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

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

are stuttering equivalent, whereas

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

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
theory Samplers
imports Main
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** \( f: \text{strict-mono} (f::\text{nat} \Rightarrow \text{nat}) \)
- **shows** \( \exists k. \ n < f k \)
- **proof** (induct \( n \))
  - from \( f \) have \( f 0 < f 1 \) by (rule strict-monoD) simp
  - hence \( 0 < f 1 \) by simp
  - thus \( \exists k. \ 0 < f k \) ..
- next
  - fix \( n \)
  - assume \( \exists k. \ n < f k \)
  - then obtain \( k \) where \( n < f k \) ..
  - hence \( \text{Suc} \ n \leq f k \) by simp
  - also from \( f \) have \( f k < f (\text{Suc} \ k) \) by (rule strict-monoD) simp
  - finally show \( \exists k. \ \text{Suc} \ n < f k \) ..
- qed

More precisely, any natural number \( n \geq f \ 0 \) lies in the interval between \( f k \) and \( f (\text{Suc} \ k) \), for some \( k \).

**lemma strict-mono-interval:**
- **assumes** \( f: \text{strict-mono} (f::\text{nat} \Rightarrow \text{nat}) \text{ and } n: f \ 0 \leq n \)
- **obtains** \( k \) where \( f k \leq n \) \text{ and } \( n < f (\text{Suc} \ k) \)
- **proof** –
  - from \( f[\text{THEN} \text{strict-mono-exceeds}] \) **obtain** \( m \) where \( m < f m \) ..
  - have \( m \neq 0 \)
  - proof
    - assume \( m = 0 \)
    - with \( m \ n \) show \( \text{False} \) by simp
  - qed
  - with \( m \) **obtain** \( m' \) where \( m': n < f (\text{Suc} \ m') \) by (auto simp: gr0-conv-Suc)
  - let \( \ ?k = \text{LEAST} \ k . \ n < f (\text{Suc} \ k) \)
  - from \( m' \) have \( 1: n < f (\text{Suc} \ ?k) \) 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)

We represent \( \omega \)-words as functions of type \( \text{nat} \Rightarrow 'a \). Suffixes of \( \omega \)-words are simply obtained by index shifting, and we introduce a convenient notation.

definition suffix :: \((\text{nat} \Rightarrow 'a)\) \Rightarrow \((\text{nat} \Rightarrow 'a)\) where
\( \sigma[n..] \equiv \lambda i. \sigma(n+i) \)

lemma suffix-elt [simp]: \( (\sigma[n..]) i = \sigma(n+i) \)
  by (simp add: suffix-def)

lemma suffix-0 [simp]: \( \sigma[0..] = \sigma \)
  by (simp add: suffix-def)

lemma suffix-suffix [simp]: \( \sigma[n..][m..] = \sigma[n+m ..] \)
  by (simp add: suffix-def add.assoc)

2 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 \( f \emptyset = \emptyset \) 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 \( \sigma \circ f \).

2.1 Definition and elementary properties

definition stutter-sampler where
  \( f \) is a stuttering sampling function for \( \sigma \)
stutter-sampler \((f::\text{nat} \Rightarrow \text{nat})\) \(\sigma \equiv \)
\(f\ 0 = 0\)
\(\land\ \text{strict-mono}\ f\)
\(\land\ (\forall k\ n.\ f\ k < n \land n < f\ (\text{Suc}\ k) \Rightarrow\ \sigma\ n = \sigma\ (f\ k))\)

**Lemma stutter-sampler-0:** stutter-sampler \(f\ \sigma \implies f\ 0 = 0\)
by \((\text{simp add: stutter-sampler-def})\)

**Lemma stutter-sampler-mono:** stutter-sampler \(f\ \sigma \implies \text{strict-mono}\ f\)
by \((\text{simp add: stutter-sampler-def})\)

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

**Lemma stutter-sampler-interval:**
assumes \(f:\ \text{stutter-sampler}\ f\ \sigma\)
obtains \(k\) where \(f\ k \leq n \land n < f\ (\text{Suc}\ k)\)
using \(f[\text{THEN}\ \text{stutter-sampler-mono}]\) proof \((\text{rule strict-mono-interval})\)
from \(f\) show \(f\ 0 \leq n\) by \((\text{simp add: stutter-sampler-0})\)
qed

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

**Lemma id-stutter-sampler [iff]:** stutter-sampler \(id\ \sigma\)
by \((\text{auto simp: stutter-sampler-def strict-mono-def})\)

Stuttering sampling functions compose, sort of.

**Lemma stutter-sampler-comp:**
assumes \(f::\ \text{stutter-sampler}\ f\ \sigma\)
and \(g::\ \text{stutter-sampler}\ g\ (\sigma\ \circ\ f)\)
shows stutter-sampler \((f\ \circ\ g)\ \sigma\)
proof \((\text{auto simp: stutter-sampler-def})\)
from \(f\ g\) show \(f\ (g\ 0) = 0\) by \((\text{simp add: stutter-sampler-0})\)
next
from \(g[\text{THEN}\ \text{stutter-sampler-mono}]\) \(f[\text{THEN}\ \text{stutter-sampler-mono}]\)
show \(\text{strict-mono}\ (f\ \circ\ g)\) by \((\text{rule strict-mono-comp})\)
next
fix \(i\ k\)
assume \(\text{lo}:: f\ (g\ i) < k \land \text{hi}:: k < f\ (g\ (\text{Suc}\ i))\)
from \(f\) obtain \(m\) where \(1:: f\ m \leq k \land \text{hi}:: k < f\ (\text{Suc}\ m)\)
by \((\text{rule stutter-sampler-interval})\)
with \(f\) have \(3:: \sigma\ k = \sigma\ (f\ m)\) by \((\text{rule stutter-sampler-between})\)
from \(\text{lo}\ 2\) have \(f\ (g\ i) < f\ (\text{Suc}\ m)\) by \(\text{simp}\)
with \(f[\text{THEN}\ \text{stutter-sampler-mono}]\) have \(4:: g\ i \leq m\) by \((\text{simp add: strict-mono-less})\)
from \(1\ \text{hi}\) have \(f\ m < f\ (g\ (\text{Suc}\ i))\) by \(\text{simp}\)
with \(f[\text{THEN}\ \text{stutter-sampler-mono}]\) have \(5:: m < g\ (\text{Suc}\ i)\) by \((\text{simp add: strict-mono-less})\)
from \( g \) 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** stutter-sampler \((\lambda k. \ f \ (n+k) - f \ n) \ (\sigma[fn ..])\)

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

fix \(i\) \(j\)
assume \(ij\): \((i::nat) < j\)

from \(f\) have \(\text{mono}\): \(\text{strict-mono} f\) by (rule stutter-sampler-mono)

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

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

moreover
from \(\text{mono} \ [\text{THEN strict-mono-mono}]\) have \(f \ (n+i) < f \ (n+j)\) by (auto intro: strict-monoD)

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

next
fix \(i\) \(k\)
assume \(lo\): \(f \ (n+i) - f \ n < k\) and \(hi\): \(k < f \ (Suc \ (n+i)) - f \ n\)

from \(lo\) have \(f \ (n+i) \leq f \ n + k\) by simp

moreover
from \(hi\) have \(f \ n + k < f \ (Suc \ (n + i))\) by simp

moreover
from \(f\) [THEN stutter-sampler-mono, THEN strict-mono-mono] have \(f \ n \leq f \ (n+i)\) by (rule monoD) simp

ultimately show \(\sigma \ (f \ n + k) = \sigma \ (f \ n + (f \ (n+i) - f \ n))\)
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** \(\sigma \ 0 = \sigma \ 0\)

