

Stuttering Equivalence and Stuttering Invariance

Stephan Merz
Inria Nancy & LORIA
Villers-lès-Nancy, France

April 10, 2026

Two ω -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

4	Stuttering Invariant LTL Formulas	17
4.1	Finite Conjunctions and Disjunctions in PLTL	17
4.2	Next-Free PLTL Formulas	18
4.3	Stuttering Invariance of PLTL Without “Next”	20
4.4	Atoms, Canonical State Sequences, and Characteristic Formulas	21
4.5	Stuttering Invariant PLTL Formulas Don’t Need Next	25
4.6	Stutter Invariance for LTL with Syntactic Sugar	32

```

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  $\Rightarrow$  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  $Suc\ n \leq f\ k$  by simp
  also from f have  $f\ k < f\ (Suc\ k)$  by (rule strict-monoD) simp
  finally show  $\exists k. Suc\ n < f\ k$  ..
qed

```

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

```

lemma strict-mono-interval:
  assumes f: strict-mono (f::nat  $\Rightarrow$  nat) and n:  $f\ 0 \leq n$ 
  obtains k where  $f\ k \leq n$  and  $n < f\ (Suc\ k)$ 
proof –
  from f[THEN strict-mono-exceeds] obtain m where  $m: n < f\ m$  ..
  have  $m \neq 0$ 
  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  $g: \text{strict-mono } (g::'a::\text{order} \Rightarrow 'b::\text{order})$ 
    and  $f: \text{strict-mono } (f::'b::\text{order} \Rightarrow 'c::\text{order})$ 
  shows  $\text{strict-mono } (f \circ g)$ 
  using assms by (auto simp: strict-mono-def)

```

2 Stuttering Sampling Functions

Given an ω -sequence σ , 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 σ and f , the (stuttering-)sampled reduction of σ is the sequence of elements of σ 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  $(f::\text{nat} \Rightarrow \text{nat})\ \sigma \equiv$ 
   $f\ 0 = 0$ 
   $\wedge \text{strict-mono } f$ 
   $\wedge (\forall k\ n. f\ k < n \wedge n < f\ (Suc\ k) \longrightarrow \sigma\ n = \sigma\ (f\ k))$ 

```

```

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

```

```

lemma stutter-sampler-mono: stutter-sampler  $f\ \sigma \implies \text{strict-mono } f$ 
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 σ*
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 σ .

lemma *id-stutter-sampler [iff]: stutter-sampler id σ*
by (*auto simp: stutter-sampler-def strict-mono-def*)

Stuttering sampling functions compose, sort of.

lemma *stutter-sampler-comp*:
assumes *f: stutter-sampler f σ*
and *g: stutter-sampler g ($\sigma \circ f$)*
shows *stutter-sampler (f \circ g) σ*
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 σ*
shows *stutter-sampler ($\lambda k. f (n+k) - f n$) (suffix (f n) σ)*
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 (rule monoD) simp
  moreover
  from mono[THEN strict-mono-mono] have  $f\ n \leq f\ (n+j)$ 
    by (rule monoD) simp
  moreover
  from mono ij 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  $\sigma\ (f\ 0) = \sigma\ 0$ 
  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 –

```

  from s obtain n where  $n: \sigma\ n = s$  by auto
  from f obtain k where  $f\ k \leq n < f\ (Suc\ k)$  by (rule stutter-sampler-interval)
  with f have  $\sigma\ n = \sigma\ (f\ k)$  by (rule stutter-sampler-between)
  with n show ?thesis by auto

```

qed

lemma *stutter-sampled-range*:

```

  range  $(\sigma \circ f) = \text{range}\ \sigma$  if stutter-sampler f  $\sigma$ 
  using that stutter-sampled-in-range [of f  $\sigma$ ] by auto

```

lemma *stutter-sampled-precedence*:

```

  assumes f: stutter-sampler f  $\sigma$  and ij:  $i \leq j$ 
  obtains k l where  $k \leq l$   $\sigma\ (f\ k) = \sigma\ i$   $\sigma\ (f\ l) = \sigma\ j$ 

```

proof –

```

  from f obtain k where  $k: f\ k \leq i < f\ (Suc\ k)$  by (rule stutter-sampler-interval)

```

with f **have** 1: $\sigma i = \sigma (f k)$ **by** (*rule stutter-sampler-between*)
from f **obtain** l **where** $l: f l \leq j j < f (Suc l)$ **by** (*rule stutter-sampler-interval*)
with f **have** 2: $\sigma j = \sigma (f l)$ **by** (*rule stutter-sampler-between*)
from $k l ij$ **have** $f k < f (Suc l)$ **by** *simp*
with f [*THEN stutter-sampler-mono*] **have** $k \leq l$ **by** (*simp add: strict-mono-less*)
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* $prev = \text{max-stutter-sampler } \sigma n$
 in *if* $(\forall k > prev. \sigma k = \sigma prev)$
 then $Suc prev$
 else (*LEAST* $k. prev < k \wedge \sigma k \neq \sigma prev$))

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

lemma *max-stutter-sampler*:

stutter-sampler (max-stutter-sampler σ) σ (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 $\exists k. ?prev < k \wedge \sigma k \neq \sigma ?prev$ **by** *simp*

from *this* [*THEN LeastI-ex*]

have $?prev < (\text{LEAST } k. ?prev < k \wedge \sigma k \neq \sigma ?prev)$..

