Stuttering Equivalence and Stuttering Invariance

Stephan Merz
Inria Nancy & LORIA
Villers-lès-Nancy, France
September 13, 2023

Two $\omega$-sequences are stuttering equivalent if they differ only by finite repetitions of elements. For example, the two sequences

$$(abccca)\omega \quad \text{and} \quad (aaaabc)\omega$$

are stuttering equivalent, whereas

$$(abac)\omega \quad \text{and} \quad (aaabcc)\omega$$

are not. Stuttering equivalence is a fundamental concept in the theory of concurrent and distributed systems. Notably, Lamport [1] argues that refinement notions for such systems should be insensitive to finite stuttering. Peled and Wilke [2] showed that all PLTL (propositional linear-time temporal logic) properties that are insensitive to stuttering equivalence can be expressed without the next-time operator. Stuttering equivalence is also important for certain verification techniques such as partial-order reduction for model checking.

We formalize stuttering equivalence in Isabelle/HOL. Our development relies on the notion of stuttering sampling functions that may skip blocks of identical sequence elements. We also encode PLTL and prove the theorem due to Peled and Wilke [2].

Contents

1 Utility Lemmas 2

2 Stuttering Sampling Functions 3
   2.1 Definition and elementary properties 3
   2.2 Preservation of properties through stuttering sampling 5
   2.3 Maximal stuttering sampling 6

3 Stuttering Equivalence 13
4 Stuttering Invariant LTL Formulas

4.1 Finite Conjunctions and Disjunctions in PLTL

4.2 Next-Free PLTL Formulas

4.3 Stuttering Invariance of PLTL Without “Next”

4.4 Atoms, Canonical State Sequences, and Characteristic Formulas

4.5 Stuttering Invariant PLTL Formulas Don’t Need Next

4.6 Stutter Invariance for LTL with Syntactic Sugar

theory Samplers
  imports Main HOL−Library.Omega-Words-Fun
begin

1 Utility Lemmas

The following lemmas about strictly monotonic functions could go to the standard library of Isabelle/HOL.

Strongly monotonic functions over the integers grow without bound.

lemma strict-mono-exceeds:
  assumes f: strict-mono (f::nat ⇒ nat)
  shows ∃k. n < f k
proof (induct n)
  from f have f 0 < f 1 by (rule strict-monoD) simp
  hence 0 < f 1 by simp
  thus ∃k. 0 < f k ..
next
  fix n
  assume ∃k. n < f k
  then obtain k where n < f k ..
  hence Suc n ≤ f k by simp
  also from f have f k < f (Suc k) by (rule strict-monoD) simp
  finally show ∃k. Suc n < f k ..
qed

More precisely, any natural number n ≥ f 0 lies in the interval between f k and f (Suc k), for some k.

lemma strict-mono-interval:
  assumes f: strict-mono (f::nat ⇒ nat) and n: f 0 ≤ n
  obtains k where f k ≤ n and n < f (Suc k)
proof –
  from f[THEN strict-mono-exceeds] obtain m where m: n < f m ..
  have m ≠ 0
  proof
    assume m = 0
    with m n show False by simp
  qed
  with m obtain m′ where m′: n < f (Suc m′) by (auto simp: gr0-conv-Suc)
  let ?k = LEAST k. n < f (Suc k)
from \( m' \) have 1: \( n < f \Suc k \) by (rule LeastI)
have \( f ?k \leq n \)
proof (rule ccontr)
  assume \( \neg \thesis \)
  hence \( k: n < f ?k \) by simp
  show \( \False \)
proof (cases \( ?k \))
  case 0 with \( k n \) show \( \False \) by simp
next
  case \Suc with \( k show \( \False \) by (auto dest: Least-le)
qed
qed

with 1 that show \( ?thesis \) by simp
qed

lemma strict-mono-comp:
  assumes \( g: \text{strict-mono} (g::'a::order \Rightarrow 'b::order) \)
  and \( f: \text{strict-mono} (f::'b::order \Rightarrow 'c::order) \)
  shows \( \text{strict-mono} (f \circ g) \)
using assms by (auto simp: strict-mono-def)

\section{Stuttering Sampling Functions}

Given an \( \omega \)-sequence \( \sigma \), a stuttering sampling function is a strictly monotonic function \( f::\text{nat} \Rightarrow \text{nat} \) such that \( f 0 = 0 \) and for all \( i \) and all \( f i \leq k < f (i+1) \), the elements \( \sigma k \) are the same. In other words, \( f \) skips some (but not necessarily all) stuttering steps, but never skips a non-stuttering step. Given such \( \sigma \) and \( f \), the (stuttering-)sampled reduction of \( \sigma \) is the sequence of elements of \( \sigma \) at the indices \( f i \), which can simply be written as \( \sigma \circ f \).

\subsection{Definition and elementary properties}

definition stutter-sampler where
  \( f \) is a stuttering sampling function for \( \sigma \)

stutter-sampler (f::nat \Rightarrow nat) \( \equiv \)
  \( f 0 = 0 \)
  \( \land \text{strict-mono } f \)
  \( \land (\forall \, k n. f k < n \land n < f (\Suc k) \rightarrow \sigma n = \sigma (f k)) \)

lemma stutter-sampler-0: stutter-sampler \( f \sigma \Rightarrow f 0 = 0 \)
  by (simp add: stutter-sampler-def)

lemma stutter-sampler-mono: stutter-sampler \( f \sigma \Rightarrow \text{strict-mono } f \)
  by (simp add: stutter-sampler-def)

lemma stutter-sampler-between:
  assumes \( f: \text{stutter-sampler } f \sigma \)
  and lo: \( f k \leq n \) and hi: \( n < f (\Suc k) \)
shows $\sigma \ n = \sigma \ (f \ k)$
using assms by (auto simp: stutter-sampler-def less-le)

lemma stutter-sampler-interval:
assumes $f$: stutter-sampler $f \ \sigma$
obtains $k$ where $f \ k \leq \ n$ and $n < f \ (Suc \ k)$
using $f$[THEN stutter-sampler-mono] proof (rule strict-mono-interval)
from $f$ show $f \ 0 \leq \ n$ by (simp add: stutter-sampler-0)
qed

The identity function is a stuttering sampling function for any $\sigma$.

lemma id-stutter-sampler [iff]: stutter-sampler $id \ \sigma$
by (auto simp: stutter-sampler-def strict-mono-def)

Stuttering sampling functions compose, sort of.

lemma stutter-sampler-comp:
assumes $f$: stutter-sampler $f \ \sigma$
and $g$: stutter-sampler $g \ (\sigma \circ \ f)$
shows stutter-sampler $(f \circ g) \ \sigma$
proof (auto simp: stutter-sampler-def)
from $f \ g$ show $f \ (g \ 0) = 0$ by (simp add: stutter-sampler-0)
next
from $g$[THEN stutter-sampler-mono] $f$[THEN stutter-sampler-mono]
show strict-mono $(f \circ g)$ by (rule strict-mono-comp)
next
fix $i \ k$
assume lo: $f \ (g \ i) < k$ and hi: $k < f \ (g \ (Suc \ i))$
from $f$ obtain $m$ where 1: $f \ m \leq k$ and 2: $k < f \ (Suc \ m)$
by (rule stutter-sampler-interval)
with $f$ have 3: $\sigma \ k = \sigma \ (f \ m)$ by (rule stutter-sampler-between)
from lo 2 have $f \ (g \ i) < f \ (Suc \ m)$ by simp
with $f$[THEN stutter-sampler-mono] have 4: $g \ i \leq \ m$ by (simp add: strict-mono-less)
from 1 hi have $f \ m < f \ (g \ (Suc \ i))$ by simp
with $f$[THEN stutter-sampler-mono] have 5: $m < g \ (Suc \ i)$ by (simp add: strict-mono-less)
from $g \ 4 \ 5$ have $(\sigma \circ f) \ m = (\sigma \circ f) \ (g \ i)$ by (rule stutter-sampler-between)
with 3 show $\sigma \ k = \sigma \ (f \ (g \ i))$ by simp
qed

Stuttering sampling functions can be extended to suffixes.

lemma stutter-sampler-suffix:
assumes $f$: stutter-sampler $f \ \sigma$
shows $(\lambda k. f \ (n+k) - f \ n) \ (suffix \ (f \ n) \ \sigma)$
proof (auto simp: stutter-sampler-def strict-mono-def)
fix $i \ j$
assume ij: $(i::nat) < j$
from $f$ have mono: strict-mono $f$ by (rule stutter-sampler-mono)

from mono[THEN strict-mono-mono] have $f \ n \leq f \ (n+i)$
by \( \text{rule monoD} \) simp

moreover

from \( \text{mono[THEN strict-mono-mono]} \) have \( f \cdot n \leq f \cdot (n+j) \)

by \( \text{rule monoD} \) simp

moreover

from mono \( ij \) have \( f \cdot (n+i) < f \cdot (n+j) \) by (auto intro: strict-monoD)

ultimately

show \( f \cdot (n+i) - f \cdot n < f \cdot (n+j) - f \cdot n \) by simp

next

fix \( i \ k \)

assume \( \text{lo: } f \cdot (n+i) - f \cdot n < k \) and \( \text{hi: } k < f \cdot (\text{Suc} \cdot (n+i)) - f \cdot n \)

from \( \text{lo} \) have \( f \cdot (n+i) \leq f \cdot n + k \) by simp

moreover

from \( \text{hi} \) have \( f \cdot n + k < f \cdot (\text{Suc} \cdot (n+i)) \) by simp

moreover

from \( f[\text{THEN stutter-sampler-mono}, \text{THEN strict-mono-mono}] \)

have \( f \cdot n \leq f \cdot (n+i) \) by \( \text{rule monoD} \) simp

ultimately show \( \sigma \cdot (f \cdot n + k) = \sigma \cdot (f \cdot n + (f \cdot (n+i) - f \cdot n)) \)

by (auto dest: stutter-sampler-between[OF \( f \)])

qed

2.2 Preservation of properties through stuttering sampling

Stuttering sampling preserves the initial element of the sequence, as well as the presence and relative ordering of different elements.