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

**lemma stutter-sampled-in-range:**

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

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

**proof** –
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 $\land$ $\sigma$ 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 = $\sigma$ $\$prev$)
We write $\natural\sigma$ for the sequence $\sigma$ 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 ($\natural$- [100] 100)** where
$\natural\sigma = \sigma \circ (\text{max-stutter-sampler } \sigma)$

**definition stutter-free** where
stutter-free $\sigma \equiv \forall k. \sigma (\text{Suc } k) = \sigma k \rightarrow (\forall n>k. \sigma n = \sigma k)$

**lemma stutter-freeI:**
assumes $\forall k. [\sigma (\text{Suc } k) = \sigma k; n>k] \Rightarrow \sigma n = \sigma k$
shows stutter-free $\sigma$
using assms unfolding stutter-free-def by blast

**lemma stutter-freeD:**
assumes stutter-free $\sigma$ and $\sigma (\text{Suc } k) = \sigma k$ and $n>k$
shows $\sigma n = \sigma k$
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
- **shows** stutter-free $\langle \sigma[k..] \rangle$

**proof** (rule stutter-freeI)
- fix $j n$
- assume $j: (\sigma[k..]) (\text{Suc} j) = (\sigma[k..]) j$ and $n: j < n$
- from $j$ have $\sigma (\text{Suc} (k+j)) = \sigma (k+j)$ by simp
- moreover from $n$ have $k+n > k+j$ by simp
- ultimately have $\sigma (k+n) = \sigma (k+j)$ by (rule stutter-freeD[OF $\sigma$])
- thus $(\sigma[k..]) n = (\sigma[k..]) j$ by simp

qed

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

**lemma stutter-free-reduced**: 
- **assumes** $\sigma$: stutter-free
- **shows** $\sharp\sigma = \sigma$

**proof** –
- fix $n$
- have max-stutter-sampler $\sigma n = n$ (is $\exists ms n = n$)
  - proof (induct $n$)
    - show $\exists ms 0 = 0$ by simp
  - next
    - fix $n$
    - assume ih: $\exists ms n = n$
    - show $\exists ms (\text{Suc} n) = \text{Suc} n$
      - proof (cases $\sigma (\text{Suc} n) = \sigma (\exists ms n)$)
        - case True
          - with ih have $\sigma (\text{Suc} n) = \sigma n$ by simp
        - with $\sigma$ have $\forall k > n. \sigma k = \sigma n$
          - unfolding stutter-free-def by blast
          - with ih show $\exists ms$ by (simp add: Let-def)
      - next
        - case False
          - with ih have (LEAST $k. k > n \land \sigma k \neq \sigma (\exists ms n)) = \text{Suc} n$
            - by (auto intro: Least-equality)
          - with ih False show $\exists ms$ by (simp add: Let-def)
    - qed
  - qed
- thus $\exists ms$ 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, ♮σ is stutter free.

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

lemma stutter-reduced-stutter-free: stutter-free (♮σ)
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)[k..]) = (♮σ)[k..]
proof (rule stutter-free-reduced)
  have stutter-free (♮σ) by (rule stutter-reduced-stutter-free)
  thus stutter-free ((♮σ)[k..]) by (rule stutter-free-suffix)
qed

lemma stutter-reduced-reduced: ♮♮σ = ♮σ
  by (rule stutter-reduced-suffix[of σ 0, 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 ◦ 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)
  thus ?P 0 ..
  next
  fix k
  assume ih: ?P k
  have b: ?B k → ?B (Suc k)
  proof
    assume a: ?A k
    show ?thesis
    proof (cases ∀ n ≥ ?mss k. σ (?mss k) = σ n)
      case True
      hence ∀ n ≥ ?mss k. σ (?mss k) = σ n by (auto simp: le-less)
      with a have ?B k by simp

qed
with $b$ show $\neg \text{thesis}$ by (simp del: max-stutter-sampler.simps)

next

case False

hence $\text{diff} : \sigma (\text{?mss} (\text{Suc} \ k)) \neq \sigma (\text{?mss} \ k)$

by (blast dest: max-stutter-sampler-nostuttering)

have $\exists A (\text{Suc} \ k)$

proof (rule antisym)

show $f (\text{?mssf} (\text{Suc} \ k)) \leq \text{?mss} (\text{Suc} \ k)$

proof (rule ccontr)

assume $\neg \neg \text{thesis}$

hence $\exists i. f ((\text{?mssf} \ k) + \text{Suc} \ i) \neq \text{?mss} (\text{Suc} \ k)$ by simp

with $a$ show $(f \circ \text{?mssf}) \ k \leq \text{?mss} (\text{Suc} \ k)$

proof (rule strict-monoD)

simp

finally show $\exists F \ i$.

next

fix $i$

have $f (\text{?mssf} \ k + \ i) < \text{?mss} (\text{Suc} \ k)$ (is $\exists i. f ((\text{?mssf} \ k) + \text{Suc} \ i) \neq \text{?mss} (\text{Suc} \ k)$ by simp)

proof (induct $i$

from $a$ have $f (\text{?mssf} \ k + \ i) = \text{?mss} \ k$ by (simp add: o-def)

also from max-stutter-sampler[of $\sigma$, THEN stutter-sampler-mono]

have $\exists i. f ((\text{?mssf} \ k) + \text{Suc} \ i) \neq \text{?mss} (\text{Suc} \ k)$

proof (rule strict-monoD)

simp

finally show $\exists F \ i$.

next

fix $i$

assume $\neg \exists i. f ((\text{?mssf} \ k) + \text{Suc} \ i) \neq \text{?mss} (\text{Suc} \ k)$ by simp

proof (rule ccontr)

assume $\neg \exists \text{thesis}$

then have $\exists i. f ((\text{?mssf} \ k) + \text{Suc} \ i) \neq \text{?mss} (\text{Suc} \ k)$

by (simp add: o-def)

moreover from $\neg \exists \text{thesis}$ have $(f \circ \text{?mssf}) (\text{Suc} \ i) \neq \text{?mss} (\text{Suc} \ i)$

by blast

ultimately have $(f \circ \text{?mssf}) (\text{Suc} \ k) < f ((\text{?mssf} \ k + \text{Suc} \ i)$
by (simp add: less_le)
from \( f \) have \( \sigma (\text{\?mss} (\text{Suc} \ k)) = \sigma (f (\text{\?mssf} \ k + \text{\?i})) \)
proof (rule stutter-sampler-between)
from \( \text{ih} \) show \( f (\text{\?mssf} \ k + \text{\?i}) \leq \text{\?mss} (\text{Suc} \ k) \)
  by (simp add: o-def)
next
from \( \text{i} \) show \( \text{\?mss} (\text{Suc} \ k) < f (\text{Suc} (\text{\?mssf} \ k + \text{\?i})) \)
  by simp
qed
also from max-stutter-sampler have ... = \( \sigma (\text{\?mss} \ k) \)
proof (rule stutter-sampler-between)
from \( f \) \[ \text{THEN} \] stutter-sampler-mono, \text{THEN} strict-mono-mono
have \( f (\text{\?mssf} \ k) \leq f (\text{\?mssf} \ k + \text{\?i}) \) by (rule monoD) simp
with \( \text{a} \) show \( \text{\?mss} \ k \leq f (\text{\?mssf} \ k + \text{\?i}) \) by (simp add: o-def)
qed (rule \( \text{ih} \))
also note \( \text{diff} \)
finally show \( \text{False} \) by simp
qed
qed

