\varphi 1$ R_n $\varphi 2$) $w \subseteq \mathcal{F}$ ($\varphi 1$ M_n $\varphi 2$) w
    by *fastforce*
  then have $\mathcal{F}$ ($\varphi 1$ R_n $\varphi 2$) $w \subseteq X$ **and** $w \models_n \varphi 1$ R_n $\varphi 2$
    using *StrongRelease-ltln.prems ltln-strong-to-weak* **by** *blast+*
  then have $w \models_n \varphi 1[X]_\nu$ R_n $\varphi 2[X]_\nu$
    using *StrongRelease-ltln.IH*
    by *simp* (*meson* $\mathcal{F}$*-suffix subset-trans sup.boundedE*)

moreover

have $w \models_n \varphi 1\ M_n\ \varphi 2$
  using *StrongRelease-ltln.prems* **by** *simp*
then have $\varphi 1\ M_n\ \varphi 2 \in \mathcal{F}\ (\varphi 1\ M_n\ \varphi 2)\ w$
  by *force*
then have $\varphi 1\ M_n\ \varphi 2 \in X$
  using *StrongRelease-ltln.prems* **by** *fast*

ultimately show *?case*
  by *auto*
qed *auto*

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 (*induction* φ *arbitrary*: *w*)
  case (*Next-ltln* φ)
  then show *?case*
    unfolding *GF-advice.simps semantics-ltln.simps*(*7*)
    using *GF-suffix* **by** *blast*
next
  case (*Until-ltln* $\varphi 1\ \varphi 2$)

  then have $\varphi 1\ U_n\ \varphi 2 \in X$
    using *ccontr*[*of* $\varphi 1\ U_n\ \varphi 2 \in X$] **by** *force*
  then have $w \models_n F_n\ \varphi 2$
    using *Until-ltln.prems* **by** *fastforce*

  moreover

  have $w \models_n (\varphi 1\ U_n\ \varphi 2)[X]_\nu$
    using *Until-ltln.prems* **by** *simp*
  then have $w \models_n (\varphi 1[X]_\nu)\ W_n\ (\varphi 2[X]_\nu)$
    unfolding *GF-advice.simps* **using** ‹$\varphi 1\ U_n\ \varphi 2 \in X$› **by** *simp*
  then have $w \models_n \varphi 1\ W_n\ \varphi 2$
    unfolding *GF-advice.simps semantics-ltln.simps*(*10*)
    by (*metis GF-suffix Until-ltln.IH Until-ltln.prems*(*1*))

  ultimately show *?case*
    using *ltln-weak-to-strong* **by** *blast*
next
  case (*Release-ltln* $\varphi 1\ \varphi 2$)
  then show *?case*

unfolding *GF-advice.simps semantics-ltln.simps*(*9*)
by (*metis GF-suffix Release-ltln.IH Release-ltln.prems*(*1*))
next
case (*WeakUntil-ltln* $\varphi 1$ $\varphi 2$)
then show *?case*
unfolding *GF-advice.simps semantics-ltln.simps*(*10*)
by (*metis GF-suffix*)
next
case (*StrongRelease-ltln* $\varphi 1$ $\varphi 2$)

then have $\varphi 1\ M_n\ \varphi 2 \in X$
using *ccontr*[*of* $\varphi 1\ M_n\ \varphi 2 \in X$] **by** *force*
then have $w \models_n F_n\ \varphi 1$
using *StrongRelease-ltln.prems* **by** *fastforce*

moreover

have $w \models_n (\varphi 1\ M_n\ \varphi 2)[X]_\nu$
using *StrongRelease-ltln.prems* **by** *simp*
then have $w \models_n (\varphi 1[X]_\nu)\ R_n\ (\varphi 2[X]_\nu)$
unfolding *GF-advice.simps* **using** ‹$\varphi 1\ M_n\ \varphi 2 \in X$› **by** *simp*
then have $w \models_n \varphi 1\ R_n\ \varphi 2$
unfolding *GF-advice.simps semantics-ltln.simps*(*9*)
by (*metis GF-suffix StrongRelease-ltln.IH StrongRelease-ltln.prems*(*1*))

ultimately show *?case*
using *ltln-weak-to-strong* **by** *blast*
qed *auto*

lemma *GF-advice-a2*:
$[\![X \subseteq \mathcal{GF}\ \varphi\ w;\ w \models_n \varphi[X]_\nu]\!] \Longrightarrow w \models_n \varphi$
by (*metis GF-advice-a2-helper* $\mathcal{GF}$*-elim subset-eq*)

lemma *GF-advice-a3*:
$[\![X = \mathcal{F}\ \varphi\ w;\ X = \mathcal{GF}\ \varphi\ w]\!] \Longrightarrow w \models_n \varphi \longleftrightarrow w \models_n \varphi[X]_\nu$
using *GF-advice-a1 GF-advice-a2* **by** *fastforce*

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 (*induction* φ *arbitrary*: w)
case (*Next-ltln* φ)
then show *?case*

using *$\mathcal{FG}$-suffix* **by** *simp blast*
next
case (*Until-ltln* $\varphi 1$ $\varphi 2$)
then show *?case*
by *simp* (*metis* *$\mathcal{FG}$-suffix*)
next
case (*Release-ltln* $\varphi 1$ $\varphi 2$)

show *?case*
proof (*cases* $\varphi 1\ R_n\ \varphi 2 \in Y$)
case *False*
then have $\varphi 1\ R_n\ \varphi 2 \notin \mathcal{FG}\ (\varphi 1\ R_n\ \varphi 2)\ w$
using *Release-ltln.prems* **by** *blast*
then have $\neg\ w \models_n G_n\ \varphi 2$
by *fastforce*
then have $w \models_n \varphi 1\ M_n\ \varphi 2$
using *Release-ltln.prems ltln-weak-to-strong* **by** *blast*

moreover

have $\mathcal{FG}\ (\varphi 1\ M_n\ \varphi 2)\ w \subseteq \mathcal{FG}\ (\varphi 1\ R_n\ \varphi 2)\ w$
by *fastforce*
then have $\mathcal{FG}\ (\varphi 1\ M_n\ \varphi 2)\ w \subseteq Y$
using *Release-ltln.prems* **by** *blast*

ultimately show *?thesis*
using *Release-ltln.IH* **by** *simp* (*metis* *$\mathcal{FG}$-suffix*)
qed *simp*
next
case (*WeakUntil-ltln* $\varphi 1$ $\varphi 2$)

show *?case*
proof (*cases* $\varphi 1\ W_n\ \varphi 2 \in Y$)
case *False*
then have $\varphi 1\ W_n\ \varphi 2 \notin \mathcal{FG}\ (\varphi 1\ W_n\ \varphi 2)\ w$
using *WeakUntil-ltln.prems* **by** *blast*
then have $\neg\ w \models_n G_n\ \varphi 1$
by *fastforce*
then have $w \models_n \varphi 1\ U_n\ \varphi 2$
using *WeakUntil-ltln.prems ltln-weak-to-strong* **by** *blast*

moreover

have $\mathcal{FG}\ (\varphi 1\ U_n\ \varphi 2)\ w \subseteq \mathcal{FG}\ (\varphi 1\ W_n\ \varphi 2)\ w$

```
      by fastforce
    then have ℱ𝒢 (φ1 U_n φ2) w ⊆ Y
      using WeakUntil-ltln.prems by blast

    ultimately show ?thesis
      using WeakUntil-ltln.IH by simp (metis ℱ𝒢-suffix)
  qed simp
next
  case (StrongRelease-ltln φ1 φ2)
  then show ?case
    by simp (metis ℱ𝒢-suffix)
qed auto

lemma FG-advice-b2-helper:
  ⟦∀ψ ∈ Y. w ⊨_n G_n ψ; w ⊨_n φ[Y]_μ⟧ ⟹ w ⊨_n φ
proof (induction φ arbitrary: w)
  case (Until-ltln φ1 φ2)
  then show ?case
    by simp (metis (no-types, lifting) suffix-suffix)
next
  case (Release-ltln φ1 φ2)
  then show ?case
  proof (cases φ1 R_n φ2 ∈ Y)
    case True
    then show ?thesis
      using Release-ltln.prems by force
  next
    case False
    then have w ⊨_n (φ1[Y]_μ) M_n (φ2[Y]_μ)
      using Release-ltln.prems by simp
    then have w ⊨_n φ1 M_n φ2
      using Release-ltln
      by simp (metis (no-types, lifting) suffix-suffix)
    then show ?thesis
      using ltln-strong-to-weak by fast
  qed
next
  case (WeakUntil-ltln φ1 φ2)
  then show ?case
  proof (cases φ1 W_n φ2 ∈ Y)
    case True
    then show ?thesis
      using WeakUntil-ltln.prems by force
  next
```

case *False*
then have $w \models_n (\varphi 1[Y]_\mu)\ U_n\ (\varphi 2[Y]_\mu)$
using *WeakUntil-ltln.prems* **by** *simp*
then have $w \models_n \varphi 1\ U_n\ \varphi 2$
using *WeakUntil-ltln*
by *simp* (*metis* (*no-types*, *lifting*) *suffix-suffix*)
then show *?thesis*
using *ltln-strong-to-weak* **by** *fast*
qed
next
case (*StrongRelease-ltln* $\varphi 1$ $\varphi 2$)
then show *?case*
by *simp* (*metis* (*no-types*, *lifting*) *suffix-suffix*)
qed *auto*

lemma *FG-advice-b2*:
$[\![Y \subseteq \mathcal{G}\ \varphi\ w;\ w \models_n \varphi[Y]_\mu]\!] \Longrightarrow w \models_n \varphi$
by (*metis FG-advice-b2-helper* $\mathcal{G}$*-elim subset-eq*)

lemma *FG-advice-b3*:
$[\![Y = \mathcal{FG}\ \varphi\ w;\ Y = \mathcal{G}\ \varphi\ w]\!] \Longrightarrow w \models_n \varphi \longleftrightarrow w \models_n \varphi[Y]_\mu$
using *FG-advice-b1 FG-advice-b2* **by** *fastforce*

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 (*induction* φ)
case (*Until-ltln* $\varphi 1$ $\varphi 2$)

then have $w \models_n \textit{af-letter}\ ((\varphi 1\ U_n\ \varphi 2)[X]_\nu)\ x$
using *af-letter-build* **by** *blast*

then show *?case*
using *Until-ltln.IH af-letter-build* **by** *fastforce*
next
case (*Release-ltln* $\varphi 1$ $\varphi 2$)

then have $w \models_n \textit{af-letter}\ ((\varphi 1\ R_n\ \varphi 2)[X]_\nu)\ x$
using *af-letter-build* **by** *blast*

then show *?case*
using *Release-ltln.IH af-letter-build* **by** *auto*
next

```
  case (WeakUntil-ltln φ1 φ2)

  then have w ⊨n af-letter ((φ1 Wn φ2)[X]ν) x
    using af-letter-build by blast

  then show ?case
    using WeakUntil-ltln.IH af-letter-build by auto
next
  case (StrongRelease-ltln φ1 φ2)

  then have w ⊨n af-letter ((φ1 Mn φ2)[X]ν) x
    using af-letter-build by blast

  then show ?case
    using StrongRelease-ltln.IH af-letter-build by force
qed auto

lemma FG-advice-af-letter:
  w ⊨n (af-letter φ x)[Y]μ ⟹ (x ## w) ⊨n φ[Y]μ
proof (induction φ)
  case (Prop-ltln a)
  then show ?case
    using semantics-ltln.simps(3) by fastforce
next
  case (Until-ltln φ1 φ2)
  then show ?case
    unfolding af-letter.simps FG-advice.simps semantics-ltln.simps(5,6)
   using af-letter-build apply (cases w ⊨n af-letter φ2 x[Y]μ) apply force
  by (metis af-letter.simps(8) semantics-ltln.simps(5) semantics-ltln.simps(6))
next
  case (Release-ltln φ1 φ2)
  then show ?case
    apply (cases φ1 Rn φ2 ∈ Y)
    apply simp
    unfolding af-letter.simps FG-advice.simps semantics-ltln.simps(5,6)
   using af-letter-build apply (cases w ⊨n af-letter φ1 x[Y]μ) apply force
   by (metis (full-types) af-letter.simps(11) semantics-ltln.simps(5) seman-
tics-ltln.simps(6))
next
  case (WeakUntil-ltln φ1 φ2)
  then show ?case
    apply (cases φ1 Wn φ2 ∈ Y)
    apply simp
    unfolding af-letter.simps FG-advice.simps semantics-ltln.simps(5,6)
```

using *af-letter-build* **apply** (*cases* $w \models_n$ *af-letter* $\varphi 2$ $x[Y]_\mu$) **apply** *force*
by (*metis* (*full-types*) *af-letter.simps*(*8*) *semantics-ltln.simps*(*5*) *semantics-ltln.simps*(*6*))
next
case (*StrongRelease-ltln* $\varphi 1$ $\varphi 2$)
then show *?case*
unfolding *af-letter.simps FG-advice.simps semantics-ltln.simps*(*5*,*6*)
using *af-letter-build* **apply** (*cases* $w \models_n$ *af-letter* $\varphi 1$ $x[Y]_\mu$) **apply** *force*
by (*metis af-letter.simps*(*11*) *semantics-ltln.simps*(*5*) *semantics-ltln.simps*(*6*))
qed *auto*

lemma *GF-advice-af*:
$(w \frown w') \models_n \varphi[X]_\nu \Longrightarrow w' \models_n (af\ \varphi\ w)[X]_\nu$
by (*induction w arbitrary*: φ) (*simp*, *insert GF-advice-af-letter*, *fastforce*)

lemma *FG-advice-af*:
$w' \models_n (af\ \varphi\ w)[X]_\mu \Longrightarrow (w \frown w') \models_n \varphi[X]_\mu$
by (*induction w arbitrary*: φ) (*simp*, *insert FG-advice-af-letter*, *fastforce*)

lemma *GF-advice-af-2*:
$w \models_n \varphi[X]_\nu \Longrightarrow suffix\ i\ w \models_n (af\ \varphi\ (prefix\ i\ w))[X]_\nu$
using *GF-advice-af* **by** *force*

lemma *FG-advice-af-2*:
$suffix\ i\ w \models_n (af\ \varphi\ (prefix\ i\ w))[X]_\mu \Longrightarrow w \models_n \varphi[X]_\mu$
using *FG-advice-af* **by** *force*

lemma *prefix-suffix-subsequence*: *prefix i* (*suffix j w*) = (w [$j \to i + j$])
by (*simp add*: *add.commute*)

We show this generic lemma to prove the following theorems:

lemma *GF-advice-sync*:
fixes *index* :: *nat* ⇒ *nat*
fixes *formula* :: *nat* ⇒ *'a ltln*
assumes $\bigwedge i.\ i < n \Longrightarrow \exists j.\ suffix\ ((index\ i) + j)\ w \models_n af\ (formula\ i)\ (w\ [index\ i \to (index\ i) + j])[X]_\nu$
shows $\exists k.\ (\forall i < n.\ k \geq index\ i \wedge suffix\ k\ w \models_n af\ (formula\ i)\ (w\ [index\ i \to k])[X]_\nu)$
using *assms*
proof (*induction n*)
case (*Suc n*)

obtain *k1* **where** *leq1*: $\bigwedge i.\ i < n \Longrightarrow k1 \geq index\ i$

and *suffix1*: $\bigwedge i.\ i < n \Longrightarrow$ *suffix k1 w* $\models_n$ *af* (*formula i*) (*w* [(*index i*) $\rightarrow$ *k1*])$[X]_\nu$
using *Suc less-SucI* **by** *blast*

obtain *k2* **where** *leq2*: *k2* $\geq$ *index n*
and *suffix2*: *suffix k2 w* $\models_n$ *af* (*formula n*) (*w* [*index n* $\rightarrow$ *k2*])$[X]_\nu$
using *le-add1 Suc.prems* **by** *blast*

define *k* **where** *k* $\equiv$ *k1* + *k2*

have $\bigwedge i.\ i < Suc\ n \Longrightarrow k \geq index\ i$
unfolding *k-def* **by** (*metis leq1 leq2 less-SucE trans-le-add1 trans-le-add2*)

moreover

{
have $\bigwedge i.\ i < n \Longrightarrow$ *suffix k w* $\models_n$ *af* (*formula i*) (*w* [(*index i*) $\rightarrow$ *k*])$[X]_\nu$
unfolding *k-def*
by (*metis GF-advice-af-2*[*OF suffix1*, *unfolded suffix-suffix prefix-suffix-subsequence*] *af-subsequence-append leq1 add.commute le-add1*)

moreover

have *suffix k w* $\models_n$ *af* (*formula n*) (*w* [*index n* $\rightarrow$ *k*])$[X]_\nu$
unfolding *k-def*
by (*metis GF-advice-af-2*[*OF suffix2*, *unfolded suffix-suffix prefix-suffix-subsequence*] *af-subsequence-append leq2 add.commute le-add1*)

ultimately

have $\bigwedge i.\ i \leq n \Longrightarrow$ *suffix k w* $\models_n$ *af* (*formula i*) (*w* [(*index i*) $\rightarrow$ *k*])$[X]_\nu$
using *nat-less-le* **by** *blast*
}

ultimately

show *?case*
by (*meson less-Suc-eq-le*)
qed *simp*

lemma *GF-advice-sync-and*:
assumes $\exists\, i.$ *suffix i w* $\models_n$ *af* φ (*prefix i w*)$[X]_\nu$
assumes $\exists\, i.$ *suffix i w* $\models_n$ *af* ψ (*prefix i w*)$[X]_\nu$
shows $\exists\, i.$ *suffix i w* $\models_n$ *af* φ (*prefix i w*)$[X]_\nu$ $\wedge$ *suffix i w* $\models_n$ *af* ψ (*prefix*

i $w)[X]_\nu$
proof −
  let *?formula* = λi :: *nat*. (*if* (i = *0*) *then* φ *else* ψ)

  have *assms*: $\bigwedge i.\ i < 2 \Longrightarrow \exists j.$ *suffix* j $w \models_n$ *af* (*?formula* i) (w [*0* → j])$[X]_\nu$
    using *assms* **by** *simp*
  obtain k **where** *k-def*: $\bigwedge i$ :: *nat*. $i < 2 \Longrightarrow$ *suffix* k $w \models_n$ *af* (*if* $i = 0$ *then* φ *else* ψ) (*prefix* k $w)[X]_\nu$
    using *GF-advice-sync*[*of 2* λi. *0* w *?formula*, *simplified*, *OF assms*, *simplified*] **by** *blast*
  show *?thesis*
    using *k-def*[*of 0*] *k-def*[*of 1*] **by** *auto*
qed

lemma *GF-advice-sync-less*:
  assumes $\bigwedge i.\ i < n \Longrightarrow \exists j.$ *suffix* $(i + j)$ $w \models_n$ *af* φ (w [$i \to j + i$])$[X]_\nu$
  assumes $\exists j.$ *suffix* $(n + j)$ $w \models_n$ *af* ψ (w [$n \to j + n$])$[X]_\nu$
  shows $\exists k \geq n.$ ($\forall j < n.$ *suffix* k $w \models_n$ *af* φ (w [$j \to k$])$[X]_\nu$) $\wedge$ *suffix* k $w \models_n$ *af* ψ (w [$n \to k$])$[X]_\nu$
proof −
  let *?index* = λi. *min* i n
  let *?formula* = λi. *if* ($i < n$) *then* φ *else* ψ

  {
    fix i
    assume $i <$ *Suc* n
    then have *min-def*: *min* i $n = i$
      by *simp*
    have $\exists j.$ *suffix* ((*?index* i) + j) $w \models_n$ *af* (*?formula* i) (w [*?index* i → (*?index* i) + j])$[X]_\nu$
      unfolding *min-def*
      by (*cases* $i < n$)
        (*metis* (*full-types*) *assms*(*1*) *add.commute*, *metis* (*full-types*) *assms*(*2*) ‹$i <$ *Suc* n› *add.commute* *less-SucE*)
  }

  then obtain k **where** *leq*: ($\bigwedge i.\ i <$ *Suc* $n \Longrightarrow$ *min* i $n \leq k$)
    and *suffix*: $\bigwedge i.\ i <$ *Suc* $n \Longrightarrow$ *suffix* k $w \models_n$ *af* (*if* $i < n$ *then* φ *else* ψ) (w [*min* i $n \to k$])$[X]_\nu$
    using *GF-advice-sync*[*of Suc n ?index w ?formula X*] **by** *metis*

  have $\forall j < n.$ *suffix* k $w \models_n$ *af* φ (w [$j \to k$])$[X]_\nu$
    using *suffix* **by** (*metis* (*full-types*) *less-SucI* *min.strict-order-iff*)

moreover

have *suffix k w* $\models_n$ *af* ψ $(w\ [n \rightarrow k])[X]_\nu$
using *suffix*[*of n*, *simplified*] **by** *blast*

moreover

have $k \geq n$
using *leq* **by** *presburger*

ultimately
show *?thesis*
by *auto*
qed

lemma *GF-advice-sync-lesseq*:
assumes $\bigwedge i.\ i \leq n \Longrightarrow \exists j.$ *suffix* $(i + j)$ *w* $\models_n$ *af* φ $(w\ [i \rightarrow j + i])[X]_\nu$
assumes $\exists j.$ *suffix* $(n + j)$ *w* $\models_n$ *af* ψ $(w\ [n \rightarrow j + n])[X]_\nu$
shows $\exists k \geq n.\ (\forall j \leq n.$ *suffix k w* $\models_n$ *af* φ $(w\ [j \rightarrow k])[X]_\nu) \wedge$ *suffix k w* $\models_n$ *af* ψ $(w\ [n \rightarrow k])[X]_\nu$
proof −
let *?index* = $\lambda i.$ *min i n*
let *?formula* = $\lambda i.$ *if* $(i \leq n)$ *then* φ *else* ψ

{
fix *i*
assume $i <$ *Suc* (*Suc n*)
hence $\exists j.$ *suffix* ((*?index i*) + *j*) *w* $\models_n$ *af* (*?formula i*) (*w* [*?index i* $\rightarrow$ (*?index i*) + *j*])$[X]_\nu$
proof (*cases i* < *Suc n*)
case *True*
then have *min-def*: *min i n* = *i*
by *simp*
show *?thesis*
unfolding *min-def* **by** (*metis* (*full-types*) *assms*(*1*) *Suc-leI Suc-le-mono True add.commute*)
next
case *False*
then have *i-def*: *i* = *Suc n*
using ‹*i* < *Suc* (*Suc n*)› *less-antisym* **by** *blast*
have *min-def*: *min i n* = *n*
unfolding *i-def* **by** *simp*
show *?thesis*

```
      using assms(2) False
      by (simp add: min-def add.commute)
    qed
  }

  then obtain k where leq: (⋀i. i ≤ Suc n ⟹ min i n ≤ k)
    and suffix: ⋀i :: nat. i < Suc (Suc n) ⟹ suffix k w ⊨n af (if i ≤ n
then φ else ψ) (w [min i n → k])[X]ν
    using GF-advice-sync[of Suc (Suc n) ?index w ?formula X]
    by (metis (no-types, opaque-lifting) less-Suc-eq min-le-iff-disj)

  have ∀ j ≤ n. suffix k w ⊨n af φ (w [j → k])[X]ν
    using suffix by (metis (full-types) le-SucI less-Suc-eq-le min.orderE)

  moreover

  have suffix k w ⊨n af ψ (w [n → k])[X]ν
    using suffix[of Suc n, simplified] by linarith

  moreover

  have k ≥ n
    using leq by presburger

  ultimately
  show ?thesis
    by auto
qed

lemma af-subsequence-U-GF-advice:
  assumes i ≤ n
  assumes suffix n w ⊨n ((af ψ (w [i → n]))[X]ν)
  assumes ⋀j. j < i ⟹ suffix n w ⊨n ((af φ (w [j → n]))[X]ν)
  shows suffix (Suc n) w ⊨n (af (φ Un ψ) (prefix (Suc n) w))[X]ν
  using assms
proof (induction i arbitrary: w n)
  case 0
  then have A: suffix n w ⊨n ((af ψ (w [0 → n]))[X]ν)
    by blast
  then have suffix (Suc n) w ⊨n (af ψ (w [0 → Suc n]))[X]ν
    using GF-advice-af-2[OF A, of 1] by simp
  then show ?case
     unfolding GF-advice.simps af-subsequence-U semantics-ltln.simps by
blast
```

next
  case (*Suc i*)
  have *suffix* (*Suc n*) *w* $\models_n$ (*af* φ (*prefix* (*Suc n*) *w*))$[X]_\nu$
    using *Suc.prems*(*3*)[*OF zero-less-Suc*, *THEN GF-advice-af-2*, *unfolded suffix-suffix*, *of 1*]
    by *simp*
  moreover
  have *B*: (*Suc* (*n* − *1*)) = *n*
    using *Suc* **by** *simp*
  note *Suc.IH*[*of n* − *1 suffix 1 w*, *unfolded suffix-suffix*] *Suc.prems*
  then have *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ U_n ψ) (*w* [*1* → (*Suc n*)]))$[X]_\nu$
    by (*metis B One-nat-def Suc-le-mono Suc-mono plus-1-eq-Suc subsequence-shift*)
  ultimately
  show *?case*
    unfolding *af-subsequence-U semantics-ltln.simps GF-advice.simps* **by** *blast*
qed

lemma *af-subsequence-M-GF-advice*:
  assumes $i \leq n$
  assumes *suffix n w* $\models_n$ ((*af* φ (*w* [*i* → *n*]))$[X]_\nu$)
  assumes $\bigwedge j.\ j \leq i \Longrightarrow$ *suffix n w* $\models_n$ ((*af* ψ (*w* [*j* → *n*]))$[X]_\nu$)
  shows *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ M_n ψ) (*prefix* (*Suc n*) *w*))$[X]_\nu$
  using *assms*
proof (*induction i arbitrary*: *w n*)
  case *0*
  then have *A*: *suffix n w* $\models_n$ ((*af* ψ (*w* [*0* → *n*]))$[X]_\nu$)
    by *blast*
  have *suffix* (*Suc n*) *w* $\models_n$ (*af* ψ (*w* [*0* → *Suc n*]))$[X]_\nu$
    using *GF-advice-af-2*[*OF A*, *of 1*] **by** *simp*
  moreover
  have *suffix* (*Suc n*) *w* $\models_n$ (*af* φ (*w* [*0* → *Suc n*]))$[X]_\nu$
    using *GF-advice-af-2*[*OF 0.prems*(*2*), *of 1*, *unfolded suffix-suffix*] **by** *auto*
  ultimately
  show *?case*
    unfolding *af-subsequence-M GF-advice.simps semantics-ltln.simps* **by** *blast*
next
  case (*Suc i*)
  have *suffix 1* (*suffix n w*) $\models_n$ *af* (*af* ψ (*prefix n w*)) [*suffix n w 0*]$[X]_\nu$
    by (*metis* (*no-types*) *GF-advice-af-2 Suc.prems*(*3*) *plus-1-eq-Suc subsequence-singleton suffix-0 suffix-suffix zero-le*)