lemma stutter-sampled-0:

assumes stutter-sampler \( f \ \sigma \)

shows \( \sigma \cdot (f \cdot 0) = \sigma \cdot 0 \)

using assms[THEN stutter-sampler-0] by simp

lemma stutter-sampled-in-range:

assumes \( f: \text{stutter-sampler } f \ \sigma \) and \( s: s \in \text{range } \sigma \)

shows \( s \in \text{range } (\sigma \circ f) \)

proof

from \( f \) obtain \( k \) where \( k \leq n \) \( n < f \cdot (\text{Suc} \cdot k) \) by (rule stutter-sampler-interval)

with \( f \) have \( \sigma \cdot n = \sigma \cdot (f \cdot k) \) by (rule stutter-sampler-between)

with \( n \) show \( ?\text{thesis} \) by auto

qed

lemma stutter-sampled-range:

\( \text{range } (\sigma \circ f) = \text{range } \sigma \) if \( \text{stutter-sampler } f \ \sigma \)

using that stutter-sampled-in-range \([of f \ \sigma]\) by auto

lemma stutter-sampled-precedence:

assumes \( f: \text{stutter-sampler } f \ \sigma \) and \( ij: i \leq j \)

obtains \( k \ l \) where \( k \leq l \sigma \cdot (f \cdot k) = \sigma \cdot i \sigma \cdot (f \cdot l) = \sigma \cdot j \)

proof

from \( f \) obtain \( k \) where \( k \cdot f \leq i \) \( i < f \cdot (\text{Suc} \cdot k) \) by (rule stutter-sampler-interval)
with \( f \) have 1: \( \sigma_i = \sigma(fk) \) by (rule stutter-sampler-between)
from \( f \) obtain \( l \) where \( l. f l \leq j < f(Suc l) \) by (rule stutter-sampler-interval)
with \( f \) have 2: \( \sigma j = \sigma(fl) \) by (rule stutter-sampler-between)
from \( klij \) have \( f l < Suc l \) by simp
with 1 2 that show \( ?thesis \) by simp
qed

2.3 Maximal stuttering sampling

We define a particular sampling function that is maximal in the sense that it eliminates all finite stuttering. If a sequence ends with infinite stuttering then it behaves as the identity over the (maximal such) suffix.

fun max-stutter-sampler where
max-stutter-sampler \( \sigma 0 = 0 \)
| max-stutter-sampler \( \sigma (Suc n) = \)
  (let prev = max-stutter-sampler \( \sigma n \)
in if (\( \forall k > prev. \sigma k = \sigma prev \))
  then Suc prev
  else (LEAST k. prev < k \& \sigma k \neq \sigma prev))

max-stutter-sampler is indeed a stuttering sampling function.

lemma max-stutter-sampler:
stutter-sampler (max-stutter-sampler \( \sigma \)) \( \sigma \) (is stutter-sampler \( ?ms - \))
proof –
  have \( ?ms 0 = 0 \) by simp
  moreover
  have \( \forall n. ?ms n < ?ms (Suc n) \)
  proof
    fix \( n \)
    show \( ?ms n < ?ms (Suc n) \) (is \( ?prev < ?next \))
    proof (cases \( \forall k > ?prev. \sigma k = \sigma ?prev \))
      case True thus \( ?thesis \) by (simp add: Let-def)
      next
case False
    hence \( \exists k. ?prev < k \& \sigma k \neq \sigma ?prev\) by simp
    from this.THEN LeastI-ex
    have \( ?prev < (LEAST k. ?prev < k \& \sigma k \neq \sigma ?prev) \).
    with False show \( ?thesis \) by (simp add: Let-def)
  qed
  qed
hence strict-mono \( ?ms \)
unfolding strict-mono-def by (blast intro: lift-Suc-mono-less)
moreover
have \( \forall n k. ?ms n < k \& \sigma k < ?ms (Suc n) \rightarrow \sigma k = \sigma (?ms n) \)
proof (clarify)
  fix \( n k \)
assume \( lo: ?ms n < k \) (is \( ?prev < k \))
and hi: k < ?ms (Suc n) (is k < ?next)

show σ k = σ ?prev

proof (cases ∀ k > ?prev. σ k = σ ?prev)

case True

hence ?next = Suc ?prev by (simp add: Let-def)

with lo hi show ?thesis by simp — no room for intermediate index

next

case False

hence ?next = (LEAST k. ?prev < k ∧ σ k ≠ σ ?prev)

by (auto simp add: Let-def)

with lo hi show ?thesis by (auto dest: not-less-Least)

qed

qed

ultimately show ?thesis unfolding stutter-sampler-def by blast

qed

We write ♮σ for the sequence σ sampled by the maximal stuttering sampler. Also, a sequence is stutter free if it contains no finite stuttering: whenever two subsequent elements are equal then all subsequent elements are the same.

definition stutter-reduced (♮- [100] 100) where

♮σ = σ ◦ (max-stutter-sampler σ)

definition stutter-free where

stutter-free σ ≡ ∀ k (Suc k) = σ k → (∀ n> k. σ n = σ k)

lemma stutter-freeI:

assumes ∨ k n. [[σ (Suc k) = σ k; n> k]] → σ n = σ k

shows stutter-free σ

using assms unfolding stutter-free-def by blast

lemma stutter-freeD:

assumes stutter-free σ and σ (Suc k) = σ k and n> k

shows σ n = σ k

using assms unfolding stutter-free-def by blast

Any suffix of a stutter free sequence is itself stutter free.

lemma stutter-free-suffix:

assumes sigma: stutter-free σ

shows stutter-free (suffix k σ)

proof (rule stutter-freeI)

fix j n

assume j: (suffix k σ) (Suc j) = (suffix k σ) j and n: j < n

from j have σ (Suc (k+j)) = σ (k+j) by simp

moreover from n have k+n > k+j by simp

ultimately have σ (k+n) = σ (k+j) by (rule stutter-freeD[OF sigma])

thus (suffix k σ) n = (suffix k σ) j by simp

qed
lemma stutter-reduced-0: $(\sigma) 0 = \sigma 0$
  by (simp add: stutter-reduced-def stutter-sampled-0 max-stutter-sampler)

lemma stutter-free-reduced:
  assumes sigma: stutter-free $\sigma$
  shows $\neg\sigma = \sigma$
proof
{ fix n
  have max-stutter-sampler $\sigma$ n = n (is ?ms n = n)
  proof (induct n)
    show ?ms 0 = 0 by simp
  next
    fix n
    assume ih: ?ms n = n
    show ?ms (Suc n) = Suc n
    proof (cases $\sigma$ (Suc n) = $\sigma$ (?ms n))
      case True
      with ih have $\sigma$ (Suc n) = $\sigma$ n by simp
      with sigma have $\forall$ k > n. $\sigma$ k = $\sigma$ n
        unfolding stutter-free-def by blast
      with ih show ?thesis by (simp add: Let-def)
    next
      case False
      with ih have (LEAST k. k > n ∧ $\sigma$ k ≠ $\sigma$ (?ms n)) = Suc n
        by (auto intro: Least-equality)
      with ih False show ?thesis by (simp add: Let-def)
    qed
  qed
} thus ?thesis by (auto simp: stutter-reduced-def)
qed

Whenever two sequence elements at two consecutive sampling points of the
maximal stuttering sampler are equal then the sequence stutters infinitely
from the first sampling point onwards. In particular, $\neg\sigma$ is stutter free.

lemma max-stutter-sampler-nostuttering:
  assumes stat: $\sigma$ (max-stutter-sampler $\sigma$ (Suc k)) = $\sigma$ (max-stutter-sampler $\sigma$ k)
  and n: n > max-stutter-sampler $\sigma$ k (is - > ?ms k)
  shows $\sigma$ n = $\sigma$ (?ms k)
proof (rule contr)
  assume contr: $\neg$ ?thesis
  with n have ?ms k < n ∧ $\sigma$ n ≠ $\sigma$ (?ms k) (is ?diff n) ..
  hence ?diff (LEAST n. ?diff n) by (rule LeastI)
  with contr have $\sigma$ (?ms (Suc k)) ≠ $\sigma$ (?ms k) by (auto simp add: Let-def)
  from this stat show False ..
  qed

lemma stutter-reduced-stutter-free: stutter-free $(\neg\sigma)$
proof (rule stutter-freeI)
  fix k n
  assume k: (∃σ) (Suc k) = (∃σ) k and n: k < n
  from n have max-stutter-sampler σ k < max-stutter-sampler σ n
    using max-stutter-sampler[THEN stutter-sampler-mono, THEN strict-monoD]
    by blast
  with k show (∃σ) n = (∃σ) k
    unfolding stutter-reduced-def
    by (auto elim: max-stutter-sampler-nostuttering simp del: max-stutter-sampler.simps)
qed

lemma stutter-reduced-suffix: (∃σ) (suffix k (∃σ)) = suffix k (∃σ)
proof (rule stutter-free-reduced)
  have stutter-free (∃σ) by (rule stutter-reduced-stutter-free)
  thus stutter-free (suffix k (∃σ)) by (rule stutter-free-suffix)
qed

lemma stutter-reduced-reduced: (∃σ) = (∃σ)
  by (insert stutter-reduced-suffix[of 0 σ, simplified])

One can define a partial order on sampling functions for a given sequence σ by saying that function g is better than function f if the reduced sequence induced by f can be further reduced to obtain the reduced sequence corresponding to g, i.e. if there exists a stuttering sampling function h for the reduced sequence σ ◦ f such that σ ◦ f ◦ h = σ ◦ g. (Note that f ◦ h is indeed a stuttering sampling function for σ, by theorem stutter-sampler-comp.)

We do not formalize this notion but prove that max-stutter-sampler σ is the best sampling function according to this order.

theorem sample-max-sample:
  assumes f: stutter-sampler f σ
  shows (∃σ) (σ ◦ f) = (∃σ)
proof
  let ?mss = max-stutter-sampler σ
  let ?mssf = max-stutter-sampler (σ ◦ f)
  from f have mssf: stutter-sampler (f ◦ ?mssf) σ
    by (blast intro: stutter-sampler-comp max-stutter-sampler)

The following is the core invariant of the proof: the sampling functions max-stutter-sampler σ and f ◦ (max-stutter-sampler (σ ◦ f)) work in lock-step (i.e., sample the same points), except if σ ends in infinite stuttering, at which point function f may make larger steps than the maximal sampling functions.