\}
note \( \text{bounded} = \text{this} \)
from \( f \) \[ \text{THEN} \] stutter-sampler-mono
have strict-mono \( (\lambda \text{\?i}. f (\text{\?mssf} \ k + \text{\?i})) \)
  by (auto simp: strict-mono-def)
then obtain \( \text{i} \) where \( \text{i} : \text{\?mss} (\text{Suc} \ k) < f (\text{\?mssf} \ k + \text{\?i}) \)
  by (blast dest: strict-mono-exceeds)
from \( \text{bounded} \) have \( f (\text{\?mssf} \ k + \text{\?i}) < \text{\?mss} (\text{Suc} \ k) \).
with \( \text{i} \) show \( \text{False} \) by (simp del: max-stutter-sampler.simps)
qed
then obtain \( \text{m} \) where \( \text{m} : \text{m} > \text{\?mssf} \ k \) and \( \text{m'} : \text{f m} = \text{\?mss} (\text{Suc} \ k) \)
  by blast
show \( \text{\?mss} (\text{Suc} \ k) \leq f (\text{\?mssf} (\text{Suc} \ k)) \)
proof (rule ccontr)
assume \( \neg \text{\?thesis} \)
hence contr: \( f (\text{\?mssf} (\text{Suc} \ k)) < \text{\?mss} (\text{Suc} \ k) \) by simp
from mssf[\( \text{THEN} \) stutter-sampler-mono]
have \( (f \circ \text{\?mssf}) \ k < (f \circ \text{\?mssf}) (\text{Suc} \ k) \)
  by (rule strict-monoD) simp
with \( \text{a} \) have \( \text{\?mss} \ k \leq f (\text{\?mssf} (\text{Suc} \ k)) \)
  by (simp add: o-def)
from this contr have \( \sigma (f (\text{\?mssf} (\text{Suc} \ k))) = \sigma (\text{\?mss} \ k) \)
  by (rule stutter-sampler-between[\( \text{OF} \) max-stutter-sampler])
with \( \text{a} \) have stat: \( (\sigma \circ f) (\text{\?mssf} (\text{Suc} \ k)) = (\sigma \circ f) (\text{\?mssf} \ k) \)
  by (simp add: o-def)
from this \( \text{m} \) have \( (\sigma \circ f) \ m = (\sigma \circ f) (\text{\?mssf} \ k) \)
  by (blast intro: max-stutter-sampler-nostuttering)
with \( \text{diff m'} \) a show \( \text{False} \)
  by (simp add: o-def)
qed
qed

12
thus ?thesis ..

qed

next

assume ?B

with 

b

show ?thesis

by (simp del: max-stutter-sampler.simps)

qed

qed

}

hence ?σ = ?σ(σ ◦ f) unfolding stutter-reduced-def by force

thus ?thesis by (rule sym)

qed

end

theory StutterEquivalence

imports Samplers

begin

3 Stuttering Equivalence

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

definition stutter-equiv (infix ≈ 50) where

σ ≈ τ ≡ ?σ = ?τ

Stuttering equivalence is an equivalence relation.

lemma stutter-equiv-refl: σ ≈ σ

unfolding stutter-equiv-def ..

lemma stutter-equiv-sym [sym]: σ ≈ τ =⇒ τ ≈ σ

unfolding stutter-equiv-def by (rule sym)

lemma stutter-equiv-trans [trans]: ϱ ≈ σ =⇒ σ ≈ τ =⇒ ϱ ≈ τ

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 σ

shows σ ◦ f ≈ σ

using assms unfolding stutter-equiv-def by (rule sample-max-sample)

lemma stutter-reduced-equivalent: ?σ ≈ σ

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 se-
quences. 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** $\exists \sigma = \exists (\sigma \circ f)$ **by** (rule sample-max-sample\[THEN sym\])
- **also from** $eq$ **have** $\ldots = \exists (\tau \circ g)$ **by** simp
- **also from** $g$ **have** $\ldots = \exists \tau$ **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** $eq$: $\sigma \approx \tau$
- **and** $p$: $\forall f g. [ stutter-sampler f \sigma \wedge stutter-sampler g \tau; \sigma \circ f = \tau \circ g ] \implies P$
- **shows** $P$

**proof** (rule $p$)

- **from** $eq$ **show** $\sigma \circ (\max-stutter-sampler \sigma) = \tau \circ (\max-stutter-sampler \tau)$ **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:**

- $\sigma \approx \tau = (\exists f g. \text{stutter-sampler } f \sigma \wedge \text{stutter-sampler } g \tau \wedge \sigma \circ f = \tau \circ g)$
- **by** (blast intro: stutter-equiv elim: stutter-equivE)

The initial elements of stutter equivalent sequences are equal.

**lemma stutter-equiv-0:**

- **assumes** $\sigma \approx \tau$
- **shows** $\sigma \ 0 = \tau \ 0$

**proof**

- **have** $\sigma \ 0 = (\exists \sigma) \ 0$ **by** (rule stutter-reduced-0\[THEN sym\])
- **with** assms\[unfolded stutter-equiv-def\] **show** ?thesis **by** (simp add: stutter-reduced-0)

**qed**

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

**lemma suffix-stutter-equiv:**

- **assumes** $f$: stutter-sampler $f\ \sigma$
- **shows** $\sigma[f\ n\.\ ] \approx (\sigma \circ f)[n\.\ ]$
proof
from \( f \) have stutter-sampler \((\lambda k. f\ (n+k) - f\ n)\ (\sigma[f\ n..])\)
by (rule stutter-sampler-suffix)
moreover
have stutter-sampler id \(((\sigma \circ f)[n..])\)
by (rule id-stutter-sampler)
moreover
have \((\sigma[f\ n..])\circ(\lambda k. f\ (n+k) - f\ n) = ((\sigma \circ f)[n..]) \circ \text{id}\)
proof (rule ext, auto)
  fix \( i \)
  from \( f \) show \( \sigma\ (n+i) = \sigma\ (\sigma[f\ n..]) \circ \text{id} \)
proof
  (rule stutter-equivI)
qed
ultimately show \( \text{thesis} \)
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 \( \text{lo}: f\ k \leq n \ \text{and} \ \text{hi}: n < f\ (\text{Suc} k) \)
shows \( \sigma[n..] \approx \sigma[f\ k..]\)
proof
  have stutter-sampler id \((\sigma[n..])\) by (rule id-stutter-sampler)
moreover
from \( \text{lo} \) 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 \( i: i < \text{Suc} n - f\ k \)
  from \( f \) show \( \sigma\ (f\ k + i) = \sigma\ (f\ k) \)
proof (rule stutter-sampler-between)
    from \( \text{hi} \) show \( f\ k + i < f\ (\text{Suc} k) \) 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 \( \text{lo} \) show \( \sigma\ (n+i) = \sigma\ (f\ k + (n + i - f\ k)) \) by simp
qed
ultimately show \( \text{thesis} \) 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 $i$: $g i \leq n$ $n < g$ (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** ... $\approx (\tau \circ g)[i ..]$
  - **By (rule suffix-stutter-equiv)**
- **Also from** $eq$ **have** ... $= (\sigma \circ f)[i ..]$
  - **By simp**
- **Also from** $f$ **have** ... $\approx \sigma[f i ..]$
  - **By (rule suffix-stutter-equiv [THEN stutter-equiv-sym])**
- **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 $a$: $f a \leq j$ $j < f$ (Suc $a$)
    - **By (rule stutter-sampler-interval)**
  - **From** $a$ $j$ **have** $f a < f i$ **by simp**
    - **With** $f[THEN$ stutter-sampler-mono] **have** $a < i$
      - **By (simp add: strict-mono-less)**
    - **With** $g[THEN$ 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** ... $\approx (\sigma \circ f)[a ..]$
    - **By (rule suffix-stutter-equiv)**
  - **Also from** $eq$ **have** ... $= (\tau \circ g)[a ..]$ **by simp**
  - **Also from** $g$ **have** ... $\approx \tau[g a ..]$
    - **By (rule suffix-stutter-equiv [THEN stutter-equiv-sym])**
  - **Finally have** $\sigma[j ..] \approx \tau[g a ..]$.
with \( I \) have \( \exists k < n. \, \sigma[j..] \approx \tau[k..] \) by blast