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 $\forall n k. ?ms n < k \wedge k < ?ms (Suc n) \longrightarrow \sigma k = \sigma (?ms n)$

proof (*clarify*)

fix $n k$

assume *lo*: $?ms n < k$ (**is** *?prev* $<$ k)

```

    and hi: k < ?ms (Suc n) (is k < ?next)
  show  $\sigma k = \sigma ?prev$ 
  proof (cases  $\forall k > ?prev. \sigma k = \sigma ?prev$ )
    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  $\wedge$   $\sigma k \neq \sigma ?prev$ )
        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 $\natural\sigma$ 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* ($\langle \natural \rightarrow [100] 100$) **where**
 $\natural\sigma = \sigma \circ (\text{max-stutter-sampler } \sigma)$

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

lemma *stutter-freeI*:
assumes $\bigwedge k n. \llbracket \sigma (\text{Suc } k) = \sigma k; n > k \rrbracket \implies \sigma n = \sigma k$
shows *stutter-free* σ
using *assms unfolding stutter-free-def by blast*

lemma *stutter-freeD*:
assumes *stutter-free* σ **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* σ
shows *stutter-free* (*suffix* $k \sigma$)
proof (*rule stutter-freeI*)
 fix $j n$
assume $j: (\text{suffix } k \sigma) (\text{Suc } j) = (\text{suffix } k \sigma) j$ **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 (*suffix* $k \sigma$) $n = (\text{suffix } k \sigma) j$ **by** *simp*
 qed

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

lemma *stutter-free-reduced*:

assumes *sigma*: *stutter-free* σ

shows $\natural\sigma = \sigma$

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  $\wedge \sigma k \neq \sigma (?ms n)$ ) = 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, $\natural\sigma$ is stutter free.

lemma *max-stutter-sampler-nostuttering*:

assumes *stut*: $\sigma (\text{max-stutter-sampler } \sigma (Suc k)) = \sigma (\text{max-stutter-sampler } \sigma k)$

and *n*: $n > \text{max-stutter-sampler } \sigma k$ (**is** *- > ?ms k*)

shows $\sigma n = \sigma (?ms k)$

proof (*rule ccontr*)

assume *contr*: $\neg ?thesis$

with *n* **have** $?ms k < n \wedge \sigma n \neq \sigma (?ms k)$ (**is** *?diff n*) ..

hence *?diff (LEAST n. ?diff n)* **by** (*rule LeastI*)

with *contr* **have** $\sigma (?ms (Suc k)) \neq \sigma (?ms k)$ **by** (*auto simp add: Let-def*)

from *this stut* **show** *False* ..

qed

lemma *stutter-reduced-stutter-free*: *stutter-free* $(\natural\sigma)$

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

lemma *stutter-reduced-suffix*: $\natural (\text{suffix}\ k\ (\natural\sigma)) = \text{suffix}\ k\ (\natural\sigma)$
proof (*rule stutter-free-reduced*)
have *stutter-free* $(\natural\sigma)$ **by** (*rule stutter-reduced-stutter-free*)
thus *stutter-free* $(\text{suffix}\ k\ (\natural\sigma))$ **by** (*rule stutter-free-suffix*)
qed

lemma *stutter-reduced-reduced*: $\natural\natural\sigma = \natural\sigma$
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 $\sigma \circ f$ such that $\sigma \circ f \circ h = \sigma \circ g$. (Note that $f \circ h$ 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 $f: \text{stutter-sampler}\ f\ \sigma$
shows $\natural(\sigma \circ f) = \natural\sigma$
proof –
let $?mss = \text{max-stutter-sampler}\ \sigma$
let $?mssf = \text{max-stutter-sampler}\ (\sigma \circ f)$
from f **have** $mssf: \text{stutter-sampler}\ (f \circ ?mssf)\ \sigma$
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 $f \circ (\text{max-stutter-sampler}\ (\sigma \circ f))$ 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 $?mss\ k = f\ (?mssf\ k)$
 $\vee\ ?mss\ k \leq f\ (?mssf\ k) \wedge (\forall n \geq ?mss\ k. \sigma\ (?mss\ k) = \sigma\ n)$
(is $?P\ k$ **is** $?A\ k \vee ?B\ k)$
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  $b: ?B\ k \longrightarrow ?B\ (Suc\ k)$ 
proof
  assume  $0: ?B\ k$  hence  $1: ?mss\ k \leq f\ (?mssf\ k)$  ..

  from  $0$  have  $2: \forall n \geq ?mss\ k. \sigma\ (?mss\ k) = \sigma\ n$  ..
  hence  $\forall n > ?mss\ k. \sigma\ (?mss\ k) = \sigma\ n$  by auto
  hence  $\forall n > ?mss\ k. \sigma\ n = \sigma\ (?mss\ k)$  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]  $2\ 3$  have  $\forall n \geq ?mss\ (Suc\ k). \sigma\ (?mss\ (Suc\ k)) = \sigma\ n$ 
    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  $?B\ (Suc\ k)$  by blast
qed
from  $ih$  show  $?P\ (Suc\ k)$ 
proof
  assume  $a: ?A\ k$ 
  show  $?thesis$ 
  proof (cases  $\forall n > ?mss\ k. \sigma\ n = \sigma\ (?mss\ k)$ )
    case True
    hence  $\forall n \geq ?mss\ k. \sigma\ (?mss\ k) = \sigma\ n$  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:  $\sigma\ (?mss\ (Suc\ k)) \neq \sigma\ (?mss\ k)$ 
    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 ccontr)
      assume  $\neg ?thesis$ 
      hence contr:  $?mss\ (Suc\ k) < f\ (?mssf\ (Suc\ k))$  by simp
      from  $mssf$  have  $\sigma\ (?mss\ (Suc\ k)) = \sigma\ ((f \circ ?mssf)\ k)$ 
      proof (rule stutter-sampler-between)
        from max-stutter-sampler [of  $\sigma$ , THEN stutter-sampler-mono]
        have  $?mss\ k < ?mss\ (Suc\ k)$  by (rule strict-monoD) simp
      qed
    qed
  qed