{ 
  fix k
  have ?mss k = f (?mssf k)
    ∨ ?mss k ≤ f (?mssf k) ∧ (∀ n ≥ ?mss k. σ (?mss k) = σ n)
    (is ?P k is ?A k ∨ ?B k)
proof (induct k)
from \( f \) mssf have \(?mss \ 0 = f \ (?mssf \ 0) \)
  by (simp add: max-stutter-sampler stutter-sampler-0)
thus \(?P \ 0 \)
next
fix \( k \)
assume ih: \(?P \ k \)
have \( b \): \( ?B \ k \rightarrow ?B \ (Suc \ k) \)
proof
  assume \( 0 \): \( ?B \ k \)
hence \( 1 \): \( ?mss \ k \leq f \ (?mssf \ k) \)
  ..
from \( 0 \) have \( 2 \): \( \forall \ n \geq ?mss \ k. \ ?mss \ k = \sigma \ n \)
  ..
hence \( \forall \ n > ?mss \ k. \ ?mss \ k = \sigma \ n \) by auto
hence \( \forall \ n > ?mss \ k. \ ?mss \ k = \sigma \ ( ?mss \ k) \) by auto
hence \( 3 \): \( ?mss \ (Suc \ k) = Suc \ (?mss \ k) \) by (simp add: Let-def)
with \( 2 \) have \( \sigma \ ( ?mss \ k) = \sigma \ ( ?mss \ (Suc \ k)) \)
  by (auto simp del: max-stutter-sampler.simps)
from \( sym[\ OF \ this] \) \( 2 \) have \( \forall \ n \geq ?mss \ (Suc \ k). \ ?mss \ (Suc \ k) = \sigma \ n \)
  by (auto simp del: max-stutter-sampler.simps)
moreover
from mssf[THEN stutter-sampler-mono, THEN strict-monoD]
have \( f \ (?mssf \ k) < f \ (?mssf \ (Suc \ k)) \)
  by (simp del: max-stutter-sampler.simps)
with \( 1 \) \( 3 \) have \( ?mss \ (Suc \ k) \leq f \ (?mssf \ (Suc \ k)) \)
  by (simp del: max-stutter-sampler.simps)
ultimately show \( \exists \ ?B \ (Suc \ k) \) by blast
qed
from \( \text{ih} \) show \( \ ?P \ (Suc \ k) \)
proof
  assume \( a \): \( \ ?A \ k \)
show \( \ ?thesis \)
proof (cases \( \forall \ n > ?mss \ k. \ ?mss \ k = \sigma \ n \))
  case True
  hence \( \forall \ n \geq ?mss \ k. \ ?mss \ k = \sigma \ n \) by (auto simp: le-less)
  with \( a \) have \( ?B \ k \) by simp
  with \( b \) show \( \ ?thesis \) by (simp del: max-stutter-sampler.simps)
next
  case False
  hence \( \diff : \ ?mss \ (Suc \ k) \neq \sigma \ ( ?mss \ k) \)
  by (blast dest: max-stutter-sampler-nostuttering)
  have \( ?A \ (Suc \ k) \)
proof (rule antisym)
  show \( f \ (?mssf \ (Suc \ k)) \leq \ ?mss \ (Suc \ k) \)
proof (rule contr)
  assume \( \neg \ ?thesis \)
  hence \( \contr : \ ?mss \ (Suc \ k) < f \ (?mssf \ (Suc \ k)) \) by simp
  from mssf have \( \ ?mss \ (Suc \ k) \ = \sigma \ ((f \ ?mssf) \ k) \)
proof (rule stutter-sampler-between)
  from max-stutter-sampler[of \ ?mss \ (Suc \ k), THEN stutter-sampler-mono]
  have \( \ ?mss \ k < \ ?mss \ (Suc \ k) \) by (rule strict-monoD) simp
with a show \((f \circ \mathsf{mssf}) \leq \mathsf{mss} (\text{Suc } k)\)
by \((\text{simp add: o-def del: max-stutter-sampler.simps})\)
next
from \text{contr} show \(\mathsf{mss} (\text{Suc } k) < (f \circ \mathsf{mssf}) (\text{Suc } k)\) by simp
qed
with a have \(\sigma (\mathsf{mss} (\text{Suc } k)) = \sigma (\mathsf{mss} k)\)
by \((\text{simp add: o-def del: max-stutter-sampler.simps})\)
with \text{diff} show False ..
qed
next
have \(\exists m > \mathsf{mssf} k. f m = \mathsf{mss} (\text{Suc } k)\)
proof (rule ccontr)
assume \(\neg \mathsf{thesis}\)
hence \text{contr: } \forall i. f ((\mathsf{mssf} k) + \text{Suc } i) \neq \mathsf{mss} (\text{Suc } k)\) by simp
\{
  fix \(i\)
  have \(f (\mathsf{mssf} k + i) < \mathsf{mss} (\text{Suc } k)\) (is \(?F i\))
  proof (induct \(i\))
  from \(a\) have \(f (\mathsf{mssf} k + 0) = \mathsf{mss} k\) by \((\text{simp add: o-def})\)
  also from \text{max-stutter-sampler[of } \sigma, THEN stutter-sampler-mono\]
  have \(\ldots < \mathsf{mss} (\text{Suc } k)\)
  by (rule \text{strict-monoD}) simp
finally show \(?F 0\).
next
  fix \(i\)
  assume \(ih: \ ?F i\)
  show \(?F (\text{Suc } i)\)
  proof (rule \text{ccontr})
    assume \(\neg \mathsf{thesis}\)
    then have \(\mathsf{mss} (\text{Suc } k) \leq f (\mathsf{mssf} k + \text{Suc } i)\)
    by \((\text{simp add: o-def})\)
    moreover from \text{contr} have \(f (\mathsf{mssf} k + \text{Suc } i) \neq \mathsf{mss} (\text{Suc } k)\)
    by \text{blast}
    ultimately have \(i: \mathsf{mss} (\text{Suc } k) < f (\mathsf{mssf} k + \text{Suc } i)\)
    by \((\text{simp add: less-le})\)
    from \(f\) have \(\sigma (\mathsf{mss} (\text{Suc } k)) = \sigma (f (\mathsf{mssf} k + i))\)
    proof (rule \text{stutter-sampler-between})
      from \(ih\) show \(f (\mathsf{mssf} k + i) \leq \mathsf{mss} (\text{Suc } k)\)
      by \((\text{simp add: o-def})\)
    next
      from \(i\) show \(\mathsf{mss} (\text{Suc } k) < f (\text{Suc } (\mathsf{mssf} k + i))\)
      by simp
    qed
  also from \text{max-stutter-sampler} have \(\ldots = \sigma (\mathsf{mss} k)\)
  proof (rule \text{stutter-sampler-between})
    from \(f[THEN \text{stutter-sampler-mono, THEN strict-mono-mono}]\)
    have \(f (\mathsf{mssf} k) \leq f (\mathsf{mssf} k + i)\) by \((\text{rule } \text{monoD})\) simp
    with a show \(\mathsf{mss} k \leq f (\mathsf{mssf} k + i)\) by \((\text{simp add: o-def})\)
  qed (rule \(ih\))
also note diff
finally show False by simp
qed

qed

} note bounded = this
from f[THEN stutter-sampler-mono]
have strict-mono (λi. (:?mssf k + i))
  by (auto simp: strict-mono-def)
then obtain i where i: ?mss (Suc k) < f (?mssf k + i)
  by (blast dest: strict-mono-exceeds)
from bounded have f (?mssf k + i) < ?mss (Suc k) .
with i show False by (simp del: max-stutter-sampler.simps)
qed
then obtain i where i: ?mss (Suc k) < f (?mssf k + i)
  by (rule strict-monoD simp)
with a have ?mss k ≤ f (?mssf (Suc k))
  by (simp add: o-def)
from this have σ (f (?mssf (Suc k))) = σ (?mssf k)
  by (rule stutter-sampler-between[OF max-stutter-sampler])
with a have stut: (σ o f) (?mssf (Suc k)) = (σ o f) (?mssf k)
  by (simp add: o-def)
from this m have (σ o f) m = (σ o f) (?mssf k)
  by (blast intro: max-stutter-sampler-nostuttering)
with diff m' a show False
  by (simp add: o-def)
qed
qed

qed

thus ?thesis ..
qed

next
assume ?B k with b show ?thesis by (simp del: max-stutter-sampler.simps)
qed
qed

} hence σ = 5(σ o f) unfolding stutter-reduced-def by force
thus ?thesis by (rule sym)
qed

end

theory StutterEquivalence

imports Samplers
3 Stuttering Equivalence

Stuttering equivalence of two sequences is formally defined as the equality of their maximally reduced versions.

**Definition**: stutter-equiv (infix \(\approx\)) where

\[
\sigma \approx \tau \equiv \exists \nu \sigma = \nu \tau
\]

Stuttering equivalence is an equivalence relation.

**Lemma**: stutter-equiv-refl: \(\sigma \approx \sigma\)

**Unfolding**: stutter-equiv-def ..

**Lemma**: stutter-equiv-sym [sym]: \(\sigma \approx \tau \implies \tau \approx \sigma\)

**Unfolding**: stutter-equiv-def by (rule sym)

**Lemma**: stutter-equiv-trans [trans]: \(\rho \approx \sigma \implies \sigma \approx \tau \implies \rho \approx \tau\)

**Unfolding**: stutter-equiv-def by simp

In particular, any sequence sampled by a stuttering sampler is stuttering equivalent to the original one.

**Lemma**: sampled-stutter-equiv:

- **Assumes**: stutter-sampler \(f\) \(\sigma\)
- **Shows**: \(\sigma \circ f \approx \sigma\)
- **Using**: assms unfolding stutter-equiv-def by (rule sample-max-sample)

**Lemma**: stutter-reduced-equivalent: \(\exists \nu \sigma \approx \sigma\)

**Unfolding**: stutter-equiv-def by (rule stutter-reduced-reduced)

For proving stuttering equivalence of two sequences, it is enough to exhibit two arbitrary sampling functions that equalize the reductions of the sequences. This can be more convenient than computing the maximal stutter-reduced version of the sequences.