moreover

note that

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

qed

theorem 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 \( \tau \approx \sigma \)

by (rule stutter-equiv-sym)

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

by (rule stutter-equiv-suffixes-left)

with that show \( \text{thesis} \)

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

proof –

from assms obtain \( m \) where \( \sigma[m..] \approx \tau[n..] \) and \( \forall j < m. \, \exists k < n. \, \sigma[k..] \approx \tau[j..] \)

by (rule stutter-equiv-suffixes-left)

with that show \( \text{thesis} \)

by (force dest: stutter-equiv-0)

qed

lemma stutter-equiv-element-right:

assumes \( \sigma \approx \tau \)

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

proof –

from assms obtain \( n \) where \( \sigma[m..] \approx \tau[n..] \) and \( \forall j < n. \, \exists k < m. \, \sigma[k..] \approx \tau[j..] \)

by (rule stutter-equiv-suffixes-right)

with that show \( \text{thesis} \)

by (force dest: stutter-equiv-0)

qed

end

theory PLTL

imports StutterEquivalence

begin
4 Stuttering Invariant PLTL Formulas

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

4.1 Syntax and semantics of PLTL

PLTL formulas are built from atomic formulas, propositional connectives, and the temporal operators “next” and “until”. The following data type definition is parameterized by the type of states over which formulas are evaluated.

```
datatype 'a pltl =
    false
  | atom 'a ⇒ bool
  | implies 'a pltl 'a pltl
  | next 'a pltl
  | until 'a pltl 'a pltl
```

The semantics of PLTL formulas is defined inductively w.r.t. \( \omega \)-sequences of states.

```
fun holds-of :: ('a ⇒ bool ⇒ bool) ⇒ 'a pltl ⇒ bool (- | - [70,70] 40) where
  (σ | false) = False
  | (σ | atom p) = p (σ 0)
  | (σ | implies ϕ ψ) = ((σ | ϕ) −→ (σ | ψ))
  | (σ | next ϕ) = (σ[1..] |= ϕ)
  | (σ | until ϕ ψ) = (∃ k. σ[k..] |= ψ ∧ (∀ i<k. σ[i..] |= ϕ))
```

Further connectives of PLTL can be defined as abbreviations.

```
definition not where not ϕ = implies ϕ false

definition true where true = not false

definition or where or ϕ ψ = implies (not ϕ) ψ

definition and where and ϕ ψ = not (or (not ϕ) (not ψ))

definition eventually where eventually ϕ = until true ϕ

definition always where always ϕ = not (eventually (not ϕ))
```

These definitions give rise to the expected semantics.

```
lemma holds-of-not [simp]: (σ |= not ϕ) = (¬(σ |= ϕ))
  by (simp add: not-def)

lemma holds-of-true [simp]: σ |= true
  by (simp add: true-def)
```
lemma holds-of-or [simp]: \( (\sigma \models \text{or} \ \varphi \ \psi) = (\sigma \models \varphi \lor \sigma \models \psi) \)
by (auto simp add: or-def)

lemma holds-of-and [simp]: \( (\sigma \models \text{and} \ \varphi \ \psi) = (\sigma \models \varphi \land \sigma \models \psi) \)
by (simp add: and-def)

lemma holds-of-eventually [simp]: \( (\sigma \models \text{eventually} \ \varphi) = (\exists k. \sigma[k..] \models \varphi) \)
by (simp add: eventually-def)

lemma holds-of-always [simp]: \( (\sigma \models \text{always} \ \varphi) = (\forall k. \sigma[k..] \models \varphi) \)
by (simp add: always-def)

We also define finite conjunctions and disjunctions.

definition OR where \( \text{OR} \ \Phi \equiv \text{SOME} \ \varphi. \ \text{fold-graph or false} \ \Phi \ \varphi \)
definition AND where \( \text{AND} \ \Phi \equiv \text{SOME} \ \varphi. \ \text{fold-graph and true} \ \Phi \ \varphi \)

lemma fold-graph-OR: finite \( \Phi \implies \text{fold-graph or false} \ (\text{OR} \ \Phi) \)
unfolding OR-def by (rule someI2-ex[OF finite-imp-fold-graph])

lemma fold-graph-AND: finite \( \Phi \implies \text{fold-graph and true} \ (\text{AND} \ \Phi) \)
unfolding AND-def by (rule someI2-ex[OF finite-imp-fold-graph])

lemma holds-of-OR [simp]:
assumes fin: finite \( (\Phi::'a \text{ pltl set}) \)
shows \( (\sigma \models \text{OR} \ \Phi) = (\exists \varphi \in \Phi. \ \sigma \models \varphi) \)
proof –
{  
  fix \( \psi::'a \text{ pltl} \)
  assume fold-graph or false \( \Phi \ \psi \)
  hence \( (\sigma \models \psi) = (\exists \varphi \in \Phi. \ \sigma \models \varphi) \)
  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 \( (\Phi::'a \text{ pltl set}) \)
shows \( (\sigma \models \text{AND} \ \Phi) = (\forall \varphi \in \Phi. \ \sigma \models \varphi) \)
proof –
{  
  fix \( \psi::'a \text{ pltl} \)
  assume fold-graph and true \( \Phi \ \psi \)
  hence \( (\sigma \models \psi) = (\forall \varphi \in \Phi. \ \sigma \models \varphi) \)
  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 \( t_{\text{next}} f \).

```plaintext
fun next-free where
  next-free false = True
  | next-free (atom p) = True
  | next-free (implies \( \varphi \) \( \psi \)) = (next-free \( \varphi \) \& next-free \( \psi \))
  | next-free (next \( \varphi \)) = False
  | next-free (until \( \varphi \) \( \psi \)) = (next-free \( \varphi \) \& next-free \( \psi \))

lemma next-free-not [simp]: next-free (not \( \varphi \)) = next-free \( \varphi \)
  by (simp add: not-def)

lemma next-free-true [simp]: next-free true
  by (simp add: true-def)

lemma next-free-or [simp]: next-free (or \( \varphi \) \( \psi \)) = (next-free \( \varphi \) \& next-free \( \psi \))
  by (simp add: or-def)

lemma next-free-and [simp]: next-free (and \( \varphi \) \( \psi \)) = (next-free \( \varphi \) \& next-free \( \psi \))
  by (simp add: and-def)

lemma next-free-eventually [simp]: next-free (eventually \( \varphi \)) = next-free \( \varphi \)
  by (simp add: eventually-def)

lemma next-free-always [simp]: next-free (always \( \varphi \)) = next-free \( \varphi \)
  by (simp add: always-def)

lemma next-free-OR [simp]:
  assumes fin: finite \( \Phi :: \prime a \text{ pltl set} \)
  shows next-free (OR \( \Phi \)) = (\( \forall \varphi \in \Phi . \) next-free \( \varphi \))
proof –
  { fix \( \psi :: \prime a \text{ pltl} \)
    assume fold-graph or false \( \Phi \) \( \psi \)
    hence next-free \( \psi \) = (\( \forall \varphi \in \Phi . \) next-free \( \varphi \))
    by (rule fold-graph.induct) auto
  }
  with fold-graph-OR[OF fin] show \( \text{thesis} \) by simp
qed

lemma next-free-AND [simp]:
  assumes fin: finite \( \Phi :: \prime a \text{ pltl set} \)
  shows next-free (AND \( \Phi \)) = (\( \forall \varphi \in \Phi . \) next-free \( \varphi \))
proof –
  { fix \( \psi :: \prime a \text{ pltl} \)
    assume fold-graph and true \( \Phi \) \( \psi \)
...}
```
hence \( \text{next-free } \psi = (\forall \varphi \in \Phi. \text{next-free } \varphi) \)