then have *suffix* (*Suc n*) *w* $\models_n$ (*af* ψ (*prefix* (*Suc n*) *w*))$[X]_\nu$
using *Suc.prems*(*3*)[*THEN GF-advice-af-2*, *unfolded suffix-suffix*, *of 1*] **by** *simp*
moreover
have *B*: (*Suc* (*n* − *1*)) = *n*
using *Suc* **by** *simp*
note *Suc.IH*[*of - suffix 1 w*, *unfolded subsequence-shift suffix-suffix*]
then have *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ M_n ψ) (*w* [*1* → (*Suc n*)]))$[X]_\nu$
by (*metis B One-nat-def Suc-le-mono plus-1-eq-Suc Suc.prems*)
ultimately
show *?case*
unfolding *af-subsequence-M semantics-ltln.simps GF-advice.simps* **by** *blast*
qed

lemma *af-subsequence-R-GF-advice*:
assumes $i \leq n$
assumes *suffix n w* $\models_n$ ((*af* φ (*w* [*i* → *n*]))$[X]_\nu$)
assumes $\bigwedge j.\ j \leq i \Longrightarrow$ *suffix n w* $\models_n$ ((*af* ψ (*w* [*j* → *n*]))$[X]_\nu$)
shows *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ R_n ψ) (*prefix* (*Suc n*) *w*))$[X]_\nu$
using *assms*
proof (*induction i arbitrary*: *w n*)
case *0*
then have *A*: *suffix n w* $\models_n$ ((*af* ψ (*w* [*0* → *n*]))$[X]_\nu$)
by *blast*
have *suffix* (*Suc n*) *w* $\models_n$ (*af* ψ (*w* [*0* → *Suc n*]))$[X]_\nu$
using *GF-advice-af-2*[*OF A*, *of 1*] **by** *simp*
moreover
have *suffix* (*Suc n*) *w* $\models_n$ (*af* φ (*w* [*0* → *Suc n*]))$[X]_\nu$
using *GF-advice-af-2*[*OF 0.prems*(*2*), *of 1*, *unfolded suffix-suffix*] **by** *auto*
ultimately
show *?case*
unfolding *af-subsequence-R GF-advice.simps semantics-ltln.simps* **by** *blast*
next
case (*Suc i*)
have *suffix 1* (*suffix n w*) $\models_n$ *af* (*af* ψ (*prefix n w*)) [*suffix n w 0*]$[X]_\nu$
by (*metis* (*no-types*) *GF-advice-af-2 Suc.prems*(*3*) *plus-1-eq-Suc subsequence-singleton suffix-0 suffix-suffix zero-le*)
then have *suffix* (*Suc n*) *w* $\models_n$ (*af* ψ (*prefix* (*Suc n*) *w*))$[X]_\nu$
using *Suc.prems*(*3*)[*THEN GF-advice-af-2*, *unfolded suffix-suffix*, *of 1*] **by** *simp*
moreover

have *B*: (*Suc* (*n* − *1*)) = *n*
using *Suc* **by** *simp*
note *Suc.IH*[*of - suffix 1 w*, *unfolded subsequence-shift suffix-suffix*]
then have *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ R_n ψ) (*w* [*1* → (*Suc n*)]))[*X*]$_\nu$
by (*metis B One-nat-def Suc-le-mono plus-1-eq-Suc Suc.prems*)
ultimately
show *?case*
unfolding *af-subsequence-R semantics-ltln.simps GF-advice.simps* **by** *blast*
qed

lemma *af-subsequence-W-GF-advice*:
assumes $i \leq n$
assumes *suffix n w* $\models_n$ ((*af* ψ (*w* [*i* → *n*]))[*X*]$_\nu$)
assumes $\bigwedge j.\ j < i \Longrightarrow$ *suffix n w* $\models_n$ ((*af* φ (*w* [*j* → *n*]))[*X*]$_\nu$)
shows *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ W_n ψ) (*prefix* (*Suc n*) *w*))[*X*]$_\nu$
using *assms*
proof (*induction i arbitrary*: *w n*)
case *0*
then have *A*: *suffix n w* $\models_n$ ((*af* ψ (*w* [*0* → *n*]))[*X*]$_\nu$)
by *blast*
have *suffix* (*Suc n*) *w* $\models_n$ (*af* ψ (*w* [*0* → *Suc n*]))[*X*]$_\nu$
using *GF-advice-af-2*[*OF A*, *of 1*] **by** *simp*
then show *?case*
unfolding *af-subsequence-W GF-advice.simps semantics-ltln.simps* **by** *blast*
next
case (*Suc i*)
have *suffix* (*Suc n*) *w* $\models_n$ (*af* φ (*prefix* (*Suc n*) *w*))[*X*]$_\nu$
using *Suc.prems*(*3*)[*OF zero-less-Suc*, *THEN GF-advice-af-2*, *unfolded suffix-suffix*, *of 1*]
by *simp*
moreover
have *B*: (*Suc* (*n* − *1*)) = *n*
using *Suc* **by** *simp*
note *Suc.IH*[*of n* − *1 suffix 1 w*, *unfolded suffix-suffix*] *Suc.prems*
then have *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ W_n ψ) (*w* [*1* → (*Suc n*)]))[*X*]$_\nu$
by (*metis B One-nat-def Suc-le-mono Suc-mono plus-1-eq-Suc subsequence-shift*)
ultimately
show *?case*
unfolding *af-subsequence-W* **unfolding** *semantics-ltln.simps GF-advice.simps* **by** *simp*
qed

lemma *af-subsequence-R-GF-advice-connect*:
  assumes $i \le n$
  assumes *suffix n w* $\models_n$ *af* $(\varphi\ R_n\ \psi)$ $(w\ [i \to n])[X]_\nu$
  assumes $\bigwedge j.\ j \le i \Longrightarrow$ *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*))$[X]_\nu$
  using *assms*
proof (*induction i arbitrary*: *w n*)
  case *0*
  then have *A*: *suffix n w* $\models_n$ $((af\ \psi\ (w\ [0 \to n]))[X]_\nu)$
    by *blast*
  have *suffix* (*Suc n*) *w* $\models_n$ (*af* ψ (*w* [*0* $\to$ *Suc n*]))$[X]_\nu$
    using *GF-advice-af-2*[*OF A*, *of 1*] **by** *simp*
  moreover
  have *suffix* (*Suc n*) *w* $\models_n$ (*af* $(\varphi\ R_n\ \psi)$ (*w* [*0* $\to$ *Suc n*]))$[X]_\nu$
    using *GF-advice-af-2*[*OF 0.prems*(*2*), *of 1*, *unfolded suffix-suffix*] **by** *simp*
  ultimately
  show *?case*
    unfolding *af-subsequence-R GF-advice.simps semantics-ltln.simps* **by** *blast*
next
  case (*Suc i*)
  have *suffix 1* (*suffix n w*) $\models_n$ *af* (*af* ψ (*prefix n w*)) [*suffix n w 0*]$[X]_\nu$
    by (*metis* (*no-types*) *GF-advice-af-2 Suc.prems*(*3*) *plus-1-eq-Suc subsequence-singleton suffix-0 suffix-suffix zero-le*)
  then have *suffix* (*Suc n*) *w* $\models_n$ (*af* ψ (*prefix* (*Suc n*) *w*))$[X]_\nu$
    using *Suc.prems*(*3*)[*THEN GF-advice-af-2*, *unfolded suffix-suffix*, *of 1*] **by** *simp*
  moreover
  have *B*: (*Suc* $(n - 1)$) $= n$
    using *Suc* **by** *simp*
  note *Suc.IH*[*of - suffix 1 w*, *unfolded subsequence-shift suffix-suffix*]
  then have *suffix* (*Suc n*) *w* $\models_n$ (*af* $(\varphi\ R_n\ \psi)$ (*w* [*1* $\to$ (*Suc n*)]))$[X]_\nu$
    by (*metis B One-nat-def Suc-le-mono plus-1-eq-Suc Suc.prems*)
  ultimately
  show *?case*
    unfolding *af-subsequence-R semantics-ltln.simps GF-advice.simps* **by** *blast*
qed

lemma *af-subsequence-W-GF-advice-connect*:
  assumes $i \le n$
  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* φ (*w* [*j* $\rightarrow$ *n*]))$[X]_\nu$)
shows *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ W_n ψ) (*prefix* (*Suc n*) *w*))$[X]_\nu$
using *assms*
proof (*induction i arbitrary*: *w n*)
case *0*
have *suffix* (*Suc n*) *w* $\models_n$ *af-letter* (*af* (φ W_n ψ) (*prefix n w*)) (*w n*)$[X]_\nu$
by (*simp add*: *0.prems*(*2*) *GF-advice-af-letter*)
moreover
have *prefix* (*Suc n*) *w* = *prefix n w* @ [*w n*]
using *subseq-to-Suc* **by** *blast*
ultimately show *?case*
by (*metis* (*no-types*) *foldl.simps*(*1*) *foldl.simps*(*2*) *foldl-append*)
next
case (*Suc i*)
have *suffix* (*Suc n*) *w* $\models_n$ (*af* φ (*prefix* (*Suc n*) *w*))$[X]_\nu$
using *Suc.prems*(*3*)[*OF zero-less-Suc*, *THEN GF-advice-af-2*, *unfolded suffix-suffix*, *of 1*] **by** *simp*
moreover
have *n* > *0* **and** *B*: (*Suc* (*n* − *1*)) = *n*
using *Suc* **by** *simp+*
note *Suc.IH*[*of n* − *1 suffix 1 w*, *unfolded suffix-suffix*] *Suc.prems*
then have *suffix* (*Suc n*) *w* $\models_n$ (*af* (φ W_n ψ) (*w* [*1* $\rightarrow$ (*Suc n*)]))$[X]_\nu$
by (*metis B One-nat-def Suc-le-mono Suc-mono plus-1-eq-Suc subsequence-shift*)
ultimately
show *?case*
unfolding *af-subsequence-W* **unfolding** *semantics-ltln.simps GF-advice.simps* **by** *simp*
qed

3.7 Advice Functions and Propositional Entailment

lemma *GF-advice-prop-entailment*:
$\mathcal{A} \models_P \varphi[X]_\nu \Longrightarrow \{\psi.\ \psi[X]_\nu \in \mathcal{A}\} \models_P \varphi$
$false_n \notin \mathcal{A} \Longrightarrow \{\psi.\ \psi[X]_\nu \in \mathcal{A}\} \models_P \varphi \Longrightarrow \mathcal{A} \models_P \varphi[X]_\nu$
by (*induction* φ) (*auto*, *meson*, *meson*)

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

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

by (*induction* φ) *auto*

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

lemma *GF-advice-subst*:
$\varphi[X]_\nu = subst\ \varphi\ (\lambda\psi.\ Some\ (\psi[X]_\nu))$
by (*induction* φ) *auto*

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

lemma *GF-advice-prop-congruent*:
$\varphi \longrightarrow_P \psi \Longrightarrow \varphi[X]_\nu \longrightarrow_P \psi[X]_\nu$
$\varphi \sim_P \psi \Longrightarrow \varphi[X]_\nu \sim_P \psi[X]_\nu$
by (*metis GF-advice-subst subst-respects-ltl-prop-entailment*)+

lemma *FG-advice-prop-congruent*:
$\varphi \longrightarrow_P \psi \Longrightarrow \varphi[X]_\mu \longrightarrow_P \psi[X]_\mu$
$\varphi \sim_P \psi \Longrightarrow \varphi[X]_\mu \sim_P \psi[X]_\mu$
by (*metis FG-advice-subst subst-respects-ltl-prop-entailment*)+

3.8 GF-advice with Equivalence Relations

locale *GF-advice-congruent* = *ltl-equivalence* +
fixes
normalise :: *'a ltln* $\Rightarrow$ *'a ltln*
assumes
normalise-eq: $\varphi \sim normalise\ \varphi$
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*]:
$w \models_n normalise\ \varphi \longleftrightarrow w \models_n \varphi$
using *normalise-eq ltl-lang-equiv-def eq-implies-lang* **by** *blast*

end

interpretation *prop-GF-advice-compatible*: *GF-advice-congruent* $(\sim_P)$ *id*
by *unfold-locales* (*simp add*: *GF-advice-af GF-advice-prop-congruent*(*2*))+

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-$\mathcal{GF}$-Y-$\mathcal{FG}$*:
assumes
X-μ: $X \subseteq subformulas_\mu\ \varphi$
and
Y-ν: $Y \subseteq subformulas_\nu\ \varphi$
and
X-GF: $\forall \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)$
and
Y-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 −
— Custom induction rule with *size* as a partial order
note *induct* = *finite-ranking-induct*[**where** *f* = *size*]

have *finite* $(X \cup Y)$
using *subformulas*$_\mu$*-finite subformulas*$_\nu$*-finite X-μ Y-ν finite-subset*
by *blast*+

then show *?thesis*
using *assms*
proof (*induction* $X \cup Y$ *arbitrary*: $X\ Y\ \varphi$ *rule*: *induct*)
case (*insert* ψ *S*)

note *IH* = *insert*(*3*)
note *insert-S* = ‹*insert* ψ $S = X \cup Y$›
note *X-μ* = ‹$X \subseteq subformulas_\mu\ \varphi$›
note *Y-ν* = ‹$Y \subseteq subformulas_\nu\ \varphi$›

note $X\text{-}GF = \langle\forall\,\psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)\rangle$
note $Y\text{-}FG = \langle\forall\,\psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu)\rangle$

from $X\text{-}\mu$ $Y\text{-}\nu$ **have** $X \cap Y = \{\}$
  using *subformulas*$_{\mu\nu}$*-disjoint* **by** *fast*

from *insert-S* $X\text{-}\mu$ $Y\text{-}\nu$ **have** $\psi \in$ *subfrmlsn* φ
  using *subformulas*$_\mu$*-subfrmlsn* *subformulas*$_\nu$*-subfrmlsn* **by** *blast*

show *?case*
proof (*cases* $\psi \notin S$, *cases* $\psi \in X$)
  assume $\psi \notin S$ **and** $\psi \in X$

  {
    — Show $X - \{\psi\} \subseteq \mathcal{GF}\ \varphi\ w$ and $Y \subseteq \mathcal{FG}\ \varphi\ w$

    then have $\psi \notin Y$
      using $\langle X \cap Y = \{\}\rangle$ **by** *auto*
    then have $S = (X - \{\psi\}) \cup Y$
      using *insert-S* $\langle\psi \notin S\rangle$ **by** *fast*

    moreover

    have $\forall\,\psi' \in Y.\ \psi'[X - \{\psi\}]_\nu = \psi'[X]_\nu$
      using *GF-advice-minus-size* *insert*(*1*,*2*,*4*) $\langle\psi \notin Y\rangle$ **by** *fast*

    ultimately have $X - \{\psi\} \subseteq \mathcal{GF}\ \varphi\ w$ **and** $Y \subseteq \mathcal{FG}\ \varphi\ w$
      using *IH*[*of* $X - \{\psi\}$ Y φ] $X\text{-}\mu$ $Y\text{-}\nu$ $X\text{-}GF$ $Y\text{-}FG$ **by** *auto*
  }

  moreover

  {
    — Show $\psi \in \mathcal{GF}\ \varphi\ w$

    have $w \models_n G_n\ (F_n\ \psi[Y]_\mu)$
      using $X\text{-}GF$ $\langle\psi \in X\rangle$ **by** *simp*
    then have $\exists_\infty i.$ *suffix* i $w \models_n \psi[Y]_\mu$
      unfolding *GF-Inf-many* **by** *simp*

    moreover

    from $Y\text{-}\nu$ **have** *finite* Y
      using *subformulas*$_\nu$*-finite* *finite-subset* **by** *auto*

have $\forall \varphi \in Y.\ w \models_n F_n\ (G_n\ \varphi)$
using ‹$Y \subseteq \mathcal{FG}\ \varphi\ w$› **by** (*blast dest*: $\mathcal{FG}$*-elim*)
then have $\forall \varphi \in Y.\ \forall_\infty i.$ *suffix i w* $\models_n G_n\ \varphi$
using *FG-suffix-G* **by** *blast*
then have $\forall_\infty i.\ \forall \varphi \in Y.$ *suffix i w* $\models_n G_n\ \varphi$
using ‹*finite Y*› *eventually-ball-finite* **by** *fast*

ultimately

have $\exists_\infty i.$ *suffix i w* $\models_n \psi[Y]_\mu \wedge (\forall \varphi \in Y.$ *suffix i w* $\models_n G_n\ \varphi)$
using *INFM-conjI* **by** *auto*
then have $\exists_\infty i.$ *suffix i w* $\models_n \psi$
by (*elim frequently-elim1*) (*metis FG-advice-b2-helper*)
then have $w \models_n G_n\ (F_n\ \psi)$
unfolding *GF-Inf-many* **by** *simp*
then have $\psi \in \mathcal{GF}\ \varphi\ w$
unfolding $\mathcal{GF}$*-semantics* **using** ‹$\psi \in X$› *X-μ* **by** *auto*
}

ultimately show *?thesis*
by *auto*
next
assume $\psi \notin S$ **and** $\psi \notin X$
then have $\psi \in Y$
using *insert* **by** *fast*

{
— Show $X \subseteq \mathcal{GF}\ \varphi\ w$ and $Y - \{\psi\} \subseteq \mathcal{FG}\ \varphi\ w$

then have $S \cap X = X$
using *insert* ‹$\psi \notin X$› **by** *fast*
then have $S = X \cup (Y - \{\psi\})$
using *insert-S* ‹$\psi \notin S$› **by** *fast*

moreover

have $\forall \psi' \in X.\ \psi'[Y - \{\psi\}]_\mu = \psi'[Y]_\mu$
using *FG-advice-minus-size insert*(*1*,*2*,*4*) ‹$\psi \notin X$› **by** *fast*

ultimately have $X \subseteq \mathcal{GF}\ \varphi\ w$ **and** $Y - \{\psi\} \subseteq \mathcal{FG}\ \varphi\ w$
using *IH*[*of X Y* $- \{\psi\}\ \varphi$] *X-μ Y-ν X-GF Y-FG* **by** *auto*
}

moreover

{
— Show $\psi \in \mathcal{FG}\ \varphi\ w$

have $w \models_n F_n\ (G_n\ \psi[X]_\nu)$
using *Y-FG* ‹$\psi \in Y$› **by** *simp*
then have $\forall_\infty i.$ *suffix i w* $\models_n \psi[X]_\nu$
unfolding *FG-Alm-all* **by** *simp*

moreover

have $\forall \varphi \in X.\ w \models_n G_n\ (F_n\ \varphi)$
using ‹$X \subseteq \mathcal{GF}\ \varphi\ w$› **by** (*blast dest*: $\mathcal{GF}$*-elim*)
then have $\forall_\infty i.\ \forall \varphi \in X.$ *suffix i w* $\models_n G_n\ (F_n\ \varphi)$
by *simp*

ultimately

have $\forall_\infty i.$ *suffix i w* $\models_n \psi[X]_\nu \wedge (\forall \varphi \in X.$ *suffix i w* $\models_n G_n\ (F_n$
$\varphi))$
using *MOST-conjI* **by** *auto*
then have $\forall_\infty i.$ *suffix i w* $\models_n \psi$
by (*elim MOST-mono*) (*metis GF-advice-a2-helper*)
then have $w \models_n F_n\ (G_n\ \psi)$
unfolding *FG-Alm-all* **by** *simp*
then have $\psi \in \mathcal{FG}\ \varphi\ w$
unfolding $\mathcal{FG}$*-semantics* **using** ‹$\psi \in Y$› *Y-ν* **by** *auto*
}

ultimately show *?thesis*
by *auto*
next
assume $\neg\ \psi \notin S$
then have $S = X \cup Y$
using *insert* **by** *fast*
then show *?thesis*
using *insert* **by** *auto*
qed
qed *fast*
qed

lemma $\mathcal{GF}$*-implies-GF*:

$\forall \psi \in \mathcal{GF}\ \varphi\ w.\ w \models_n G_n\ (F_n\ \psi[\mathcal{FG}\ \varphi\ w]_\mu)$
proof *safe*
fix ψ
assume $\psi \in \mathcal{GF}\ \varphi\ w$

then have $\exists_\infty i.$ *suffix i w* $\models_n \psi$
using $\mathcal{GF}$*-elim GF-Inf-many* **by** *blast*

moreover

have $\psi \in$ *subfrmlsn* φ
using ‹$\psi \in \mathcal{GF}\ \varphi\ w$› $\mathcal{GF}$*-subfrmlsn* **by** *blast*

then have $\bigwedge i\ w.\ \mathcal{FG}\ \psi$ (*suffix i w*) $\subseteq \mathcal{FG}\ \varphi\ w$
using $\mathcal{FG}$*-suffix* $\mathcal{FG}$*-subset* **by** *blast*

ultimately have $\exists_\infty i.$ *suffix i w* $\models_n \psi[\mathcal{FG}\ \varphi\ w]_\mu$
by (*elim frequently-elim1*) (*metis FG-advice-b1*)

then show $w \models_n G_n\ (F_n\ \psi[\mathcal{FG}\ \varphi\ w]_\mu)$
unfolding *GF-Inf-many* **by** *simp*
qed

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

then have $\forall_\infty i.$ *suffix i w* $\models_n \psi$
using $\mathcal{FG}$*-elim FG-Alm-all* **by** *blast*

moreover

{
have $\psi \in$ *subfrmlsn* φ
using ‹$\psi \in \mathcal{FG}\ \varphi\ w$› $\mathcal{FG}$*-subfrmlsn* **by** *blast*

moreover have $\forall_\infty i.\ \mathcal{GF}\ \psi$ (*suffix i w*) $= \mathcal{F}\ \psi$ (*suffix i w*)
using *suffix-*μ*-stable* **unfolding** μ*-stable-def* **by** *blast*

ultimately have $\forall_\infty i.\ \mathcal{F}\ \psi$ (*suffix i w*) $\subseteq \mathcal{GF}\ \varphi\ w$
unfolding *MOST-nat-le* **by** (*metis* $\mathcal{GF}$*-subset* $\mathcal{GF}$*-suffix*)

}

ultimately have $\forall_\infty i.\ \mathcal{F}\ \psi\ (\mathit{suffix}\ i\ w) \subseteq \mathcal{GF}\ \varphi\ w \wedge \mathit{suffix}\ i\ w \models_n \psi$
  using *eventually-conj* **by** *auto*

then have $\forall_\infty i.\ \mathit{suffix}\ i\ w \models_n \psi[\mathcal{GF}\ \varphi\ w]_\nu$
  using *GF-advice-a1* **by** (*elim eventually-mono*) *auto*

then show $w \models_n F_n\ (G_n\ \psi[\mathcal{GF}\ \varphi\ w]_\nu)$
  unfolding *FG-Alm-all* **by** *simp*
qed

4.2 Putting the pieces together: The Master Theorem

theorem *master-theorem-ltr*:
  assumes
    $w \models_n \varphi$
  obtains X **and** Y **where**
    $X \subseteq \mathit{subformulas}_\mu\ \varphi$
  and
    $Y \subseteq \mathit{subformulas}_\nu\ \varphi$
  and
    $\exists\, i.\ \mathit{suffix}\ i\ w \models_n \mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w)[X]_\nu$
  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
  show $\mathcal{GF}\ \varphi\ w \subseteq \mathit{subformulas}_\mu\ \varphi$
    by (*rule* $\mathcal{GF}$*-subformulas*$_\mu$)
next
  show $\mathcal{FG}\ \varphi\ w \subseteq \mathit{subformulas}_\nu\ \varphi$
    by (*rule* $\mathcal{FG}$*-subformulas*$_\nu$)
next
  obtain i **where** $\mathcal{GF}\ \varphi\ (\mathit{suffix}\ i\ w) = \mathcal{F}\ \varphi\ (\mathit{suffix}\ i\ w)$
    using *suffix-*μ*-stable* **unfolding** *MOST-nat* μ*-stable-def* **by** *fast*
  then have $\mathcal{F}\ (\mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w))\ (\mathit{suffix}\ i\ w) \subseteq \mathcal{GF}\ \varphi\ w$
    using $\mathcal{GF}$*-af* $\mathcal{F}$*-af* $\mathcal{GF}$*-suffix* **by** *fast*

  moreover

  have $\mathit{suffix}\ i\ w \models_n \mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w)$
    using *af-ltl-continuation* ‹$w \models_n \varphi$› **by** *fastforce*

ultimately show $\exists\, i.$ *suffix i w* $\models_n$ *af* φ *(prefix i w)*$[\mathcal{GF}\ \varphi\ w]_\nu$
  using *GF-advice-a1* **by** *blast*
next
  show $\forall\, \psi \in \mathcal{GF}\ \varphi\ w.\ w \models_n G_n\ (F_n\ \psi[\mathcal{FG}\ \varphi\ w]_\mu)$
    by (*rule* $\mathcal{GF}$*-implies-GF*)
next
  show $\forall\, \psi \in \mathcal{FG}\ \varphi\ w.\ w \models_n F_n\ (G_n\ \psi[\mathcal{GF}\ \varphi\ w]_\nu)$
    by (*rule* $\mathcal{FG}$*-implies-FG*)
qed

theorem *master-theorem-rtl*:
  assumes
    $X \subseteq$ *subformulas*$_\mu$ φ
  and
    $Y \subseteq$ *subformulas*$_\nu$ φ
  and
    1: $\exists\, i.$ *suffix i w* $\models_n$ *af* φ *(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 −
  from *1* **obtain** *i* **where** *suffix i w* $\models_n$ *af* φ *(prefix i w)*$[X]_\nu$
    by *blast*

  moreover

  from *assms* **have** $X \subseteq \mathcal{GF}\ \varphi\ w$
    using *X-*$\mathcal{GF}$*-Y-*$\mathcal{FG}$ **by** *blast*
  then have $X \subseteq \mathcal{GF}\ \varphi$ (*suffix i w*)
    using $\mathcal{GF}$*-suffix* **by** *fast*

  ultimately have *suffix i w* $\models_n$ *af* φ *(prefix i w)*
    using *GF-advice-a2* $\mathcal{GF}$*-af* **by** *metis*
  then show $w \models_n \varphi$
    using *af-ltl-continuation* **by** *force*
qed

theorem *master-theorem*:
  $w \models_n \varphi \longleftrightarrow$
    $(\exists\, X \subseteq$ *subformulas*$_\mu$ φ.
      $(\exists\, Y \subseteq$ *subformulas*$_\nu$ φ.