```

```

with a show  $(f \circ ?mssf) k \leq ?mss (Suc k)$ 
  by (simp add: o-def del: max-stutter-sampler.simps)
next
from contr show  $?mss (Suc k) < (f \circ ?mssf) (Suc k)$  by simp
qed
with a have  $\sigma (?mss (Suc k)) = \sigma (?mss k)$ 
  by (simp add: o-def del: max-stutter-sampler.simps)
with diff show False ..
qed
next
have  $\exists m > ?mssf k. f m = ?mss (Suc k)$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  hence contr:  $\forall i. f ((?mssf k) + Suc i) \neq ?mss (Suc k)$  by simp
  {
    fix i
    have  $f (?mssf k + i) < ?mss (Suc k)$  (is ?F i)
    proof (induct i)
      from a have  $f (?mssf k + 0) = ?mss k$  by (simp add: o-def)
      also from max-stutter-sampler[of  $\sigma$ , THEN stutter-sampler-mono]
        have  $\dots < ?mss (Suc k)$ 
        by (rule strict-monoD) simp
      finally show ?F 0 .
    next
      fix i
      assume ih: ?F i
      show ?F (Suc i)
      proof (rule ccontr)
        assume  $\neg ?thesis$ 
        then have  $?mss (Suc k) \leq f (?mssf k + Suc i)$ 
          by (simp add: o-def)
        moreover from contr have  $f (?mssf k + Suc i) \neq ?mss (Suc k)$ 
          by blast
        ultimately have  $i: ?mss (Suc k) < f (?mssf k + Suc i)$ 
          by (simp add: less-le)
        from f have  $\sigma (?mss (Suc k)) = \sigma (f (?mssf k + i))$ 
        proof (rule stutter-sampler-between)
          from ih show  $f (?mssf k + i) \leq ?mss (Suc k)$ 
            by (simp add: o-def)
        next
          from i show  $?mss (Suc k) < f (Suc (?mssf k + i))$ 
            by simp
        qed
      also from max-stutter-sampler have  $\dots = \sigma (?mss k)$ 
      proof (rule stutter-sampler-between)
        from f[THEN stutter-sampler-mono, THEN strict-mono-mono]
          have  $f (?mssf k) \leq f (?mssf k + i)$  by (rule monoD) simp
        with a show  $?mss k \leq f (?mssf k + i)$  by (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 ( $\lambda i. f (?mssf\ k + i)$ )
  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 m where m:  $m > ?mssf\ k$  and m':  $f\ m = ?mss (Suc\ k)$ 
  by blast
show  $?mss (Suc\ k) \leq f (?mssf (Suc\ k))$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  hence contr:  $f (?mssf (Suc\ k)) < ?mss (Suc\ k)$  by simp
  from mssf[THEN stutter-sampler-mono]
  have  $(f \circ ?mssf)\ k < (f \circ ?mssf) (Suc\ k)$ 
    by (rule strict-monoD) simp
  with a have  $?mss\ k \leq f (?mssf (Suc\ k))$ 
    by (simp add: o-def)
  from this contr have  $\sigma (f (?mssf (Suc\ k))) = \sigma (?mss\ k)$ 
    by (rule stutter-sampler-between[OF max-stutter-sampler])
  with a have stut:  $(\sigma \circ f) (?mssf (Suc\ k)) = (\sigma \circ f) (?mssf\ k)$ 
    by (simp add: o-def)
  from this m have  $(\sigma \circ f)\ m = (\sigma \circ f) (?mssf\ k)$ 
    by (blast intro: max-stutter-sampler-nostuttering)
  with diff m' a show False
    by (simp add: o-def)
  qed
  qed
  thus ?thesis ..
  qed
next
  assume ?B k with b show ?thesis by (simp del: max-stutter-sampler.simps)
  qed
  qed
}
hence  $\mathfrak{h}\sigma = \mathfrak{h}(\sigma \circ f)$  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 $\langle \approx \rangle$ 50) where

$$\sigma \approx \tau \equiv \natural\sigma = \natural\tau$$

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*]: $\varrho \approx \sigma \implies \sigma \approx \tau \implies \varrho \approx \tau$

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 σ

shows $\sigma \circ f \approx \sigma$

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

lemma *stutter-reduced-equivalent*: $\natural\sigma \approx \sigma$

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 σ **and** g : *stutter-sampler* g τ

and eq : $\sigma \circ f = \tau \circ g$

shows $\sigma \approx \tau$

proof –

from f **have** $\natural\sigma = \natural(\sigma \circ f)$ **by** (*rule sample-max-sample*[*THEN sym*])

also from eq **have** $\dots = \natural(\tau \circ g)$ **by** *simp*

also from g **have** $\dots = \natural\tau$ **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*: $\bigwedge f g. \llbracket \text{stutter-sampler } f \ \sigma; \text{stutter-sampler } g \ \tau; \sigma \circ f = \tau \circ g \rrbracket \implies P$
shows *P*
proof (*rule p*)
from *eq* **show** $\sigma \circ (\text{max-stutter-sampler } \sigma) = \tau \circ (\text{max-stutter-sampler } \tau)$
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-equivI 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 $\sigma \ 0 = (\natural\sigma) \ 0$ **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* ($\langle \cdot \ [-..] \rangle$)
where
suffix-notation *w k* \equiv *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) $\sigma \approx$ suffix n ($\sigma \circ f$)*
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*[*THEN stutter-sampler-mono, THEN strict-mono-mono*]
have $f \ n \leq f \ (n+i)$ **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 : *stutter-sampler* f σ
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 $(\sigma[n ..]) \circ id = (\sigma[f k ..]) \circ ?f$

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 σ and τ , 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  $\dots \approx (\tau \circ g)[i\ ..]$ 
  by (rule suffix-stutter-equiv)
also from  $eq$  have  $\dots = (\sigma \circ f)[i\ ..]$ 
  by simp
also from  $f$  have  $\dots \approx \sigma[f\ i\ ..]$ 
  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  $\dots \approx (\sigma \circ f)[a\ ..]$ 
  by (rule suffix-stutter-equiv)
  also from  $eq$  have  $\dots = (\tau \circ g)[a\ ..]$  by simp
  also from  $g$  have  $\dots \approx \tau[g\ a\ ..]$ 
  by (rule suffix-stutter-equiv[THEN stutter-equiv-sym])
  finally have  $\sigma[j\ ..] \approx \tau[g\ a\ ..]$  .
  with  $1$  have  $\exists k < n.$   $\sigma[j..] \approx \tau[k\ ..]$  by blast
}
moreover
note that
ultimately show ?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 $\tau[n..] \approx \sigma[m..] \forall j < n. \exists k < m. \tau[j..] \approx \sigma[k..]$
 by (rule stutter-equiv-suffixes-left)
with that show ?thesis
 by (blast dest: stutter-equiv-sym)
qed

In particular, if σ and τ 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 $OR \Phi \equiv SOME \varphi. fold-graph Or-ltlp False-ltlp \Phi \varphi$

definition *AND* where $AND \Phi \equiv SOME \varphi. fold-graph And-ltlp True-ltlp \Phi \varphi$

lemma *fold-graph-OR*: $finite \Phi \implies fold-graph Or-ltlp False-ltlp \Phi (OR \Phi)$

unfolding *OR-def* by (rule someI2-ex[OF finite-imp-fold-graph])

lemma *fold-graph-AND*: $finite\ \Phi \implies fold_graph\ And_ltlp\ True_ltlp\ \Phi\ (AND\ \Phi)$
unfolding *AND-def* **by** (*rule someI2-ex[OF finite-imp-fold-graph]*)

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

lemma *holds-of-AND* [*simp*]:
assumes *fin*: $finite\ (\Phi::'a\ pltl\ set)$
shows $(\sigma \models_p AND\ \Phi) = (\forall \varphi \in \Phi. \sigma \models_p \varphi)$
proof –
{
 fix $\psi::'a\ pltl$
 assume *fold-graph And-ltlp True-ltlp* $\Phi\ \psi$
 hence $(\sigma \models_p \psi) = (\forall \varphi \in \Phi. \sigma \models_p \varphi)$
 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_p = *True*
 | *next-free (atom_p(p))* = *True*
 | *next-free ($\varphi\ implies_p\ \psi$)* = (*next-free* $\varphi \wedge next_free\ \psi$)
 | *next-free ($X_p\ \varphi$)* = *False*
 | *next-free ($\varphi\ U_p\ \psi$)* = (*next-free* $\varphi \wedge next_free\ \psi$)

lemma *next-free-not* [*simp*]:
next-free (not_p φ) = *next-free* φ
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*]:

```

next-free ( $\varphi$  orp  $\psi$ ) = (next-free  $\varphi$   $\wedge$  next-free  $\psi$ )
by (simp add: Or-ltlp-def)

lemma next-free-and [simp]: next-free ( $\varphi$  andp  $\psi$ ) = (next-free  $\varphi$   $\wedge$  next-free  $\psi$ )
by (simp add: And-ltlp-def)

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

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

lemma next-free-release [simp]:
next-free ( $\varphi$  Rp  $\psi$ ) = (next-free  $\varphi$   $\wedge$  next-free  $\psi$ )
by (simp add: Release-ltlp-def)

lemma next-free-weak-until [simp]:
next-free ( $\varphi$  Wp  $\psi$ ) = (next-free  $\varphi$   $\wedge$  next-free  $\psi$ )
by (auto simp: WeakUntil-ltlp-def)

lemma next-free-strong-release [simp]:
next-free ( $\varphi$  Mp  $\psi$ ) = (next-free  $\varphi$   $\wedge$  next-free  $\psi$ )
by (auto simp: StrongRelease-ltlp-def)

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

lemma next-free-AND [simp]:
assumes fin: finite ( $\Phi::'a$  pltl set)
shows next-free (AND  $\Phi$ ) = ( $\forall \varphi \in \Phi. \text{next-free } \varphi$ )
proof -
{
fix  $\psi::'a$  pltl
assume fold-graph And-ltlp True-ltlp  $\Phi$   $\psi$ 
hence next-free  $\psi$  = ( $\forall \varphi \in \Phi. \text{next-free } \varphi$ )
by (rule fold-graph.induct) auto
}

```

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 σ iff it holds of τ .

definition *stutter-invariant* **where**

$$\text{stutter-invariant } \varphi = (\forall \sigma \tau. (\sigma \approx \tau) \longrightarrow (\sigma \models_p \varphi) = (\tau \models_p \varphi))$$

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 $\bigwedge \sigma \tau. [\sigma \approx \tau; \sigma \models_p \varphi] \Longrightarrow \tau \models_p \varphi$

shows *stutter-invariant* φ

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* φ 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 $\varphi \Longrightarrow$ *stutter-invariant* ($\varphi::'a$ *pltl*)

proof (*induct* φ)

show *stutter-invariant* *false_p* by *auto*

next

fix $p :: 'a \Rightarrow \text{bool}$

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  $\varphi \ \psi :: 'a \ \text{ptll}$ 
assume ih: next-free  $\varphi \implies$  stutter-invariant  $\varphi$ 
        next-free  $\psi \implies$  stutter-invariant  $\psi$ 
assume next-free ( $\varphi \text{ implies}_p \psi$ )
with ih show stutter-invariant ( $\varphi \text{ implies}_p \psi$ ) by auto
next
fix  $\varphi :: 'a \ \text{ptll}$ 
assume next-free ( $X_p \ \varphi$ ) — hence contradiction
thus stutter-invariant ( $X_p \ \varphi$ ) by simp
next
fix  $\varphi \ \psi :: 'a \ \text{ptll}$ 
assume ih: next-free  $\varphi \implies$  stutter-invariant  $\varphi$ 
        next-free  $\psi \implies$  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 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:  $(\sigma[m..]) \approx (\tau[n..])$  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
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 $'a \Rightarrow \text{bool}$) of a PLTL formula.