by \( \text{rule fold-graph.induct auto} \)

\}

with fold-graph-AND[OF fin] show \( \text{thesis by simp} \)

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

\[
\text{stutter-invariant } \varphi = (\forall \sigma \tau. \sigma \approx \tau \rightarrow (\sigma \models \varphi) = (\tau \models \varphi))
\]

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 \( \langle \sigma \tau. [[\sigma \approx \tau; \sigma \models \varphi] \implies \tau \models \varphi \rangle \)

shows stutter-invariant \( \varphi \)

proof −

\{ fix \( \sigma \tau \)

assume st: \( \sigma \approx \tau \) and f: \( \sigma \models \varphi \)

hence \( \tau \models \varphi \) by \( \text{rule assms} \)

\}

moreover

\{ fix \( \sigma \tau \)

assume st: \( \sigma \approx \tau \) and f: \( \tau \models \varphi \)

from st have \( \tau \approx \sigma \) by \( \text{rule stutter-equiv-sym} \)

from this f have \( \sigma \models \varphi \) by \( \text{rule assms} \)

\}

ultimately show \( \text{thesis by } (\text{auto simp: stutter-invariant-def}) \)

qed

lemma stutter-invariantD [dest]:

assumes stutter-invariant \( \varphi \) and \( \sigma \approx \tau \)

shows \( (\sigma \models \varphi) = (\tau \models \varphi) \)

using assms by \( (\text{auto simp: stutter-invariant-def}) \)

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 \( \varphi \implies \text{stutter-invariant } (\varphi::'a \text{ pltl}) \)

proof \( \text{(induct } \varphi) \)

\shows \( \text{stutter-invariant false by auto} \)

next

fix \( p :: 'a \Rightarrow \text{bool} \)

21
show stutter-invariant (atom p)
proof
  fix σ τ
  assume σ ≈ τ σ |= atom p
  thus τ |= atom p by (simp add: stutter-equiv-0)
qed
next
fix ϕ ψ :: 'a pltl
assume ih: next-free ϕ ==> stutter-invariant ϕ
next-free ψ ==> stutter-invariant ψ
assume next-free (implies ϕ ψ)
with ih show stutter-invariant (implies ϕ ψ) by auto
next
fix ϕ ψ :: 'a pltl
assume ih: next-free (next ϕ) — hence contradiction
thus stutter-invariant (next ϕ) by simp
next
fix ϕ ψ :: 'a pltl
assume ih: next-free ϕ ==> stutter-invariant ϕ
next-free ψ ==> stutter-invariant ψ
assume next-free (until ϕ ψ)
with ih have stinv: stutter-invariant ϕ stutter-invariant ψ by auto
show stutter-invariant (until ϕ ψ)
proof
  fix σ τ
  assume st: σ ≈ τ and unt: σ |= until ϕ ψ
  from unt obtain m
  where 1: σ[m ..] |= ψ and 2: ∀ j < m. σ[j ..] |= ϕ by auto
  from st obtain n
  where 3: σ[m ..] ≈ τ[n ..] and 4: ∀ i < n. ∃ j < m. σ[j ..] ≈ τ[i ..]
  by (rule stutter-equiv-suffixes-right)
  from 1 3 stinv have τ[n ..] |= ψ by auto
  moreover
  from 2 4 stinv have ∀ i < n. τ[i ..] |= ϕ by force
  ultimately show τ |= until ϕ ψ by auto
qed
qed

4.4 Atoms, Canonical State Sequences, and Characteristic Formulas

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

The following function collects the atoms (of type 'a ⇒ bool) of a PLTL formula.
fun atoms where
  atoms false = {}
| atoms (atom p) = {p}
| atoms (implies φ ψ) = atoms φ ∪ atoms ψ
| atoms (next φ) = atoms φ
| atoms (until φ ψ) = atoms φ ∪ atoms ψ

lemma atoms-finite [iff]: finite (atoms φ)
  by (induct φ) auto

lemma atoms-not [simp]: atoms (not φ) = atoms φ
  by (simp add: not-def)

lemma atoms-true [simp]: atoms true = {}
  by (simp add: true-def)

lemma atoms-or [simp]: atoms (or φ ψ) = atoms φ ∪ atoms ψ
  by (simp add: or-def)

lemma atoms-and [simp]: atoms (and φ ψ) = atoms φ ∪ atoms ψ
  by (simp add: and-def)

lemma atoms-eventually [simp]: atoms (eventually φ) = atoms φ
  by (simp add: eventually-def)

lemma atoms-always [simp]: atoms (always φ) = atoms φ
  by (simp add: always-def)

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

lemma atoms-AND [simp]:
  assumes fin: finite (Φ::'a pltl set)
  shows atoms (AND Φ) = (⋃ϕ∈Φ. atoms ϕ)
proof –
  { fix ψ::'a pltl
    assume fold-graph and true Φ ψ
    hence atoms ψ = (⋃ϕ∈Φ. atoms 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 $\sigma$ and $\tau$ are $A$-similar if corresponding states are $A$-equal.

**definition** state-sim :: ['a, ('a ⇒ bool) set, 'a ⇒ bool
(· ~ · - [70,100,70] 50) where
\begin{align*}
\sigma \sim A \sim \tau &= (\forall p \in A. \ p s \leftrightarrow p t)
\end{align*}

**definition** seq-sim :: [nat ⇒ 'a, ('a ⇒ bool) set, nat ⇒ 'a ⇒ bool
(· ⊳ · - [70,100,70] 50) where
\begin{align*}
\sigma \simeq A \simeq \tau &= (\forall n. \ (\sigma n) \sim A \sim (\tau n))
\end{align*}

These relations are (indexed) equivalence relations. Moreover $s \sim A \sim t$ implies $s \sim B \sim t$ for $B \subseteq A$, and similar for $\sigma \simeq A \simeq \tau$ and $\sigma \simeq B \simeq \tau$.