$(\exists\, i.\ \mathit{suffix}\ i\ w \models_n \mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w)[X]_\nu)$
$\wedge\ (\forall\, \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu))$
$\wedge\ (\forall\, \psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu))))$
by (*metis master-theorem-ltr master-theorem-rtl*)

4.3 The Master Theorem on Languages

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

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*:
$\mathit{language\text{-}ltln}\ \varphi = \bigcup\ \{L_1\ \varphi\ X \cap L_2\ X\ Y \cap L_3\ X\ Y \mid X\ Y.\ X \subseteq \mathit{subformulas}_\mu\ \varphi \wedge Y \subseteq \mathit{subformulas}_\nu\ \varphi\}$
proof *safe*
fix w
assume $w \in \mathit{language\text{-}ltln}\ \varphi$

then have $w \models_n \varphi$
unfolding *language-ltln-def* **by** *simp*

then obtain $X\ Y$ **where** $X \subseteq \mathit{subformulas}_\mu\ \varphi$ **and** $Y \subseteq \mathit{subformulas}_\nu\ \varphi$
and $\exists\, i.\ \mathit{suffix}\ i\ w \models_n \mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w)[X]_\nu$
and $\forall\, \psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)$
and $\forall\, \psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu)$
using *master-theorem-ltr* **by** *metis*

then have $w \in L_1\ \varphi\ X$ **and** $w \in L_2\ X\ Y$ **and** $w \in L_3\ X\ Y$
unfolding L_1*-def* L_2*-def* L_3*-def* **by** *simp+*

then show $w \in \bigcup\ \{L_1\ \varphi\ X \cap L_2\ X\ Y \cap L_3\ X\ Y \mid X\ Y.\ X \subseteq \mathit{subformulas}_\mu\ \varphi \wedge Y \subseteq \mathit{subformulas}_\nu\ \varphi\}$
using ‹$X \subseteq \mathit{subformulas}_\mu\ \varphi$› ‹$Y \subseteq \mathit{subformulas}_\nu\ \varphi$› **by** *blast*
next
fix $w\ X\ Y$
assume $X \subseteq \mathit{subformulas}_\mu\ \varphi$ **and** $Y \subseteq \mathit{subformulas}_\nu\ \varphi$
and $w \in L_1\ \varphi\ X$ **and** $w \in L_2\ X\ Y$ **and** $w \in L_3\ X\ Y$

then show $w \in \mathit{language\text{-}ltln}\ \varphi$
unfolding *language-ltln-def* L_1*-def* L_2*-def* L_3*-def*
using *master-theorem-rtl* **by** *blast*
qed

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*:
  $\psi \in subfrmlsn\ \varphi \Longrightarrow suffix\ i\ w \models_n \psi \Longrightarrow suffix\ i\ w \models_n \psi[\mathcal{FG}\ \varphi\ w]_\mu$
proof −
  assume $\psi \in subfrmlsn\ \varphi$

  then have $\mathcal{FG}\ \psi\ (suffix\ i\ w) \subseteq \mathcal{FG}\ \varphi\ w$
    using *$\mathcal{FG}$-suffix subformulas$_\nu$-subset* **unfolding** *$\mathcal{FG}$-semantics′* **by** *fast*

  moreover

  assume $suffix\ i\ w \models_n \psi$

  ultimately show $suffix\ i\ w \models_n \psi[\mathcal{FG}\ \varphi\ w]_\mu$
    using *FG-advice-b1* **by** *blast*
qed

lemma *FG-advice-b2-helper*:
  $S \subseteq \mathcal{G}\ \varphi\ (suffix\ i\ w) \Longrightarrow i \leq j \Longrightarrow suffix\ j\ w \models_n \psi[S]_\mu \Longrightarrow suffix\ j\ w \models_n \psi$
proof −
  fix $i\ j$
  assume $S \subseteq \mathcal{G}\ \varphi\ (suffix\ i\ w)$ **and** $i \leq j$ **and** $suffix\ j\ w \models_n \psi[S]_\mu$

  then have $suffix\ j\ w \models_n \psi[S \cap subformulas_\nu\ \psi]_\mu$
    using *FG-advice-inter-subformulas* **by** *metis*

  moreover

  have $S \cap subformulas_\nu\ \psi \subseteq \mathcal{G}\ \psi\ (suffix\ i\ w)$
    using ‹$S \subseteq \mathcal{G}\ \varphi\ (suffix\ i\ w)$› **unfolding** *$\mathcal{G}$-semantics′* **by** *blast*

then have $S \cap subformulas_\nu\ \psi \subseteq \mathcal{G}\ \psi\ (suffix\ j\ w)$
using $\mathcal{G}$-*suffix* ‹$i \leq j$› *inf.absorb-iff2 le-Suc-ex* **by** *fastforce*

ultimately show $suffix\ j\ w \models_n \psi$
using *FG-advice-b2* **by** *blast*
qed

lemma *Y*-$\mathcal{G}$:
assumes
Y-ν: $Y \subseteq subformulas_\nu\ \varphi$
and
Y-G-1: $\forall \psi_1\ \psi_2.\ \psi_1\ R_n\ \psi_2 \in Y \longrightarrow suffix\ i\ w \models_n G_n\ (\psi_2[Y]_\mu)$
and
Y-G-2: $\forall \psi_1\ \psi_2.\ \psi_1\ W_n\ \psi_2 \in Y \longrightarrow suffix\ i\ w \models_n G_n\ (\psi_1[Y]_\mu\ or_n\ \psi_2[Y]_\mu)$
shows
$Y \subseteq \mathcal{G}\ \varphi\ (suffix\ i\ w)$
proof –
— Custom induction rule with *size* as a partial order
note *induct* = *finite-ranking-induct*[**where** *f* = *size*]

have *finite Y*
using *Y*-ν *finite-subset subformulas*$_\nu$*-finite* **by** *auto*

then show *?thesis*
using *assms*
proof (*induction Y rule*: *induct*)
case (*insert* ψ *S*)

show *?case*
proof (*cases* $\psi \notin S$)
assume $\psi \notin S$

note *FG-advice-insert* = *FG-advice-insert*[*OF* ‹$\psi \notin S$›]

{
— Show $S \subseteq \mathcal{G}\ \varphi\ (suffix\ i\ w)$

{
fix $\psi_1\ \psi_2$
assume $\psi_1\ R_n\ \psi_2 \in S$

then have $suffix\ i\ w \models_n G_n\ \psi_2[insert\ \psi\ S]_\mu$
using *insert*(*5*) **by** *blast*

then have *suffix i w* $\models_n$ G_n $\psi_2[S]_\mu$
using ‹ψ_1 R_n $\psi_2 \in S$› *FG-advice-insert insert.hyps*(*2*)
by *fastforce*
}

moreover

{
fix ψ_1 ψ_2
assume ψ_1 W_n $\psi_2 \in S$

then have *suffix i w* $\models_n$ G_n (ψ_1[*insert* ψ S]$_\mu$ or_n ψ_2[*insert* ψ S]$_\mu$)
using *insert*(*6*) **by** *blast*

then have *suffix i w* $\models_n$ G_n ($\psi_1[S]_\mu$ or_n $\psi_2[S]_\mu$)
using ‹ψ_1 W_n $\psi_2 \in S$› *FG-advice-insert insert.hyps*(*2*)
by *fastforce*
}

ultimately

have $S \subseteq \mathcal{G}$ φ (*suffix i w*)
using *insert.IH insert.prems*(*1*) **by** *blast*
}

moreover

{
— Show $\psi \in \mathcal{G}$ φ (*suffix i w*)

have $\psi \in$ *subformulas*$_\nu$ φ
using *insert.prems*(*1*) **by** *fast*
then have *suffix i w* $\models_n$ G_n ψ
using *subformulas*$_\nu$*-semantics*
proof (*cases* ψ)
case (*Release-ltln* ψ_1 ψ_2)

then have *suffix i w* $\models_n$ G_n ψ_2[*insert* ψ S]$_\mu$
using *insert.prems*(*2*) **by** *blast*
then have *suffix i w* $\models_n$ G_n $\psi_2[S]_\mu$
using *Release-ltln FG-advice-insert* **by** *simp*
then have *suffix i w* $\models_n$ G_n ψ_2
using *FG-advice-b2-helper*[*OF* ‹$S \subseteq \mathcal{G}$ φ (*suffix i w*)›] **by** *auto*

then show *?thesis*
using *Release-ltln globally-release*
by *blast*
next
case (*WeakUntil-ltln* ψ_1 ψ_2)

then have *suffix i w* $\models_n$ G_n (ψ_1[*insert* ψ *S*]$_\mu$ *or*$_n$ ψ_2[*insert* ψ *S*]$_\mu$)
using *insert.prems*(*3*) **by** *blast*
then have *suffix i w* $\models_n$ G_n (ψ_1 *or*$_n$ ψ_2)[*S*]$_\mu$
using *WeakUntil-ltln FG-advice-insert* **by** *simp*
then have *suffix i w* $\models_n$ G_n (ψ_1 *or*$_n$ ψ_2)
using *FG-advice-b2-helper*[*OF* ‹*S* $\subseteq$ $\mathcal{G}$ φ (*suffix i w*)›, *of* - ψ_1 *or*$_n$
ψ_2]
by *force*
then show *?thesis*
unfolding *WeakUntil-ltln semantics-ltln.simps*
by (*metis order-refl suffix-suffix*)
qed *fast+*

then have $\psi \in \mathcal{G}$ φ (*suffix i w*)
unfolding $\mathcal{G}$*-semantics* **using** ‹$\psi \in$ *subformulas*$_\nu$ φ›
by *simp*
}

ultimately show *?thesis*
by *blast*
next
assume $\neg$ $\psi \notin S$
then have *insert* ψ *S* = *S*
by *auto*
then show *?thesis*
using *insert* **by** *simp*
qed
qed *simp*
qed

theorem *asymmetric-master-theorem-ltr*:
assumes
w $\models_n$ φ
obtains *Y* **and** *i* **where**
Y $\subseteq$ *subformulas*$_\nu$ φ
and
suffix i w $\models_n$ *af* φ (*prefix i w*)[*Y*]$_\mu$
and

```
    ∀ψ₁ ψ₂. ψ₁ Rₙ ψ₂ ∈ Y ⟶ suffix i w ⊨ₙ Gₙ (ψ₂[Y]μ)
  and
    ∀ψ₁ ψ₂. ψ₁ Wₙ ψ₂ ∈ Y ⟶ suffix i w ⊨ₙ Gₙ (ψ₁[Y]μ orₙ ψ₂[Y]μ)
proof
  let ?Y = ℱ𝒢 φ w

  show ?Y ⊆ subformulasν φ
    by (rule ℱ𝒢-subformulasν)
next
  let ?Y = ℱ𝒢 φ w
  let ?i = SOME i. ?Y = 𝒢 φ (suffix i w)

  have suffix ?i w ⊨ₙ af φ (prefix ?i w)
    using af-ltl-continuation ‹w ⊨ₙ φ› by fastforce
  then show suffix ?i w ⊨ₙ af φ (prefix ?i w)[?Y]μ
    by (metis ℱ𝒢-suffix FG-advice-b1 ℱ𝒢-af order-refl)
next
  let ?Y = ℱ𝒢 φ w
  let ?i = SOME i. ?Y = 𝒢 φ (suffix i w)

  have ∃i. ?Y = 𝒢 φ (suffix i w)
    using suffix-ν-stable ℱ𝒢-suffix unfolding ν-stable-def MOST-nat
    by fast
  then have Y-G: ?Y = 𝒢 φ (suffix ?i w)
    by (metis (mono-tags, lifting) someI-ex)

  show ∀ψ₁ ψ₂. ψ₁ Rₙ ψ₂ ∈ ?Y ⟶ suffix ?i w ⊨ₙ Gₙ (ψ₂[?Y]μ)
  proof safe
    fix ψ₁ ψ₂
    assume ψ₁ Rₙ ψ₂ ∈ ?Y

    then have suffix ?i w ⊨ₙ Gₙ (ψ₁ Rₙ ψ₂)
      using Y-G 𝒢-semantics' by blast
    then have suffix ?i w ⊨ₙ Gₙ ψ₂
      by force

    moreover

    have ψ₂ ∈ subfrmlsn φ
      using ℱ𝒢-subfrmlsn ‹ψ₁ Rₙ ψ₂ ∈ ?Y› subfrmlsn-subset by force

    ultimately show suffix ?i w ⊨ₙ Gₙ (ψ₂ [?Y]μ)
      using FG-advice-b1-helper by fastforce
  qed
```

next
  let $?Y = \mathcal{FG}\ \varphi\ w$
  let $?i = SOME\ i.\ ?Y = \mathcal{G}\ \varphi\ (suffix\ i\ w)$

  have $\exists\, i.\ ?Y = \mathcal{G}\ \varphi\ (suffix\ i\ w)$
    using *suffix-ν-stable $\mathcal{FG}$-suffix* **unfolding** *ν-stable-def MOST-nat*
    by *fast*
  then have *Y-G*: $?Y = \mathcal{G}\ \varphi\ (suffix\ ?i\ w)$
    by (*rule someI-ex*)

  show $\forall\, \psi_1\ \psi_2.\ \psi_1\ W_n\ \psi_2 \in\ ?Y \longrightarrow suffix\ ?i\ w \models_n G_n\ (\psi_1[?Y]_\mu\ or_n\ \psi_2[?Y]_\mu)$
  proof *safe*
    fix $\psi_1\ \psi_2$
    assume $\psi_1\ W_n\ \psi_2 \in\ ?Y$

    then have $suffix\ ?i\ w \models_n G_n\ (\psi_1\ W_n\ \psi_2)$
      using *Y-G $\mathcal{G}$-semantics′* **by** *blast*
    then have $suffix\ ?i\ w \models_n G_n\ (\psi_1\ or_n\ \psi_2)$
      by *force*

    moreover

    have $\psi_1 \in subfrmlsn\ \varphi$ **and** $\psi_2 \in subfrmlsn\ \varphi$
      using *$\mathcal{FG}$-subfrmlsn* ‹$\psi_1\ W_n\ \psi_2 \in\ ?Y$› *subfrmlsn-subset* **by** *force+*

    ultimately show $suffix\ ?i\ w \models_n G_n\ (\psi_1[?Y]_\mu\ or_n\ \psi_2[?Y]_\mu)$
      using *FG-advice-b1-helper* **by** *fastforce*
  qed
qed

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

by (*metis assms Y-$\mathcal{G}$ FG-advice-b2 $\mathcal{G}$-af*)

then show $w \models_n \varphi$
using *af-ltl-continuation* **by** *force*
qed

theorem *asymmetric-master-theorem*:
$w \models_n \varphi \longleftrightarrow$
$(\exists\, i.\ \exists\, Y \subseteq \mathit{subformulas}_\nu\ \varphi.$
$\mathit{suffix}\ i\ w \models_n \mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w)[Y]_\mu$
$\wedge (\forall\, \psi_1\ \psi_2.\ \psi_1\ R_n\ \psi_2 \in Y \longrightarrow \mathit{suffix}\ i\ w \models_n G_n\ (\psi_2[Y]_\mu))$
$\wedge (\forall\, \psi_1\ \psi_2.\ \psi_1\ W_n\ \psi_2 \in Y \longrightarrow \mathit{suffix}\ i\ w \models_n G_n\ (\psi_1[Y]_\mu\ \mathit{or}_n\ \psi_2[Y]_\mu)))$
by (*metis asymmetric-master-theorem-ltr asymmetric-master-theorem-rtl*)

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 $(\varphi\ \mathit{and}_n\ \psi)$ = *restricted-subformulas-inner* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas-inner* $(\varphi\ \mathit{or}_n\ \psi)$ = *restricted-subformulas-inner* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas-inner* $(X_n\ \varphi)$ = *restricted-subformulas-inner* φ
| *restricted-subformulas-inner* $(\varphi\ U_n\ \psi)$ = $\mathit{subformulas}_\nu\ (\varphi\ U_n\ \psi) \cup \mathit{subformulas}_\mu\ (\varphi\ U_n\ \psi)$
| *restricted-subformulas-inner* $(\varphi\ R_n\ \psi)$ = *restricted-subformulas-inner* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas-inner* $(\varphi\ W_n\ \psi)$ = *restricted-subformulas-inner* φ ∪ *restricted-subformulas-inner* ψ
| *restricted-subformulas-inner* $(\varphi\ M_n\ \psi)$ = $\mathit{subformulas}_\nu\ (\varphi\ M_n\ \psi) \cup \mathit{subformulas}_\mu\ (\varphi\ M_n\ \psi)$
| *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 ($\varphi[X]_\nu$) = {}
by (*induction* φ) *simp-all*

lemma *GF-advice-restricted-subformulas*:
restricted-subformulas ($\varphi[X]_\nu$) = {}
by (*induction* φ) (*simp-all add*: *GF-advice-restricted-subformulas-inner*)

lemma *restricted-subformulas-inner-subset*:
restricted-subformulas-inner $\varphi \subseteq$ *subformulas*$_\nu$ φ ∪ *subformulas*$_\mu$ φ
by (*induction* φ) *auto*

lemma *restricted-subformulas-subset'*:
restricted-subformulas $\varphi \subseteq$ *restricted-subformulas-inner* φ
by (*induction* φ) (*insert restricted-subformulas-inner-subset*, *auto*)

lemma *restricted-subformulas-subset*:
restricted-subformulas $\varphi \subseteq$ *subformulas*$_\nu$ φ ∪ *subformulas*$_\mu$ φ
using *restricted-subformulas-inner-subset restricted-subformulas-subset'* **by** *auto*

lemma *restricted-subformulas-size*:
$\psi \in$ *restricted-subformulas* $\varphi \Longrightarrow$ *size* $\psi <$ *size* φ
proof –
have $\bigwedge\varphi$. *restricted-subformulas-inner* $\varphi \subseteq$ *subfrmlsn* φ
using *restricted-subformulas-inner-subset subformulas*$_{\mu\nu}$*-subfrmlsn* **by** *blast*

then have *inner*: $\bigwedge\psi\ \varphi$. $\psi \in$ *restricted-subformulas-inner* $\varphi \Longrightarrow$ *size* ψ

$\leq$ *size* φ
  using *subfrmlsn-size dual-order.strict-implies-order*
  by *blast*

 show $\psi \in$ *restricted-subformulas* $\varphi \Longrightarrow$ *size* $\psi <$ *size* φ
  by (*induction* φ *arbitrary*: ψ) (*fastforce dest*: *inner*)+
qed

lemma *restricted-subformulas-notin*:
 $\varphi \notin$ *restricted-subformulas* φ
 using *restricted-subformulas-size* **by** *auto*

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

 then obtain χ x **where**
  $\psi \in$ *restricted-subformulas-inner* χ **and** (x R_n χ) $\in$ *subformulas*$_\nu$ φ $\vee$ (χ W_n x) $\in$ *subformulas*$_\nu$ φ
  by (*induction* φ) *auto*

 moreover

 have $\bigwedge\psi_1\ \psi_2.$ (ψ_1 R_n ψ_2) $\in$ *subformulas*$_\nu$ $\varphi \Longrightarrow$ *restricted-subformulas-inner* $\psi_2 \subseteq$ *restricted-subformulas* φ
  $\bigwedge\psi_1\ \psi_2.$ (ψ_1 W_n ψ_2) $\in$ *subformulas*$_\nu$ $\varphi \Longrightarrow$ *restricted-subformulas-inner* $\psi_1 \subseteq$ *restricted-subformulas* φ
  by (*induction* φ) (*simp-all*; *insert restricted-subformulas-subset*′, *blast*)+

 moreover

 have *subformulas*$_\nu$ $\psi \cup$ *subformulas*$_\mu$ $\psi \subseteq$ *restricted-subformulas-inner* χ
  using ‹$\psi \in$ *restricted-subformulas-inner* χ›
 proof (*induction* χ)
  case (*Until-ltln* $\chi 1$ $\chi 2$)
  then show *?case*
   apply (*cases* $\psi = \chi 1$ U_n $\chi 2$)
    apply *auto*[*1*]
   apply *simp*
   apply (*cases* $\psi \in$ *subformulas*$_\nu$ $\chi 1$)
   apply (*meson le-supI1 le-supI2 subformulas*$_\mu$*-subset subformulas*$_\nu$*-subfrmlsn*

subformulas$_\nu$*-subset subset-eq subset-insertI2*)
  apply (*cases* $\psi \in$ *subformulas*$_\nu$ $\chi 2$)
  apply (*meson le-supI1 le-supI2 subformulas*$_\mu$*-subset subformulas*$_\nu$*-subfrmlsn subformulas*$_\nu$*-subset subset-eq subset-insertI2*)
  apply (*cases* $\psi \in$ *subformulas*$_\mu$ $\chi 1$)
  apply (*metis* (*no-types*, *opaque-lifting*) *Un-insert-right subformulas*$_\mu$*-subfrmlsn subformulas*$_\mu$*-subset subformulas*$_\nu$*-subset subsetD sup.coboundedI2 sup-commute*)
  apply *simp*
  by (*metis* (*no-types*, *opaque-lifting*) *Un-insert-right subformulas*$_\mu$*-subfrmlsn subformulas*$_\mu$*-subset subformulas*$_\nu$*-subset subsetD sup.coboundedI2 sup-commute*)
next
  case (*Release-ltln* $\chi 1$ $\chi 2$)
  then show *?case* **by** *simp blast*
next
  case (*WeakUntil-ltln* $\chi 1$ $\chi 2$)
  then show *?case* **by** *simp blast*
next
  case (*StrongRelease-ltln* $\chi 1$ $\chi 2$)
  then show *?case*
  apply (*cases* $\psi = \chi 1\ M_n\ \chi 2$)
  apply *auto*[*1*]
  apply *simp*
  apply (*cases* $\psi \in$ *subformulas*$_\nu$ $\chi 1$)
  apply (*meson le-supI1 le-supI2 subformulas*$_\mu$*-subset subformulas*$_\nu$*-subfrmlsn subformulas*$_\nu$*-subset subset-eq subset-insertI2*)
  apply (*cases* $\psi \in$ *subformulas*$_\nu$ $\chi 2$)
  apply (*meson le-supI1 le-supI2 subformulas*$_\mu$*-subset subformulas*$_\nu$*-subfrmlsn subformulas*$_\nu$*-subset subset-eq subset-insertI2*)
  apply (*cases* $\psi \in$ *subformulas*$_\mu$ $\chi 1$)
  apply (*metis* (*no-types*, *opaque-lifting*) *Un-insert-right subformulas*$_\mu$*-subfrmlsn subformulas*$_\mu$*-subset subformulas*$_\nu$*-subset subsetD sup.coboundedI2 sup-commute*)
  apply *simp*
  by (*metis* (*no-types*, *opaque-lifting*) *Un-insert-right subformulas*$_\mu$*-subfrmlsn subformulas*$_\mu$*-subset subformulas*$_\nu$*-subset subsetD sup.coboundedI2 sup-commute*)
qed *auto*

ultimately

show *subformulas*$_\nu$ $\psi \cup$ *subformulas*$_\mu$ $\psi \subseteq$ *restricted-subformulas* φ
  by *blast*
qed

lemma *restricted-subformulas-W-μ*:
subformulas$_\mu$ $\varphi \subseteq$ *restricted-subformulas* (φ W_n ψ)
by (*induction* φ) *auto*

lemma *restricted-subformulas-R-μ*:
subformulas$_\mu$ $\psi \subseteq$ *restricted-subformulas* (φ R_n ψ)
by (*induction* ψ) *auto*

lemma *restrict-af-letter*:
restricted-subformulas (*af-letter* φ σ) = *restricted-subformulas* φ
proof (*induction* φ)
case (*Release-ltln* φ*1* φ*2*)
then show *?case*
using *restricted-subformulas-subset′* **by** *simp blast*
next
case (*WeakUntil-ltln* φ*1* φ*2*)
then show *?case*
using *restricted-subformulas-subset′* **by** *simp blast*
qed *auto*

lemma *restrict-af*:
restricted-subformulas (*af* φ *w*) = *restricted-subformulas* φ
by (*induction w rule*: *rev-induct*) (*auto simp*: *restrict-af-letter*)