```

lemma atoms-ptll-OR [simp]:
  assumes fin: finite ( $\Phi :: 'a \ \text{ptll set}$ )
  shows atoms-ptll (OR  $\Phi$ ) = ( $\bigcup \varphi \in \Phi. \text{atoms-ptll } \varphi$ )

```

proof –
 {
 fix $\psi :: 'a \text{ pttl}$
 assume $\text{fold-graph Or-ltlp False-ltlp } \Phi \ \psi$
 hence $\text{atoms-pttl } \psi = (\bigcup \varphi \in \Phi. \text{atoms-pttl } \varphi)$
 by ($\text{rule fold-graph.induct}$) auto
 }
with $\text{fold-graph-OR}[OF \text{ fin}]$ **show** $?thesis$ **by** simp
qed

lemma atoms-pttl-AND [simp]:
assumes $\text{fin: finite } (\Phi :: 'a \text{ pttl set})$
shows $\text{atoms-pttl } (\text{AND } \Phi) = (\bigcup \varphi \in \Phi. \text{atoms-pttl } \varphi)$
proof –
 {
 fix $\psi :: 'a \text{ pttl}$
 assume $\text{fold-graph And-ltlp True-ltlp } \Phi \ \psi$
 hence $\text{atoms-pttl } \psi = (\bigcup \varphi \in \Phi. \text{atoms-pttl } \varphi)$
 by ($\text{rule fold-graph.induct}$) auto
 }
with $\text{fold-graph-AND}[OF \text{ fin}]$ **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 $\text{state-sim} :: ['a, ('a \Rightarrow \text{bool}) \text{ set}, 'a] \Rightarrow \text{bool}$
 $(\langle - \sim - \rangle \rightarrow [70, 100, 70] \ 50)$ **where**
 $s \sim A \sim t = (\forall p \in A. p \ s \longleftrightarrow p \ t)$

definition $\text{seq-sim} :: [\text{nat} \Rightarrow 'a, ('a \Rightarrow \text{bool}) \text{ set}, \text{nat} \Rightarrow 'a] \Rightarrow \text{bool}$
 $(\langle - \simeq - \rangle \rightarrow [70, 100, 70] \ 50)$ **where**
 $\sigma \simeq A \simeq \tau = (\forall n. (\sigma \ n) \sim A \sim (\tau \ n))$

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 ($\text{simp add: state-sim-def}$)

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

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 state-sim-mono :
assumes $s \sim A \sim t$ **and** $B \subseteq A$
shows $s \sim B \sim t$
using $\text{assms unfolding state-sim-def}$ **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*]: $\varrho \simeq_A \simeq \sigma \implies \sigma \simeq_A \simeq \tau \implies \varrho \simeq_A \simeq \tau$
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*: $\sigma \simeq \text{atoms-pltl } \varphi \simeq \tau \implies (\sigma \models_p \varphi) = (\tau \models_p \varphi)$
(is *?sim* $\sigma \varphi \tau \implies ?P \sigma \varphi \tau$)

proof (*induct* φ *arbitrary*: $\sigma \tau$)

fix $\sigma \tau$

show $?P \sigma \text{ false}_p \tau$ **by** *simp*

next

fix $p \sigma \tau$

assume $?sim \sigma (\text{atom}_p(p)) \tau$ **thus** $?P \sigma (\text{atom}_p(p)) \tau$

by (*auto simp: seq-sim-def state-sim-def*)

next

fix $\varphi \psi \sigma \tau$

assume *ih*: $\bigwedge \sigma \tau. ?sim \sigma \varphi \tau \implies ?P \sigma \varphi \tau$

$\bigwedge \sigma \tau. ?sim \sigma \psi \tau \implies ?P \sigma \psi \tau$

and *sim*: $?sim \sigma (\varphi \text{ implies}_p \psi) \tau$

from *sim* **have** $?sim \sigma \varphi \tau$ $?sim \sigma \psi \tau$

by (*auto elim: seq-sim-mono*)

with *ih* **show** $?P \sigma (\varphi \text{ implies}_p \psi) \tau$ **by** *simp*

next

fix $\varphi \sigma \tau$

assume *ih*: $\bigwedge \sigma \tau. ?sim \sigma \varphi \tau \implies ?P \sigma \varphi \tau$

and *sim*: $\sigma \simeq \text{atoms-pltl } (X_p \varphi) \simeq \tau$

from *sim* **have** $(\sigma[1..]) \simeq \text{atoms-pltl } \varphi \simeq (\tau[1..])$

by (*auto simp: seq-sim-def*)

with *ih* **show** $?P \sigma (X_p \varphi) \tau$ **by** *auto*

next

fix $\varphi \psi \sigma \tau$

assume *ih*: $\bigwedge \sigma \tau. ?sim \sigma \varphi \tau \implies ?P \sigma \varphi \tau$

$\bigwedge \sigma \tau. ?sim \sigma \psi \tau \implies ?P \sigma \psi \tau$

and *sim*: $?sim \sigma (\varphi U_p \psi) \tau$

from *sim* **have** $\forall i. (\sigma[i..]) \simeq \text{atoms-pltl } \varphi \simeq (\tau[i..])$ $\forall j. (\sigma[j..]) \simeq \text{atoms-pltl } \psi \simeq (\tau[j..])$

by (*auto simp: seq-sim-def state-sim-def*)

with *ih* **have** $\forall i. ?P (\sigma[i..]) \varphi (\tau[i..]) \forall j. ?P (\sigma[j..]) \psi (\tau[j..])$
by *blast+*
thus $?P \sigma (\varphi U_p \psi) \tau$
by (*meson semantics-pltl.simps(5)*)
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**
 $canonize A s \equiv SOME t. t \sim A \sim s$

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* **where**
 $canonical-sequence A \sigma \equiv \forall m (n::nat). \sigma m \sim A \sim \sigma n \longrightarrow \sigma m = \sigma n$

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

lemma *canonical-suffix*:
 $canonical-sequence A \sigma \implies canonical-sequence A (\sigma[k..])$
by (*auto simp: canonical-sequence-def*)

lemma *canonical-sampled*:
 $canonical-sequence A \sigma \implies canonical-sequence A (\sigma \circ f)$
by (*auto simp: canonical-sequence-def*)

lemma *canonical-reduced*:
 $canonical-sequence A \sigma \implies canonical-sequence A (\natural \sigma)$
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** $\tau \simeq A \simeq \sigma$ *canonical-sequence* A τ
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 $s \equiv$
 $((\text{AND } \{ \text{atom}_p(p) \mid p \cdot p \in A \wedge p \ s \}) \text{ and}_p (\text{AND } \{ \text{not}_p(\text{atom}_p(p)) \mid p \cdot p \in A \wedge \neg(p \ s) \}))$

lemma *characteristic-holds*:
 $\text{finite } A \implies \sigma \models_p \text{characteristic-formula } A (\sigma \ 0)$
by (*auto simp: characteristic-formula-def*)

lemma *characteristic-state-sim*:
assumes *fin: finite* A
shows $(\sigma \ 0 \sim A \sim \tau \ 0) = (\tau \models_p \text{characteristic-formula } A (\sigma \ (0::\text{nat})))$
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** $\tau \models_p \text{characteristic-formula } A (\sigma \ 0)$
by (*auto simp: characteristic-formula-def*) (*blast+*)
next
assume $\tau \models_p \text{characteristic-formula } A (\sigma \ 0)$