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

**lemma** state-sim-sym: $s \sim A \sim t \Longrightarrow t \sim A \sim s$
by (auto simp: state-sim-def)

**lemma** state-sim-trans [trans]: $s \sim A \sim t \Longrightarrow t \sim A \sim u \Longrightarrow s \sim A \sim u$
unfolding state-sim-def by blast

**lemma** seq-sim-refl [simp]: $\sigma \simeq A \simeq \sigma$
by (simp add: seq-sim-def)

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

**lemma** seq-sim-trans [trans]: $\sigma \simeq A \simeq \sigma \Longrightarrow \sigma \simeq A \simeq \tau \Longrightarrow \sigma \simeq A \simeq \tau$
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 state-sim-def by auto

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: $\sigma \simeq (\text{atoms } \varphi) \simeq \tau \Longrightarrow (\sigma \models \varphi) = (\tau \models \varphi)$
proof (induct \(\varphi\) arbitrary: \(\sigma\) \(\tau\))
  fix \(\sigma\) \(\tau\)
  show \(\neg P\ \sigma\ \false\ \tau\) by simp
next
  fix \(p\ \sigma\ \tau\)
  assume \(?\mathrm{sim}\ \sigma\ (\text{atom}\ p)\ \tau\ \therefore\ \neg P\ \sigma\ \text{atom}\ p\ \tau\)
  by (auto simp: seq-sim-def state-sim-def)
next
  fix \(\varphi\ \psi\ \sigma\ \tau\)
  assume \(\mathrm{ih}\): \(\forall\ \sigma\ \tau.\ \neg P\ \sigma\ \varphi\ \tau\ \Rightarrow\ \neg P\ \sigma\ \psi\ \tau\ \Rightarrow\ \neg P\ \sigma\ \psi\ \tau\)
  and \(\mathrm{sim}\): \(?\mathrm{sim}\ \sigma\ (\text{implies} \ \varphi\ \psi)\ \tau\)
  from \(\mathrm{sim}\) have \(?\mathrm{sim}\ \sigma\ \varphi\ \tau\ \neg P\ \sigma\ \psi\ \tau\)
  by (auto elim: seq-sim-mono)
with \(\mathrm{ih}\) show \(\neg P\ \sigma\ (\text{implies} \ \varphi\ \psi)\ \tau\) by simp
next
  fix \(\varphi\ \psi\ \sigma\ \tau\)
  assume \(\mathrm{ih}\): \(\forall\ \sigma\ \tau.\ \neg P\ \sigma\ \varphi\ \tau\ \Rightarrow\ \neg P\ \sigma\ \psi\ \tau\ \Rightarrow\ \neg P\ \sigma\ \psi\ \tau\)
  and \(\mathrm{sim}\): \(?\mathrm{sim}\ \sigma\ \text{atoms} \ (\text{next} \ \varphi)\ \tau\)
  from \(\mathrm{sim}\) have \(?\mathrm{sim}\ \sigma\ [1..] \text{atoms} \ ((\text{next} \ \varphi) \ \tau[1..]\)
  by (auto simp: seq-sim-def)
with \(\mathrm{ih}\) show \(?P\ \sigma\ \text{next} \ \varphi\ \tau\) by auto
next
  fix \(\varphi\ \psi\ \sigma\ \tau\)
  assume \(\mathrm{ih}\): \(\forall\ \sigma\ \tau.\ \neg P\ \sigma\ \varphi\ \tau\ \Rightarrow\ \neg P\ \sigma\ \psi\ \tau\ \Rightarrow\ \neg P\ \sigma\ \psi\ \tau\)
  and \(\mathrm{sim}\): \(?\mathrm{sim}\ \sigma\ \text{until} \ \varphi\ \psi\ \tau\)
  from \(\mathrm{sim}\) have \(\forall\ i.\ \sigma[i..] \text{atoms} \ \varphi\ \tau[i..] \ \forall\ j.\ \sigma[j..] \text{atoms} \ \psi\ \tau[j..]\)
  by (auto simp: seq-sim-def state-sim-def)
with \(\mathrm{ih}\) have \(\forall\ i.\ \neg P\ \sigma[i..] \ \varphi\ \tau[i..] \ \forall\ j.\ \neg P\ \sigma[j..] \ \psi\ \tau[j..]\)
  by (auto simp del: suffix-elt)
thus \(?P\ \sigma\ \text{until} \ \varphi\ \psi\ \tau\) by (auto simp del: suffix-elt)
qed

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 where
  canonize \(A\) \(s\) \equiv SOME \(t\). \(\sim A\) \(s\)

lemma canonize-state-sim: canonize \(A\) \(s\) \(\sim A\) \(s\)
unfolding canonize-def by (rule someI, rule state-sim-refl)

lemma canonize-canonical:
  assumes \(st\): \(\sim A\) \(t\)
  shows canonize \(A\) \(s\) = canonize \(A\) \(t\)
proof
  from \(st\) have \(\forall\ u.\ (u \sim A\ s) = (u \sim A\ t)\)
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 where
canonical-sequence A σ ≡ ∀ m (n::nat). σ m ~ A~ σ n → σ m = σ 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 σ → canonical-sequence A (σ[k..])
by (auto simp: canonical-sequence-def)

lemma canonical-sampled:
canonical-sequence A σ → canonical-sequence A (σ o f)
by (auto simp: canonical-sequence-def)

lemma canonical-reduced:
canonical-sequence A σ → canonical-sequence A (♮σ)
unfolding stutter-reduced-def by (rule canonical-sampled)

For any sequence σ there exists a canonical A-similar sequence τ. Such a τ can be obtained by canonizing all states of σ.

lemma canonical-exists:
obtains τ where τ ≃ A≃ σ canonical-sequence A τ
proof –
have (canonize A o σ) ≃ A≃ σ
  by (simp add: seq-sim-def canonize-state-sim)
moreover
have canonical-sequence A (canonize A o σ)
  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 ≡
  (and (AND { atom p | p . p ∈ A ∧ p s })
  (AND { not (atom p) | p . p ∈ A ∧ ¬(p s) }))
lemma characteristic-holds:
finite $A \implies \sigma \models \text{characteristic-formula } A (\sigma \ 0)$
by (auto simp: characteristic-formula-def)

lemma characteristic-state-sim:
assumes fin: finite $A$
shows $(\sigma \ 0 \sim A \sim \tau \ 0) = (\tau \models \text{characteristic-formula } A (\sigma \ (0::\text{nat})))$
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 $\tau \models \text{characteristic-formula } A (\sigma \ 0)$
  by (auto simp: characteristic-formula-def) (blast+)
next
assume $\tau \models \text{characteristic-formula } A (\sigma \ 0)$
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 $A \supseteq \text{atoms } \varphi$).

lemma ex-next-free-stutter-free-canonical:
assumes A: $\text{atoms } \varphi \subseteq A$ and fin: finite $A$
shows $\exists \psi. \ \text{next-free } \psi \land \text{atoms } \psi \subseteq A \land$
  $(\forall \sigma. \ \text{stutter-free } \sigma \land \text{canonical-sequence } A \sigma \rightarrow (\sigma \models \psi) = (\sigma \models \varphi))$
(is $\exists \psi. \ \text{P } \varphi \ \psi$)
using A proof (induct $\varphi$)

The cases of false and atomic formulas are trivial.

  have $\text{P } \text{false false}$ by auto
  thus $\exists \psi. \ \text{P } \text{false } \psi$ ..