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* $\varphi]_\nu$
using *assms*
proof (*induction* φ *arbitrary*: *w*)
case (*Prop-ltln x*)
then show *?case*
by (*metis GF-advice.simps*(*10*) *GF-advice-af prefix-suffix*)
next
case (*Nprop-ltln x*)
then show *?case*
by (*metis GF-advice.simps*(*11*) *GF-advice-af prefix-suffix*)
next
case (*And-ltln* φ*1* φ*2*)

let *?X* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* (φ*1* *and*$_n$ φ*2*)

let *?X1* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 1$
let *?X2* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 2$

have *?X1* $\subseteq$ *?X* **and** *?X2* $\subseteq$ *?X*
by *auto*

moreover

obtain *i j* **where** *suffix i w* $\models_n$ *af* $\varphi 1$ (*prefix i w*)[*?X1*]$_\nu$
and *suffix j w* $\models_n$ *af* $\varphi 2$ (*prefix j w*)[*?X2*]$_\nu$
using μ*-stable-subfrmlsn*[*OF* ‹μ*-stable* ($\varphi 1$ *and*$_n$ $\varphi 2$) *w*›] *And-ltln* **by** *fastforce*

ultimately

obtain *k* **where** *suffix k w* $\models_n$ *af* $\varphi 1$ (*prefix k w*)[*?X*]$_\nu$
and *suffix k w* $\models_n$ *af* $\varphi 2$ (*prefix k w*)[*?X*]$_\nu$
using *GF-advice-sync-and GF-advice-monotone* **by** *blast*

thus *?case*
unfolding *af-decompose semantics-ltln.simps*(*5*) *GF-advice.simps* **by** *blast*
next
case (*Or-ltln* $\varphi 1$ $\varphi 2$)
let *?X* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* ($\varphi 1$ *and*$_n$ $\varphi 2$)
let *?X1* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 1$
let *?X2* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 2$

have *?X1* $\subseteq$ *?X* **and** *?X2* $\subseteq$ *?X*
by *auto*

moreover

obtain *i j* **where** *suffix i w* $\models_n$ *af* $\varphi 1$ (*prefix i w*)[*?X1*]$_\nu$ $\vee$ *suffix j w* $\models_n$ *af* $\varphi 2$ (*prefix j w*)[*?X2*]$_\nu$
using μ*-stable-subfrmlsn*[*OF* ‹μ*-stable* ($\varphi 1$ *or*$_n$ $\varphi 2$) *w*›] *Or-ltln* **by** *fastforce*

ultimately

obtain *k* **where** *suffix k w* $\models_n$ *af* $\varphi 1$ (*prefix k w*)[*?X*]$_\nu$ $\vee$ *suffix k w* $\models_n$ *af* $\varphi 2$ (*prefix k w*)[*?X*]$_\nu$
using *GF-advice-monotone* **by** *blast*

thus *?case*
unfolding *af-decompose semantics-ltln.simps*(*6*) *GF-advice.simps* **by** *auto*
next
case (*Next-ltln* φ)
then have *stable*: μ*-stable* φ (*suffix 1 w*)
and *suffix*: *suffix 1 w* $\models_n \varphi$
using $\mathcal{F}$*-suffix* $\mathcal{GF}$*-*$\mathcal{F}$*-subset* $\mathcal{GF}$*-suffix*
by (*simp-all add*: μ*-stable-def*) *fast*
show *?case*
by (*metis* (*no-types*, *lifting*) *Next-ltln.IH*[*OF stable suffix*, *unfolded suffix-suffix prefix-suffix-subsequence GF-suffix*] *One-nat-def add.commute af-simps*(*3*) *foldl-Nil foldl-append restricted-subformulas.simps*(*3*) *subsequence-append subsequence-singleton*)
next
case (*Until-ltln* $\varphi 1$ $\varphi 2$)
let *?X* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* ($\varphi 1\ U_n\ \varphi 2$)
let *?X1* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 1$
let *?X2* = $\{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 2$

have *stable-1*: $\bigwedge i.$ μ*-stable* $\varphi 1$ (*suffix i w*)
and *stable-2*: $\bigwedge i.$ μ*-stable* $\varphi 2$ (*suffix i w*)
using μ*-stable-subfrmlsn*[*OF Until-ltln.prems*(*1*)] **by** (*simp add*: μ*-stable-suffix*)+

obtain *i* **where** $\bigwedge j.\ j < i \Longrightarrow$ *suffix j w* $\models_n \varphi 1$ **and** *suffix i w* $\models_n \varphi 2$
using *Until-ltln* **by** *auto*

then have $\bigwedge j.\ j < i \Longrightarrow \exists k.$ *suffix* $(j + k)$ *w* $\models_n$ *af* $\varphi 1$ (*w* $[j \rightarrow k + j]$)[*?X1*]$_\nu$
and $\exists k.$ *suffix* $(i + k)$ *w* $\models_n$ *af* $\varphi 2$ (*w* $[i \rightarrow k + i]$)[*?X2*]$_\nu$
using *Until-ltln.IH*(*1*)[*OF stable-1*, *unfolded suffix-suffix prefix-suffix-subsequence GF-suffix*]
using *Until-ltln.IH*(*2*)[*OF stable-2*, *unfolded suffix-suffix prefix-suffix-subsequence GF-suffix*]
by *blast+*

moreover

have *?X1* $\subseteq$ *?X*
and *?X2* $\subseteq$ *?X*
by *auto*

ultimately

obtain k **where** $k \geq i$
and *suffix* k $w \models_n$ *af* $\varphi 2$ $(w\ [i \to k])[?X]_\nu$
and $\bigwedge j.\ j < i \Longrightarrow$ *suffix* k $w \models_n$ *af* $\varphi 1$ $(w\ [j \to k])[?X]_\nu$
using *GF-advice-sync-less*[*of* i w $\varphi 1$ $?X$ $\varphi 2$] *GF-advice-monotone*[*of* - $?X$] **by** *meson*

hence *suffix* (*Suc* k) $w \models_n$ *af* $(\varphi 1\ U_n\ \varphi 2)$ (*prefix* (*Suc* k) $w)[?X]_\nu$
by (*rule af-subsequence-U-GF-advice*)

then show *?case*
by *blast*
next
case (*WeakUntil-ltln* $\varphi 1$ $\varphi 2$)

let $?X\ = \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $(\varphi 1\ W_n\ \varphi 2)$
let $?X1 = \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 1$
let $?X2 = \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 2$

have *stable-1*: $\bigwedge i.$ μ*-stable* $\varphi 1$ (*suffix* i w)
and *stable-2*: $\bigwedge i.$ μ*-stable* $\varphi 2$ (*suffix* i w)
using μ*-stable-subfrmlsn*[*OF WeakUntil-ltln.prems*(*1*)] **by** (*simp add*: μ*-stable-suffix*)+

{
assume *Until-ltln*: $w \models_n \varphi 1\ U_n\ \varphi 2$
then obtain i **where** $\bigwedge j.\ j < i \Longrightarrow$ *suffix* j $w \models_n \varphi 1$ **and** *suffix* i $w \models_n \varphi 2$
by *auto*

then have $\bigwedge j.\ j < i \Longrightarrow \exists k.$ *suffix* $(j + k)$ $w \models_n$ *af* $\varphi 1$ $(w\ [j \to k + j])[?X1]_\nu$
and $\exists k.$ *suffix* $(i + k)$ $w \models_n$ *af* $\varphi 2$ $(w\ [i \to k + i])[?X2]_\nu$
using *WeakUntil-ltln.IH*(*1*)[*OF stable-1*, *unfolded suffix-suffix prefix-suffix-subsequence GF-suffix*]
using *WeakUntil-ltln.IH*(*2*)[*OF stable-2*, *unfolded suffix-suffix prefix-suffix-subsequence GF-suffix*]
by *blast*+

moreover

have $?X1 \subseteq ?X$
and $?X2 \subseteq ?X$
using *restricted-subformulas-subset′* **by** *force*+

ultimately

obtain *k* **where** $k \geq i$
and *suffix k w* $\models_n$ *af* $\varphi 2$ $(w\ [i \rightarrow k])[?X]_\nu$
and $\bigwedge j.\ j < i \Longrightarrow$ *suffix k w* $\models_n$ *af* $\varphi 1$ $(w\ [j \rightarrow k])[?X]_\nu$
using *GF-advice-sync-less*[*of i w* $\varphi 1$ *?X* $\varphi 2$] *GF-advice-monotone*[*of* - *?X*] **by** *meson*

hence *suffix* (*Suc k*) *w* $\models_n$ *af* $(\varphi 1\ W_n\ \varphi 2)$ (*prefix* (*Suc k*) *w*)$[?X]_\nu$
by (*rule af-subsequence-W-GF-advice*)
hence *?case*
by *blast*
}
moreover
{
assume *Globally-ltln*: *w* $\models_n$ G_n $\varphi 1$

{
fix *j*
have *suffix j w* $\models_n$ $\varphi 1$
using *Globally-ltln* **by** *simp*
then have *suffix j w* $\models_n$ $\varphi 1[\{\psi.\ w \models_n G_n\ (F_n\ \psi)\}]_\nu$
by (*metis stable-1 GF-advice-a1* $\mathcal{GF}$*-suffix* μ*-stable-def* $\mathcal{GF}$*-elim mem-Collect-eq subsetI*)
then have *suffix j w* $\models_n$ $\varphi 1[?X]_\nu$
by (*metis GF-advice-inter restricted-subformulas-W-*μ *le-infI2*)
}

then have *w* $\models_n$ $(\varphi 1\ W_n\ \varphi 2)[?X]_\nu$
by *simp*
then have *?case*
using *GF-advice-af-2* **by** *blast*
}
ultimately
show *?case*
using *WeakUntil-ltln.prems*(*2*) *ltln-weak-to-strong*(*1*) **by** *blast*
next
case (*Release-ltln* $\varphi 1$ $\varphi 2$)

let *?X* $= \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $(\varphi 1\ R_n\ \varphi 2)$
let *?X1* $= \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 1$
let *?X2* $= \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 2$

have *stable-1*: $\bigwedge i.$ μ*-stable* $\varphi 1$ (*suffix i w*)

and *stable-2*: $\bigwedge i.$ *μ-stable φ2 (suffix i w)*
using *μ-stable-subfrmlsn[OF Release-ltln.prems(1)]* **by** *(simp add: μ-stable-suffix)+*

{
assume *Until-ltln*: $w \models_n \varphi 1\ M_n\ \varphi 2$
then obtain *i* **where** $\bigwedge j.\ j \leq i \Longrightarrow$ *suffix j w* $\models_n \varphi 2$ **and** *suffix i w* $\models_n \varphi 1$
by *auto*

then have $\bigwedge j.\ j \leq i \Longrightarrow \exists k.$ *suffix* $(j + k)$ *w* $\models_n$ *af* $\varphi 2$ $(w\ [j \rightarrow k + j])[?X2]_\nu$
and $\exists k.$ *suffix* $(i + k)$ *w* $\models_n$ *af* $\varphi 1$ $(w\ [i \rightarrow k + i])[?X1]_\nu$
using *Release-ltln.IH(1)[OF stable-1, unfolded suffix-suffix prefix-suffix-subsequence GF-suffix]*
using *Release-ltln.IH(2)[OF stable-2, unfolded suffix-suffix prefix-suffix-subsequence GF-suffix]*
by *blast+*

moreover

have $?X1 \subseteq ?X$
and $?X2 \subseteq ?X$
using *restricted-subformulas-subset′* **by** *force+*

ultimately

obtain *k* **where** $k \geq i$
and *suffix k w* $\models_n$ *af* $\varphi 1$ $(w\ [i \rightarrow k])[?X]_\nu$
and $\bigwedge j.\ j \leq i \Longrightarrow$ *suffix k w* $\models_n$ *af* $\varphi 2$ $(w\ [j \rightarrow k])[?X]_\nu$
using *GF-advice-sync-lesseq[of i w φ2 ?X φ1] GF-advice-monotone[of - ?X]* **by** *meson*

hence *suffix (Suc k) w* $\models_n$ *af* $(\varphi 1\ R_n\ \varphi 2)$ *(prefix (Suc k) w)*$[?X]_\nu$
by *(rule af-subsequence-R-GF-advice)*
hence *?case*
by *blast*
}
moreover
{
assume *Globally-ltln*: $w \models_n G_n\ \varphi 2$

{
fix *j*
have *suffix j w* $\models_n \varphi 2$

using *Globally-ltln* **by** *simp*
then have *suffix j w* $\models_n$ $\varphi 2[\{\psi.\ w \models_n G_n\ (F_n\ \psi)\}]_\nu$
by (*metis stable-2 GF-advice-a1* $\mathcal{GF}$*-suffix* μ*-stable-def* $\mathcal{GF}$*-elim mem-Collect-eq subsetI*)
then have *suffix j w* $\models_n$ $\varphi 2[?X]_\nu$
by (*metis GF-advice-inter restricted-subformulas-R-*μ *le-infI2*)
}

then have $w \models_n (\varphi 1\ R_n\ \varphi 2)[?X]_\nu$
by *simp*
then have *?case*
using *GF-advice-af-2* **by** *blast*
}
ultimately
show *?case*
using *Release-ltln.prems*(*2*) *ltln-weak-to-strong*(*3*) **by** *blast*
next
case (*StrongRelease-ltln* $\varphi 1$ $\varphi 2$)

let $?X = \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $(\varphi 1\ M_n\ \varphi 2)$
let $?X1 = \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 1$
let $?X2 = \{\psi.\ w \models_n G_n\ (F_n\ \psi)\} \cap$ *restricted-subformulas* $\varphi 2$

have *stable-1*: $\bigwedge i.$ μ*-stable* $\varphi 1$ (*suffix i w*)
and *stable-2*: $\bigwedge i.$ μ*-stable* $\varphi 2$ (*suffix i w*)
using μ*-stable-subfrmlsn*[*OF StrongRelease-ltln.prems*(*1*)] **by** (*simp add*: μ*-stable-suffix*)+

obtain *i* **where** $\bigwedge j.\ j \leq i \Longrightarrow$ *suffix j w* $\models_n \varphi 2$ **and** *suffix i w* $\models_n \varphi 1$
using *StrongRelease-ltln* **by** *auto*

then have $\bigwedge j.\ j \leq i \Longrightarrow \exists k.$ *suffix* $(j + k)$ $w \models_n$ *af* $\varphi 2$ $(w\ [j \rightarrow k + j])[?X2]_\nu$
and $\exists k.$ *suffix* $(i + k)$ $w \models_n$ *af* $\varphi 1$ $(w\ [i \rightarrow k + i])[?X1]_\nu$
using *StrongRelease-ltln.IH*(*1*)[*OF stable-1*, *unfolded suffix-suffix prefix-suffix-subsequence GF-suffix*]
using *StrongRelease-ltln.IH*(*2*)[*OF stable-2*, *unfolded suffix-suffix prefix-suffix-subsequence GF-suffix*]
by *blast*+

moreover

have $?X1 \subseteq ?X$
and $?X2 \subseteq ?X$

by *auto*

ultimately

obtain k **where** $k \geq i$
and *suffix* k $w \models_n$ *af* $\varphi 1$ $(w\ [i \to k])[?X]_\nu$
and $\bigwedge j.\ j \leq i \Longrightarrow$ *suffix* k $w \models_n$ *af* $\varphi 2$ $(w\ [j \to k])[?X]_\nu$
using *GF-advice-sync-lesseq*[*of* i w $\varphi 2$ $?X$ $\varphi 1$] *GF-advice-monotone*[*of* - $?X$] **by** *meson*

hence *suffix* (*Suc* k) $w \models_n$ *af* $(\varphi 1\ M_n\ \varphi 2)$ (*prefix* (*Suc* k) w)$[?X]_\nu$
by (*rule af-subsequence-M-GF-advice*)

then show *?case*
by *blast*
qed *simp+*

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
assume *?lhs*

obtain i **where** μ*-stable* φ (*suffix* i w)
by (*metis MOST-nat less-Suc-eq suffix-*μ*-stable*)

hence *stable*: μ*-stable* (*af* φ (*prefix* i w)) (*suffix* i w)
by (*simp add*: $\mathcal{F}$*-af* $\mathcal{GF}$*-af* μ*-stable-def*)

let $?\varphi' =$ *af* φ (*prefix* i w)
let $?X' = \mathcal{GF}\ \varphi\ w \cap$ *restricted-subformulas* φ
let $?Y' = \mathcal{FG}\ \varphi\ w \cap$ *restricted-subformulas* φ

have *1*: *suffix* i $w \models_n ?\varphi'$
using ‹*?lhs*› *af-ltl-continuation* **by** *force*

have *2*: $\bigwedge j.$ *af* (*af* φ (*prefix* i w)) (*prefix* j (*suffix* i w)) $=$ *af* φ (*prefix* (i + j) w)
by (*simp add*: *subsequence-append*)

have *3*: $\mathcal{GF}\ \varphi\ w = \mathcal{GF}\ \varphi\ (\mathit{suffix}\ i\ w)$
using $\mathcal{GF}$*-af* $\mathcal{GF}$*-suffix* **by** *blast*

have $\exists j.\ \mathit{suffix}\ (i + j)\ w \models_n \mathit{af}\ (?\varphi')\ (\mathit{prefix}\ j\ (\mathit{suffix}\ i\ w))[?X']_\nu$
using *delay-2*[*OF stable 1*] **unfolding** *suffix-suffix 2 restrict-af 3* **unfolding** $\mathcal{GF}$*-semantics'*
by (*metis* (*no-types*, *lifting*) *GF-advice-inter-subformulas af-subformulas*$_\mu$ *inf-assoc inf-commute*)

hence $\exists i.\ \mathit{suffix}\ i\ w \models_n \mathit{af}\ \varphi\ (\mathit{prefix}\ i\ w)[?X']_\nu$
using *2* **by** *auto*

moreover

{
fix ψ
have $\bigwedge X.\ \psi \in \mathit{restricted\text{-}subformulas}\ \varphi \Longrightarrow \psi[X \cap \mathit{restricted\text{-}subformulas}\ \varphi]_\mu = \psi[X]_\mu$
by (*metis le-supE restricted-subformulas-superset FG-advice-inter inf.coboundedI2*)
hence $\psi \in\ ?X' \Longrightarrow\ w \models_n G_n\ (F_n\ \psi[?Y']_\mu)$
using $\mathcal{GF}$*-implies-GF* **by** *force*
}

moreover

{
fix ψ
have $\bigwedge X.\ \psi \in \mathit{restricted\text{-}subformulas}\ \varphi \Longrightarrow \psi[X \cap \mathit{restricted\text{-}subformulas}\ \varphi]_\nu = \psi[X]_\nu$
by (*metis le-supE restricted-subformulas-superset GF-advice-inter inf.coboundedI2*)
hence $\psi \in\ ?Y' \Longrightarrow w \models_n F_n\ (G_n\ \psi[?X']_\nu)$
using $\mathcal{FG}$*-implies-FG* **by** *force*
}

moreover

have $?X' \subseteq \mathit{subformulas}_\mu\ \varphi \cap \mathit{restricted\text{-}subformulas}\ \varphi$
using $\mathcal{GF}$*-subformulas*$_\mu$ **by** *blast*

moreover

have *?Y'* $\subseteq$ *subformulas*$_\nu$ $\varphi \cap$ *restricted-subformulas* φ
using $\mathcal{FG}$*-subformulas*$_\nu$ **by** *blast*

ultimately show *?rhs*
by *meson*
qed (*insert master-theorem*, *fast*)

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-subformulas* $\varphi \wedge Y \subseteq$ *subformulas*$_\nu$ $\varphi \cap$ *restricted-subformulas* $\varphi\}$
proof *safe*
fix *w*
assume $w \in$ *language-ltln* φ

then have $w \models_n \varphi$
unfolding *language-ltln-def* **by** *simp*

then obtain *X Y* **where**
1: $X \subseteq$ *subformulas*$_\mu$ $\varphi \cap$ *restricted-subformulas* φ
and *2*: $Y \subseteq$ *subformulas*$_\nu$ $\varphi \cap$ *restricted-subformulas* φ
and $\exists\, i.$ *suffix i w* $\models_n$ *af* φ (*prefix i w*)$[X]_\nu$
and $\forall\,\psi \in X.\ w \models_n G_n\ (F_n\ \psi[Y]_\mu)$
and $\forall\,\psi \in Y.\ w \models_n F_n\ (G_n\ \psi[X]_\nu)$
using *master-theorem-restricted* **by** *metis*

then have $w \in L_1\ \varphi\ X$ **and** $w \in L_2\ X\ Y$ **and** $w \in L_3\ X\ Y$
unfolding L_1*-def* L_2*-def* L_3*-def* **by** *simp+*

then show $w \in \bigcup \{L_1\ \varphi\ X \cap L_2\ X\ Y \cap L_3\ X\ Y \mid X\ Y.\ X \subseteq$ *subformulas*$_\mu$ $\varphi \cap$ *restricted-subformulas* $\varphi \wedge Y \subseteq$ *subformulas*$_\nu$ $\varphi \cap$ *restricted-subformulas* $\varphi\}$
using *1 2* **by** *blast*
next
fix *w X Y*
assume $X \subseteq$ *subformulas*$_\mu$ $\varphi \cap$ *restricted-subformulas* φ **and** $Y \subseteq$ *subformulas*$_\nu$ $\varphi \cap$ *restricted-subformulas* φ
and $w \in L_1\ \varphi\ X$ **and** $w \in L_2\ X\ Y$ **and** $w \in L_3\ X\ Y$

then show $w \in$ *language-ltln* φ
unfolding *language-ltln-def* L_1*-def* L_2*-def* L_3*-def*
using *master-theorem-restricted* **by** *blast*
qed

6.3 Definitions with Lists for Code Export

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

lemma *subseqs-subformulas*$_\mu$*-restricted-list*:
$X \subseteq$ *subformulas*$_\mu$ φ ∩ *restricted-subformulas* φ ⟷ (∃ *xs*. X = *set xs* ∧ *xs* ∈ *set* (*subseqs* (*List.filter* (λx. $x \in$ *restricted-subformulas* φ) (*subformulas*$_\mu$*-list* φ))))
by (*metis in-set-subseqs inf-commute inter-set-filter subformulas*$_\mu$*-list-set subset-subseq*)

lemma *subseqs-subformulas*$_\nu$*-restricted-list*:
$Y \subseteq$ *subformulas*$_\nu$ φ ∩ *restricted-subformulas* φ ⟷ (∃ *ys*. Y = *set ys* ∧ *ys* ∈ *set* (*subseqs* (*List.filter* (λx. $x \in$ *restricted-subformulas* φ) (*subformulas*$_\nu$*-list* φ))))
by (*metis in-set-subseqs inf-commute inter-set-filter subformulas*$_\nu$*-list-set subset-subseq*)

lemma *restricted-advice-sets-subformulas*:
$X \subseteq$ *subformulas*$_\mu$ φ ∩ *restricted-subformulas* φ ∧ $Y \subseteq$ *subformulas*$_\nu$ φ ∩ *restricted-subformulas* φ ⟷ (∃ *xs ys*. X = *set xs* ∧ Y = *set ys* ∧ (*xs*, *ys*) ∈ *set* (*restricted-advice-sets* φ))
unfolding *restricted-advice-sets-def set-product subseqs-subformulas*$_\mu$*-restricted-list subseqs-subformulas*$_\nu$*-restricted-list* **by** *blast*

lemma *restricted-advice-sets-not-empty*:
restricted-advice-sets $\varphi \neq$ []
unfolding *restricted-advice-sets-def* **using** *subseqs-not-empty product-not-empty* **by** *blast*

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

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

lemma af_F*-step*:
$\mathit{af}_F\ \varphi\ \psi\ w \sim \mathit{true}_n \Longrightarrow \mathit{af}_F\ \varphi\ \psi\ (w\ @\ [\nu]) = F_n\ \varphi$
by (*induction w rule*: *rev-induct*) (*auto simp*: $\mathit{af\text{-}letter}_F$*-def*)

lemma af_G*-step*:
$\mathit{af}_G\ \varphi\ \psi\ w \sim \mathit{false}_n \Longrightarrow \mathit{af}_G\ \varphi\ \psi\ (w\ @\ [\nu]) = G_n\ \varphi$
by (*induction w rule*: *rev-induct*) (*auto simp*: $\mathit{af\text{-}letter}_G$*-def*)

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

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

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 (*induction w rule*: *rev-induct*)
  case (*snoc x xs*)

  then have $af_F\ \varphi\ \psi\ xs = af\ \psi\ xs$
    by *simp*

  moreover

  have $\neg\ af\ \psi\ xs \sim true_n$
    using *snoc.prems* **by** *blast*

  ultimately show *?case*
    by (*metis af-letter*$_F$*-def foldl-Cons foldl-Nil foldl-append*)
qed *simp*

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 (*induction w rule*: *rev-induct*)
  case (*snoc x xs*)

  then have $af_G\ \varphi\ \psi\ xs = af\ \psi\ xs$
    by *simp*

  moreover

  have $\neg\ af\ \psi\ xs \sim false_n$
    using *snoc.prems* **by** *blast*

  ultimately show *?case*
    by (*metis af-letter*$_G$*-def foldl-Cons foldl-Nil foldl-append*)
qed *simp*

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 (*induction w rule*: *rev-induct*)
  case (*snoc x xs*)

  then have $af_F\ \varphi\ \psi\ xs = af\ \psi\ xs$
    by *simp*

  moreover

have $\neg$ *af*$_F$ φ ψ *xs* $\sim$ *true*$_n$
using *snoc.prems* **by** *blast*

ultimately show *?case*
by (*metis af-letter*$_F$*-def foldl-Cons foldl-Nil foldl-append*)
qed *simp*

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 (*induction w rule*: *rev-induct*)
case (*snoc x xs*)

then have $af_G\ \varphi\ \psi\ xs = af\ \psi\ xs$
by *simp*

moreover

have $\neg$ *af*$_G$ φ ψ *xs* $\sim$ *false*$_n$
using *snoc.prems* **by** *blast*

ultimately show *?case*
by (*metis af-letter*$_G$*-def foldl-Cons foldl-Nil foldl-append*)
qed *simp*

lemma *af*$_F$*-true-implies-af-true*:
$af_F\ \varphi\ \psi\ w \sim true_n \Longrightarrow af\ \psi\ w \sim true_n$
by (*metis af-append-true af-not-true-implies-af-equals-af*$_F$)

lemma *af*$_G$*-false-implies-af-false*:
$af_G\ \varphi\ \psi\ w \sim false_n \Longrightarrow af\ \psi\ w \sim false_n$
by (*metis af-append-false af-not-false-implies-af-equals-af*$_G$)

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 (*induction w rule*: *rev-induct*)
case (*snoc a w*)
then show *?case*
proof (*cases* $af\ \psi\ w \sim true_n$)
case *False*

then have $\bigwedge xs\ ys.\ w = xs\ @\ ys \Longrightarrow \neg\ af\ \psi\ xs \sim true_n$

using *af-append-true* **by** *blast*

then have $af_F\ \varphi\ \psi\ w = af\ \psi\ w$
using *af-not-true-implies-af-equals-af*$_F$ **by** *auto*

then have $af_F\ \varphi\ \psi\ (w\ @\ [a]) = af\ \psi\ (w\ @\ [a])$
by (*simp add*: *False af-letter*$_F$*-def*)

then show *?thesis*
by (*metis append-Nil2 snoc.prems*)
qed (*insert snoc*, *force*)
qed (*simp add*: *const-implies-eq*)

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 (*induction w rule*: *rev-induct*)
case (*snoc a w*)
then show *?case*
proof (*cases* $af\ \psi\ w \sim false_n$)
case *False*

then have $\bigwedge xs\ ys.\ w = xs\ @\ ys \Longrightarrow \neg\ af\ \psi\ xs \sim false_n$
using *af-append-false* **by** *blast*

then have $af_G\ \varphi\ \psi\ w = af\ \psi\ w$
using *af-not-false-implies-af-equals-af*$_G$ **by** *auto*

then have $af_G\ \varphi\ \psi\ (w\ @\ [a]) = af\ \psi\ (w\ @\ [a])$
by (*simp add*: *False af-letter*$_G$*-def*)

then show *?thesis*
by (*metis append-Nil2 snoc.prems*)
qed (*insert snoc*, *force*)
qed (*simp add*: *const-implies-eq*)

lemma *append-take-drop*:
$w = xs\ @\ ys \longleftrightarrow xs = take\ (length\ xs)\ w \wedge ys = drop\ (length\ xs)\ w$
by (*metis append-eq-conv-conj*)

lemma *subsequence-split*:
$(w\ [i \to j]) = xs\ @\ ys \Longrightarrow xs = (w\ [i \to i + length\ xs])$
by (*simp add*: *append-take-drop*) (*metis add-diff-cancel-left′ subsequence-length subsequence-prefix-suffix*)

lemma *subsequence-append-general*:
$i \le k \Longrightarrow k \le j \Longrightarrow (w\ [i \to j]) = (w\ [i \to k])\ @\ (w\ [k \to j])$
by (*metis le-Suc-ex map-append subsequence-def upt-add-eq-append*)