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 φ there exists a next-free formula ψ 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 : *atoms-pltl* $\varphi \subseteq A$ **and** *fin*: *finite* A

shows $\exists \psi$. *next-free* $\psi \wedge \text{atoms-pltl } \psi \subseteq A \wedge$

$(\forall \sigma$. *stutter-free* $\sigma \wedge \text{canonical-sequence } A \sigma \longrightarrow (\sigma \models_p \psi) = (\sigma \models_p \varphi))$

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

using A **proof** (*induct* φ)

The cases of *false* and atomic formulas are trivial.

have $?P \text{false}_p \text{false}_p$ **by** *auto*

thus $\exists \psi$. $?P \text{false}_p \psi$..

next

fix p

assume *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 *atoms-pltl* $\varphi \subseteq A \implies \exists \varphi'$. $?P \varphi \varphi'$

and *atoms-pltl* $\psi \subseteq A \implies \exists \psi'$. $?P \psi \psi'$

and *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 *atoms-pltl* $\varphi \subseteq A \implies \exists \varphi'$. $?P \varphi \varphi'$

and *atoms-pltl* $\psi \subseteq A \implies \exists \psi'$. $?P \psi \psi'$

and *atoms-pltl* $(\varphi U_p \psi) \subseteq A$

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

{

fix σ

assume *sigma*: *stutter-free* σ *canonical-sequence* $A \sigma$

hence $\bigwedge k$. *stutter-free* $(\sigma[k..]) \bigwedge k$. *canonical-sequence* $A (\sigma[k..])$

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

with 1 2

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

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

by (*blast+*)

hence $(\sigma \models_p \varphi' U_p \psi') = (\sigma \models_p \varphi U_p \psi)$

by *auto*

}

with 1 2 **have** $?P (\varphi U_p \psi) (\varphi' U_p \psi')$ **by** *auto*

thus $\exists \chi$. $?P (\varphi U_p \psi) \chi$..

next

The interesting case is the one of the *next*-operator.

fix φ
assume *ih*: $\text{atoms-pltl } \varphi \subseteq A \implies \exists \psi. ?P \varphi \psi$ **and** *at*: $\text{atoms-pltl } (X_p \varphi) \subseteq A$
then obtain ψ **where** *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 $X_p \varphi$ over the stutter-free and canonical sequence σ .

define *stval* **where** $\text{stval} = (\lambda s. \{ p \in A . p s \})$
define *chi* **where** $\text{chi} = (\lambda \text{val}. ((\text{AND } \{ \text{atom}_p(p) \mid p . p \in \text{val} \}) \text{and}_p (\text{AND } \{ \text{not}_p (\text{atom}_p(p)) \mid p . p \in A - \text{val} \})))$
define *psi'* **where** $\text{psi}' = ((\psi \text{and}_p (\text{OR } \{ G_p (\text{chi } \text{val}) \mid \text{val} . \text{val} \subseteq A \})) \text{or}_p (\text{OR } \{ (\text{chi } \text{val}) \text{and}_p ((\text{chi } \text{val}) U_p (\psi \text{and}_p (\text{chi } \text{val}')) \mid \text{val } \text{val}' . \text{val} \subseteq A \wedge \text{val}' \subseteq A \wedge \text{val}' \neq \text{val} \}))$
(is $- = ((- \text{and}_p (\text{OR } ?ALW)) \text{or}_p (\text{OR } ?UNT))$ **)**

have $\bigwedge s. \{ \text{not}_p (\text{atom}_p(p)) \mid p . p \in A - \text{stval } s \}$
 $= \{ \text{not}_p (\text{atom}_p(p)) \mid p . p \in A \wedge \neg(p s) \}$
by (*auto simp: stval-def*)
hence *chi1*: $\bigwedge s. \text{chi} (\text{stval } s) = \text{characteristic-formula } A s$
by (*auto simp: chi-def stval-def characteristic-formula-def*)
{
fix *val* τ
assume *val*: $\text{val} \subseteq A$ **and** *tau*: $\tau \models_p \text{chi } \text{val}$
with *fin* **have** $\text{val} = \text{stval } (\tau 0)$
by (*auto simp: stval-def chi-def finite-subset*)
}
note *chi2* $= \text{this}$

have $?UNT \subseteq (\lambda(\text{val}, \text{val}'). (\text{chi } \text{val}) \text{and}_p ((\text{chi } \text{val}) U_p (\psi \text{and}_p (\text{chi } \text{val}'))))$
 $' (\text{Pow } A \times \text{Pow } A)$
(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** $\bigwedge \text{val}. \text{val} \subseteq A \implies \text{next-free } (\text{chi } \text{val})$
by (*auto simp: chi-def finite-subset*)
with *fin* *fin-UNT* *psi* **show** *?thesis*
by (*force simp: psi'-def finite-subset*)
qed

have *atoms-pltl*: $\text{atoms-pltl } \text{psi}' \subseteq A$
proof $-$
from *fin* **have** *at-chi*: $\bigwedge \text{val}. \text{val} \subseteq A \implies \text{atoms-pltl } (\text{chi } \text{val}) \subseteq A$
by (*auto simp: chi-def finite-subset*)

with *fin psi* **have** *at-alm: atoms-pltl* (ψ *and*_p (*OR ?ALW*)) $\subseteq A$
by *auto blast?*
from *fin fin-UNT psi at-chi* **have** *atoms-pltl* (*OR ?UNT*) $\subseteq A$
by *auto (blast+)?*
with *at-alm* **show** *?thesis* **by** (*auto simp: psi'-def*)
qed

{
fix σ
assume *st: stutter-free* σ **and** *can: canonical-sequence* $A \sigma$
have ($\sigma \models_p X_p \varphi$) = ($\sigma \models_p \psi'$)
proof (*cases* σ (*Suc* 0) = σ 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 *alleg* = *this*
have *suffix*: $\bigwedge n. \sigma[n..] = \sigma$
proof (*rule ext*)
fix $n i$
have ($\sigma[n..]$) $i = \sigma 0$ **by** (*auto intro: alleg*)
moreover **have** $\sigma i = \sigma 0$ **by** (*rule alleg*)
ultimately **show** ($\sigma[n..]$) $i = \sigma i$ **by** *simp*
qed
with *st can psi* **have** 1 : ($\sigma \models_p X_p \varphi$) = ($\sigma \models_p \psi$) **by** *simp*

from *fin* **have** $\sigma \models_p \text{chi}$ (*stval* ($\sigma 0$)) **by** (*simp add: chi1 characteristic-holds*)
with *suffix* **have** $\sigma \models_p G_p$ (*chi* (*stval* ($\sigma 0$))) (*is -* \models_p *?alm*) **by** *simp*
moreover **have** *?alm* \in *?ALW* **by** (*auto simp: stval-def*)
ultimately **have** 2 : $\sigma \models_p \text{OR ?ALW}$
using *fin* **by** (*auto simp: finite-subset simp del: semantics-pltl-sugar*)

have 3 : $\neg(\sigma \models_p \text{OR ?UNT})$
proof
assume *unt*: $\sigma \models_p \text{OR ?UNT}$
with *fin-UNT* **obtain** *val val' k* **where**
val: $val \subseteq A$ *val'* $\subseteq A$ *val' \neq val* **and**
now: $\sigma \models_p \text{chi } val$ **and** *k*: $\sigma[k..] \models_p \text{chi } val'$