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

next

Implication is easy, using the induction hypothesis.

  fix $\varphi \ \psi$
assume $\text{atoms } \varphi \subseteq A \implies \exists \psi'. \ ?P \ \varphi \ \varphi'$

and $\text{atoms } \psi \subseteq A \implies \exists \psi'. \ ?P \ \psi \ \psi'$

and $\text{atoms } (\text{implies } \varphi \ \psi) \subseteq A$

then obtain $\varphi' \ \psi'$ where $\ ?P \ \varphi \ \varphi'$ $\ ?P \ \psi \ \psi'$ by auto

hence $\ ?P \ (\text{implies } \varphi \ \psi) \ (\text{implies } \varphi' \ \psi')$ by auto

thus $\exists \chi. \ ?P \ (\text{implies } \varphi \ \chi)$ ..

next

The case of $\text{until}$ follows similarly.

fix $\varphi \ \psi$

assume $\text{atoms } \varphi \subseteq A \implies \exists \varphi'. \ ?P \ \varphi \ \varphi'$

and $\text{atoms } \psi \subseteq A \implies \exists \psi'. \ ?P \ \psi \ \psi'$

and $\text{atoms } (\text{until } \varphi \ \psi) \subseteq A$

then obtain $\varphi' \ \psi'$ where $1: \ ?P \ \varphi \ \varphi'$ and $2: \ ?P \ \psi \ \psi'$ by auto

{ fix $\sigma$

assume sigma: $\text{stutter-free } \sigma \ \text{canonical-sequence } A \ \sigma$

hence $\bigwedge \ k. \ (\sigma[k..] \models \varphi) \ \bigwedge \ k. \ (\text{canonical-sequence } A \ (\sigma[k..]))$

by (auto simp: stutter-free-suffix canonical-suffix)

with 1 2

have $\bigwedge \ k. \ (\sigma[k..] \models \varphi) = (\sigma[k..] \models \varphi)$

and $\bigwedge \ k. \ (\sigma[k..] \models \psi) = (\sigma[k..] \models \psi)$

by (blast+)

hence $(\sigma \models \text{until } \varphi' \ \psi') = (\sigma \models \text{until } \varphi \ \psi)$

by auto

}

with 1 2 have $\ ?P \ (\text{until } \varphi \ \psi) \ (\text{until } \varphi' \ \psi')$ by auto

thus $\exists \chi. \ ?P \ (\text{until } \varphi \ \psi) \ \chi$ ..

next

The interesting case is the one of the $\text{next}$-operator.

fix $\varphi$

assume $\text{ih}: \text{atoms } \varphi \subseteq A \implies \exists \psi. \ ?P \ \varphi \ \psi$ and $\text{at}: \text{atoms } (\text{next } \varphi) \subseteq A$

then obtain $\psi$ where $\text{psi}: \ ?P \ \varphi \ \psi$ by auto

A valuation (over $A$) is a set $\text{val} \subseteq A$ 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 $\text{next } \varphi$ over the stutter-free and canonical sequence $\sigma$.

\[
\begin{align*}
\text{def } \text{stval} & \equiv \lambda s. \{ p \in A . \ p \ S \} \\
\text{def } \text{chi} & \equiv \lambda \text{val}. \ (\text{and } \ (\text{AND } \{ \text{atom } p \ | \ p . \ p \in \text{val}) \}) \\
& \ (\text{AND } \{ \text{not } (\text{atom } p) \ | \ p . \ p \in A \ - \ \text{val})\}) \\
\text{def } \psi' & \equiv (\text{or } (\text{and } \psi \ (\text{OR } \{ \text{always } (\text{chi val}) \ | \ \text{val} . \ \text{val} \subseteq A \}) \)) \\
& \ (\text{OR } \{ \text{and } \text{chi val} \ (\text{until } \text{chi val} \ (\text{and } \psi \ (\text{chi val}')) | \ \text{val}' . \ \text{val} \subseteq A \ - \ \text{val}') \subseteq A \and \text{val}' \ - \ \text{val} \}) \\
& \ (\text{is } (\text{or } (\text{and } (\text{OR } \text{ALW})) \ (\text{OR } \text{UNT})))) \\
\text{have } \bigwedge s. \ { \text{not } (\text{atom } p) | p . p \in A \ - \ \text{stval} s} \\
& \ = \ { \text{not } (\text{atom } p) | p . p \in A \ - \ \text{stval} s} \\
\end{align*}
\]
by (auto simp: stval-def)

hence \chi_1: \bigwedge s. \chi (stval s) = \text{characteristic-formula } A \ s
  by (auto simp: chi-def stval-def characteristic-formula-def)

\{ 
  fix val \tau
  assume val: val \subseteq A \text{ and } tau: \tau \models \chi val
  with fin have val = stval (\tau 0)
  by (auto simp: stval-def chi-def finite-subset)
\}

note chi2 = this

have \(\exists \text{UNT} \subseteq (\lambda(val,val'). \text{ and } (\chi val) \text{ (until } (\chi val) \text{ (and } \psi (\chi val'))))\)
  \(\text{ (Pow } A \times \text{ Pow } A)\)
  (is - \subseteq \?S)
  by auto

with fin have fin-UNT: finite \?UNT
  by (auto simp: finite-subset)

have nf: next-free psi’
proof –
  from fin have \(\forall val. \text{ val } \subseteq A \Rightarrow \text{ next-free } (\chi val)\)
  by (auto simp: chi-def finite-subset)

with fin fin-UNT psi show ?thesis
  by (force simp: psi’-def finite-subset)

qed

have atoms: atoms psi’ \subseteq A
proof –
  from fin have at-chi: \(\forall val. \text{ val } \subseteq A \Rightarrow \text{ atoms } (\chi val) \subseteq A\)
  by (auto simp: chi-def finite-subset)

with fin psi have at-alw: atoms (\text{ and } \psi (\text{ OR } \text{ ?ALW}) ) \subseteq A
  by auto blast?

from fin fin-UNT psi at-chi have atoms (\text{ OR } \text{ ?UNT}) \subseteq A
  by auto (blast+)?

with at-alw show ?thesis by (auto simp: psi’-def)

qed

\{ 
  fix \sigma
  assume st: stutter-free \sigma \text{ and } can: \text{ canonical-sequence } A \ \sigma
  have (\sigma \models \text{ next } \varphi) = (\sigma \models psi’)
  proof (cases \sigma (\text{Suc } 0) = \sigma 0)
    case True
  
  In the case of a stuttering transition at the beginning, we must have infinite stuttering, and the first disjunct of psi’ holds, whereas the second does not.