**Lemma**: stutter-equivI:

- **Assumes**: \(f\): stutter-sampler \(f\) \(\sigma\) and \(g\): stutter-sampler \(g\) \(\tau\)
- and **eq**: \(\sigma \circ f = \tau \circ g\)
- **Shows**: \(\sigma \approx \tau\)

**Proof** –

- from \(f\) have \(\exists \nu \sigma = \nu(\sigma \circ f)\) by (rule sample-max-sample[THEN sym])
- also from \(eq\) have ... = \(\nu(\tau \circ g)\) by simp
- also from \(g\) have ... = \(\nu \tau\) by (rule sample-max-sample)
- finally show ?thesis by (unfold stutter-equiv-def)

**Qed**

The corresponding elimination rule is easy to prove, given that the maximal stuttering sampling function is a stuttering sampling function.
lemma stutter-equivE:
assumes eq: σ ≈ τ
and p: ∀ f g. [ stutter-sampler f σ; stutter-sampler g τ; σ o f = τ o g ] ⇒ P
shows P
proof (rule p)
  from eq show σ o (max-stutter-sampler σ) = τ o (max-stutter-sampler τ)
  by (unfold stutter-equiv-def stutter-reduced-def)
qed (rule max-stutter-sampler)+

Therefore we get the following alternative characterization: two sequences are stuttering equivalent iff there are stuttering sampling functions that equalize the two sequences.

lemma stutter-equiv-eq:
σ ≈ τ = (∃ f g. stutter-sampler f σ ∧ stutter-sampler g τ ∧ σ o f = τ o g)
by (blast intro: stutter-equivI elim: stutter-equivE)

The initial elements of stutter equivalent sequences are equal.

lemma stutter-equiv-0:
assumes σ ≈ τ
shows σ 0 = τ 0
proof −
  have σ 0 = (τ 0) 0 by (rule stutter-reduced-0[THEN sym])
  with assms[unfolded stutter-equiv-def] show ?thesis
    by (simp add: stutter-reduced-0)
qed

abbreviation suffix-notation (- [..-])
where
  suffix-notation w k ≡ suffix k w

Given any stuttering sampling function f for sequence σ, any suffix of σ starting at index f n is stuttering equivalent to the suffix of the stutter-reduced version of σ starting at n.

lemma suffix-stutter-equiv:
assumes f: stutter-sampler f σ
shows suffix (f n) σ ≈ suffix n (σ o f)
proof −
  from f have stutter-sampler (λk. f (n+k) − f n) (σ[f n ..])
    by (rule stutter-sampler-suffix)
  moreover
  have stutter-sampler id ((σ o f)[n ..])
    by (rule id-stutter-sampler)
  moreover
  have (σ[f n ..]) o (λk. f (n+k) − f n) = ((σ o f)[n ..]) o id
  proof (rule ext, auto)
    fix i
    from f[THEN stutter-sampler-mono, THEN strict-sampler-mono]
  have f n ≤ f (n+i) by (rule monoD) simp
Thus \( \sigma (f n + (f (n+i) - f n)) = \sigma (f (n+i)) \) by simp 

qed 

ultimately show \( ? \)thesis 

by (rule stutter-equivI) 

qed 

Given a stuttering sampling function \( f \) and a point \( n \) within the interval from \( f k \) to \( f (k+1) \), the suffix starting at \( n \) is stuttering equivalent to the suffix starting at \( f k \). 

**Lemma** stutter-equiv-within-interval: 

assumes \( f: \text{stutter-sampler} \ f \sigma \) and \( lo: f k \leq n \) and \( hi: n < f (Suc k) \) 

shows \( \sigma [n ..] \approx \sigma [f k ..] \) 

**proof** --

have stutter-sampler id \( (\sigma [n ..]) \) by (rule id-stutter-sampler) 

moreover from \( lo \) have stutter-sampler \( (\lambda i. \text{if} \ i=0 \ \text{then} \ 0 \ \text{else} \ n + i - f k) \) \( (\sigma [f k ..]) \) 

(is stutter-sampler \( ?f \) -) 

**proof** (auto simp; stutter-sampler-def strict mono-def) 

fix \( i \) 

assume \( i: i < Suc \ n - f k \) 

from \( f \) show \( \sigma (f k + i) = \sigma (f k) \) 

**proof** (rule stutter-sampler-between) 

from \( i \ hi \) show \( f k + i < f (Suc k) \) by simp 

qed simp 

qed 

moreover 

have \( \sigma [n ..] \circ \text{id} = (\sigma [f k ..]) \circ ?f \) 

**proof** (rule ext, auto) 

from \( f \) \( lo \) \( hi \) show \( \sigma n = \sigma (f k) \) by (rule stutter-sampler-between) 

next 

fix \( i \) 

from \( lo \) show \( \sigma (n+i) = \sigma (f k + (n + i - f k)) \) by simp 

qed 

ultimately show \( ? \)thesis by (rule stutter-equivI) 

qed 

Given two stuttering equivalent sequences \( \sigma \) and \( \tau \), we obtain a zig-zag relationship as follows: for any suffix \( \tau [n ..] \) there is a suffix \( \sigma [m ..] \) such that:

1. \( \sigma [m ..] \approx \tau [n ..] \) and 

2. for every suffix \( \sigma [j ..] \) where \( j < m \) there is a corresponding suffix \( \tau [k ..] \) for some \( k < n \). 

**Theorem** stutter-equiv-suffixes-left: 

assumes \( \sigma \approx \tau \) 

obtains \( m \) where \( \sigma [m ..] \approx \tau [n ..] \) and \( \forall j < m. \ \exists k < n. \ \sigma [j ..] \approx \tau [k ..] \) 

using assms proof (rule stutter-equivE)
fix $f, g$
assume $f$: stutter-sampler $f \sigma$
    and $g$: stutter-sampler $g \tau$
    and $eq$: $\sigma \circ f = \tau \circ g$
from $g$ obtain $i$ where $i$: $g \; i \le n < g \: (\text{Suc} \: i)$
    by (rule stutter-sampler-interval)
with $g$ have $\tau[n..] \approx \tau[g \; i ..]$ 
    by (rule stutter-equiv-within-interval)
also from $g$ have $\ldots \approx (\tau \circ g)[i ..]$
    by (rule suffix-stutter-equiv)
also from $eq$ have $\ldots = (\sigma \circ f)[i ..]$
    by simp
also from $f$ have $\ldots \approx \sigma[f \; i ..]$
    by (rule stutter-equiv-within-interval)
finally have $\sigma[f \; i ..] \approx \tau[n ..]$
    by (rule stutter-equiv-sym)
moreover
    
    {  
        fix $j$
assume $j$: $j < f \; i$
from $f$ obtain $a$ where $a$: $f \; a \le j < f \: (\text{Suc} \: a)$
    by (rule stutter-sampler-interval)
from $a \; j$ have $f \; a < f \; i$ by simp
with $f[\text{THEN} \: \text{stutter-sampler-mono}]$ have $a < i$
    by (simp add: strict-mono-less)
with $g[\text{THEN} \: \text{stutter-sampler-mono}]$ have $g \; a < g \; i$
    by (simp add: strict-mono-less)
with $i$ have $1$: $g \; a < n$ by simp
from $f \; a$ have $\sigma[j..] \approx \sigma[f \; a ..]$ 
    by (rule stutter-equiv-within-interval)
also from $f$ have $\ldots \approx (\sigma \circ f)[a ..]$
    by (rule suffix-stutter-equiv)
also from $eq$ have $\ldots = (\tau \circ g)[a ..]$ by simp
also from $g$ have $\ldots \approx \tau[g \; a ..]$ 
    by (rule suffix-stutter-equiv[\text{THEN} \: \text{stutter-equiv-sym}])
finally have $\sigma[j ..] \approx \tau[g \; a ..]$ .
    with $f$ have $\exists k < n. \; \sigma[j ..] \approx \tau[k ..]$ by blast
    }
moreover
note that
ultimately show \textit{thesis} by blast
qed

\textbf{theorem \textit{stutter-equiv-suffixes-right}}:  
assumes $\sigma \approx \tau$
obtains $n$ where $\sigma[m..] \approx \tau[n..]$ and $\forall j < n. \; \exists k < m. \; \sigma[k..] \approx \tau[j ..]$
proof –
from assms have $\tau \approx \sigma$

16
by (rule stutter-equiv-sym)
then obtain \( n \) where \( \tau[n..] \approx \sigma[m..] \forall j<m. \exists k<m. \tau[j..] \approx \sigma[k..] \)
by (rule stutter-equiv-suffixes-left)
with that show \(?thesis
by (blast dest: stutter-equiv-sym)
qed

In particular, if \( \sigma \) and \( \tau \) are stutter equivalent then every element that occurs in one sequence also occurs in the other.

**lemma stutter-equiv-element-left:**

assumes \( \sigma \approx \tau \)
obtains \( m \) where \( \sigma m = \tau n \) and \( \forall j<m. \exists k<n. \sigma j = \tau k \)

**proof** –

from assms obtain \( m \) where \( \sigma[m..] \approx \tau[n..] \forall j<m. \exists k<n. \sigma[j..] \approx \tau[k..] \)
by (rule stutter-equiv-suffixes-left)
with that show \(?thesis
by (force dest: stutter-equiv-0)
qed

**lemma stutter-equiv-element-right:**

assumes \( \sigma \approx \tau \)
obtains \( n \) where \( \sigma m = \tau n \) and \( \forall j<n. \exists k<m. \sigma k = \tau j \)

**proof** –

from assms obtain \( n \) where \( \sigma[m..] \approx \tau[n..] \forall j<n. \exists k<m. \sigma[k..] \approx \tau[j..] \)
by (rule stutter-equiv-suffixes-right)
with that show \(?thesis
by (force dest: stutter-equiv-0)
qed

end
theory PLTL
imports Main LTL LTL Samplers StutterEquivalence
begin

4 Stuttering Invariant LTL Formulas

We show that the next-free fragment of propositional linear-time temporal logic PLTL is invariant to finite stuttering.