```

    by auto (blast+)?
  from ⟨val ⊆ A⟩ now have val = stval (σ 0) by (rule chi2)
  moreover
  from ⟨val' ⊆ A⟩ k suffix have val' = stval (σ 0) by (simp add: chi2)
  moreover note ⟨val' ≠ val⟩
  ultimately show False by simp
qed

```

```

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

```

```

next
case False

```

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

```

let ?val = stval (σ 0)
let ?val' = stval (σ 1)
from False can have vals: ?val' ≠ ?val
  by (auto simp: canonical-sequence-def state-sim-def stval-def)

```

```

show ?thesis

```

```

proof

```

```

  assume phi: σ ⊨p Xp φ
  from fin have 1: σ ⊨p chi ?val by (simp add: chi1 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..] ⊨p ψ by auto

```

```

  from fin have σ[1..] ⊨p characteristic-formula A ((σ[1..]) 0)
  by (rule characteristic-holds)
  hence 3: σ[1..] ⊨p chi ?val' by (simp add: chi1)

```

```

  from 1 2 3 have σ ⊨p And-ltlp (chi ?val) ((chi ?val) Up (And-ltlp ψ (chi
?val')))

```

```

    (is - ⊨p ?unt)
  by auto

```

```

  moreover from vals have ?unt ∈ ?UNT
  by (auto simp: stval-def)

```

```

  ultimately have σ ⊨p OR ?UNT
  using fin-UNT[THEN holds-of-OR] by blast
  thus σ ⊨p psi' by (simp add: psi'-def)

```

```

next

```

```

  assume psi': σ ⊨p psi'
  have ¬(σ ⊨p OR ?ALW)

```

```

  proof

```

```

    assume σ ⊨p OR ?ALW

```

```

    with fin obtain val where 1: val ⊆ A and 2: ∀ n. (σ[n..] ⊨p chi val)

```

```

    by (force simp: finite-subset)
  from 2 have  $\sigma[0..] \models_p \text{chi val} ..$ 
  with 1 have  $\text{val} = ?\text{val}$  by (simp add: chi2)
  moreover
  from 2 have  $\sigma[1..] \models_p \text{chi val} ..$ 
  with 1 have  $\text{val} = ?\text{val}'$  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:  $\text{val} \subseteq A$  val'  $\subseteq A$  val'  $\neq \text{val}$  and
  now:  $\sigma \models_p \text{chi val}$  and
  k:  $\sigma[k..] \models_p \psi$   $\sigma[k..] \models_p \text{chi val}'$  and
  i:  $\forall i < k. (\sigma[i..] \models_p \text{chi val})$ 
  by auto (blast+)?

from val now have 1:  $\text{val} = ?\text{val}$  by (simp add: chi2)

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

have 3:  $k \leq 1$ 
proof (rule ccontr)
  assume  $\neg(k \leq 1)$ 
  with i have  $\sigma[1..] \models_p \text{chi val}$  by simp
  with 1 have  $\sigma[1..] \models_p \text{characteristic-formula } A (\sigma 0)$ 
    by (simp add: chi1)
  hence  $(\sigma 0) \sim A \sim ((\sigma[1..]) 0)$ 
    using characteristic-state-sim[OF fin] by blast
  with can have  $\sigma 0 = \sigma 1$ 
    by (simp add: canonical-sequence-def)
  with  $\langle \sigma (\text{Suc } 0) \neq \sigma 0 \rangle$  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  $\langle \sigma[k..] \models_p \psi \rangle$  psi by auto
qed
qed
}
with nf atoms-ptl 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 *stutter-invariant-next-free:*

assumes phi : *stutter-invariant* φ

obtains ψ **where** *next-free* ψ *atoms-pltl* $\psi \subseteq$ *atoms-pltl* φ

$$\forall \sigma. (\sigma \models_p \psi) = (\sigma \models_p \varphi)$$

proof –

have *atoms-pltl* $\varphi \subseteq$ *atoms-pltl* φ *finite* (*atoms-pltl* φ) **by** *simp-all*

then obtain ψ **where**

psi: *next-free* ψ *atoms-pltl* $\psi \subseteq$ *atoms-pltl* φ **and**

equiv: $\forall \sigma. \text{stutter-free } \sigma \wedge \text{canonical-sequence } (\text{atoms-pltl } \varphi) \sigma \longrightarrow (\sigma \models_p \psi)$