  \{ 
    fix n
    have \sigma \ n = \sigma\ 0
  \}
proof (cases n)
  case 0 thus ?thesis by simp
next
  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 \( (\sigma[n..]) i = \sigma 0 \) by (auto intro: alleq)
  moreover have \( \sigma i = \sigma 0 \) by (rule alleq)
  ultimately show \( (\sigma[n..]) i = \sigma i \) by simp
qed

with st can psi have 1: \( (\sigma \models next \varphi) = (\sigma \models \psi) \) by simp

from fin have \( \sigma \models \chi (stval (\sigma 0)) \) by (simp add: chi1 characteristic-holds)
with suffix have \( \sigma \models always (chi (stval (\sigma 0))) \) (is - \( \models ?alw \) by simp
moreover have \( ?alw \in ?ALW \) by (auto simp: stval-def)
ultimately have 2: \( \sigma \models OR ?ALW \)
  using fin by (auto simp: finite-subset simp del: holds-of-always)

have 3: \( \neg (\sigma \models OR ?UNT) \)
proof
  assume unt: \( \sigma \models OR ?UNT \)
  with fin-UNT obtain val val' k where
    val: \( val \subseteq A \) \( val' \subseteq A \) \( val' \neq val \) and
    now: \( \sigma \models \chi val \) and \( k: \sigma[k..] = \chi val' \)
    by auto (blast+)
  from \( val \subseteq A \) now have \( val = stval (\sigma 0) \) by (rule chi2)
  moreover
  from \( val' \subseteq A \) \( k \) suffix have \( val' = stval (\sigma 0) \) by (simp add: chi2)
  moreover note \( val' \neq val \)
  ultimately show False by simp
qed

from 1 2 3 show ?thesis by (simp add: psi'-def)

next
  case False

Otherwise, \( \sigma \models next \varphi \) is equivalent to \( \sigma \) satisfying the second disjunct of \( psi' \). We show both implications separately.

let \( ?val = stval (\sigma 0) \)
let \( ?val' = stval (\sigma 1) \)
from False can have vals: \( ?val' \neq ?val \)
  by (auto simp: canonical-sequence-def state-sim-def stval-def)
show ¬thesis

proof
assume phi: σ |= next φ
from fin have 1: σ |= chi ?val by (simp add: chi characteristic-holds)
from st can have stutter-free (σ[1..]) canonical-sequence A (σ[1..])
  by (auto simp: stutter-free-suffix canonical-suffix)
with phi psi have 2: σ[1..] |= ψ by auto
from fin have σ[1..] |= characteristic-formula A ((σ[1..]) 0)
  by (rule characteristic-holds)
hence 3: σ[1..] |= chi ?val’ by (simp add: chi)
from 1 2 3 have σ |= and (chi ?val) (until (chi ?val) (and ψ (chi ?val’)))
  (is |- ?unt)
  by auto
moreover from vals have ?unt ∈ ?UNT
  by (auto simp: stval-def)
ultimately have σ |= OR ?UNT
  using fin-UNT[THEN holds-of-OR] by blast
thus σ |= psi’ by (simp add: psi’-def)

next
assume psi’: σ |= psi’
have ¬(σ |= OR ?ALW)
proof
assume σ |= OR ?ALW
with fin obtain val where 1: val ⊆ A and 2: ∀ n. σ[n..] |= chi val
  by (force simp: finite-subset)
from 2 have σ[0..] |= chi val ..
with 1 have val = ?val by (simp add: chi2)
moreover
from 2 have σ[1..] |= chi val ..
with 1 have val = ?val’ by (force dest: chi2)
ultimately
show False using vals by simp
qed

with psi’ have σ |= OR ?UNT by (simp add: psi’-def)
with fin-UNT obtain val val’ k where
  val: val ⊆ A val’ ⊆ A val’ ≠ val and
  now: σ |= chi val and
  k: σ[k..] |= ψ σ[k..] |= chi val’ and
  i: ∀ i<k. σ[i..] |= chi val
  by auto (blast+)?
from val now have 1: val = ?val by (simp add: chi2)
have 2: k ≠ 0

31
Comparing the definition of the next-free formula in the case of formulas
next $\varphi$ with the one that appears in [2], there is a subtle difference. Peled
and Wilke define the second disjunct as a disjunction of formulas

$$\text{until (chi val) (and (psi (chi val')))}$$

for subsets val, val' $\subseteq A$ whereas we conjoin the formula \textit{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 \textit{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 next operator.

\textbf{theorem} stutter-invariant-next-free:

```plaintext
proof
  assume k=0
  with val k have val' = ?val by (simp add: chi2)
  with l (val' $\neq$ val) show False by simp
qed

have 3: $k \leq 1$
proof (rule ccontr)
  assume $\neg (k \leq 1)$
  with i have $\sigma[1..] \models \text{chi val}$ by simp
  with i have $\sigma[1..] \models \text{characteristic-formula A (} \sigma \ 0)$
    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 $\sigma \models \text{next } \varphi$ using ($\sigma[k..] \models \psi$) psi by auto
qed
```

32
assumes phi: stutter-invariant \( \varphi \)
obtains \( \psi \) where next-free \( \psi \) atoms \( \psi \subseteq \text{atoms} \ \varphi \)
\[ \forall \sigma. (\sigma \models \psi) = (\sigma \models \varphi) \]

proof –

have atoms \( \varphi \subseteq \text{atoms} \ \varphi \) finite (\( \text{atoms} \ \varphi \)) by simp\-all
then obtain \( \psi \) where

\( \psi \): next-free \( \psi \) atoms \( \psi \subseteq \text{atoms} \ \varphi \) and

equiv: \( \forall \sigma. \text{stutter-free} \ \sigma \land \text{canonical-sequence} (\text{atoms} \ \varphi \ \sigma \rightarrow (\sigma \models \psi) = (\sigma \models \varphi) \]
by (blast dest: ex\-next-free-stutter\-free\-canonical)

from (next-free \( \psi \)) have sinv: stutter\-invariant \( \psi \)
by (rule next-free\-stutter\-invariant)

\{
fix \( \sigma \)

obtain \( \tau \) where

1: \( \tau \simeq \text{atoms} \ \varphi \simeq \sigma \) and

by (rule canonical\-exists)

from 1 (atoms \( \psi \subseteq \text{atoms} \ \varphi \)) have 3: \( \tau \simeq \text{atoms} \ \psi \simeq \sigma \)
by (rule seq\-sim\-mono)

from 1 have \( (\sigma \models \varphi) = (\tau \models \varphi) \) by (simp add: pltl\-seq\-sim)
also from phi stutter\-reduced\-equivalent have \( ... = (\tau \models \varphi) \) by auto
also from 2 THEN canonical\-reduced\-stutter\-free
have \( ... = (\tau \models \psi) \) by auto
also from sinv stutter\-reduced\-equivalent have \( ... = (\tau \models \psi) \) by auto
also from 3 have \( ... = (\sigma \models \psi) \) by (simp add: pltl\-seq\-sim)
finally have \( (\sigma \models \psi) = (\sigma \models \varphi) \) by (rule sym)
\}

with psi that show \( ? \)thesis by blast

qed

Combining theorems next\-free\-stutter\-invariant and stutter\-invariant\-next\-free,
it follows that a PLTL formula is stuttering invariant iff it is equivalent to
a next\-free formula.

theorem pltl\-stutter\-invariant:

\[ \text{stutter-invariant} \ \varphi \iff \]
\[ (\exists \psi. \text{next-free} \ \psi \land \text{atoms} \ \psi \subseteq \text{atoms} \ \varphi \land (\forall \sigma. \sigma \models \psi \iff \sigma \models \varphi)) \]

proof –

\{

assume stutter\-invariant \( \varphi \)

hence \( \exists \psi. \text{next-free} \ \psi \land \text{atoms} \ \psi \subseteq \text{atoms} \ \varphi \land (\forall \sigma. \sigma \models \psi \iff \sigma \models \varphi) \)

by (rule stutter\-invariant\-next\-free) blast
\}

moreover

\{

fix \( \psi \)

assume 1: next\-free \( \psi \) and 2: \( \forall \sigma. \sigma \models \psi \iff \sigma \models \varphi \)

from 1 have stutter\-invariant \( \psi \) by (rule next\-free\-stutter\-invariant)

with 2 have stutter\-invariant \( \varphi \) by blast
\}
ultimately show \textit{?thesis by blast}

qed

end

References