4.1 Finite Conjunctions and Disjunctions in PLTL

definition OR where OR \( \Phi \equiv \exists \varphi. \text{fold-graph Or-llp False-llp} \Phi \varphi \)
definition AND where AND \( \Phi \equiv \exists \varphi. \text{fold-graph And-llp True-llp} \Phi \varphi \)

**lemma** fold-graph-OR: finite \( \Phi \implies \text{fold-graph Or-llp False-llp} \Phi \) \( (OR \Phi) \)

**unfolding** OR-def by (rule someI2-ex[OF finite-imp-fold-graph])
lemma fold-graph-AND: finite \( \Phi \implies \text{fold-graph } \text{And-ltlp } \text{True-ltlp } \Phi \) \((\text{AND } \Phi)\)
unfolding AND-def by (rule someI2-ex[OF finite-imp-fold-graph])

lemma holds-of-OR [simp]:
assumes fin: finite \((\Phi::\text{pltl set})\)
shows \((\sigma |\to_p \text{ OR } \Phi) = (\exists \varphi\in\Phi. \sigma |\to_p \varphi)\)
proof –
\{ 
  fix \(\psi::\text{pltl}\)
  assume fold-graph Or-ltlp False-ltlp \(\Phi \psi\)
  hence \((\sigma |\to_p \psi) = (\exists \varphi\in\Phi. \sigma |\to_p \varphi)\)
  by (rule fold-graph.induct) auto
\} 
with fold-graph-OR[OF fin] show ?thesis by simp
qed

lemma holds-of-AND [simp]:
assumes fin: finite \((\Phi::\text{pltl set})\)
shows \((\sigma |\to_p \text{ AND } \Phi) = (\forall \varphi\in\Phi. \sigma |\to_p \varphi)\)
proof –
\{ 
  fix \(\psi::\text{pltl}\)
  assume fold-graph And-ltlp True-ltlp \(\Phi \psi\)
  hence \((\sigma |\to_p \psi) = (\forall \varphi\in\Phi. \sigma |\to_p \varphi)\)
  by (rule fold-graph.induct) auto
\} 
with fold-graph-AND[OF fin] show ?thesis by simp
qed

4.2 Next-Free PLTL Formulas

A PLTL formula is called next-free if it does not contain any subformula.

fun next-free :: 'a pltl \Rightarrow bool
where
  next-free false = True
| next-free (atom(p)) = True
| next-free (\(\varphi \implies_{p} \psi) = (\text{next-free } \varphi \land \text{next-free } \psi)\)
| next-free (\(X_{p} \varphi) = \text{False}\)
| next-free (\(U_{p} \psi) = (\text{next-free } \varphi \land \text{next-free } \psi)\)

lemma next-free-not [simp]:
  next-free (not_{p} \varphi) = next-free \(\varphi\)
  by (simp add: Not-ltlp-def)

lemma next-free-true [simp]:
  next-free true_{p}
  by (simp add: True-ltlp-def)

lemma next-free-or [simp]:
\[ \text{next-free (} \varphi \text{ or}_p \psi \text{)} = (\text{next-free } \varphi \land \text{next-free } \psi) \]

by \((\text{simp add: Or-ltlp-def})\)

\begin{itemize}
  \item \textbf{Lemma next-free-and [simp]}: \text{next-free (} \varphi \text{ and}_p \psi \text{)} = (\text{next-free } \varphi \land \text{next-free } \psi)
  
  \begin{itemize}
    \item \text{by (simp add: And-ltlp-def)}
  \end{itemize}
\end{itemize}

\begin{itemize}
  \item \textbf{Lemma next-free-eventually [simp]}:
    \text{next-free (} F_p \varphi \text{)} = \text{next-free } \varphi
    
    \begin{itemize}
      \item \text{by (simp add: Eventually-ltlp-def)}
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item \textbf{Lemma next-free-always [simp]}:
    \text{next-free (} G_p \varphi \text{)} = \text{next-free } \varphi
    
    \begin{itemize}
      \item \text{by (simp add: Always-ltlp-def)}
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item \textbf{Lemma next-free-release [simp]}:
    \text{next-free (} \varphi \text{ R}_p \psi \text{)} = (\text{next-free } \varphi \land \text{next-free } \psi)
    
    \begin{itemize}
      \item \text{by (simp add: Release-ltlp-def)}
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item \textbf{Lemma next-free-weak-until [simp]}:
    \text{next-free (} \varphi \text{ W}_p \psi \text{)} = (\text{next-free } \varphi \land \text{next-free } \psi)
    
    \begin{itemize}
      \item \text{by (auto simp: WeakUntil-ltlp-def)}
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item \textbf{Lemma next-free-strong-release [simp]}:
    \text{next-free (} \varphi \text{ M}_p \psi \text{)} = (\text{next-free } \varphi \land \text{next-free } \psi)
    
    \begin{itemize}
      \item \text{by (auto simp: StrongRelease-ltlp-def)}
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item \textbf{Lemma next-free-OR [simp]}:
    \text{assumes fin: finite (} \Phi::'a \texttt{pltl set})
    \text{shows next-free (} \text{OR} \Phi \text{)} = (\forall \varphi \in \Phi. \text{next-free } \varphi)
    
    \begin{itemize}
      \item \text{proof –}
        \begin{itemize}
          \item fix \psi::'a \texttt{pltl}
          \item assume fold-graph Or-ltlp False-ltlp \Phi \psi
          \item hence next-free \psi = (\forall \varphi \in \Phi. \text{next-free } \varphi)
            \begin{itemize}
              \item \text{by (rule fold-graph.induct) auto}
            \end{itemize}
        \end{itemize}
    \end{itemize}
  \end{itemize}

  \begin{itemize}
    \item \text{with fold-graph-OR[OF fin] show ?thesis by simp}
  \end{itemize}

  \begin{itemize}
    \item \text{qed}
  \end{itemize}
\end{itemize}

\begin{itemize}
  \item \textbf{Lemma next-free-AND [simp]}:
    \text{assumes fin: finite (} \Phi::'a \texttt{pltl set})
    \text{shows next-free (} \text{AND} \Phi \text{)} = (\forall \varphi \in \Phi. \text{next-free } \varphi)
    
    \begin{itemize}
      \item \text{proof –}
        \begin{itemize}
          \item fix \psi::'a \texttt{pltl}
          \item assume fold-graph And-ltlp True-ltlp \Phi \psi
          \item hence next-free \psi = (\forall \varphi \in \Phi. \text{next-free } \varphi)
            \begin{itemize}
              \item \text{by (rule fold-graph.induct) auto}
            \end{itemize}
        \end{itemize}
    \end{itemize}
\end{itemize}
with fold-graph-AND[OF fin] show ?thesis by simp
qed

4.3 Stuttering Invariance of PLTL Without “Next”

A PLTL formula is stuttering invariant if for any stuttering equivalent state sequences \( \sigma \approx \tau \), the formula holds of \( \sigma \) iff it holds of \( \tau \).

definition stutter-invariant where
  stutter-invariant \( \varphi \) = (\( \forall \sigma \tau. (\sigma \approx \tau) \rightarrow (\sigma \models_p \varphi) = (\tau \models_p \varphi) \))

Since stuttering equivalence is symmetric, it is enough to show an implication in the above definition instead of an equivalence.

lemma stutter-invariantI [intro!]:
  assumes \( \sigma \approx \tau; \sigma \models_p \varphi \) \rightarrow \( \tau \models_p \varphi \)
  shows stutter-invariant \( \varphi \)
proof –
  { fix \( \sigma \tau \) assume st: \( \sigma \approx \tau \) and f: \( \sigma \models_p \varphi \) hence \( \tau \models_p \varphi \) by (rule assms) }
moreover
  { fix \( \sigma \tau \) assume st: \( \sigma \approx \tau \) and f: \( \tau \models_p \varphi \) from st have \( \tau \approx \sigma \) by (rule stutter-equiv-sym) from this f have \( \sigma \models_p \varphi \) by (rule assms) }
ultimately show ?thesis by (auto simp: stutter-invariant-def)
qed

lemma stutter-invariantD [dest]:
  assumes stutter-invariant \( \varphi \) and \( \sigma \approx \tau \)
  shows \( (\sigma \models_p \varphi) = (\tau \models_p \varphi) \)
  using assms by (auto simp: stutter-invariant-def)

We first show that next-free PLTL formulas are indeed stuttering invariant. The proof proceeds by straightforward induction on the syntax of PLTL formulas.

theorem next-free-stutter-invariant:
  next-free \( \varphi \) \implies stutter-invariant (\( \varphi::\text{a} \text{ pltl} \))
proof (induct \( \varphi \))
  show stutter-invariant false_p by auto
next
  fix p :: \( \text{\textquotesingle}a \Rightarrow \text{bool} \)
  show stutter-invariant (atom_p(p))
  proof
    fix \( \sigma \tau \)
assume $\sigma \approx \tau \sigma \models_p \text{atom}_p(p)$
thus $\tau \models_p \text{atom}_p(p)$ by (simp add: stutter-equiv-0)
qed

next
fix $\varphi \psi :: \text{'a pltl}$
assume $ih$: next-free $\varphi \Rightarrow$ stutter-invariant $\varphi$
next-free $\psi \Rightarrow$ stutter-invariant $\psi$
assume next-free $(\varphi \implies \psi)$
with $ih$ show stutter-invariant $(\varphi \implies \psi)$ by auto
next
fix $\varphi \psi :: \text{'a pltl}$
assume $ih$: next-free $\varphi \Rightarrow$ stutter-invariant $\varphi$
next-free $\psi \Rightarrow$ stutter-invariant $\psi$
assume next-free $(\varphi U_p \psi)$
with $ih$ have stinv: stutter-invariant $\varphi$ stutter-invariant $\psi$ by auto
show stutter-invariant $(\varphi U_p \psi)$
proof
fix $\sigma \tau$
assume $st$: $\sigma \approx \tau$ and $\text{unt}: \sigma \models_p \varphi \ U_p \psi$
from unt obtain $m$
where 1: $\sigma[m..] \models_p \psi$ and 2: $\forall j < m. (\sigma[j..] \models_p \varphi)$ by auto
from st obtain $n$
where 3: $(\sigma[n..]) \approx (\tau[n..])$ and 4: $\forall i < n. (\exists j < m. (\sigma[j..] \approx (\tau[i..]))$
by (rule stutter-equiv-suffixes-right)
from 1 3 stinv have $\tau[n..] \models_p \psi$ by auto
moreover
from 2 4 stinv have $\forall i < n. (\tau[i..] \models_p \varphi)$ by force
ultimately show $\tau \models_p \varphi \ U_p \psi$ by auto
qed

4.4 Atoms, Canonical State Sequences, and Characteristic Formulas

We now address the converse implication: any stutter invariant PLTL formula $\varphi$ can be equivalently expressed by a next-free formula. The construction of that formula requires attention to the atomic formulas that appear in $\varphi$. We will also prove that the next-free formula does not need any new atoms beyond those present in $\varphi$.