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
fix *i*
from *assms* **obtain** *j* **where** $j > i$ **and** $af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \to j]) \sim true_n$
by *blast*
then have *X*: $af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \to Suc\ j]) = F_n\ \varphi$
using af_F*-step* **by** *auto*

obtain *k* **where** $k > j$ **and** $af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \to k]) \sim true_n$
using *assms* **using** *Suc-le-eq* **by** *blast*
then have $af_F\ \varphi\ (F_n\ \varphi)\ (w\ [Suc\ j \to k]) \sim true_n$
using af_F*-segments*[*OF X*] **by** (*metis le-Suc-ex le-simps*(*3*) *subsequence-append*)
then have $af\ (F_n\ \varphi)\ (w\ [Suc\ j \to k]) \sim true_n$
using af_F*-true-implies-af-true* **by** *blast*
then show $\exists k.\ af\ (F_n\ \varphi)\ (w\ [i \to k]) \sim_L true_n$
by (*metis* (*full-types*) *af-F-prefix-lang-equiv-true eq-implies-lang subsequence-append-general Suc-le-eq* ‹$i < j$› ‹$j < k$› *less-SucI order.order-iff-strict*)
qed

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 (*rule ccontr*)
define *resets* **where** $resets = \{i.\ af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \to i]) \sim true_n\}$
define *m* **where** $m = (if\ resets = \{\}\ then\ 0\ else\ Suc\ (Max\ resets))$

assume $\neg\ (\forall i.\ \exists j{>}i.\ af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \to j]) \sim true_n)$

then have *finite resets*
using *infinite-nat-iff-unbounded resets-def* **by** *force*
then have $resets \ne \{\} \Longrightarrow af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \to Max\ resets]) \sim true_n$

unfolding *resets-def* **using** *Max-in* **by** *blast*
then have *m-reset*: $af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \rightarrow m]) = F_n\ \varphi$
unfolding *m-def* **using** af_F*-step* **by** *auto*

{
fix *i*
assume $i \geq m$

with *m-reset* **have** $\neg\ af_F\ \varphi\ (F_n\ \varphi)\ (w\ [0 \rightarrow i]) \sim true_n$
by (*metis* (*mono-tags*, *lifting*) *Max-ge-iff Suc-n-not-le-n* ‹*finite resets*› *empty-Collect-eq m-def mem-Collect-eq resets-def*)
with *m-reset* **have** $\neg\ af_F\ \varphi\ \ (F_n\ \varphi)(w\ [m \rightarrow i]) \sim true_n$
by (*metis* (*mono-tags*, *opaque-lifting*) ‹$m \leq i$› af_F*-segments bot-nat-def le0 subsequence-append-general*)
}

then have $\nexists j.\ af_F\ \varphi\ (F_n\ \varphi)\ (w\ [m \rightarrow j]) \sim true_n$
by (*metis le-cases subseq-to-smaller*)
then have $\nexists j.\ af\ \ (F_n\ \varphi)\ (w\ [m \rightarrow j]) \sim true_n$
by (*metis af-equiv-true-*af_F*-prefix-true subsequence-split*)
then show *False*
using *assms* **by** *blast*
qed

lemma af_G*-semantics-rtl*:
assumes
$\exists\, i.\ \forall\, j{>}i.\ \neg\ af_G\ \varphi\ (G_n\ \varphi)\ (w\ [0 \rightarrow j]) \sim false_n$
shows
$\exists\, i.\ \forall\, j.\ \neg\ af\ \ (G_n\ \varphi)\ (w\ [i \rightarrow j]) \sim false_n$
proof
define *resets* **where** $resets = \{i.\ af_G\ \varphi\ (G_n\ \varphi)\ (w\ [0 \rightarrow i]) \sim false_n\}$
define *m* **where** $m = (\mathit{if}\ resets = \{\}\ \mathit{then}\ 0\ \mathit{else}\ Suc\ (Max\ resets))$

from *assms* **have** *finite resets*
by (*metis* (*mono-tags*, *lifting*) *infinite-nat-iff-unbounded mem-Collect-eq resets-def*)
then have $resets \neq \{\} \Longrightarrow af_G\ \varphi\ (G_n\ \varphi)\ (w\ [0 \rightarrow Max\ resets]) \sim false_n$
unfolding *resets-def* **using** *Max-in* **by** *blast*
then have *m-reset*: $af_G\ \varphi\ (G_n\ \varphi)\ (w\ [0 \rightarrow m]) = G_n\ \varphi$
unfolding *m-def* **using** af_G*-step* **by** *auto*

{
fix *i*

assume $i \geq m$

with *m-reset* **have** $\neg$ af_G φ (G_n φ) (*w* [*0* $\rightarrow$ *i*]) $\sim$ $false_n$
by (*metis* (*mono-tags*, *lifting*) *Max-ge-iff Suc-n-not-le-n* ‹*finite resets*› *empty-Collect-eq m-def mem-Collect-eq resets-def*)
with *m-reset* **have** $\neg$ af_G φ (G_n φ) (*w* [*m* $\rightarrow$ *i*]) $\sim$ $false_n$
by (*metis* (*mono-tags*, *opaque-lifting*) ‹$m \leq i$› af_G*-segments bot-nat-def le0 subsequence-append-general*)
}

then have $\forall$ *j*. $\neg$ af_G φ (G_n φ) (*w* [*m* $\rightarrow$ *j*]) $\sim$ $false_n$
by (*metis le-cases subseq-to-smaller*)
then show $\forall$ *j*. $\neg$ *af* (G_n φ) (*w* [*m* $\rightarrow$ *j*]) $\sim$ $false_n$
by (*metis af-equiv-false-af*$_G$*-prefix-false subsequence-split*)
qed

lemma af_G*-semantics-ltr*:
assumes
$\exists$ *i*. $\forall$ *j*. $\neg$ *af* (G_n φ) (*w* [*i* $\rightarrow$ *j*]) $\sim_L$ $false_n$
shows
$\exists$ *i*. $\forall$ *j*>*i*. $\neg$ af_G φ (G_n φ) (*w* [*0* $\rightarrow$ *j*]) $\sim$ $false_n$
proof (*rule ccontr*, *auto*)
assume *1*: $\forall$ *i*. $\exists$ *j*>*i*. af_G φ (G_n φ) (*w* [*0* $\rightarrow$ *j*]) $\sim$ $false_n$

{
fix *i*
obtain *j* **where** $j > i$ **and** af_G φ (G_n φ) (*w* [*0* $\rightarrow$ *j*]) $\sim$ $false_n$
using *1* **by** *blast*
then have *X*: af_G φ (G_n φ) (*w* [*0* $\rightarrow$ *Suc j*]) = G_n φ
using af_G*-step* **by** *auto*

obtain *k* **where** $k > j$ **and** af_G φ (G_n φ) (*w* [*0* $\rightarrow$ *k*]) $\sim$ $false_n$
using *1* **using** *Suc-le-eq* **by** *blast*
then have af_G φ (G_n φ) (*w* [*Suc j* $\rightarrow$ *k*]) $\sim$ $false_n$
using af_G*-segments*[*OF X*] **by** (*metis le-Suc-ex le-simps*(*3*) *subsequence-append*)
then have *af* (G_n φ) (*w* [*Suc j* $\rightarrow$ *k*]) $\sim$ $false_n$
using af_G*-false-implies-af-false* **by** *fastforce*
then have *af* (G_n φ) (*w* [*Suc j* $\rightarrow$ *k*]) $\sim_L$ $false_n$
using *eq-implies-lang* **by** *fastforce*
then have *af* (G_n φ) (*w* [*i* $\rightarrow$ *k*]) $\sim_L$ $false_n$
by (*metis* (*full-types*) *af-G-prefix-lang-equiv-false subsequence-append-general Suc-le-eq* ‹$i < j$› ‹$j < k$› *less-SucI order.order-iff-strict*)
then have $\exists$ *j*. *af* (G_n φ) (*w* [*i* $\rightarrow$ *j*]) $\sim_L$ $false_n$

by *fast*
}

then show *False*
using *assms* **by** *blast*
qed

7.2 After Function using GF-advice

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

abbreviation af_ν :: $'a\ \mathit{ltln}\ \mathit{set} \Rightarrow 'a\ \mathit{ltln} \times 'a\ \mathit{ltln} \Rightarrow 'a\ \mathit{set}\ \mathit{list} \Rightarrow 'a\ \mathit{ltln} \times 'a\ \mathit{ltln}$
where
$\mathit{af}_\nu\ X\ p\ w \equiv \mathit{foldl}\ (\mathit{af\text{-}letter}_\nu\ X)\ p\ w$

lemma $\mathit{af\text{-}letter}_\nu\text{-}\mathit{fst}[\mathit{simp}]$:
$\mathit{fst}\ (\mathit{af\text{-}letter}_\nu\ X\ p\ \nu) = \mathit{af\text{-}letter}\ (\mathit{fst}\ p)\ \nu$
by (*simp add*: $\mathit{af\text{-}letter}_\nu\text{-}\mathit{def}$)

lemma $\mathit{af\text{-}letter}_\nu\text{-}\mathit{snd}[\mathit{simp}]$:
$\mathit{snd}\ p \sim \mathit{false}_n \Longrightarrow \mathit{snd}\ (\mathit{af\text{-}letter}_\nu\ X\ p\ \nu) = (\mathit{normalise}\ (\mathit{af\text{-}letter}\ (\mathit{fst}\ p)\ \nu))[X]_\nu$
$\neg\ (\mathit{snd}\ p) \sim \mathit{false}_n \Longrightarrow \mathit{snd}\ (\mathit{af\text{-}letter}_\nu\ X\ p\ \nu) = \mathit{af\text{-}letter}\ (\mathit{snd}\ p)\ \nu$
by (*simp-all add*: $\mathit{af\text{-}letter}_\nu\text{-}\mathit{def}$)

lemma $\mathit{af}_\nu\text{-}\mathit{fst}$:
$\mathit{fst}\ (\mathit{af}_\nu\ X\ p\ w) = \mathit{af}\ (\mathit{fst}\ p)\ w$
by (*induction w rule*: *rev-induct*) *simp+*

lemma $\mathit{af}_\nu\text{-}\mathit{snd}$:
$\neg\ \mathit{af}\ (\mathit{snd}\ p)\ w \sim \mathit{false}_n \Longrightarrow \mathit{snd}\ (\mathit{af}_\nu\ X\ p\ w) = \mathit{af}\ (\mathit{snd}\ p)\ w$
by (*induction w rule*: *rev-induct*) (*simp-all*, *metis* $\mathit{af\text{-}letter}_\nu\text{-}\mathit{snd}(2)$ *af-letter.simps*(*2*) *af-letter-congruent*)

lemma $\mathit{af}_\nu\text{-}\mathit{snd}'$:
$\forall i.\ \neg\ \mathit{snd}\ (\mathit{af}_\nu\ X\ p\ (\mathit{take}\ i\ w)) \sim \mathit{false}_n \Longrightarrow \mathit{snd}\ (\mathit{af}_\nu\ X\ p\ w) = \mathit{af}\ (\mathit{snd}\ p)\ w$
by (*induction w rule*: *rev-induct*) (*simp-all*, *metis* $\mathit{af\text{-}letter}_\nu\text{-}\mathit{snd}(2)$ *diff-is-0-eq*

foldl-Nil le-cases take-all take-eq-Nil)

lemma af_ν*-step*:
snd (af_ν *X* (ξ, ζ) *w*) $\sim$ $false_n$ $\Longrightarrow$ *snd* (af_ν *X* (ξ, ζ) (*w* @ [ν])) = (*normalise* (*af* ξ (*w* @ [ν])))$[X]_\nu$
by (*simp add*: af_ν*-fst*)

lemma af_ν*-segments*:
af_ν *X* (ξ, ζ) *w* = (*af* ξ *w*, (*af* ξ *w*)$[X]_\nu$) $\Longrightarrow$ af_ν *X* (ξ, ζ) (*w* @ *w'*) = af_ν *X* (*af* ξ *w*, (*af* ξ *w*)$[X]_\nu$) *w'*
by (*induction w' rule*: *rev-induct*) *fastforce+*

lemma af_ν*-semantics-ltr*:
assumes
$\exists\, i.$ *suffix i w* $\models_n$ (*af* φ (*prefix i w*))$[X]_\nu$
shows
$\exists\, m.\ \forall\, k \geq m.\ \neg$ *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix* (*Suc k*) *w*)) $\sim$ $false_n$
proof
define *resets* **where** *resets* = {*i*. *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix i w*)) $\sim$ $false_n$}
define *m* **where** *m* = (*if resets* = {} *then 0 else Suc* (*Max resets*))

from *assms* **obtain** *i* **where** *1*: *suffix i w* $\models_n$ (*af* φ (*prefix i w*))$[X]_\nu$
by *blast*

{
fix *j*
assume $i \leq j$ **and** $j \in$ *resets*

let *?*φ = *af* φ (*prefix* (*Suc j*) *w*)

from *1* **have** $\forall\, n.$ *suffix n* (*suffix i w*) $\models_n$ (*normalise* (*af* φ (*prefix i w* @ *prefix n* (*suffix i w*))))$[X]_\nu$
using *normalise-monotonic* **by** (*simp add*: *GF-advice-af*)

then have *suffix* (*Suc j*) *w* $\models_n$ (*normalise* (*af* φ (*prefix* (*Suc j*) *w*)))$[X]_\nu$
by (*metis* (*no-types*) ‹$i \leq j$› *add.right-neutral le-SucI le-Suc-ex subsequence-append subsequence-shift suffix-suffix*)

then have $\forall\, k>j.\ \neg$ *af* ((*normalise* (*af* φ (*prefix* (*Suc j*) *w*)))$[X]_\nu$) (*w* [*Suc j* $\rightarrow$ *k*]) $\sim$ $false_n$
by (*metis ltl-implies-satisfiable-prefix subsequence-prefix-suffix*)

then have $\forall k>j.\ \neg\ snd\ (af_\nu\ X\ (?\varphi,\ (normalise\ ?\varphi)[X]_\nu)\ (w\ [Suc\ j \rightarrow k]))\ \sim false_n$
by (*metis* af_ν*-snd snd-eqD*)

moreover

{
have $fst\ (af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ (prefix\ (Suc\ j)\ w)) = ?\varphi$
by (*simp add*: af_ν*-fst*)

moreover

have $snd\ (af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ (prefix\ j\ w))\ \sim false_n$
using ‹$j \in resets$› *resets-def* **by** *blast*

ultimately have $af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ (prefix\ (Suc\ j)\ w) = (?\varphi,\ (normalise\ ?\varphi)[X]_\nu)$
by (*metis* (*no-types*) af_ν*-step prod.collapse subseq-to-Suc zero-le*)
}

ultimately have $\forall k>j.\ \neg\ snd\ (af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ ((w\ [0 \rightarrow Suc\ j])\ @\ (w\ [Suc\ j \rightarrow k])))\ \sim false_n$
by (*simp add*: af_ν*-segments*)

then have $\forall k>j.\ \neg\ snd\ (af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ (prefix\ k\ w))\ \sim false_n$
by (*metis Suc-leI le0 subsequence-append-general*)

then have $\forall k \in resets.\ k \leq j$
using ‹$j \in resets$› *resets-def le-less-linear* **by** *blast*
}

then have *finite resets*
by (*meson finite-nat-set-iff-bounded-le infinite-nat-iff-unbounded-le*)

then have $resets \neq \{\} \Longrightarrow snd\ (af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ (prefix\ (Max\ resets)\ w))\ \sim false_n$
using *Max-in resets-def* **by** *blast*

then have $\forall k \geq m.\ \neg\ snd\ (af_\nu\ X\ (\varphi,\ (normalise\ \varphi)[X]_\nu)\ (prefix\ k\ w))\ \sim false_n$
by (*metis* (*mono-tags, lifting*) *Max-ge Suc-n-not-le-n* ‹*finite resets*› *empty-Collect-eq m-def mem-Collect-eq order.trans resets-def*)

then show $\forall k \geq m.\ \neg$ *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix* (*Suc k*) *w*)) $\sim false_n$
  using *le-SucI* **by** *blast*
qed

lemma af_ν*-semantics-rtl*:
 assumes
  $\exists n.\ \forall k \geq n.\ \neg$ *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix* (*Suc k*) *w*)) $\sim false_n$
 shows
  $\exists i.$ *suffix i w* $\models_n$ *af* φ (*prefix i w*)$[X]_\nu$
proof −
 define *resets* **where** *resets* = {*i*. *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix i w*)) $\sim false_n$}
 define *m* **where** *m* = (*if resets* = {} *then 0 else Suc* (*Max resets*))

 from *assms* **obtain** *n* **where** $\forall k \geq n.\ \neg$ *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix* (*Suc k*) *w*)) $\sim false_n$
  by *blast*

 then have $\forall k > n.\ \neg$ *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix k w*)) $\sim false_n$
  by (*metis le-SucE lessE less-imp-le-nat*)

 then have *finite resets*
  by (*metis* (*mono-tags*, *lifting*) *infinite-nat-iff-unbounded mem-Collect-eq resets-def*)

 then have $\forall i \geq m.\ \neg$ *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix i w*)) $\sim false_n$
  unfolding *resets-def m-def* **using** *Max-ge not-less-eq-eq* **by** *auto*

 then have $\forall i.\ \neg$ *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) ((*w* [*0* $\rightarrow$ *m*]) @ (*w* [*m* $\rightarrow$ *i*]))) $\sim false_n$
  by (*metis le0 nat-le-linear subseq-to-smaller subsequence-append-general*)

 moreover

 let *?*φ = *af* φ (*prefix m w*)

 have *resets* $\neq$ {} $\Longrightarrow$ *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix* (*Max resets*) *w*)) $\sim false_n$
  using *Max-in* ‹*finite resets*› *resets-def* **by** *blast*

then have *prefix-φ'*: *snd* (af_ν *X* (φ, (*normalise* φ)$[X]_\nu$) (*prefix m w*)) = (*normalise ?φ*)$[X]_\nu$
by (*auto simp*: *GF-advice-congruent m-def* af_ν*-fst*)

ultimately have ∀ *i*. ¬ *snd* (af_ν *X* (*?φ*, (*normalise ?φ*)$[X]_\nu$) (*w* [*m* → *i*])) ∼ $false_n$
by (*metis* af_ν*-fst foldl-append fst-conv prod.collapse*)

then have ∀ *i*. ¬ *af* ((*normalise ?φ*)$[X]_\nu$) (*w* [*m* → *i*]) ∼ $false_n$
by (*metis prefix-φ'* af_ν*-fst* af_ν*-snd' fst-conv prod.collapse subsequence-take*)

then have *suffix m w* $\models_n$ (*normalise* (*af* φ (*prefix m w*)))$[X]_\nu$
by (*metis GF-advice-νLTL(1) satisfiable-prefix-implies-νLTL add.right-neutral subsequence-shift*)

from *this*[*THEN normalise-eventually-equivalent*]
show ∃ *i*. *suffix i w* $\models_n$ *af* φ (*prefix i w*)$[X]_\nu$
by (*metis add.commute af-subsequence-append le-add1 le-add-same-cancel1 prefix-suffix-subsequence suffix-suffix*)
qed

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* φ ⊇ *nested-prop-atoms* (*normalise* φ)
begin

lemma *af-letter*$_F$*-nested-prop-atoms*:
nested-prop-atoms ψ ⊆ *nested-prop-atoms* (F_n φ) ⟹ *nested-prop-atoms* (*af-letter*$_F$ φ ψ ν) ⊆ *nested-prop-atoms* (F_n φ)
by (*induction* ψ) (*auto simp*: *af-letter*$_F$*-def*, *insert af-letter-nested-prop-atoms*, *blast+*)

lemma af_F*-nested-prop-atoms*:
nested-prop-atoms ψ ⊆ *nested-prop-atoms* (F_n φ) ⟹ *nested-prop-atoms* (af_F φ ψ *w*) ⊆ *nested-prop-atoms* (F_n φ)

by (*induction w rule*: *rev-induct*) (*insert af-letter*$_F$*-nested-prop-atoms*, *auto*)

lemma *af-letter*$_F$*-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* (F_n φ) $\Longrightarrow$ *range* (*af-letter*$_F$ φ ψ) $\subseteq$ {ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (F_n φ)}
using *af-letter*$_F$*-nested-prop-atoms* **by** *blast*

lemma *af*$_F$*-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* (F_n φ) $\Longrightarrow$ *range* (*af*$_F$ φ ψ) $\subseteq$ {ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (F_n φ)}
using *af*$_F$*-nested-prop-atoms* **by** *blast*

lemma *af-letter*$_G$*-nested-prop-atoms*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* (G_n φ) $\Longrightarrow$ *nested-prop-atoms* (*af-letter*$_G$ φ ψ ν) $\subseteq$ *nested-prop-atoms* (G_n φ)
by (*induction* ψ) (*auto simp*: *af-letter*$_G$*-def*, *insert af-letter-nested-prop-atoms*, *blast*+)

lemma *af*$_G$*-nested-prop-atoms*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* (G_n φ) $\Longrightarrow$ *nested-prop-atoms* (*af*$_G$ φ ψ *w*) $\subseteq$ *nested-prop-atoms* (G_n φ)
by (*induction w rule*: *rev-induct*) (*insert af-letter*$_G$*-nested-prop-atoms*, *auto*)

lemma *af-letter*$_G$*-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* (G_n φ) $\Longrightarrow$ *range* (*af-letter*$_G$ φ ψ) $\subseteq$ {ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (G_n φ)}
using *af-letter*$_G$*-nested-prop-atoms* **by** *blast*

lemma *af*$_G$*-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* (G_n φ) $\Longrightarrow$ *range* (*af*$_G$ φ ψ) $\subseteq$ {ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* (G_n φ)}
using *af*$_G$*-nested-prop-atoms* **by** *blast*

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*)
by (*simp-all add*: *af-letter-nested-prop-atoms nested-prop-atoms*$_\nu$*-def*)
(*metis GF-advice-nested-prop-atoms*$_\nu$ *af-letter-nested-prop-atoms nested-prop-atoms*$_\nu$*-subset dual-order.trans nested-prop-atoms*$_\nu$*-def normalise-nested-propos*)

lemma *af-letter*$_\nu$*-fst-nested-prop-atoms*:
nested-prop-atoms (*fst* (*af-letter*$_\nu$ *X p* ν)) $\subseteq$ *nested-prop-atoms* (*fst p*)
by (*simp add*: *af-letter-nested-prop-atoms*)

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*))
using *af-letter*$_\nu$*-snd-nested-prop-atoms-helper* **by** *blast*

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*)$\}$
using *af-letter*$_\nu$*-fst-nested-prop-atoms* **by** *force*

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*)$\}$
using *af-letter*$_\nu$*-snd-nested-prop-atoms* **by** *force*

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 –
have *range* (*af-letter*$_\nu$ *X p*) $\subseteq$ *range* (*fst* $\circ$ *af-letter*$_\nu$ *X p*) $\times$ *range* (*snd* $\circ$ *af-letter*$_\nu$ *X p*)
by (*simp add*: *image-subset-iff mem-Times-iff*)

also have ... $\subseteq$ $\{\psi$. *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*)$\}$
using *af-letter*$_\nu$*-fst-range af-letter*$_\nu$*-snd-range* **by** *blast*

finally show *?thesis* **.**
qed

lemma *af*$_\nu$*-fst-nested-prop-atoms*:
nested-prop-atoms (*fst* (*af*$_\nu$ *X p w*)) $\subseteq$ *nested-prop-atoms* (*fst p*)
by (*induction w rule*: *rev-induct*) (*auto*, *insert af-letter-nested-prop-atoms*, *blast*)

lemma *af-letter-nested-prop-atoms*$_\nu$:
nested-prop-atoms$_\nu$ (*af-letter* φ ν) *X* $\subseteq$ *nested-prop-atoms*$_\nu$ φ *X*
by (*induction* φ) (*simp-all add*: *nested-prop-atoms*$_\nu$*-def*, *blast*+)

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*
by (*induction w rule*: *rev-induct*) (*auto*, *insert af-letter-nested-prop-atoms$_\nu$*, *blast*)

lemma *af$_\nu$-fst-range*:
range (*fst* $\circ$ *af$_\nu$* *X p*) $\subseteq$ {ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms* (*fst p*)}
using *af$_\nu$-fst-nested-prop-atoms* **by** *fastforce*

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 (*induction w arbitrary*: *p rule*: *rev-induct*)
case (*snoc x xs*)

let *?p* = *af$_\nu$* *X p xs*

have *nested-prop-atoms* (*snd* (*af$_\nu$* *X p* (*xs* @ [*x*]))) $\subseteq$ (*nested-prop-atoms$_\nu$* (*fst ?p*) *X*) $\cup$ (*nested-prop-atoms* (*snd ?p*))
by (*simp add*: *af-letter$_\nu$-snd-nested-prop-atoms*)

then show *?case*
using *snoc af$_\nu$-fst-nested-prop-atoms$_\nu$* **by** *blast*
qed (*simp add*: *nested-prop-atoms$_\nu$-def*)

lemma *af$_\nu$-snd-range*:
range (*snd* $\circ$ *af$_\nu$* *X p*) $\subseteq$ {ψ. *nested-prop-atoms* ψ $\subseteq$ (*nested-prop-atoms$_\nu$* (*fst p*) *X*) $\cup$ *nested-prop-atoms* (*snd p*)}
using *af$_\nu$-snd-nested-prop-atoms* **by** *fastforce*

lemma *af$_\nu$-range*:
range (*af$_\nu$* *X p*) $\subseteq$ {ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms* (*fst p*)} $\times$ {ψ. *nested-prop-atoms* ψ $\subseteq$ (*nested-prop-atoms$_\nu$* (*fst p*) *X*) $\cup$ *nested-prop-atoms* (*snd p*)}
proof −
have *range* (*af$_\nu$* *X p*) $\subseteq$ *range* (*fst* $\circ$ *af$_\nu$* *X p*) $\times$ *range* (*snd* $\circ$ *af$_\nu$* *X p*)
by (*simp add*: *image-subset-iff mem-Times-iff*)

also have ... $\subseteq$ {ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms* (*fst p*)} $\times$ {ψ. *nested-prop-atoms* ψ $\subseteq$ (*nested-prop-atoms$_\nu$* (*fst p*) *X*) $\cup$ *nested-prop-atoms* (*snd p*)}
using *af$_\nu$-fst-range af$_\nu$-snd-range* **by** *blast*

finally show *?thesis* **.**
qed

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*:
$Rep\ x = Rep\ y \longleftrightarrow x = y$
by (*metis Rep-inverse*)

lemma *Rep-Abs-eq*:
$eq\ x\ (Rep\ (Abs\ x))$
by (*metis Abs-eq Rep-inverse*)