$= (\sigma \models_p \varphi)$

by (*blast dest: ex-next-free-stutter-free-canonical*)

from $\langle \text{next-free } \psi \rangle$ **have** *sinv*: *stutter-invariant* ψ

by (*rule next-free-stutter-invariant*)

{

fix σ

obtain τ **where** *1*: $\tau \simeq \text{atoms-pltl } \varphi \simeq \sigma$ **and** *2*: *canonical-sequence* (*atoms-pltl* φ) τ

by (*rule canonical-exists*)

from *1* $\langle \text{atoms-pltl } \psi \subseteq \text{atoms-pltl } \varphi \rangle$ **have** *3*: $\tau \simeq \text{atoms-pltl } \psi \simeq \sigma$

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** $\dots = (\natural\tau \models_p \varphi)$ **by** *auto*

also from *2*[*THEN canonical-reduced*] *equiv* *stutter-reduced-stutter-free*

have $\dots = (\natural\tau \models_p \psi)$ **by** *auto*

also from *sinv* *stutter-reduced-equivalent* **have** $\dots = (\tau \models_p \psi)$ **by** *auto*

also from *3* **have** $\dots = (\sigma \models_p \psi)$ **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*

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*:

stutter-invariant $\varphi \longleftrightarrow$

$(\exists \psi. \text{next-free } \psi \wedge \text{atoms-pltl } \psi \subseteq \text{atoms-pltl } \varphi \wedge (\forall \sigma. \sigma \models_p \psi \longleftrightarrow \sigma \models_p \varphi))$

proof –

{
 assume *stutter-invariant* φ
 hence $\exists \psi. \text{next-free } \psi \wedge \text{atoms-pltl } \psi \subseteq \text{atoms-pltl } \varphi \wedge (\forall \sigma. \sigma \models_p \psi \longleftrightarrow \sigma \models_p \varphi)$
 by (*rule stutter-invariant-next-free*) *blast*
}

moreover

{
 fix ψ
 assume *1*: *next-free* ψ **and** *2*: $\forall \sigma. \sigma \models_p \psi \longleftrightarrow \sigma \models_p \varphi$
 from *1* **have** *stutter-invariant* ψ **by** (*rule next-free-stutter-invariant*)
 with *2* **have** *stutter-invariant* φ **by** *blast*
}

ultimately show *?thesis* **by** *blast*

qed

4.6 Stutter Invariance for LTL with Syntactic Sugar

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

primrec *ltlc-next-free* :: 'a ltlc \Rightarrow bool

where

ltlc-next-free $\text{true}_c = \text{True}$
| *ltlc-next-free* $\text{false}_c = \text{True}$
| *ltlc-next-free* $(\text{prop}_c(q)) = \text{True}$
| *ltlc-next-free* $(\text{not}_c \varphi) = \text{ltlc-next-free } \varphi$
| *ltlc-next-free* $(\varphi \text{ and}_c \psi) = (\text{ltlc-next-free } \varphi \wedge \text{ltlc-next-free } \psi)$
| *ltlc-next-free* $(\varphi \text{ or}_c \psi) = (\text{ltlc-next-free } \varphi \wedge \text{ltlc-next-free } \psi)$
| *ltlc-next-free* $(\varphi \text{ implies}_c \psi) = (\text{ltlc-next-free } \varphi \wedge \text{ltlc-next-free } \psi)$
| *ltlc-next-free* $(X_c \varphi) = \text{False}$
| *ltlc-next-free* $(F_c \varphi) = \text{ltlc-next-free } \varphi$
| *ltlc-next-free* $(G_c \varphi) = \text{ltlc-next-free } \varphi$
| *ltlc-next-free* $(\varphi U_c \psi) = (\text{ltlc-next-free } \varphi \wedge \text{ltlc-next-free } \psi)$
| *ltlc-next-free* $(\varphi R_c \psi) = (\text{ltlc-next-free } \varphi \wedge \text{ltlc-next-free } \psi)$
| *ltlc-next-free* $(\varphi W_c \psi) = (\text{ltlc-next-free } \varphi \wedge \text{ltlc-next-free } \psi)$
| *ltlc-next-free* $(\varphi M_c \psi) = (\text{ltlc-next-free } \varphi \wedge \text{ltlc-next-free } \psi)$

lemma *ltlc-next-free-iff[simp]*: *next-free* $(\text{ltlc-to-pltl } \varphi) \longleftrightarrow \text{ltlc-next-free } \varphi$

by (*induction* φ) *auto*

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

theorem *ltlc-next-free-stutter-invariant*:
assumes *next-free: ltlc-next-free* φ
assumes *eq*: $r \approx r'$
shows $r \models_c \varphi \longleftrightarrow r' \models_c \varphi$
proof –
{
 fix $r r'$
 assume *eq*: $r \approx r'$ **and** *holds*: $r \models_c \varphi$
 then have $r \models_p (\text{ltlc-to-pltl } \varphi)$ **by** *simp*

 from *next-free-stutter-invariant*[*of ltlc-to-pltl* φ] *next-free*
 have *PLTL.stutter-invariant* (*ltlc-to-pltl* φ) **by** *simp*
 from *stutter-invariantD*[*OF this eq*] *holds* **have** $r' \models_c \varphi$ **by** *simp*
} **note** *aux=this*

from *aux*[*of r r'*] *aux*[*of r' r*] *eq stutter-equiv-sym*[*OF eq*] **show** *?thesis*
 by *blast*
qed

end

References

- [1] L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Information Processing 83: Proceedings of the IFIP 9th World Congress*, pages 657–668, Paris, Sept. 1983. IFIP, North-Holland.
- [2] D. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Inf. Proc. Lett.*, 63(5):243–246, 1997.