The following function collects the atoms (of type `'a ⇒ bool`) of a PLTL formula.

lemma atoms-pltl-OR [simp]:
assumes fin: finite $(\Phi::\text{'a pltl set})$
shows atoms-pltl $(OR \Phi) = (\bigcup_{\varphi \in \Phi}. \text{atoms-pltl} \ \varphi)$
proof –
{
fix ψ::'a pltl
assume fold-graph Or-ltlp False-ltlp Φ ψ
hence atoms-pltl ψ = (∪ϕ∈Φ. atoms-pltl ϕ)
by (rule fold-graph.induct) auto
}
with fold-graph-OR[OF fin] show ?thesis by simp
qed

lemma atoms-pltl-AND [simp]:
assumes fin: finite (Φ::'a pltl set)
shows atoms-pltl (AND Φ) = (∪ϕ∈Φ. atoms-pltl ϕ)
proof –
{
fix ψ::'a pltl
assume fold-graph And-ltlp True-ltlp Φ ψ
hence atoms-pltl ψ = (∪ϕ∈Φ. atoms-pltl ϕ)
by (rule fold-graph.induct) auto
}
with fold-graph-AND[OF fin] show ?thesis by simp
qed

Given a set of atoms A as above, we say that two states are A-similar if they agree on all atoms in A. Two state sequences σ and τ are A-similar if corresponding states are A-equal.

definition state-sim :: ['a, ('a ⇒ bool) set, 'a] ⇒ bool
(- ~ ~ - [70,100,70] 50) where
s ~A~ t = (∀p∈A. p s ←→ p t)

definition seq-sim :: [nat ⇒ 'a, ('a ⇒ bool) set, nat ⇒ 'a] ⇒ bool
(- ~ ~ - [70,100,70] 50) where
σ ~A~ τ = (∀n. (σ n) ~A~ (τ n))

These relations are (indexed) equivalence relations. Moreover s ~A~ t implies s ~B~ t for B ⊆ A, and similar for σ ~A~ τ and σ ~B~ τ.

lemma state-sim-refl [simp]: s ~A~ s
by (simp add: state-sim-def)

lemma state-sim-sym: s ~A~ t ⇒ t ~A~ s
by (auto simp: state-sim-def)

lemma state-sim-trans[trans]: s ~A~ t ⇒ t ~A~ u ⇒ s ~A~ u
unfolding state-sim-def by blast

lemma state-sim-mono:
assumes s ~A~ t and B ⊆ A
shows s ~B~ t
using assms unfolding state-sim-def by auto
lemma seq-sim-refl [simp]: \( \sigma \simeq A \simeq \sigma \)
by (simp add: seq-sim-def)

lemma seq-sim-sym: \( \sigma \simeq A \simeq \tau \implies \tau \simeq A \simeq \sigma \)
by (auto simp: seq-sim-def state-sim-sym)

lemma seq-sim-trans [trans]: \( \rho \simeq A \simeq \sigma \implies \sigma \simeq A \simeq \tau \)
\( \rho \simeq A \simeq \tau \)
unfolding seq-sim-def by (blast intro: state-sim-trans)

lemma seq-sim-mono:
assumes \( \sigma \simeq A \simeq \tau \) and \( B \subseteq A \)
shows \( \sigma \simeq B \simeq \tau \)
using assms unfolding seq-sim-def by (blast intro: state-sim-mono)

State sequences that are similar w.r.t. the atoms of a PLTL formula evaluate
that formula to the same value.

lemma pltl-seq-sim: \( \sigma \simeq \text{atoms-pltl} \varphi \simeq \tau \)
\( \sigma \text{ | [1..]} \varphi \text{ | [1..]} \tau \)
proof (induct \( \varphi \) arbitrary: \( \sigma \tau \))
fix \( \sigma \tau \)
show \( \exists P \sigma \text{ false}_p \tau \)
by simp
next
fix \( p \sigma \tau \)
assume \( \exists P \sigma \text{ (atom}_p \text{ (p))} \tau \)
thus \( \exists P \sigma \text{ (atom}_p \text{ (p))} \tau \)
by (auto simp: seq-sim-def state-sim-def)
next
fix \( \varphi \psi \sigma \tau \)
assume \( \exists P \sigma \text{ (atom}_p \text{ (p))} \)
thus \( \exists P \sigma \text{ (atom}_p \text{ (p))} \)
by (auto simp: seq-sim-def state-sim-def)
next
fix \( \varphi \psi \sigma \tau \)
assume \( \exists P \sigma \text{ (atom}_p \text{ (p))} \)
thus \( \exists P \sigma \text{ (atom}_p \text{ (p))} \)
by (auto simp: seq-sim-def state-sim-def)
The following function picks an arbitrary representative among $A$-similar states. Because the choice is functional, any two $A$-similar states are mapped to the same state.

**definition canonize**


canonize $A$ $s$ ≡ SOME $t$. $t$ $\sim_{A\sim}s$

**lemma canonize-state-sim:**
canonize $A$ $s$ $\sim_{A\sim}s$

unfolding canonize-def by (rule someI, rule state-sim-refl)

**lemma canonize-canonical:**

assumes $st$: $s$ $\sim_{A\sim}t$

shows canonize $A$ $s$ = canonize $A$ $t$

proof

- from $st$ have $\forall u$. $(u \sim_{A\sim}s) = (u \sim_{A\sim}t)$
  
  by (auto elim: state-sim-sym state-sim-trans)

  thus $?thesis$ unfolding canonize-def by simp

qed

**lemma canonize-idempotent:**

canonize $A$ (canonize $A$ $s$) = canonize $A$ $s$

by (rule canonize-canonical[OF canonize-state-sim])

In a canonical state sequence, any two $A$-similar states are in fact equal.

**definition canonical-sequence**


canonical-sequence $A$ $\sigma$ ≡ $\forall m$ ($n :: \text{nat}$). $\sigma$ $m$ $\sim_{A\sim} \sigma$ $n$ $\rightarrow$ $\sigma$ $m$ = $\sigma$ $n$

Every suffix of a canonical sequence is canonical, as is any (sampled) subsequence, in particular any stutter-sampling.

**lemma canonical-suffix:**

canonical-sequence $A$ $\sigma$ $\rightarrow$ canonical-sequence $A$ ($\sigma[k:]$)

by (auto simp: canonical-sequence-def)

**lemma canonical-sampled:**

canonical-sequence $A$ $\sigma$ $\rightarrow$ canonical-sequence $A$ ($\sigma \circ f$)

by (auto simp: canonical-sequence-def)

**lemma canonical-reduced:**

canonical-sequence $A$ $\sigma$ $\rightarrow$ canonical-sequence $A$ ($\varphi\sigma$)

unfolding stutter-reduced-def by (rule canonical-sampled)

For any sequence $\sigma$ there exists a canonical $A$-similar sequence $\tau$. Such a $\tau$ can be obtained by canonizing all states of $\sigma$. 

> 24
lemma canonical-exists:
obtains $\tau$ where $\tau \simeq A \simeq \sigma$
canonical-sequence $A \tau$

proof
have $(\text{canonize } A \circ \sigma) \simeq A \simeq \sigma$
  by (simp add: seq-sim-def canonize-state-sim)
moreover
have canonical-sequence $A$ $(\text{canonize } A \circ \sigma)$
  by (auto simp: canonical-sequence-def canonize-idempotent
dest: canonize-canonical)
ultimately
show ?thesis using that by blast
qed

Given a state $s$ and a set $A$ of atoms, we define the characteristic formula
of $s$ as the conjunction of all atoms in $A$ that hold of $s$ and the negation of
the atoms in $A$ that do not hold of $s$.

definition characteristic-formula where
characteristic-formula $A$ $s$ $\equiv$
  $((\text{AND } \{ \text{atom}_p(p) \mid p : p \in A \land p \ s \} ) \text{ and } p : p \in A \land \neg (p \ s))$)

lemma characteristic-holds:
finite $A$ $\implies$ $\sigma \models p$ characteristic-formula $A$ $(\sigma 0)$
by (auto simp: characteristic-formula-def)

lemma characteristic-state-sim:
assumes fin: finite $A$
shows $(\sigma 0 \sim A^\sim \tau 0) \equiv (\tau \models_p$ characteristic-formula $A$ $(\sigma (0::\text{nat})))$
proof
assume sim: $\sigma 0 \sim A^\sim \tau 0$
  { fix $p$
    assume $p \in A$
    with sim have $p \ (\tau 0) = p \ (\sigma 0)$ by (auto simp: state-sim-def)
  }
with fin show $\tau \models_p$ characteristic-formula $A$ $(\sigma 0)$
  by (auto simp: characteristic-formula-def) (blast+)
next
assume $\tau \models_p$ characteristic-formula $A$ $(\sigma 0)$
with fin show $\sigma 0 \sim A^\sim \tau 0$
  by (auto simp: characteristic-formula-def state-sim-def)
qed

4.5 Stuttering Invariant PLTL Formulas Don’t Need Next

The following is the main lemma used in the proof of the completeness
theorem: for any PLTL formula $\varphi$ there exists a next-free formula $\psi$ such that
the two formulas evaluate to the same value over stutter-free and canonical
sequences (w.r.t. some $A \supseteq \text{atoms-pltl } \varphi$).

**Lemma** ex-next-free-stutter-free-canonical:

- **Assumes**: $A : \text{atoms-pltl } \varphi \subseteq A$ and $\text{fin} : \text{finite } A$
- **Shows**: $\exists \psi. \text{next-free } \psi \land \text{atoms-pltl } \psi \subseteq A \land$
  
  $(\forall \sigma. \text{stutter-free } \sigma \land \text{canonical-sequence } A \sigma \rightarrow (\sigma \models_p \psi) = (\sigma \models_p \varphi))$

(is $\exists \psi. ?P \varphi \psi$)

**Using** A proof (induct $\varphi$)

The cases of false and atomic formulas are trivial.

- have $?P \ false_p \ false_p$ by auto
- thus $\exists \psi. ?P \ false_p \psi$ ..

**Next**

- fix $p$
  
  - assume $\text{atoms-pltl } (\text{atom}_p(p)) \subseteq A$
  - hence $?P (\text{atom}_p(p)) (\text{atom}_p(p))$ by auto
  
  - thus $\exists \psi. ?P (\text{atom}_p(p)) \psi$ ..

**Next**

Implication is easy, using the induction hypothesis.

- fix $\varphi \psi$
  
  - assume $\text{atoms-pltl } \varphi \subseteq A \implies \exists \psi'. ?P \varphi \psi'$
    
    - and $\text{atoms-pltl } \psi \subseteq A \implies \exists \psi'. ?P \psi \psi'$
    
    - and $\text{atoms-pltl } (\varphi \text{ implies}_p \psi) \subseteq A$
  
  then obtain $\varphi' \psi'$ where $?P \varphi \varphi' ?P \psi \psi'$ by auto
  
  hence $?P (\varphi \text{ implies}_p \psi) (\varphi' \text{ implies}_p \psi')$ by auto
  
  thus $\exists \chi. ?P (\varphi \text{ implies}_p \psi) \chi$ ..