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* ⇒ *'a word* **where**
to-omega ≡ *snth*

definition *to-stream* :: *'a word* ⇒ *'a stream* **where**
to-stream w ≡ *smap w nats*

lemma *to-omega-to-stream*[*simp*]:
to-omega (*to-stream w*) = *w*
unfolding *to-omega-def to-stream-def*
by *auto*

lemma *to-stream-to-omega*[*simp*]:
to-stream (*to-omega s*) = *s*
unfolding *to-omega-def to-stream-def*
by (*metis stream-smap-nats*)

lemma *bij-to-omega*:
bij to-omega
by (*metis bijI' to-omega-to-stream to-stream-to-omega*)

lemma *bij-to-stream*:
bij to-stream
by (*metis bijI' to-omega-to-stream to-stream-to-omega*)

lemma *image-intersection*[*simp*]:
to-omega ‘ (*A* ∩ *B*) = *to-omega* ‘ *A* ∩ *to-omega* ‘ *B*
to-stream ‘ (*C* ∩ *D*) = *to-stream* ‘ *C* ∩ *to-stream* ‘ *D*
by (*simp-all add*: *bij-is-inj bij-to-omega bij-to-stream image-Int*)

lemma *to-stream-snth*[*simp*]:
(*to-stream w*) !! *k* = *w k*
by (*simp add*: *to-stream-def*)

lemma *to-omega-index*[*simp*]:
(*to-omega s*) *k* = *s* !! *k*
by (*metis to-stream-snth to-stream-to-omega*)

lemma *to-stream-stake*[*simp*]:
stake k (*to-stream w*) = *prefix k w*
by (*induction k*) (*simp add*: *to-stream-def*)+

lemma *to-omega-prefix*[*simp*]:
prefix k (*to-omega s*) = *stake k s*
by (*metis to-stream-stake to-stream-to-omega*)

lemma *in-image*[*simp*]:
x ∈ *to-omega* ‘ *X* ⟷ *to-stream x* ∈ *X*
y ∈ *to-stream* ‘ *Y* ⟷ *to-omega y* ∈ *Y*
by *force*+

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)
by (*metis Rep-Abs-eq af-letter-congruent Abs-eq*)

lemma *af-lifted-semantics*:
↑*af* (*Abs* φ) *w* = *Abs* (*af* φ *w*)
by (*induction w rule*: *rev-induct*) (*auto simp*: *Abs-eq*, *insert Rep-Abs-eq af-letter-congruent eq-sym*, *blast*)

lemma *af-lifted-range*:
range (↑*af* (*Abs* φ)) ⊆ {*Abs* ψ | ψ. *nested-prop-atoms* ψ ⊆ *nested-prop-atoms* φ}
using *af-lifted-semantics af-nested-prop-atoms* **by** *blast*

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 $\mathit{af\text{-}letter_G\text{-}lifted}$ :: $'a\ ltln \Rightarrow 'a\ set \Rightarrow 'ltlq \Rightarrow 'ltlq$ (‹↑$\mathit{afletter_G}$›)
where
↑$\mathit{afletter_G}\ \varphi\ \nu\ \psi \equiv \mathit{Abs}\ (\mathit{af\text{-}letter_G}\ \varphi\ (\mathit{Rep}\ \psi)\ \nu)$

lemma $\mathit{af\text{-}letter_F\text{-}lifted\text{-}semantics}$:
↑$\mathit{afletter_F}\ \varphi\ \nu\ (\mathit{Abs}\ \psi) = \mathit{Abs}\ (\mathit{af\text{-}letter_F}\ \varphi\ \psi\ \nu)$
by (*metis* $\mathit{af\text{-}letter_F\text{-}lifted\text{-}def}$ *Rep-inverse* $\mathit{af\text{-}letter_F\text{-}def}$ *af-letter-congruent Abs-eq*)

lemma $\mathit{af\text{-}letter_G\text{-}lifted\text{-}semantics}$:
↑$\mathit{afletter_G}\ \varphi\ \nu\ (\mathit{Abs}\ \psi) = \mathit{Abs}\ (\mathit{af\text{-}letter_G}\ \varphi\ \psi\ \nu)$
by (*metis* $\mathit{af\text{-}letter_G\text{-}lifted\text{-}def}$ *Rep-inverse* $\mathit{af\text{-}letter_G\text{-}def}$ *af-letter-congruent Abs-eq*)

abbreviation $\mathit{af_F\text{-}lifted}$ :: $'a\ ltln \Rightarrow 'ltlq \Rightarrow 'a\ set\ list \Rightarrow 'ltlq$ (‹↑$\mathit{af_F}$›)
where
↑$\mathit{af_F}\ \varphi\ \psi\ w \equiv \mathit{fold}$ (↑$\mathit{afletter_F}\ \varphi)\ w\ \psi$

abbreviation $\mathit{af_G\text{-}lifted}$ :: $'a\ ltln \Rightarrow 'ltlq \Rightarrow 'a\ set\ list \Rightarrow 'ltlq$ (‹↑$\mathit{af_G}$›)
where
↑$\mathit{af_G}\ \varphi\ \psi\ w \equiv \mathit{fold}$ (↑$\mathit{afletter_G}\ \varphi)\ w\ \psi$

lemma $\mathit{af_F\text{-}lifted\text{-}semantics}$:
↑$\mathit{af_F}\ \varphi\ (\mathit{Abs}\ \psi)\ w = \mathit{Abs}\ (\mathit{af_F}\ \varphi\ \psi\ w)$
by (*induction w rule*: *rev-induct*) (*auto simp*: $\mathit{af\text{-}letter_F\text{-}lifted\text{-}semantics}$)

lemma $\mathit{af_G\text{-}lifted\text{-}semantics}$:
↑$\mathit{af_G}\ \varphi\ (\mathit{Abs}\ \psi)\ w = \mathit{Abs}\ (\mathit{af_G}\ \varphi\ \psi\ w)$
by (*induction w rule*: *rev-induct*) (*auto simp*: $\mathit{af\text{-}letter_G\text{-}lifted\text{-}semantics}$)

definition $\mathit{af\text{-}letter_\nu\text{-}lifted}$ :: $'a\ ltln\ set \Rightarrow 'a\ set \Rightarrow 'ltlq \times 'ltlq \Rightarrow 'ltlq \times 'ltlq$ (‹↑$\mathit{afletter_\nu}$›)
where
↑$\mathit{afletter_\nu}\ X\ \nu\ p \equiv$
$(\mathit{Abs}\ (\mathit{fst}\ (\mathit{af\text{-}letter_\nu}\ X\ (\mathit{Rep}\ (\mathit{fst}\ p),\ \mathit{Rep}\ (\mathit{snd}\ p))\ \nu)),$
$\mathit{Abs}\ (\mathit{snd}\ (\mathit{af\text{-}letter_\nu}\ X\ (\mathit{Rep}\ (\mathit{fst}\ p),\ \mathit{Rep}\ (\mathit{snd}\ p))\ \nu)))$

abbreviation $\mathit{af_\nu\text{-}lifted}$ :: $'a\ ltln\ set \Rightarrow 'ltlq \times 'ltlq \Rightarrow 'a\ set\ list \Rightarrow 'ltlq \times 'ltlq$ (‹↑$\mathit{af_\nu}$›)
where
↑$\mathit{af_\nu}\ X\ p\ w \equiv \mathit{fold}$ (↑$\mathit{afletter_\nu}\ X)\ w\ p$

lemma $\mathit{af\text{-}letter_\nu\text{-}lifted\text{-}semantics}$:

$\uparrow afletter_\nu\ X\ \nu\ (Abs\ x,\ Abs\ y) = (Abs\ (fst\ (af\text{-}letter_\nu\ X\ (x,\ y)\ \nu)),\ Abs\ (snd\ (af\text{-}letter_\nu\ X\ (x,\ y)\ \nu)))$
by (*simp add*: *af-letter*$_\nu$*-def af-letter*$_\nu$*-lifted-def*) (*insert GF-advice-congruent Rep-Abs-eq Rep-inverse af-letter-lifted-semantics eq-trans Abs-eq*, *blast*)

lemma *af*$_\nu$*-lifted-semantics*:
$\uparrow af_\nu\ X\ (Abs\ \xi,\ Abs\ \zeta)\ w = (Abs\ (fst\ (af_\nu\ X\ (\xi,\ \zeta)\ w)),\ Abs\ (snd\ (af_\nu\ X\ (\xi,\ \zeta)\ w)))$
apply (*induction w rule*: *rev-induct*)
apply (*auto simp*: *af-letter*$_\nu$*-lifted-def af-letter*$_\nu$*-lifted-semantics af-letter-lifted-semantics*)
by (*metis* (*no-types*, *opaque-lifting*) *af-letter*$_\nu$*-lifted-def af*$_\nu$*-fst af-letter*$_\nu$*-lifted-semantics eq-fst-iff prod.sel*(*2*))

10.2 Büchi automata for basic languages

definition $\mathfrak{A}_\mu$ :: *'a ltln* ⇒ (*'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* **unfolding** *dba.run-alt-def*
by *simp*

lemma *dca-run*:
DCA.run (*dca UNIV p* δ α) (*to-stream w*) *p* **unfolding** *dca.run-alt-def*
by *simp*

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

then have $w \models_n \varphi \longleftrightarrow (\forall n.\ \exists k{\geq}n.\ af\ \varphi\ (w[0 \to k]) \sim true_n)$
by (*meson af-*μ*LTL af-prefix-true le-cases*)

also have ... $\longleftrightarrow$ ($\forall\, n.\ \exists\, k{\geq}n.\ af\ \varphi\ (w[0 \to Suc\ k]) \sim true_n$)
  by (*meson af-prefix-true le-SucI order-refl*)

also have ... $\longleftrightarrow$ *infs* ($\lambda\psi.\ \psi = \uparrow true_n$) (*DBA.trace* ($\mathfrak{A}_\mu\ \varphi$) (*to-stream* w) (*Abs* φ))
  by (*simp add*: *infs-snth* $\mathfrak{A}_\mu$*-def DBA.transition-def af-lifted-semantics Abs-eq*[*symmetric*] *af-letter-lifted-semantics*)

also have ... $\longleftrightarrow$ *to-stream* $w \in$ *DBA.language* ($\mathfrak{A}_\mu\ \varphi$)
  unfolding $\mathfrak{A}_\mu$*-def dba.initial-def dba.accepting-def* **by** (*auto simp*: *dba-run*)

finally show *?thesis*
  by *simp*
qed

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 –
  assume $\varphi \in \mu LTL$

  then have $w \models_n G_n\ (F_n\ \varphi) \longleftrightarrow (\forall\, n.\ \exists\, k.\ af\ (F_n\ \varphi)\ (w[n \to k]) \sim_L true_n)$
    using *ltl-lang-equivalence.af-*μ*LTL-GF* **by** *blast*

  also have ... $\longleftrightarrow$ ($\forall\, n.\ \exists\, k{>}n.\ af_F\ \varphi\ (F_n\ \varphi)\ (w[0 \to k]) \sim true_n$)
    using af_F*-semantics-ltr* af_F*-semantics-rtl*
    using ‹$\varphi \in \mu LTL$› *af-*μ*LTL-GF calculation* **by** *blast*

  also have ... $\longleftrightarrow$ ($\forall\, n.\ \exists\, k{\geq}n.\ af_F\ \varphi\ (F_n\ \varphi)\ (w[0 \to Suc\ k]) \sim true_n$)
    by (*metis less-Suc-eq-le less-imp-Suc-add*)

  also have ... $\longleftrightarrow$ *infs* ($\lambda\psi.\ \psi = \uparrow true_n$) (*DBA.trace* ($\mathfrak{A}_\mu$*-GF* φ) (*to-stream* w) (*Abs* ($F_n\ \varphi$)))
    by (*simp add*: *infs-snth* $\mathfrak{A}_\mu$*-GF-def DBA.transition-def* af_F*-lifted-semantics Abs-eq*[*symmetric*] *af-letter*$_F$*-lifted-semantics*)

  also have ... $\longleftrightarrow$ *to-stream* $w \in$ *DBA.language* ($\mathfrak{A}_\mu$*-GF* φ)
    unfolding $\mathfrak{A}_\mu$*-GF-def dba.initial-def dba.accepting-def* **by** (*auto simp*: *dba-run*)

  finally show *?thesis*
    by *simp*

qed

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

then have $w \models_n \varphi \longleftrightarrow (\exists\, n.\ \forall\, k{\geq}n.\ \neg\ af\ \varphi\ (w[0 \to k]) \sim false_n)$
by (*meson af-νLTL af-prefix-false le-cases order-refl*)

also have ... $\longleftrightarrow (\exists\, n.\ \forall\, k{\geq}n.\ \neg\ af\ \varphi\ (w[0 \to Suc\ k]) \sim false_n)$
by (*meson af-prefix-false le-SucI order-refl*)

also have ... $\longleftrightarrow$ *fins* $(\lambda\psi.\ \psi = \uparrow\! false_n)$ (*DCA.trace* $(\mathfrak{A}_\nu\ \varphi)$ (*to-stream* w) (*Abs* φ))
by (*simp add*: *infs-snth* $\mathfrak{A}_\nu$*-def DBA.transition-def af-lifted-semantics Abs-eq*[*symmetric*] *af-letter-lifted-semantics*)

also have ... $\longleftrightarrow$ *to-stream* $w \in$ *DCA.language* $(\mathfrak{A}_\nu\ \varphi)$
unfolding $\mathfrak{A}_\nu$*-def dca.initial-def dca.rejecting-def* **by** (*auto simp*: *dca-run*)

finally show *?thesis*
by *simp*
qed

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

then have $w \models_n F_n\ (G_n\ \varphi) \longleftrightarrow (\exists\, k.\ \forall\, j.\ \neg\ af\ (G_n\ \varphi)\ (w[k \to j]) \sim_L false_n)$
using *ltl-lang-equivalence.af-νLTL-FG* **by** *blast*

also have ... $\longleftrightarrow (\exists\, n.\ \forall\, k{>}n.\ \neg\ af_G\ \varphi\ (G_n\ \varphi)\ (w[0 \to k]) \sim false_n)$
using af_G*-semantics-ltr* af_G*-semantics-rtl*
using ‹$\varphi \in \nu LTL$› *af-νLTL-FG calculation* **by** *blast*

also have ... $\longleftrightarrow (\exists\, n.\ \forall\, k{\geq}n.\ \neg\ af_G\ \varphi\ (G_n\ \varphi)\ (w[0 \to Suc\ k]) \sim false_n)$
by (*metis less-Suc-eq-le less-imp-Suc-add*)

also have ... $\longleftrightarrow$ *fins* $(\lambda\psi.\ \psi = \uparrow\! false_n)$ (*DCA.trace* $(\mathfrak{A}_\nu\text{-}FG\ \varphi)$ (*to-stream* w) (*Abs* $(G_n\ \varphi)$))

by (*simp add*: *infs-snth* $\mathfrak{A}_\nu$*-FG-def DBA.transition-def af*$_G$*-lifted-semantics Abs-eq*[*symmetric*] *af-letter*$_G$*-lifted-semantics*)

also have ... $\longleftrightarrow$ *to-stream w* $\in$ *DCA.language* ($\mathfrak{A}_\nu$*-FG* φ)
unfolding $\mathfrak{A}_\nu$*-FG-def dca.initial-def dca.rejecting-def* **by** (*auto simp*: *dca-run*)

finally show *?thesis*
by *simp*
qed

10.3 A DCA checking the GF-advice Function

definition $\mathfrak{C}$ :: $'a$ *ltln* $\Rightarrow$ $'a$ *ltln set* $\Rightarrow$ ($'a$ *set*, $'ltlq$ $\times$ $'ltlq$) *dca* **where**
$\mathfrak{C}$ φ *X* = *dca UNIV* (*Abs* φ, *Abs* ((*normalise* φ)[*X*]$_\nu$)) ($\uparrow$*afletter*$_\nu$ *X*) (λp. *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*]$_\nu$)
proof –

have ($\exists$ *i*. *suffix i w* $\models_n$ *af* φ (*prefix i w*)[*X*]$_\nu$)
$\longleftrightarrow$ ($\exists$ *m*. $\forall$ *k*$\geq$*m*. $\neg$ *snd* (*af*$_\nu$ *X* (φ, (*normalise* φ)[*X*]$_\nu$) (*prefix* (*Suc k*) *w*)) $\sim$ *false*$_n$)
using *af*$_\nu$*-semantics-ltr af*$_\nu$*-semantics-rtl* **by** *blast*

also have ... $\longleftrightarrow$ *fins* (λp. *snd p* = $\uparrow$*false*$_n$) (*DCA.trace* ($\mathfrak{C}$ φ *X*) (*to-stream w*) (*Abs* φ, *Abs* ((*normalise* φ)[*X*]$_\nu$)))
by(*simp add*: *infs-snth* $\mathfrak{C}$*-def DCA.transition-def af*$_\nu$*-lifted-semantics af-letter*$_\nu$*-lifted-semantics Abs-eq*)

also have ... $\longleftrightarrow$ *to-stream w* $\in$ *DCA.language* ($\mathfrak{C}$ φ *X*)
by (*simp add*: $\mathfrak{C}$*-def dca.initial-def dca.rejecting-def dca.language-def dca-run*)

finally show *?thesis*
by *blast*
qed

10.4 A DRA for each combination of sets X and Y

lemma *dba-language*:
($\bigwedge$*w*. *to-stream w* $\in$ *DBA.language* $\mathfrak{A}$ $\longleftrightarrow$ *w* $\models_n$ φ) $\Longrightarrow$ *DBA.language* $\mathfrak{A}$

$= \{w.\ \textit{to-omega}\ w \models_n \varphi\}$
by (*metis* (*mono-tags*, *lifting*) *Collect-cong dba.language-def mem-Collect-eq to-stream-to-omega*)

lemma *dca-language*:
$(\bigwedge w.\ \textit{to-stream}\ w \in \textit{DCA.language}\ \mathfrak{A} \longleftrightarrow w \models_n \varphi) \Longrightarrow \textit{DCA.language}\ \mathfrak{A} = \{w.\ \textit{to-omega}\ w \models_n \varphi\}$
by (*metis* (*mono-tags*, *lifting*) *Collect-cong dca.language-def mem-Collect-eq to-stream-to-omega*)

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

lemma $\mathfrak{A}_1$-*language*:
to-omega ‘ $\textit{DCA.language}\ (\mathfrak{A}_1\ \varphi\ xs) = L_1\ \varphi\ (\textit{set}\ xs)$
by (*simp add*: $\mathfrak{A}_1$-*def* L_1-*def set-eq-iff* $\mathfrak{C}$-*language*)

lemma $\mathfrak{A}_1$-*alphabet*:
$\textit{DCA.alphabet}\ (\mathfrak{A}_1\ \varphi\ xs) = \textit{UNIV}$
unfolding $\mathfrak{A}_1$-*def* $\mathfrak{C}$-*def* **by** *simp*

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

lemma $\mathfrak{A}_2$-*language*:
to-omega ‘ $\textit{DBA.language}\ (\mathfrak{A}_2\ xs\ ys) = L_2\ (\textit{set}\ xs)\ (\textit{set}\ ys)$
by (*simp add*: $\mathfrak{A}_2$-*def* L_2-*def set-eq-iff dba-language*[*OF* $\mathfrak{A}_\mu$-*GF-language*[*OF FG-advice-μLTL*(*1*)]])

lemma $\mathfrak{A}_2$-*alphabet*:
$\textit{DBA.alphabet}\ (\mathfrak{A}_2\ xs\ ys) = \textit{UNIV}$
by (*simp add*: $\mathfrak{A}_2$-*def* $\mathfrak{A}_\mu$-*GF-def*)

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

lemma $\mathfrak{A}_3$-*language*:
to-omega ‘ $\textit{DCA.language}\ (\mathfrak{A}_3\ xs\ ys) = L_3\ (\textit{set}\ xs)\ (\textit{set}\ ys)$

by (*simp add*: $\mathfrak{A}_3$-*def* L_3-*def set-eq-iff dca-language*[*OF* $\mathfrak{A}_\nu$-*FG-language*[*OF GF-advice-*ν*LTL*(*1*)]])

lemma $\mathfrak{A}_3$-*alphabet*:
DCA.alphabet ($\mathfrak{A}_3$ *xs ys*) = *UNIV*
by (*simp add*: $\mathfrak{A}_3$-*def* $\mathfrak{A}_\nu$-*FG-def*)

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*))
by (*simp add*: $\mathfrak{A}'$-*def* $\mathfrak{A}_1$-*language* $\mathfrak{A}_2$-*language* $\mathfrak{A}_3$-*language*) *fastforce*

lemma $\mathfrak{A}'$-*alphabet*:
DRA.alphabet ($\mathfrak{A}'$ φ *xs ys*) = *UNIV*
by (*simp add*: $\mathfrak{A}'$-*def* $\mathfrak{A}_1$-*alphabet* $\mathfrak{A}_2$-*alphabet* $\mathfrak{A}_3$-*alphabet*)

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 −
have ($\bigcap$(*a*, *b*)$\in$*set* (*advice-sets* φ). *dra.alphabet* ($\mathfrak{A}'$ φ *a b*)) =
($\bigcup$(*a*, *b*)$\in$*set* (*advice-sets* φ). *dra.alphabet* ($\mathfrak{A}'$ φ *a b*))
using *advice-sets-not-empty* **by** (*simp add*: $\mathfrak{A}'$-*alphabet*)
then have $*$: *DRA.language* (*DRA-Combine.union-list* (*map* (λ(*x*, *y*). $\mathfrak{A}'$ φ *x y*) (*advice-sets* φ))) =
$\bigcup$ (*DRA.language* ‘ *set* (*map* (λ(*x*, *y*). $\mathfrak{A}'$ φ *x y*) (*advice-sets* φ)))
by (*simp add*: *split-def*)
have *language-ltln* $\varphi = \bigcup \{(L_1\ \varphi\ X \cap L_2\ X\ Y \cap L_3\ X\ Y) \mid X\ Y.\ X \subseteq subformulas_\mu\ \varphi \wedge Y \subseteq subformulas_\nu\ \varphi\}$
unfolding *master-theorem-language* **by** *auto*
also have ... = $\bigcup$ {L_1 φ (*set xs*) $\cap$ L_2 (*set xs*) (*set ys*) $\cap$ L_3 (*set xs*) (*set*

ys) | *xs ys*. (*xs*, *ys*) ∈ *set* (*advice-sets* φ)}
  unfolding *advice-sets-subformulas* **by** (*metis* (*no-types*, *lifting*))
 also have ... = ⋃ {*to-omega* ‘ *DRA.language* (𝔄′ φ *xs ys*) | *xs ys*. (*xs*, *ys*) ∈ *set* (*advice-sets* φ)}
  by (*simp add*: 𝔄′*-language*)
 finally show *?thesis*
  using ∗ **by** (*auto simp add*: *ltl-to-dra-def*)
qed

lemma *ltl-to-dra-alphabet*:
 alphabet (*ltl-to-dra* φ) = *UNIV*
 by (*auto simp*: *ltl-to-dra-def* 𝔄′*-alphabet*)

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

The following constant uses the *?w* $\models_n$ *?φ* = (∃ *X*⊆*subformulas*$_\mu$ *?φ* ∩ *restricted-subformulas ?φ*. ∃ *Y*⊆*subformulas*$_\nu$ *?φ* ∩ *restricted-subformulas ?φ*. ∃ *i*. *suffix i ?w* $\models_n$ *af ?φ* (*prefix i ?w*)$[X]_\nu$ ∧ (∀ ψ∈*X*. *?w* $\models_n$ G_n (F_n $\psi[Y]_\mu$)) ∧ (∀ ψ∈*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*). 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 −
 have (⋂(*a*, *b*)∈*set* (*restricted-advice-sets* φ). *dra.alphabet* (𝔄′ φ *a b*)) =
  (⋃(*a*, *b*)∈*set* (*restricted-advice-sets* φ). *dra.alphabet* (𝔄′ φ *a b*))
  using *restricted-advice-sets-not-empty* **by** (*simp add*: 𝔄′*-alphabet*)
 then have ∗: *DRA.language* (*DRA-Combine.union-list* (*map* (λ(*x*, *y*). 𝔄′ φ *x y*) (*restricted-advice-sets* φ))) =
  ⋃ (*DRA.language* ‘ *set* (*map* (λ(*x*, *y*). 𝔄′ φ *x y*) (*restricted-advice-sets* φ)))
  by (*simp add*: *split-def*)
 have *language-ltln* φ = ⋃ {(L_1 φ *X* ∩ L_2 *X Y* ∩ L_3 *X Y*) | *X Y*. *X* ⊆ *subformulas*$_\mu$ φ ∩ *restricted-subformulas* φ ∧ *Y* ⊆ *subformulas*$_\nu$ φ ∩ *restricted-subformulas* φ}
  unfolding *master-theorem-restricted-language* **by** *auto*
 also have ... = ⋃ {L_1 φ (*set xs*) ∩ L_2 (*set xs*) (*set ys*) ∩ L_3 (*set xs*) (*set ys*) | *xs ys*. (*xs*, *ys*) ∈ *set* (*restricted-advice-sets* φ)}
  unfolding *restricted-advice-sets-subformulas* **by** (*metis* (*no-types*, *lifting*))
 also have ... = ⋃ {*to-omega* ‘ *DRA.language* (𝔄′ φ *xs ys*) | *xs ys*. (*xs*,

ys) ∈ *set* (*restricted-advice-sets* φ)}
  by (*simp add*: 𝔄′*-language*)
 finally show *?thesis*
  using ∗ **by** (*auto simp add*: *ltl-to-dra-restricted-def*)
qed

lemma *ltl-to-dra-restricted-alphabet*:
 alphabet (*ltl-to-dra-restricted* φ) = *UNIV*
 by (*auto simp*: *ltl-to-dra-restricted-def* 𝔄′*-alphabet*)

10.7 A DRA for *L* φ 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 𝔄 Σ = *dra* Σ (*initial* 𝔄) (*transition* 𝔄) (*condition* 𝔄)

lemma *dra-set-alphabet-language*:
 Σ ⊆ *alphabet* 𝔄 ⟹ *language* (*dra-set-alphabet* 𝔄 Σ) = *language* 𝔄 ∩ {*s*.
sset s ⊆ Σ}
 by (*auto simp add*: *dra-set-alphabet-def dra.language-def set-eq-iff dra.run-alt-def streams-iff-sset*)

lemma *dra-set-alphabet-alphabet*[*simp*]:
 alphabet (*dra-set-alphabet* 𝔄 Σ) = Σ
 unfolding *dra-set-alphabet-def* **by** *simp*

lemma *dra-set-alphabet-nodes*:
 Σ ⊆ *alphabet* 𝔄 ⟹ *DRA.nodes* (*dra-set-alphabet* 𝔄 Σ) ⊆ *DRA.nodes* 𝔄
 unfolding *dra-set-alphabet-def dra.nodes-alt-def dra.reachable-alt-def dra.path-alt-def*
 by *auto*