**Next**

The case of until follows similarly.

- fix $\varphi \psi$
  
  - assume $\text{atoms-pltl } \varphi \subseteq A \implies \exists \psi'. ?P \varphi \psi'$
    
    - and $\text{atoms-pltl } \psi \subseteq A \implies \exists \psi'. ?P \psi \psi'$
    
    - and $\text{atoms-pltl } (\varphi \text{ U } p \psi) \subseteq A$
  
  then obtain $\varphi' \psi'$ where 1: $?P \varphi \varphi'$ and 2: $?P \psi \psi'$ by auto

  
  \[
  \begin{align*}
  &\text{fix } \sigma \\
  &\text{assume } \text{sigma: stutter-free } \sigma \text{ canonical-sequence } A \sigma \\
  &\text{hence } \land k. \text{ stutter-free } (\sigma[k..]) \land k. \text{ canonical-sequence } A (\sigma[k..]) \\
  &\text{by } (\text{auto simp: stutter-free-suffix canonical-suffix}) \\
  &\text{with } 1 2 \\
  &\text{have } \land k. (\sigma[k..] \models_p \varphi') = (\sigma[k..] \models_p \varphi) \\
  &\text{and } \land k. (\sigma[k..] \models_p \psi') = (\sigma[k..] \models_p \psi) \\
  &\text{by } (\text{blast+}) \\
  &\text{hence } (\sigma \models_p \varphi' U_p \psi') = (\sigma \models_p \varphi U_p \psi) \\
  &\text{by } \text{auto} \\
  \end{align*}
  \]

  

  with 1 2 have $?P (\varphi U_p \psi) (\varphi' U_p \psi')$ by auto
  
  thus $\exists \chi. ?P (\varphi U_p \psi) \chi$ ..
The interesting case is the one of the \textit{next}-operator.

\begin{align*}
\text{fix } \varphi \\
\text{assume ih: atoms-pltl } \varphi \subseteq A \implies \exists \psi. \ ?P \varphi \psi \text{ and at: atoms-pltl } (X_p \varphi) \subseteq A \\
\text{then obtain } \psi \text{ where } psi: \ ?P \varphi \psi \text{ by auto}
\end{align*}

A valuation \((A)\) is a set \(val \subseteq A\) of atoms. We define some auxiliary notions: the valuation corresponding to a state and the characteristic formula for a valuation. Finally, we define the formula \(\psi'\) that we will prove to be equivalent to \(X_p \varphi\) over the stutter-free and canonical sequence \(\sigma\).

\begin{align*}
\text{define } stval \text{ where } stval &= (\lambda s. \{ p \in A . p \in s \}) \\
\text{define } chi \text{ where } chi &= (\lambda val. (\ AND \{ \text{atom}_p(p) \mid p . p \in val \}) \ AND (\ NOT \{ \text{atom}_p(p) \mid p . p \in \neg \ val \})) \\
\text{define } psi' \text{ where } psi' &= ((\ AND \{ \text{atom}_p(p) \mid p . p \in val \}) \ OR \ (\ NOT \{ \text{atom}_p(p) \mid p . p \in \neg \ val \})) \\
& \quad \text{is } - = ((\ AND \{ \text{atom}_p(p) \mid p . p \in val \}) \ OR \ (\ NOT \{ \text{atom}_p(p) \mid p . p \in \neg \ val \})) \\
\text{have } \bigwedge \ s . \ \{ \text{not} \ p \ (\ AND \{ \text{atom}_p(p) \mid p . p \in \neg \ s \}) \\
& \quad = \{ \text{not} \ p \ (\ AND \{ \text{atom}_p(p) \mid p . p \in A \land \neg \ s \}) \} \\
& \quad \text{by (auto simp: stval-def)} \\
\text{hence chi1: } \bigwedge \ s . \ \text{stval}(s) = \text{characteristic-formula } A \ s \\
& \quad \text{by (auto simp: chi-def stval-def characteristic-formula-def)} \\
\{ \\
\text{fix } val \tau \\
\text{assume val: val } \subseteq A \text{ and tau: } \tau \models_p chi val \\
\text{with fin have val = stval } (\tau 0) \\
& \quad \text{by (auto simp: stval-def chi-def finite-subset)} \\
\} \\
\text{note chi2 = this} \\
\text{have } ?UNT \subseteq (\lambda (val, val'). (chi val) \ AND_p ((chi val) \ U_p (\ psi \ AND_p (chi val')))) \\
& \quad : (Pow A \times Pow A) \\
& \quad (is - \subseteq S) \\
& \quad \text{by auto} \\
\text{with fin have fin-UNT: finite } ?UNT \\
& \quad \text{by (auto simp: finite-subset)} \\
\text{have nf: next-free psi'} \\
\text{proof} \\
& \quad \text{from fin have } \bigwedge \ val. \ val \subseteq A \implies \text{next-free } (chi val) \\
& \quad \quad \text{by (auto simp: chi-def finite-subset)} \\
& \quad \text{with fin fin-UNT psi show } ?thesis \\
& \quad \quad \text{by (force simp: psi'-def finite-subset)} \\
\text{qed} \\
\text{have atoms-pltl: atoms-pltl } psi' \subseteq A \\
\text{proof} \\
& \quad \text{from fin have at-chi: } \bigwedge \ val. \ val \subseteq A \implies \text{atoms-pltl } (chi val) \subseteq A \\
& \quad \quad \text{by (auto simp: chi-def finite-subset)}
\end{align*}
with fin psi have at-alw: atoms-pltl (ψ andₚ (OR ?ALW)) ⊆ A by auto blast?
from fin fin-UNT psi at-chi have atoms-pltl (OR ?UNT) ⊆ A by auto (blast+)?
with at-alw show ?thesis by (auto simp: psi'-def)
qed

\{
  fix σ
  assume st: stutter-free σ and can: canonical-sequence A σ
  have (σ |=ₚ Xₚ ϕ) = (σ |=ₚ psi')
  proof (cases σ (Suc 0) = σ 0)
    case True
    thus ?thesis by simp
  next
    case Suc
    hence n > 0 by simp
    with True st show ?thesis unfolding stutter-free-def by blast
  qed

  { fix σ
    assume st: stutter-free σ and can: canonical-sequence A σ
    have (σ |=ₚ Xₚ ϕ) = (σ |=ₚ psi')
    proof (cases σ (Suc 0) = σ 0)
      case Suc
      hence n > 0 by simp
      with True st show ?thesis unfolding stutter-free-def by blast
    qed

    note alleq = this
    have suffix: \(\forall n. \sigma[n..] = \sigma\)
    proof (rule ext)
      fix n i
      have (σ[n..]) i = σ 0 by (auto intro: alleq)
      moreover have σ i = σ 0 by (rule alleq)
      ultimately show (σ[n..]) i = σ i by simp
    qed

    with st can psi have 1: (σ |=ₚ Xₚ ϕ) = (σ |=ₚ ψ) by simp
  }
  from fin have σ |=ₚ chi (steal (σ 0)) by (simp add: chi1 characteristic-holds)
  with suffix have σ |=ₚ Gₚ (chi (steal (σ 0))) (is - |=ₚ ?alw) by simp
  moreover have ?alw ∈ ?ALW by (auto simp: steal-def)
  ultimately have 2: σ |=ₚ OR ?ALW
  using fin by (auto simp: finite-subset simp del: semantics-pltl-sugar)

  have 3: ¬(σ |=ₚ OR ?UNT)
  proof
    assume unt: σ |=ₚ OR ?UNT
    with fin-UNT obtain val val' k where
      val: val ⊆ A val' ⊆ A val' ≠ val and
      now: σ |=ₚ chi val and k: σ[k..] |=ₚ chi val'
  qed
by auto (blast+)?
from $\langle \text{val} \subseteq A \rangle$ now have $\text{val} = \text{stval} (\sigma 0)$ by (rule chi2)

moreover
from $\langle \text{val}' \subseteq A \rangle \kappa$ suffix have $\text{val}' = \text{stval} (\sigma 0)$ by (simp add: chi2)

moreover note $\langle \text{val}' \neq \text{val} \rangle$
ultimately show $\text{False}$ by simp

qed

from $1 2 3$ show $\text{thesis}$ by (simp add: $\psi'$-def)

next

\text{case False}

Otherwise, $\sigma \models_p X_p \varphi$ is equivalent to $\sigma$ satisfying the second disjunct of $\psi'$. We show both implications separately.

let $\langle \text{val} = \text{stval} (\sigma 0) \rangle$
let $\langle \text{val}' = \text{stval} (\sigma 1) \rangle$

from $\text{False}$ can have vals: $\langle \text{val}' \neq \text{val} \rangle$
by (auto simp: canonical-sequence-def state-sim-def stval-def)

show $\text{thesis}$

proof
assume $\langle \text{phi}: \sigma \models_p X_p \varphi \rangle$

from $\text{fin}$ have $1: \sigma \models_p \chi \langle \text{val} \rangle$ by (simp add: chi1 characteristic-holds)

from $\text{st}$ can have stutter-free $(\sigma[1..])$ canonical-sequence $A \ (\sigma[1..])$
with $\langle \text{phi \ psi \ have} \ 2: \sigma[1..] \models_p \psi \rangle$ by auto

from $\text{fin}$ have $\langle \sigma[1..] \models_p \chi \text{val} \rangle$
by (rule characteristic-holds)

hence $3: \sigma[1..] \models_p \chi \text{val}'$ by (simp add: chi1)

from $1 2 3$ have $\sigma \models_p \text{And-ltlp} (\chi \langle \text{val} \rangle) ((\chi \langle \text{val} \rangle) \ U_p \ \text{And-ltlp} \ \psi \ ((\chi \langle \text{val} \rangle)))$
(is $\models_p \langle \text{ant} \rangle$
by auto

moreover from vals have $\langle \text{ant} \in \text{UNT} \rangle$
by (auto simp: stval-def)
ultimately have $\sigma \models_p \text{OR} \langle \text{UNT} \rangle$
using $\langle \text{fin-UNT[THEN holds-of-OR]} \ $ by blast
thus $\sigma \models_p \psi'$ by (simp add: $\psi'$-def)

next

assume $\langle \text{psi'}: \sigma \models_p \psi' \rangle$

have $\langle \neg(\sigma \models_p \text{OR} \langle \text{ALW} \rangle) \rangle$

proof
assume $\langle \sigma \models_p \text{OR} \langle \text{ALW} \rangle \rangle$
with $\text{fin}$ obtain $\text{val}$ where $1: \text{val} \subseteq A$ and $2: \forall n. \ (\sigma[n..] \models_p \chi \text{val})$

29
by (force simp: finite-subset)
from 2 have \( \sigma[0..] \models_p \chi \text{ val } \)
with 1 have val = ?val by (simp add: chi2)
moreover
from 2 have \( \sigma[1..] \models_p \chi \text{ val } \)
with 1 have val = ?val' by (force dest: chi2)
ultimately
show False using vals by simp
qed
with psi' have \( \sigma \models_p \text{ OR } ?\text{UNT} \)
by (simp add: psi'-def)
with fin-UNT obtain val val' k where
val: val \subseteq A val' \subseteq A val' \neq val
\text{and}
now: \( \sigma \models_p \chi \text{ val } \)
\( \text{and} \)
k: \( \sigma[k..] \models_p \psi \sigma[k..] \models_p \text{ val’ \ and } \)
i: \( \forall i<k. \ (\sigma[i..] \models_p \chi \text{ val } \)
by auto (blast+)

from val now have 1: val = ?val by (simp add: chi2)

have 2: \( k \neq 0 \)
proof
assume k=0
with val k have val' = ?val by (simp add: chi2)
with 1 (val' \neq val) show False by simp
qed

have 3: \( k \leq 1 \)
proof (rule ccontr)
assume \( \neg(k \leq 1) \)
with 1 have \( \sigma[1..] \models_p \chi \text{ val } \)
by simp
with 1 have \( \sigma[1..] \models_p \text{ characteristic-formula } A (\sigma \ 0) \)
by (simp add: chi1)
hence \( (\sigma \ 0) \sim_A (\sigma[1..] \ 0) \)
using characteristic-state-sim[OF fin] by blast
with can have \( \sigma \ 0 = \sigma \ 1 \)
by (simp add: canonical-sequence-def)
with \( \sigma (Suc \ 0) \neq \sigma \ 0 \) show False by simp
qed

from 2 3 have k=1 by simp
moreover
from st can have stutter-free \( \sigma[1..] \) canonical-sequence \( A (\sigma[1..]) \)
by (auto simp: stutter-free-suffix canonical-suffix)
ultimately show \( \sigma \models_p X_p \varphi \)
using \( \sigma[k..] \models_p \psi \)
by auto (blast+)
qed

with nf atoms-pltl show \( \exists \psi'. \ ?P (X_p \varphi) \psi' \)
by blast
qed
Comparing the definition of the next-free formula in the case of formulas $X_p$ $\varphi$ with the one that appears in [2], there is a subtle difference. Peled and Wilke define the second disjunct as a disjunction of formulas

$$(chi\ val)\ U_p\ (\psi\ and_p\ (chi\ val'))$$

for subsets $val, val' \subseteq A$ whereas we conjoin the formula $chi\ val$ to the “until” formula. This conjunct is indeed necessary in order to rule out the case of the “until” formula being true because of $chi\ val'$ being true immediately.

The subtle error in the definition of the formula was acknowledged by Peled and Wilke and apparently had not been noticed since the publication of [2] in 1996 (which has been cited more than a hundred times according to Google Scholar). Although the error was corrected easily, the fact that authors, reviewers, and readers appear to have missed it for so long underscores the usefulness of formal proofs.

We now show that any stuttering invariant PLTL formula can be expressed without the $X_p$ operator.

**Theorem 1:** stutter-invariant-next-free:

- assumes $\phi$: stutter-invariant $\varphi$
- obtains $\psi$ where next-free $\psi$ atoms-pltl $\psi \subseteq$ atoms-pltl $\varphi$
  $$\forall \sigma.\ (\sigma \models_p \psi) = (\sigma \models_p \varphi)$$

- proof
  - have atoms-pltl $\varphi \subseteq$ atoms-pltl $\varphi$ finite (atoms-pltl $\varphi$) by simp-all
  - then obtain $\psi$ where
    - psi: next-free $\psi$ atoms-pltl $\psi \subseteq$ atoms-pltl $\varphi$ and
    - equiv: $\forall \sigma.\ stutter-free\ \sigma \land canonical-sequence\ (atoms-pltl \varphi)\ \sigma \rightarrow (\sigma \models_p \psi)$
  - (sim add: $\varphi$)
  - by (blast dest: ex-next-free-stutter-free-canonical)
  - from $\langle next-free\ \psi\ \rangle$ have $sinv$: stutter-invariant $\psi$
  - by (rule next-free-stutter-invariant)
  
  \{
    fix $\sigma$
    obtain $\tau$ where $1$: $\tau \simeq atoms-pltl \varphi \simeq \sigma$ and $2$: canonical-sequence $\langle atoms-pltl \varphi \rangle\ \tau$
    - by (rule canonical-exists)
    - from $1$ $\langle atoms-pltl \psi \rangle \subseteq$ atoms-pltl $\varphi$, have $3$: $\tau \simeq atoms-pltl \psi \simeq \sigma$
    - by (rule seq-sim-mono)

    - from $1$ have $(\sigma \models_p \varphi) = (\tau \models_p \varphi)$ by (simp add: pltl-seq-sim)
    - also from $phi$ stutter-reduced-equivalent have ...
    - also from $2[THEN\ canonical-reduced]$ equiv stutter-reduced-stutter-free
    - have ...
    - also from $sinv$ stutter-reduced-equivalent have ...
    - also from $3$ have ...
    - finally have $(\sigma \models_p \psi)$ by (simp add: pltl-seq-sim)

  
  finally have $(\sigma \models_p \psi) = (\sigma \models_p \varphi)$ by (rule sym)

  with $\psi$ that show ?thesis by blast
Combining theorems \textit{next-free-stutter-invariant} and \textit{stutter-invariant-next-free}, it follows that a PLTL formula is stuttering invariant iff it is equivalent to a next-free formula.

\textbf{theorem pltl-stutter-invariant:}
\begin{align*}
stutter-invariant \varphi & \iff (\exists \psi. \text{next-free } \psi \land \text{atoms-pltl } \psi \subseteq \text{atoms-pltl } \varphi \land (\forall \sigma. \sigma \models_p \psi \iff \sigma \models_p \varphi)) \\
\end{align*}

\textbf{proof --}
\begin{align*}
\{ & \text{assume stutter-invariant } \varphi \\
& \text{hence } \exists \psi. \text{next-free } \psi \land \text{atoms-pltl } \psi \subseteq \text{atoms-pltl } \varphi \land (\forall \sigma. \sigma \models_p \psi \iff \sigma \models_p \varphi) \\
& \text{by (rule stutter-invariant-next-free) blast} \\
\}
\end{align*}

moreover
\begin{align*}
\{ & \text{fix } \psi \\
& \text{assume 1: next-free } \psi \text{ and 2: } \forall \sigma. \sigma \models_p \psi \iff \sigma \models_p \varphi \\
& \text{from 1 have stutter-invariant } \psi \text{ by (rule next-free-stutter-invariant)} \\
& \text{with 2 have stutter-invariant } \varphi \text{ by blast} \\
\}
\end{align*}

ultimately show \( \neg \text{thesis by blast} \)

\textbf{4.6 Stutter Invariance for LTL with Syntactic Sugar}

We lift the results for PLTL to an extensive version of LTL.

\textbf{primrec ltlc-next-free :: } `a ltlc \Rightarrow bool
\begin{align*}
\text{where} \\
\text{ltilc-next-free true}_e & = \text{True} \\
\text{ltilc-next-free false}_e & = \text{True} \\
\text{ltilc-next-free (prop}_c(q)) & = \text{True} \\
\text{ltilc-next-free (not}_c \varphi & = \text{ltilc-next-free } \varphi \\
\text{ltilc-next-free (\varphi \land_c \psi) & = (ltilc-next-free } \varphi \land ltilc-next-free \psi \\
\text{ltilc-next-free (\varphi \lor_c \psi) & = (ltilc-next-free } \varphi \lor ltilc-next-free \psi \\
\text{ltilc-next-free (\varphi \implies_c \psi) & = (ltilc-next-free } \varphi \land ltilc-next-free \psi \\
\text{ltilc-next-free (X}_c \varphi) & = \text{False} \\
\text{ltilc-next-free (F}_c \varphi & = ltilc-next-free \varphi \\
\text{ltilc-next-free (G}_c \varphi & = ltilc-next-free \varphi \\
\text{ltilc-next-free (U}_c \varphi & = (ltilc-next-free } \varphi \land ltilc-next-free \psi \\
\text{ltilc-next-free (R}_c \varphi & = (ltilc-next-free } \varphi \land ltilc-next-free \psi \\
\text{ltilc-next-free (W}_c \varphi & = (ltilc-next-free } \varphi \land ltilc-next-free \psi \\
\text{ltilc-next-free (M}_c \varphi & = (ltilc-next-free } \varphi \land ltilc-next-free \psi \\
\end{align*}

\textbf{lemma ltlc-next-free-iff[simp]: next-free (ltilc-to-pltl } \varphi \iff \text{ltilc-next-free } \varphi \\
\text{by (induction } \varphi) \text{ auto} \\

A next free formula cannot distinguish between stutter-equivalent runs.

32
\textbf{theorem ltlc-next-free-stutter-invariant:}
assumes \text{next-free: ltlc-next-free } \varphi
assumes eq: r \approx r'
shows r \models_c \varphi \leftrightarrow r' \models_c \varphi
\begin{proof}
  \begin{array}{l}
  \{ \\
  \text{fix } r r' \\
  \text{assume eq: } r \approx r' \text{ and holds: } r \models_c \varphi \\
  \text{then have } r \models_p (\text{ltlc-to-pltl } \varphi) \text{by simp} \\
  \text{from next-free-stutter-invariant[of ltlc-to-pltl } \varphi \text{] next-free} \\
  \text{have PLTL.stutter-invariant (ltlc-to-pltl } \varphi \text{) by simp} \\
  \text{from stutter-invariantD[of this eq] holds have } r' \models_c \varphi \text{ by simp} \\
  \text{note aux=this} \\
  \text{from aux[of r r'] aux[of r' r] eq stutter-equiv-sym[of eq] show ?thesis} \\
  \text{by blast} \\
  \}
\end{array}
\end{proof}
\begin{flushright}
\text{qed}
\end{flushright}

\textbf{References}