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 φ ⊆ *Ap*
 shows
  to-omega ‘ *language* (*ltl-to-dra-alphabet* φ *Ap*) = *language-ltln* φ ∩ {*w*.
range w ⊆ *Pow Ap*}

proof −
  have *1*: *Pow Ap* ⊆ *alphabet* (*ltl-to-dra-restricted* φ)
    unfolding *ltl-to-dra-restricted-alphabet* **by** *simp*

  show *?thesis*
    unfolding *ltl-to-dra-alphabet-def dra-set-alphabet-language*[*OF 1*]
    by (*simp add*: *ltl-to-dra-restricted-language sset-range*) *force*
qed

lemma *ltl-to-dra-alphabet-alphabet*[*simp*]:
  alphabet (*ltl-to-dra-alphabet* φ *Ap*) = *Pow Ap*
  unfolding *ltl-to-dra-alphabet-def* **by** *simp*

lemma *ltl-to-dra-alphabet-nodes*:
  DRA.nodes (*ltl-to-dra-alphabet* φ *Ap*) ⊆ *DRA.nodes* (*ltl-to-dra-restricted* φ)
  unfolding *ltl-to-dra-alphabet-def*
  by (*rule dra-set-alphabet-nodes*) (*simp add*: *ltl-to-dra-restricted-alphabet*)

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* ⇒ *nat*
  by (*induction xs*) (*auto simp*: *le-funD mult-le-mono*)

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* ⇒ *nat*
  by (*induction xs*) (*auto simp*: *mult-le-mono*)

lemma *card-insert-Suc*:
  card (*insert x S*) ≤ *Suc* (*card S*)
  by (*metis Suc-n-not-le-n card.infinite card-insert-if finite-insert linear*)

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

by (*metis leD linorder-le-less-linear nat-power-less-imp-less neq0-conv power-eq-0-iff*)

lemma *const-less-power*:
$n < x \,\hat{}\, n$ **if** $x > 1$
using *that* **by** (*induction n*) (*auto simp*: *less-trans-Suc*)

lemma *floorlog-le-const*:
floorlog x n $\leq n$
by (*induction n*) (*simp add*: *floorlog-eq-zero-iff*, *metis Suc-lessI floorlog-le-iff le-SucI power-inject-exp*)

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

lemma af_F*-lifted-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi) \Longrightarrow$ *range* $(\uparrow af_F\ \varphi$ (*Abs* $\psi)) \subseteq \{$*Abs* $\psi \mid \psi$. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi)\}$
using af_F*-lifted-semantics* af_F*-nested-prop-atoms* **by** *blast*

lemma af_G*-lifted-range*:
nested-prop-atoms $\psi \subseteq$ *nested-prop-atoms* $(G_n\ \varphi) \Longrightarrow$ *range* $(\uparrow af_G\ \varphi$ (*Abs* $\psi)) \subseteq \{$*Abs* $\psi \mid \psi$. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $(G_n\ \varphi)\}$
using af_G*-lifted-semantics* af_G*-nested-prop-atoms* **by** *blast*

lemma $\mathfrak{A}_\mu$*-nodes*:
DBA.nodes $(\mathfrak{A}_\mu\ \varphi) \subseteq \{$*Abs* $\psi \mid \psi$. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $\varphi\}$
unfolding $\mathfrak{A}_\mu$*-def*
using *af-lifted-semantics af-nested-prop-atoms* **by** *fastforce*

lemma $\mathfrak{A}_\mu$*-GF-nodes*:
DBA.nodes $(\mathfrak{A}_\mu$*-GF* $\varphi) \subseteq \{$*Abs* $\psi \mid \psi$. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $(F_n\ \varphi)\}$
unfolding $\mathfrak{A}_\mu$*-GF-def dba.nodes-alt-def dba.reachable-alt-def*

using af_F*-nested-prop-atoms*[*of* F_n φ] **by** (*auto simp*: af_F*-lifted-semantics*)

lemma $\mathfrak{A}_\nu$*-nodes*:
DCA.nodes ($\mathfrak{A}_\nu$ φ) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms* φ}
unfolding $\mathfrak{A}_\nu$*-def*
using *af-lifted-semantics af-nested-prop-atoms* **by** *fastforce*

lemma $\mathfrak{A}_\nu$*-FG-nodes*:
DCA.nodes ($\mathfrak{A}_\nu$*-FG* φ) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms* (G_n φ)}
unfolding $\mathfrak{A}_\nu$*-FG-def dca.nodes-alt-def dca.reachable-alt-def*
using af_G*-nested-prop-atoms*[*of* G_n φ] **by** (*auto simp*: af_G*-lifted-semantics*)

lemma $\mathfrak{C}$*-nodes-normalise*:
DCA.nodes ($\mathfrak{C}$ φ X) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms* φ} $\times$ {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms*$_\nu$ (*normalise* φ) X}
unfolding $\mathfrak{C}$*-def dca.nodes-alt-def dca.reachable-alt-def*
apply (*auto simp add*: af_ν*-lifted-semantics af-letter*$_\nu$*-lifted-semantics*)
using af_ν*-fst-nested-prop-atoms* **apply** *force*
by (*metis GF-advice-nested-prop-atoms*$_\nu$ af_ν*-snd-nested-prop-atoms Abs-eq* af_ν*-lifted-semantics fst-conv normalise-eq snd-conv sup.absorb-iff1*)

lemma $\mathfrak{C}$*-nodes*:
DCA.nodes ($\mathfrak{C}$ φ X) $\subseteq$ {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms* φ} $\times$ {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms*$_\nu$ φ X}
unfolding $\mathfrak{C}$*-def dca.nodes-alt-def dca.reachable-alt-def*
apply (*auto simp add*: af_ν*-lifted-semantics af-letter*$_\nu$*-lifted-semantics*)
using af_ν*-fst-nested-prop-atoms* **apply** *force*
by (*metis* (*no-types*, *opaque-lifting*) *GF-advice-nested-prop-atoms*$_\nu$ af_ν*-snd-nested-prop-atoms fst-eqD nested-prop-atoms*$_\nu$*-subset normalise-nested-propos order-refl order-trans snd-eqD sup.order-iff*)

lemma *equiv-subset*:
{*Abs* ψ |ψ. *nested-prop-atoms* ψ $\subseteq$ P} $\subseteq$ {*Abs* ψ |ψ. *prop-atoms* ψ $\subseteq$ P}
using *prop-atoms-nested-prop-atoms* **by** *blast*

lemma *equiv-finite'*:
finite P $\Longrightarrow$ *finite* {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ P}
using *equiv-finite equiv-subset finite-subset* **by** *fast*

lemma *equiv-card′*:
finite P $\Longrightarrow$ *card* $\{$*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq P\} \leq$ *2* ^ *2* ^ *card* *P*
by (*metis* (*mono-tags*, *lifting*) *equiv-card equiv-subset equiv-finite card-mono le-trans*)

lemma *nested-prop-atoms-finite*:
finite $\{$*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $\varphi\}$
using *equiv-finite′*[*OF Equivalence-Relations.nested-prop-atoms-finite*] **.**

lemma *nested-prop-atoms-card*:
card $\{$*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms* $\varphi\} \leq$ *2* ^ *2* ^ *card* (*nested-prop-atoms* φ)
using *equiv-card′*[*OF Equivalence-Relations.nested-prop-atoms-finite*] **.**

lemma *nested-prop-atoms*$_\nu$*-finite*:
finite $\{$*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms*$_\nu$ φ $X\}$
using *equiv-finite′*[*OF nested-prop-atoms*$_\nu$*-finite*] **by** *fast*

lemma *nested-prop-atoms*$_\nu$*-card*:
card $\{$*Abs* ψ | ψ. *nested-prop-atoms* $\psi \subseteq$ *nested-prop-atoms*$_\nu$ φ $X\} \leq$ *2* ^ *2* ^ *card* (*nested-prop-atoms* φ) (**is** *?lhs* $\leq$ *?rhs*)
proof –
have *finite* $\{$*Abs* ψ | ψ. *prop-atoms* $\psi \subseteq$ *nested-prop-atoms*$_\nu$ φ $X\}$
by (*simp add*: *nested-prop-atoms*$_\nu$*-finite Advice.nested-prop-atoms*$_\nu$*-finite equiv-finite*)

then have *?lhs* $\leq$ *card* $\{$*Abs* ψ | ψ. *prop-atoms* $\psi \subseteq$ (*nested-prop-atoms*$_\nu$ φ X)$\}$
using *card-mono equiv-subset* **by** *blast*

also have ... $\leq$ *2* ^ *2* ^ *card* (*nested-prop-atoms*$_\nu$ φ X)
using *equiv-card*[*OF Advice.nested-prop-atoms*$_\nu$*-finite*] **by** *fast*

also have ... $\leq$ *?rhs*
using *nested-prop-atoms*$_\nu$*-card* **by** *auto*

finally show *?thesis* **.**
qed

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

using *finite-subset*[*OF* $\mathfrak{A}_\mu$*-GF-nodes nested-prop-atoms-finite*] **.**

lemma $\mathfrak{A}_\nu$*-FG-nodes-finite*:
finite (*DCA.nodes* ($\mathfrak{A}_\nu$*-FG* φ))
using *finite-subset*[*OF* $\mathfrak{A}_\nu$*-FG-nodes nested-prop-atoms-finite*] **.**

lemma $\mathfrak{A}_\mu$*-GF-nodes-card*:
card (*DBA.nodes* ($\mathfrak{A}_\mu$*-GF* φ)) $\leq$ *2* ^ *2* ^ *card* (*nested-prop-atoms* (F_n φ))
using *le-trans*[*OF card-mono*[*OF nested-prop-atoms-finite* $\mathfrak{A}_\mu$*-GF-nodes*] *nested-prop-atoms-card*] **.**

lemma $\mathfrak{A}_\nu$*-FG-nodes-card*:
card (*DCA.nodes* ($\mathfrak{A}_\nu$*-FG* φ)) $\leq$ *2* ^ *2* ^ *card* (*nested-prop-atoms* (G_n φ))
using *le-trans*[*OF card-mono*[*OF nested-prop-atoms-finite* $\mathfrak{A}_\nu$*-FG-nodes*] *nested-prop-atoms-card*] **.**

lemma $\mathfrak{A}_2$*-nodes-finite-helper*:
list-all (*finite* $\circ$ *DBA.nodes*) (*map* ($\lambda\psi$. $\mathfrak{A}_\mu$*-GF* (ψ[*set ys*]$_\mu$)) *xs*)
by (*auto simp*: *list.pred-map list-all-iff* $\mathfrak{A}_\mu$*-GF-nodes-finite*)

lemma $\mathfrak{A}_2$*-nodes-finite*:
finite (*DBA.nodes* ($\mathfrak{A}_2$ *xs ys*))
unfolding $\mathfrak{A}_2$*-def* **using** *DBA-Combine.intersect-list-nodes-finite* $\mathfrak{A}_2$*-nodes-finite-helper*
.

lemma $\mathfrak{A}_3$*-nodes-finite-helper*:
list-all (*finite* $\circ$ *DCA.nodes*) (*map* ($\lambda\psi$. $\mathfrak{A}_\nu$*-FG* (ψ[*set xs*]$_\nu$)) *ys*)
by (*auto simp*: *list.pred-map list-all-iff* $\mathfrak{A}_\nu$*-FG-nodes-finite*)

lemma $\mathfrak{A}_3$*-nodes-finite*:
finite (*DCA.nodes* ($\mathfrak{A}_3$ *xs ys*))
unfolding $\mathfrak{A}_3$*-def* **using** *DCA-Combine.intersect-list-nodes-finite* $\mathfrak{A}_3$*-nodes-finite-helper*
.

lemma $\mathfrak{A}_2$*-nodes-card*:
assumes
length xs $\leq$ *n*
and
$\bigwedge\psi$. $\psi \in$ *set xs* $\Longrightarrow$ *card* (*nested-prop-atoms* ψ) $\leq$ *n*
shows
card (*DBA.nodes* ($\mathfrak{A}_2$ *xs ys*)) $\leq$ *2* ^ *2* ^ (*n* + *floorlog 2 n* + *2*)
proof –
have *1*: $\bigwedge\psi$. $\psi \in$ *set xs* $\Longrightarrow$ *card* (*nested-prop-atoms* (F_n ψ[*set ys*]$_\mu$)) $\leq$

Suc n
proof −
fix ψ
assume $\psi \in$ *set xs*

have *card* (*nested-prop-atoms* (F_n (ψ[*set ys*]$_\mu$)))
≤ *Suc* (*card* (*nested-prop-atoms* (ψ[*set ys*]$_\mu$)))
by (*simp add*: *card-insert-Suc*)

also have ... ≤ *Suc* (*card* (*nested-prop-atoms* ψ))
by (*simp add*: *FG-advice-nested-prop-atoms-card*)

also have ... ≤ *Suc n*
by (*simp add*: *assms*(*2*) ‹$\psi \in$ *set xs*›)

finally show *card* (*nested-prop-atoms* (F_n (ψ[*set ys*]$_\mu$))) ≤ *Suc n* **.**
qed

have ($\prod\psi$←*xs*. *card* (*DBA.nodes* ($\mathfrak{A}_\mu$-*GF* (ψ[*set ys*]$_\mu$))))
≤ ($\prod\psi$←*xs*. *2* ^ *2* ^ *card* (*nested-prop-atoms* (F_n (ψ[*set ys*]$_\mu$))))
by (*rule list-prod-mono*) (*insert* $\mathfrak{A}_\mu$-*GF-nodes-card le-fun-def*, *blast*)

also have ... ≤ (*2* ^ *2* ^ *Suc n*) ^ *length xs*
by (*rule list-prod-const*) (*metis 1 Suc-leI nat-power-le-imp-le nat-power-eq-Suc-0-iff neq0-conv pos2 zero-less-power*)

also have ... ≤ (*2* ^ *2* ^ *Suc n*) ^ *n*
using *assms*(*1*) *nat-power-le-imp-le* **by** *fastforce*

also have ... = *2* ^ (*n* ∗ *2* ^ *Suc n*)
by (*metis Groups.mult-ac*(*2*) *power-mult*)

also have ... ≤ *2* ^ (*2* ^ *floorlog 2 n* ∗ *2* ^ *Suc n*)
by (*cases n* = *0*) (*auto simp*: *floorlog-bounds less-imp-le-nat*)

also have ... = *2* ^ *2* ^ (*Suc n* + *floorlog 2 n*)
by (*simp add*: *power-add*)

finally have *2*: ($\prod\psi$←*xs*. *card* (*DBA.nodes* ($\mathfrak{A}_\mu$-*GF* (ψ[*set ys*]$_\mu$)))) ≤ *2* ^ *2* ^ (*Suc n* + *floorlog 2 n*) **.**

have *card* (*DBA.nodes* ($\mathfrak{A}_2$ *xs ys*)) ≤ *max 1* (*length xs*) ∗ ($\prod\psi$←*xs*. *card* (*DBA.nodes* ($\mathfrak{A}_\mu$-*GF* (ψ[*set ys*]$_\mu$))))
using *DBA-Combine.intersect-list-nodes-card*[*OF* $\mathfrak{A}_2$-*nodes-finite-helper*]

by (*auto simp*: $\mathfrak{A}_2$-*def comp-def*)

also have ... $\leq$ *max 1 n* $*$ *2* ^ *2* ^ (*Suc n* + *floorlog 2 n*)
using *assms*(*1*) *2* **by** (*simp add*: *mult-le-mono*)

also have ... $\leq$ *2* ^ (*floorlog 2 n*) $*$ *2* ^ *2* ^ (*Suc n* + *floorlog 2 n*)
by (*cases n* = *0*) (*auto simp*: *floorlog-bounds less-imp-le-nat*)

also have ... = *2* ^ (*floorlog 2 n* + *2* ^ (*Suc n* + *floorlog 2 n*))
by (*simp add*: *power-add*)

also have ... $\leq$ *2* ^ (*n* + *2* ^ (*Suc n* + *floorlog 2 n*))
by (*simp add*: *floorlog-le-const*)

also have ... $\leq$ *2* ^ *2* ^ (*n* + *floorlog 2 n* + *2*)
by *simp* (*metis const-less-power Suc-1 add-Suc-right add-leE lessI less-imp-le-nat power-Suc*)

finally show *?thesis* **.**
qed

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 −
have *1*: $\bigwedge\psi$. $\psi \in$ *set ys* $\Longrightarrow$ *card* (*DCA.nodes* ($\mathfrak{A}_\nu$-*FG* (ψ[*set xs*]$_\nu$))) $\leq$ *2* ^ *2* ^ *Suc n*
proof −
fix ψ
assume $\psi \in$ *set ys*

have *card* (*nested-prop-atoms* (G_n ψ[*set xs*]$_\nu$))
$\leq$ *Suc* (*card* (*nested-prop-atoms* (ψ[*set xs*]$_\nu$)))
by (*simp add*: *card-insert-Suc*)

also have ... $\leq$ *Suc* (*card* (*nested-prop-atoms* ψ))
by (*simp add*: *GF-advice-nested-prop-atoms-card*)

also have ... $\leq$ *Suc n*

by (*simp add*: *assms*(*2*) ‹$\psi \in$ *set ys*›)

finally have *2*: *card* (*nested-prop-atoms* (G_n ψ[*set xs*]$_\nu$)) $\leq$ *Suc n* **.**

then show *?thesis* ψ
by (*intro le-trans*[*OF* $\mathfrak{A}_\nu$*-FG-nodes-card*]) (*meson one-le-numeral power-increasing*)
qed

have *card* (*DCA.nodes* ($\mathfrak{A}_3$ *xs ys*)) $\leq$ ($\prod \psi \leftarrow$*ys*. *card* (*DCA.nodes* ($\mathfrak{A}_\nu$*-FG* (ψ[*set xs*]$_\nu$))))
unfolding $\mathfrak{A}_3$*-def* **using** *DCA-Combine.intersect-list-nodes-card*[*OF* $\mathfrak{A}_3$*-nodes-finite-helper*]
by (*auto simp*: *comp-def*)

also have ... $\leq$ (*2* ^ *2* ^ *Suc n*) ^ *length ys*
by (*rule list-prod-const*) (*rule 1*)

also have ... $\leq$ (*2* ^ *2* ^ *Suc n*) ^ *n*
by (*simp add*: *assms*(*1*) *power-increasing*)

also have ... $\leq$ *2* ^ (*n* $*$ *2* ^ *Suc n*)
by (*metis le-refl mult.commute power-mult*)

also have ... $\leq$ *2* ^ (*2* ^ *floorlog 2 n* $*$ *2* ^ *Suc n*)
by (*cases* ‹*n* > *0*›) (*simp-all add*: *floorlog-bounds less-imp-le-nat*)

also have ... = *2* ^ *2* ^ (*n* + *floorlog 2 n* + *1*)
by (*simp add*: *power-add*)

finally show *?thesis* **.**
qed

lemma $\mathfrak{A}_1$*-nodes-finite*:
finite (*DCA.nodes* ($\mathfrak{A}_1$ φ *xs*))
unfolding $\mathfrak{A}_1$*-def*
by (*metis* (*no-types*, *lifting*) *finite-subset* $\mathfrak{C}$*-nodes finite-SigmaI nested-prop-atoms*$_\nu$*-finite nested-prop-atoms-finite*)

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 −
let *?fst* = {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms* φ}
let *?snd* = {*Abs* ψ | ψ. *nested-prop-atoms* ψ $\subseteq$ *nested-prop-atoms*$_\nu$ φ (*set* *xs*)}

have *1*: *card* (*nested-prop-atoms* φ) $\leq$ *n*
by (*meson card-mono*[*OF subfrmlsn-finite nested-prop-atoms-subfrmlsn*] *assms le-trans*)

have *card* (*DCA.nodes* ($\mathfrak{A}_1$ φ *xs*)) $\leq$ *card* (*?fst* $\times$ *?snd*)
unfolding $\mathfrak{A}_1$-*def*
by (*rule card-mono*) (*simp-all add*: $\mathfrak{C}$-*nodes nested-prop-atoms*$_\nu$-*finite* *nested-prop-atoms-finite*)

also have ... = *card ?fst* ∗ *card ?snd*
using *nested-prop-atoms*$_\nu$-*finite card-cartesian-product* **by** *blast*

also have ... $\leq$ *2* ^ *2* ^ *card* (*nested-prop-atoms* φ) ∗ *2* ^ *2* ^ *card* (*nested-prop-atoms* φ)
using *nested-prop-atoms*$_\nu$-*card nested-prop-atoms-card mult-le-mono* **by** *blast*

also have ... = *2* ^ *2* ^ (*card* (*nested-prop-atoms* φ) + *1*)
by (*simp add*: *semiring-normalization-rules*(*36*))

also have ... $\leq$ *2* ^ *2* ^ (*n* + *1*)
using *assms 1* **by** *simp*

finally show *?thesis* **.**
qed

lemma $\mathfrak{A}'$-*nodes-finite*:
finite (*DRA.nodes* ($\mathfrak{A}'$ φ *xs ys*))
unfolding $\mathfrak{A}'$-*def*
using *intersect-nodes-finite intersect-bc-nodes-finite*
using $\mathfrak{A}_1$-*nodes-finite* $\mathfrak{A}_2$-*nodes-finite* $\mathfrak{A}_3$-*nodes-finite*
by *fast*

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 −
have *n* + *1* $\leq$ *n* + *floorlog 2 n* + *2*
by *auto*

then have *1*: (*2*::*nat*) ^ (*n* + *1*) $\leq$ *2* ^ (*n* + *floorlog 2 n* + *2*)
using *one-le-numeral power-increasing* **by** *blast*

have *card* (*DRA.nodes* ($\mathfrak{A}'$ φ *xs ys*)) $\leq$ *card* (*DCA.nodes* ($\mathfrak{A}_1$ φ *xs*)) $*$ *card* (*DBA.nodes* ($\mathfrak{A}_2$ *xs ys*)) $*$ *card* (*DCA.nodes* ($\mathfrak{A}_3$ *xs ys*)) (**is** *?lhs* $\leq$ *?rhs*)
proof (*unfold* $\mathfrak{A}'$*-def*)
have *card* (*DBA.nodes* ($\mathfrak{A}_2$ *xs ys*)) $*$ *card* (*DCA.nodes* (*DCA-Combine.intersect* ($\mathfrak{A}_1$ φ *xs*) ($\mathfrak{A}_3$ *xs ys*))) $\leq$ *?rhs*
by (*simp add*: *intersect-nodes-card*[*OF* $\mathfrak{A}_1$*-nodes-finite* $\mathfrak{A}_3$*-nodes-finite*])
then show *card* (*DRA.nodes* (*intersect-bc* ($\mathfrak{A}_2$ *xs ys*) (*DCA-Combine.intersect* ($\mathfrak{A}_1$ φ *xs*) ($\mathfrak{A}_3$ *xs ys*)))) $\leq$ *?rhs*
by (*meson intersect-bc-nodes-card*[*OF* $\mathfrak{A}_2$*-nodes-finite intersect-nodes-finite*[*OF* $\mathfrak{A}_1$*-nodes-finite* $\mathfrak{A}_3$*-nodes-finite*]] *basic-trans-rules*(*23*))
qed

also have ... $\leq$ *2* ^ *2* ^ (*n* + *1*) $*$ *2* ^ *2* ^ (*n* + *floorlog 2 n* + *2*) $*$ *2* ^ *2* ^ (*n* + *floorlog 2 n* + *1*)
using $\mathfrak{A}_1$*-nodes-card*[*OF assms*(*5*)] $\mathfrak{A}_2$*-nodes-card*[*OF assms*(*1*,*2*)] $\mathfrak{A}_3$*-nodes-card*[*OF assms*(*3*,*4*)]
by (*metis mult-le-mono*)

also have ... = *2* ^ (*2* ^ (*n* + *1*) + *2* ^ (*n* + *floorlog 2 n* + *2*) + *2* ^ (*n* + *floorlog 2 n* + *1*))
by (*metis power-add*)

also have ... $\leq$ *2* ^ (*4* $*$ *2* ^ (*n* + *floorlog 2 n* + *2*))
using *1* **by** *auto*

finally show *?thesis*
by (*simp add*: *numeral.simps*(*2*) *power-add*)
qed

lemma *subformula-nested-prop-atoms-subfrmlsn*:
$\psi \in$ *subfrmlsn* $\varphi \Longrightarrow$ *nested-prop-atoms* $\psi \subseteq$ *subfrmlsn* φ
using *nested-prop-atoms-subfrmlsn subfrmlsn-subset* **by** *blast*

lemma *ltl-to-dra-nodes-finite*:
finite (*DRA.nodes* (*ltl-to-dra* φ))
unfolding *ltl-to-dra-def*
apply (*rule DRA-Combine.union-list-nodes-finite*)
apply (*simp add*: *split-def* $\mathfrak{A}'$*-alphabet advice-sets-not-empty*)
apply (*simp add*: *list.pred-set split-def* $\mathfrak{A}'$*-nodes-finite*)
done

lemma *ltl-to-dra-restricted-nodes-finite*:
finite (*DRA.nodes* (*ltl-to-dra-restricted* φ))
unfolding *ltl-to-dra-restricted-def*
apply (*rule DRA-Combine.union-list-nodes-finite*)
apply (*simp add*: *split-def* $\mathfrak{A}'$*-alphabet advice-sets-not-empty*)
apply (*simp add*: *list.pred-set split-def* $\mathfrak{A}'$*-nodes-finite*)
done

lemma *ltl-to-dra-alphabet-nodes-finite*:
finite (*DRA.nodes* (*ltl-to-dra-alphabet* φ *AP*))
using *ltl-to-dra-alphabet-nodes ltl-to-dra-restricted-nodes-finite finite-subset*
by *fast*

lemma *ltl-to-dra-nodes-card*:
assumes
card (*subfrmlsn* φ) $\leq n$
shows
card (*DRA.nodes* (*ltl-to-dra* φ)) $\leq 2 \mathbin{\hat{}} 2 \mathbin{\hat{}} (2 * n +$ *floorlog* $2\ n + 4)$
proof –
let *?map* = *map* ($\lambda(x, y)$. $\mathfrak{A}'$ φ x y) (*advice-sets* φ)

have *1*: $\bigwedge x$::*nat*. $x > 0 \Longrightarrow x \mathbin{\hat{}}$ *length* (*advice-sets* φ) $\leq x \mathbin{\hat{}} 2 \mathbin{\hat{}}$ *card* (*subfrmlsn* φ)
by (*metis advice-sets-length linorder-not-less nat-power-less-imp-less*)

have *card* (*DRA.nodes* (*ltl-to-dra* φ)) $\leq$ *prod-list* (*map* (*card* $\circ$ *DRA.nodes*)

?map)
  unfolding *ltl-to-dra-def*
  apply (*rule DRA-Combine.union-list-nodes-card*)
  unfolding *list.pred-set* **using** $\mathfrak{A}'$*-nodes-finite* **by** *auto*

also have ... = $(\prod (x, y) \leftarrow$ *advice-sets* φ. *card* (*DRA.nodes* ($\mathfrak{A}'$ φ *x y*)))
  by (*induction advice-sets* φ) (*auto*, *metis* (*no-types*, *lifting*) *comp-apply split-def*)

also have ... $\leq$ (*2* ^ *2* ^ (*n* + *floorlog 2 n* + *4*)) ^ *length* (*advice-sets* φ)
proof (*rule list-prod-const*, *unfold split-def*, *rule* $\mathfrak{A}'$*-nodes-card*)
  show $\bigwedge x.\ x \in$ *set* (*advice-sets* φ) $\Longrightarrow$ *length* (*fst x*) $\leq n$
    using *advice-sets-element-length assms* **by** *fastforce*

  show $\bigwedge x\ \psi.\ [\![x \in$ *set* (*advice-sets* φ); $\psi \in$ *set* (*fst x*)$]\!] \Longrightarrow$ *card* (*nested-prop-atoms* ψ) $\leq n$
    using *advice-sets-element-subfrmlsn*(*1*) *assms subformula-nested-prop-atoms-subfrmlsn* $subformulas_\mu$*-subfrmlsn*
      by (*metis* (*no-types*, *lifting*) *card-mono subfrmlsn-finite subset-iff sup.absorb-iff2 sup.coboundedI1 surjective-pairing*)

  show $\bigwedge x.\ x \in$ *set* (*advice-sets* φ) $\Longrightarrow$ *length* (*snd x*) $\leq n$
    using *advice-sets-element-length assms* **by** *fastforce*

  show $\bigwedge x\ \psi.\ [\![x \in$ *set* (*advice-sets* φ); $\psi \in$ *set* (*snd x*)$]\!] \Longrightarrow$ *card* (*nested-prop-atoms* ψ) $\leq n$
    using *advice-sets-element-subfrmlsn*(*2*) *assms subformula-nested-prop-atoms-subfrmlsn* $subformulas_\nu$*-subfrmlsn*
      by (*metis* (*no-types*, *lifting*) *card-mono subfrmlsn-finite subset-iff sup.absorb-iff2 sup.coboundedI1 surjective-pairing*)
qed (*insert assms*, *blast*)

also have ... $\leq$ (*2* ^ *2* ^ (*n* + *floorlog 2 n* + *4*)) ^ (*2* ^ *card* (*subfrmlsn* φ))
  by (*simp add*: *1*)

also have ... $\leq$ (*2* ^ *2* ^ (*n* + *floorlog 2 n* + *4*)) ^ (*2* ^ *n*)
  by (*simp add*: *assms power-increasing*)

also have ... = *2* ^ (*2* ^ *n* * *2* ^ (*n* + *floorlog 2 n* + *4*))
  by (*simp add*: *ac-simps power-mult* [*symmetric*])

also have ... = *2* ^ *2* ^ (*2* * *n* + *floorlog 2 n* + *4*)
  by (*simp add*: *power-add*) (*simp add*: *mult-2 power-add*)

finally show *?thesis* **.**
qed

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* ^ *2* ^ (*2* $*$ *size* φ + *floorlog 2* (*size* φ) + *4*)
using *ltl-to-dra-nodes-card subfrmlsn-card* **by** *blast*

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* ⇒ '*a list* ⇒ ('*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* $\mathfrak{A}$ ⟹ *language* (*drai-dra* (*dra-to-drai* $\mathfrak{A}$ Σ)) = *language* $\mathfrak{A}$
by (*simp add*: *dra-to-drai-def drai-dra-def*)

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

lemma *dra-to-drai-rel*:
assumes

(Σ, *alphabet A*) ∈ ⟨*Id*⟩ *list-set-rel*
shows
(*dra-to-drai A* Σ, *A*) ∈ ⟨*Id*, *Id*⟩*drai-dra-rel*
proof −
have (*A*, *A*) ∈ ⟨*Id*, *Id*⟩*dra-rel*
by *simp*

then have (*dra-to-drai A* Σ, *dra* (*alphabet A*) (*initial A*) (*transition A*) (*condition A*)) ∈ ⟨*Id*, *Id*⟩*drai-dra-rel*
unfolding *dra-to-drai-def* **using** *assms* **by** *parametricity*

then show *?thesis*
by *simp*
qed

lemma *draei-language-rel*:
fixes
A :: (′*label*, ′*state* :: *hashable*) *dra*
assumes
(Σ, *alphabet A*) ∈ ⟨*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 −
have (*dra-to-drai A* Σ, *A*) ∈ ⟨*Id*, *Id*⟩*drai-dra-rel*
using *dra-to-drai-rel assms* **by** *fast*

then have (*drai-to-draei hms* (*dra-to-drai A* Σ), *to-draei A*) ∈ ⟨*Id-on* (*dra.alphabet A*), *rel* (*dra-to-drai A* Σ) *A* (=) *bounded-hashcode-nat hms*⟩ *draei-dra-rel*
unfolding *drai-to-draei-def*
using *to-draei-impl-refine*[*unfolded autoref-tag-defs*]
by *parametricity* (*simp-all add*: *assms is-bounded-hashcode-def bounded-hashcode-nat-bounds*)

then have (*DRA.language* ((*drae-dra* ∘ *draei-drae*) (*drai-to-draei hms* (*dra-to-drai A* Σ))), *DRA.language* (*id* (*to-draei A*))) ∈ ⟨⟨*Id-on* (*dra.alphabet A*)⟩ *stream-rel*⟩ *set-rel*
by *parametricity*

then show *?thesis*

by (*simp add*: *to-draei-def*)
qed

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
by (*induction* φ) *simp-all*

lemma *atoms-ltlc-list-distinct*:
distinct (*atoms-ltlc-list* φ)
by (*induction* φ) *simp-all*

definition *ltl-alphabet* :: $'a$ *list* $\Rightarrow$ $'a$ *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*
unfolding *ltl-to-dra-alphabet-alphabet ltl-alphabet-def*
by (*simp add*: *list-set-rel-def in-br-conv subseqs-powset distinct-set-subseqs*)

lemma *ltlc-to-ltln-simplify-atoms*:
atoms-ltln (*simplify Slow* (*ltlc-to-ltln* φ)) ⊆ *atoms-ltlc* φ
using *ltlc-to-ltln-atoms simplify-atoms* **by** *fast*

lemma *valid-def-hm-size*:
is-valid-def-hm-size TYPE(*'state*) (*Suc* (*size* φ)) **for** φ :: *'a ltln*
unfolding *is-valid-def-hm-size-def*
using *ltln.size-neq* **by** *auto*

theorem *final-correctness*:
to-omega ' *language* (*drae-dra* (*draei-drae* (*ltlc-to-draei* φ)))
= *language-ltlc* φ ∩ {*w*. *range w* ⊆ *Pow* (*atoms-ltlc* φ)}
unfolding *ltlc-to-draei-def ltln-to-draei-def*
unfolding *draei-language-rel*[*OF ltl-to-dra-alphabet-rel*[*OF atoms-ltlc-list-distinct*] *ltl-to-dra-alphabet-nodes-finite valid-def-hm-size*]
unfolding *atoms-ltlc-list-set*
unfolding *ltl-to-dra-alphabet-language*[*OF ltlc-to-ltln-simplify-atoms*]
unfolding *ltlc-to-ltln-atoms language-ltln-def language-ltlc-def ltlc-to-ltln-semantics simplify-correct* **..**

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* :: $'a\ ltln \Rightarrow 'a\ ltln$
where
  $Unf\ (\varphi\ U_n\ \psi) = ((\varphi\ U_n\ \psi)\ and_n\ Unf\ \varphi)\ or_n\ Unf\ \psi$
| $Unf\ (\varphi\ W_n\ \psi) = ((\varphi\ W_n\ \psi)\ and_n\ Unf\ \varphi)\ or_n\ Unf\ \psi$
| $Unf\ (\varphi\ M_n\ \psi) = ((\varphi\ M_n\ \psi)\ or_n\ Unf\ \varphi)\ and_n\ Unf\ \psi$
| $Unf\ (\varphi\ R_n\ \psi) = ((\varphi\ R_n\ \psi)\ or_n\ Unf\ \varphi)\ and_n\ Unf\ \psi$
| $Unf\ (\varphi\ and_n\ \psi) = Unf\ \varphi\ and_n\ Unf\ \psi$
| $Unf\ (\varphi\ or_n\ \psi) = Unf\ \varphi\ or_n\ Unf\ \psi$
| $Unf\ \varphi = \varphi$

lemma *Unf-sound*:
  $w \models_n Unf\ \varphi \longleftrightarrow w \models_n \varphi$
proof (*induction* φ *arbitrary*: *w*)
  case (*Until-ltln* $\varphi 1$ $\varphi 2$)
  then show *?case*
    by (*simp*, *metis less-linear not-less0 suffix-0*)
next
  case (*Release-ltln* $\varphi 1$ $\varphi 2$)
  then show *?case*
    by (*simp*, *metis less-linear not-less0 suffix-0*)
next
  case (*WeakUntil-ltln* $\varphi 1$ $\varphi 2$)
  then show *?case*
    by (*simp*, *metis bot.extremum-unique bot-nat-def less-eq-nat.simps*(*1*) *suffix-0*)
qed (*simp-all*, *fastforce*)

lemma *Unf-lang-equiv*:
  $\varphi \sim_L Unf\ \varphi$
  by (*simp add*: *Unf-sound ltl-lang-equiv-def*)

lemma *Unf-idem*:
  $Unf\ (Unf\ \varphi) \sim_P Unf\ \varphi$
  by (*induction* φ) (*auto simp*: *ltl-prop-equiv-def*)

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 $(\sim_Q)$
by (*metis ltl-prop-equiv-equivp ltl-prop-unfold-equiv-def equivpI equivp-def reflpI sympI transpI*)

lemma *unfolding-prop-unfold-idem*:
$Unf\ \varphi \sim_Q \varphi$
unfolding *ltl-prop-unfold-equiv-def* **by** (*rule Unf-idem*)

lemma *unfolding-is-subst*: $Unf\ \varphi = subst\ \varphi\ (\lambda\psi.\ Some\ (Unf\ \psi))$
by (*induction* φ) *auto*

lemma *ltl-prop-equiv-implies-ltl-prop-unfold-equiv*:
$\varphi \sim_P \psi \Longrightarrow \varphi \sim_Q \psi$
by (*metis ltl-prop-unfold-equiv-def unfolding-is-subst subst-respects-ltl-prop-entailment*(*2*))

lemma *ltl-prop-unfold-equiv-implies-ltl-lang-equiv*:
$\varphi \sim_Q \psi \Longrightarrow \varphi \sim_L \psi$
by (*metis ltl-prop-equiv-implies-ltl-lang-equiv ltl-lang-equiv-def Unf-sound ltl-prop-unfold-equiv-def*)

lemma *ltl-prop-unfold-equiv-gt-and-lt*:
$(\sim_C) \le (\sim_Q)$ $(\sim_P) \le (\sim_Q)$ $(\sim_Q) \le (\sim_L)$
using *ltl-prop-equiv-implies-ltl-prop-unfold-equiv ltl-prop-equivalence.ge-const-equiv ltl-prop-unfold-equiv-implies-ltl-lang-equiv*
by *blast+*

quotient-type $'a\ ltln_Q = 'a\ ltln\ /\ (\sim_Q)$
by (*rule ltl-prop-unfold-equiv-equivp*)

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

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

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

instance

by (*standard*; *simp add*: eq_Q $ltln_Q$*-eq-test.rep-eq*, *metis Quotient-ltln*$_Q$ *Quotient-rel-rep*)

end

lemma *af-letter-unfolding*:
af-letter (*Unf* φ) $\nu \sim_P$ *af-letter* φ ν
by (*induction* φ) (*simp-all add*: *ltl-prop-equiv-def*, *blast*+)

lemma *af-letter-prop-unfold-congruent*:
assumes $\varphi \sim_Q \psi$
shows *af-letter* φ $\nu \sim_Q$ *af-letter* ψ ν
proof −
have *Unf* $\varphi \sim_P$ *Unf* ψ
using *assms* **by** (*simp add*: *ltl-prop-unfold-equiv-def ltl-prop-equiv-def*)
then have *af-letter* (*Unf* φ) $\nu \sim_P$ *af-letter* (*Unf* ψ) ν
by (*simp add*: *prop-af-congruent.af-letter-congruent*)
then have *af-letter* φ $\nu \sim_P$ *af-letter* ψ ν
by (*metis af-letter-unfolding ltl-prop-equivalence.eq-sym ltl-prop-equivalence.eq-trans*)
then show *af-letter* φ $\nu \sim_Q$ *af-letter* ψ ν
by (*rule ltl-prop-equiv-implies-ltl-prop-unfold-equiv*)
qed

lemma *GF-advice-prop-unfold-congruent*:
assumes $\varphi \sim_Q \psi$
shows (*Unf* φ)$[X]_\nu \sim_Q$ (*Unf* ψ)$[X]_\nu$
proof −
have *Unf* $\varphi \sim_P$ *Unf* ψ
using *assms*
by (*simp add*: *ltl-prop-unfold-equiv-def ltl-prop-equiv-def*)
then have (*Unf* φ)$[X]_\nu \sim_P$ (*Unf* ψ)$[X]_\nu$
by (*simp add*: *GF-advice-prop-congruent*(*2*))
then show (*Unf* φ)$[X]_\nu \sim_Q$ (*Unf* ψ)$[X]_\nu$
by (*simp add*: *ltl-prop-equiv-implies-ltl-prop-unfold-equiv*)
qed

interpretation *prop-unfold-equivalence*: *ltl-equivalence* ($\sim_Q$)
by *unfold-locales* (*metis ltl-prop-unfold-equiv-equivp ltl-prop-unfold-equiv-gt-and-lt*)+

interpretation *af-congruent* ($\sim_Q$)
by *unfold-locales* (*rule af-letter-prop-unfold-congruent*)

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

```
proof (induction φ)
  case (Until-ltln φ1 φ2)
  then show ?case
    by (cases (φ1 U_n φ2) ∈ X) force+
next
  case (Release-ltln φ1 φ2)
  then show ?case
    using ltln-expand-Release by auto
next
  case (WeakUntil-ltln φ1 φ2)
  then show ?case
    using ltln-expand-WeakUntil by auto
next
  case (StrongRelease-ltln φ1 φ2)
  then show ?case
    by (cases (φ1 M_n φ2) ∈ X) force+
qed auto

lemma unfolding-next-step-equivalent:
  w ⊨_n (Unf φ)[X]_ν ⟹ suffix 1 w ⊨_n (af-letter φ (w 0))[X]_ν
proof (induction φ)
  case (Next-ltln φ)
  then show ?case
    unfolding Unf.simps by (metis GF-advice-af-letter build-split)
next
  case (Until-ltln φ1 φ2)
  then show ?case
    unfolding Unf.simps
     by (metis GF-advice.simps(2) GF-advice.simps(3) GF-advice-af-letter
af-letter.simps(8) build-split semantics-ltln.simps(5) semantics-ltln.simps(6))
next
  case (Release-ltln φ1 φ2)
  then show ?case
    unfolding Unf.simps
     by (metis GF-advice.simps(2) GF-advice.simps(3) GF-advice-af-letter
One-nat-def af-letter.simps(9) build-first semantics-ltln.simps(5) semantics-ltln.simps(6))
next
  case (WeakUntil-ltln φ1 φ2)
  then show ?case
    unfolding Unf.simps
     by (metis GF-advice.simps(2) GF-advice.simps(3) GF-advice-af-letter
af-letter.simps(10) build-split semantics-ltln.simps(5) semantics-ltln.simps(6))
next
  case (StrongRelease-ltln φ1 φ2)
```

then show *?case*
unfolding *Unf.simps*
by (*metis GF-advice.simps(2) GF-advice.simps(3) GF-advice-af-letter af-letter.simps(11) build-split semantics-ltln.simps(5) semantics-ltln.simps(6)*)
qed *auto*

lemma *nested-prop-atoms-Unf*:
nested-prop-atoms (*Unf* φ) $\subseteq$ *nested-prop-atoms* φ
by (*induction* φ) *auto*

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 –
obtain Y **where** $Y \subseteq X$ **and** *finite* Y **and** *Y-def*: f ` $X$ = $f$ ` Y
using *assms* **by** (*meson finite-subset-image subset-refl*)
moreover
{
fix x
assume $x \in X$
then have $g\ x \in g$ ` Y
by (*metis* (*no-types*, *opaque-lifting*) ‹$x \in X$› *assms*(*1*) *Y-def image-iff*)
}
then have g ` $X$ = $g$ ` Y
using *assms* ‹$Y \subseteq X$› **by** *blast*
ultimately
show *finite* (g ` X)
by *simp*

from ‹*finite* Y› **have** *card* (f ` $Y$) $\geq$ *card* ($g$ ` Y)
proof (*induction* Y *rule*: *finite-induct*)
case (*insert* x F)

then have *1*: *finite* (g ` $F$) **and** *2*: *finite* ($f$ ` F)
by *simp-all*

have $f\ x \in f$ ` $F \Longrightarrow g\ x \in g$ ` F
using *assms*(*1*) **by** *blast*

then show *?case*
using *insert* **by** (*simp add*: *card-insert-if*[*OF 1*] *card-insert-if*[*OF 2*])

qed *simp*

then show *card* $(f \text{ ` } X) \geq$ *card* $(g \text{ ` } X)$
by (*simp add*: *Y-def* ‹g ` $X = g$ ` Y›)
qed

lemma *abs-ltln*$_P$*-implies-abs-ltln*$_Q$:
abs-ltln$_P$ φ = *abs-ltln*$_P$ $\psi \longrightarrow$ *abs-ltln*$_Q$ φ = *abs-ltln*$_Q$ ψ
by (*simp add*: *ltl-prop-equiv-implies-ltl-prop-unfold-equiv ltln*$_P$*.abs-eq-iff ltln*$_Q$*.abs-eq-iff*)

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* $\psi \subseteq P$}
using *prop-unfold-equiv-helper*(*1*)[*OF prop-equiv-finite*[*unfolded image-Collect*[*symmetric*]]]
unfolding *image-Collect*[*symmetric*] **.**

lemma *prop-unfold-equiv-card*:
finite $P \Longrightarrow$ *card* {*abs-ltln*$_Q$ ψ |ψ. *prop-atoms* $\psi \subseteq P$} $\leq$ *2* ^ *2* ^ *card P*
using *prop-unfold-equiv-helper*(*2*)[*OF prop-equiv-finite*[*unfolded image-Collect*[*symmetric*]]] *prop-equiv-card*
unfolding *image-Collect*[*symmetric*]
by *fastforce*

lemma *Unf-eventually-equivalent*:
$w \models_n$ *Unf* $\varphi[X]_\nu \Longrightarrow \exists\, i.$ *suffix i w* $\models_n$ *af* φ (*prefix i w*)$[X]_\nu$
by (*metis* (*full-types*) *One-nat-def foldl.simps*(*1*) *foldl.simps*(*2*) *subsequence-singleton unfolding-next-step-equivalent*)

interpretation *prop-unfold-GF-advice-compatible*: *GF-advice-congruent* ($\sim_Q$) *Unf*
by *unfold-locales* (*simp-all add*: *unfolding-prop-unfold-idem prop-unfold-equivalence.eq-sym unfolding-monotonic Unf-eventually-equivalent GF-advice-prop-unfold-congruent*)

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
  from *def-hashmap-size*[**where** *?′a* = *′a*]
  show *1* < *def-hashmap-size TYPE*(*′a set*)
    by (*simp add*: *def-hashmap-size-set-def*)
qed

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
  from *def-hashmap-size*[**where** *?′a* = *′a*]
  show *1* < *def-hashmap-size TYPE*(*′a fset*)
    by (*simp add*: *def-hashmap-size-fset-def*)

qed

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
  from *def-hashmap-size*[**where** *?'a = 'a*]
  show *1 < def-hashmap-size TYPE*(*'a ltln*$_P$)
    by (*simp add*: *def-hashmap-size-ltln*$_P$*-def def-hashmap-size-ltln-def*)
qed

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
  from *def-hashmap-size*[**where** *?'a = 'a*]
  show *1 < def-hashmap-size TYPE*(*'a ltln*$_Q$)
    by (*simp add*: *def-hashmap-size-ltln*$_Q$*-def def-hashmap-size-ltln-def*)
qed

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-letter*$_\nu$-*lifted*$_P$ = *ltl-to-dra*$_P$.*af-letter*$_\nu$-*lifted*
and $\mathfrak{C}_P$ = *ltl-to-dra*$_P$.$\mathfrak{C}$
and *af-letter*$_{\nu P}$ = *ltl-to-dra*$_P$.*af-letter*$_\nu$
and *ltln-to-draei*$_P$ = *ltl-to-dra*$_P$.*ltln-to-draei*
and *ltlc-to-draei*$_P$ = *ltl-to-dra*$_P$.*ltlc-to-draei*
by *unfold-locales* (*meson Quotient-abs-rep Quotient-ltln*$_P$, *simp-all add*: *Quotient-abs-rep Quotient-ltln*$_P$ *ltln*$_P$.*abs-eq-iff prop-equiv-card prop-equiv-finite*)

thm *ltl-to-dra*$_P$.*ltl-to-dra-language*
thm *ltl-to-dra*$_P$.*ltl-to-dra-size*
thm *ltl-to-dra*$_P$.*final-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-to-dra*$_Q$: *dra-implementation* $(\sim_Q)$ *Unf rep-ltln*$_Q$ *abs-ltln*$_Q$
defines *ltl-to-dra*$_Q$ = *ltl-to-dra*$_Q$.*ltl-to-dra*
and *ltl-to-dra-restricted*$_Q$ = *ltl-to-dra*$_Q$.*ltl-to-dra-restricted*
and *ltl-to-dra-alphabet*$_Q$ = *ltl-to-dra*$_Q$.*ltl-to-dra-alphabet*
and $\mathfrak{A}'_Q$ = *ltl-to-dra*$_Q$.$\mathfrak{A}'$
and $\mathfrak{A}_{1Q}$ = *ltl-to-dra*$_Q$.$\mathfrak{A}_1$
and $\mathfrak{A}_{2Q}$ = *ltl-to-dra*$_Q$.$\mathfrak{A}_2$
and $\mathfrak{A}_{3Q}$ = *ltl-to-dra*$_Q$.$\mathfrak{A}_3$
and $\mathfrak{A}_\nu$-FG_Q = *ltl-to-dra*$_Q$.$\mathfrak{A}_\nu$-FG
and $\mathfrak{A}_\mu$-GF_Q = *ltl-to-dra*$_Q$.$\mathfrak{A}_\mu$-GF
and *af-letter*$_{GQ}$ = *ltl-to-dra*$_Q$.*af-letter*$_G$
and *af-letter*$_{FQ}$ = *ltl-to-dra*$_Q$.*af-letter*$_F$
and *af-letter*$_G$-*lifted*$_Q$ = *ltl-to-dra*$_Q$.*af-letter*$_G$-*lifted*

and $\mathit{af\text{-}letter}_F\mathit{\text{-}lifted}_Q = \mathit{ltl\text{-}to\text{-}dra}_Q.\mathit{af\text{-}letter}_F\mathit{\text{-}lifted}$
and $\mathit{af\text{-}letter}_\nu\mathit{\text{-}lifted}_Q = \mathit{ltl\text{-}to\text{-}dra}_Q.\mathit{af\text{-}letter}_\nu\mathit{\text{-}lifted}$
and $\mathfrak{C}_Q = \mathit{ltl\text{-}to\text{-}dra}_Q.\mathfrak{C}$
and $\mathit{af\text{-}letter}_{\nu Q} = \mathit{ltl\text{-}to\text{-}dra}_Q.\mathit{af\text{-}letter}_\nu$
and $\mathit{ltln\text{-}to\text{-}draei}_Q = \mathit{ltl\text{-}to\text{-}dra}_Q.\mathit{ltln\text{-}to\text{-}draei}$
and $\mathit{ltlc\text{-}to\text{-}draei}_Q = \mathit{ltl\text{-}to\text{-}dra}_Q.\mathit{ltlc\text{-}to\text{-}draei}$
by *unfold-locales* (*meson Quotient-abs-rep* $\mathit{Quotient\text{-}ltln}_Q$, *simp-all add*: *Quotient-abs-rep* $\mathit{Quotient\text{-}ltln}_Q$ $\mathit{ltln}_Q.\mathit{abs\text{-}eq\text{-}iff}$ *nested-prop-atoms-Unf prop-unfold-equiv-finite prop-unfold-equiv-card*)

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

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 = $\mathit{ltlc\text{-}to\text{-}draei}_P$
| *ltlc-to-draei PropUnfold* = $\mathit{ltlc\text{-}to\text{-}draei}_Q$

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*
by *unfold-locales* (*auto simp*: *cube-def*)

lemma [*code*]:
hashcode (*set xs*) = *fold* (*plus o cube o hashcode*) (*remdups xs*) (*uint32-of-nat* (*length* (*remdups xs*)))
by (*simp add*: *fold-set-fold-remdups length-remdups-card-conv*)

lemma [*code*]:
hashcode (*abs-ltln*$_P$ φ) = *hashcode* (*min-dnf* φ)
by *simp*

lemma *min-dnf-rep-abs*[*simp*]:
min-dnf (*Unf* (*rep-ltln*$_Q$ (*abs-ltln*$_Q$ φ))) = *min-dnf* (*Unf* φ)
using *Quotient3-ltln*$_Q$ *ltl-prop-equiv-min-dnf ltl-prop-unfold-equiv-def rep-abs-rsp*
by *fastforce*

lemma [*code*]:
hashcode (*abs-ltln*$_Q$ φ) = *hashcode* (*min-dnf* (*Unf* φ))
by *simp*

14.2 LTL to DRA

declare *ltl-to-dra*$_P$.*af-letter*$_F$-*lifted-semantics* [*code*]
declare *ltl-to-dra*$_P$.*af-letter*$_G$-*lifted-semantics* [*code*]
declare *ltl-to-dra*$_P$.*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.