Stuttering Equivalence and Stuttering Invariance

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

March 17, 2025

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

$$(abbccca)^\omega \qquad \text{and} \qquad (aaaabc)^\omega$$

are stuttering equivalent, whereas

$$(abac)^\omega \qquad \text{and} \qquad (aaaabcc)^\omega$$

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

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

1 Utility Lemmas

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

Strongly monotonic functions over the integers grow without bound.

lemma *strict-mono-exceeds*:
assumes *f*: *strict-mono* (*f*::*nat* ⇒ *nat*)
shows ∃ *k*. *n* < *f k*
⟨*proof*⟩

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* ⇒ *nat*) **and** *n*: *f 0* ≤ *n*
obtains *k* **where** *f k* ≤ *n* **and** *n* < *f* (*Suc k*)
⟨*proof*⟩

lemma *strict-mono-comp*:
assumes *g*: *strict-mono* (*g*::'*a*::*order* ⇒ '*b*::*order*)
and *f*: *strict-mono* (*f*::'*b*::*order* ⇒ '*c*::*order*)
shows *strict-mono* (*f* ∘ *g*)
⟨*proof*⟩

2 Stuttering Sampling Functions

Given an ω-sequence σ, a stuttering sampling function is a strictly monotonic function $f::nat \Rightarrow nat$ 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 σ
stutter-sampler $(f::nat \Rightarrow nat)\ \sigma \equiv$
$f\ 0 = 0$
$\wedge$ *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* $f\ \sigma \Longrightarrow f\ 0 = 0$
⟨*proof*⟩

lemma *stutter-sampler-mono*: *stutter-sampler* $f\ \sigma \Longrightarrow$ *strict-mono* f
⟨*proof*⟩

lemma *stutter-sampler-between*:
assumes f: *stutter-sampler* $f\ \sigma$
and *lo*: $f\ k \le n$ **and** *hi*: $n < f\ (Suc\ k)$
shows $\sigma\ n = \sigma\ (f\ k)$
⟨*proof*⟩

lemma *stutter-sampler-interval*:
assumes f: *stutter-sampler* $f\ \sigma$
obtains k **where** $f\ k \le n$ **and** $n < f\ (Suc\ k)$
⟨*proof*⟩

The identity function is a stuttering sampling function for any σ.

lemma *id-stutter-sampler* [*iff*]: *stutter-sampler id* σ
⟨*proof*⟩

Stuttering sampling functions compose, sort of.

lemma *stutter-sampler-comp*:
assumes f: *stutter-sampler* $f\ \sigma$
and g: *stutter-sampler* $g\ (\sigma \circ f)$
shows *stutter-sampler* $(f \circ g)\ \sigma$
⟨*proof*⟩

Stuttering sampling functions can be extended to suffixes.

lemma *stutter-sampler-suffix*:
assumes f: *stutter-sampler* $f\ \sigma$
shows *stutter-sampler* $(\lambda k.\ f\ (n+k) - f\ n)$ (*suffix* $(f\ n)\ \sigma$)
⟨*proof*⟩

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 $\sigma\ (f\ 0) = \sigma\ 0$
⟨*proof*⟩

lemma *stutter-sampled-in-range*:
assumes f: *stutter-sampler* $f\ \sigma$ **and** s: $s \in$ *range* σ
shows $s \in$ *range* $(\sigma \circ f)$
⟨*proof*⟩

lemma *stutter-sampled-range*:
range $(\sigma \circ f) =$ *range* σ **if** *stutter-sampler* $f\ \sigma$
⟨*proof*⟩

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*⟩

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* σ (*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* $< k \wedge \sigma\ k \neq \sigma$ *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*⟩

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* (‹♮-› [*100*] *100*) **where**
$\natural\sigma = \sigma \circ$ (*max-stutter-sampler* σ)

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

lemma *stutter-freeI*:
assumes $\bigwedge k\ n.\ [\![\sigma\ (Suc\ k) = \sigma\ k;\ n{>}k]\!] \Longrightarrow \sigma\ n = \sigma\ k$

shows *stutter-free* σ
⟨*proof*⟩

lemma *stutter-freeD*:
assumes *stutter-free* σ **and** σ (*Suc* k) = σ k **and** $n>k$
shows σ n = σ k
⟨*proof*⟩

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

lemma *stutter-free-suffix*:
assumes *sigma*: *stutter-free* σ
shows *stutter-free* (*suffix* k σ)
⟨*proof*⟩

lemma *stutter-reduced-0*: ($\natural\sigma$) *0* = σ *0*
⟨*proof*⟩

lemma *stutter-free-reduced*:
assumes *sigma*: *stutter-free* σ
shows $\natural\sigma = \sigma$
⟨*proof*⟩

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*: 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*⟩

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

lemma *stutter-reduced-suffix*: $\natural$ (*suffix* k ($\natural\sigma$)) = *suffix* k ($\natural\sigma$)
⟨*proof*⟩

lemma *stutter-reduced-reduced*: $\natural\natural\sigma = \natural\sigma$
⟨*proof*⟩

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*: *stutter-sampler f σ*
  shows $\natural(\sigma \circ f) = \natural\sigma$
⟨*proof*⟩

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

Stuttering equivalence is an equivalence relation.

lemma *stutter-equiv-refl*: $\sigma \approx \sigma$
  ⟨*proof*⟩

lemma *stutter-equiv-sym* [*sym*]: $\sigma \approx \tau \Longrightarrow \tau \approx \sigma$
  ⟨*proof*⟩

lemma *stutter-equiv-trans* [*trans*]: $\varrho \approx \sigma \Longrightarrow \sigma \approx \tau \Longrightarrow \varrho \approx \tau$
  ⟨*proof*⟩

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 $\sigma \circ f \approx \sigma$
  ⟨*proof*⟩

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

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 $\sigma \approx \tau$

$\langle proof \rangle$

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*: $\bigwedge f\ g.\ [\![\ \textit{stutter-sampler}\ f\ \sigma;\ \textit{stutter-sampler}\ g\ \tau;\ \sigma \circ f = \tau \circ g\ ]\!] \Longrightarrow P$
  shows P
$\langle proof \rangle$

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*:
  $\sigma \approx \tau = (\exists f\ g.\ \textit{stutter-sampler}\ f\ \sigma \wedge \textit{stutter-sampler}\ g\ \tau \wedge \sigma \circ f = \tau \circ g)$
  $\langle proof \rangle$

The initial elements of stutter equivalent sequences are equal.

lemma *stutter-equiv-0*:
  assumes $\sigma \approx \tau$
  shows $\sigma\ 0 = \tau\ 0$
$\langle proof \rangle$

abbreviation *suffix-notation* (‹- [-..]›)
where
  $\textit{suffix-notation}\ w\ k \equiv \textit{suffix}\ k\ w$

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*: $\textit{stutter-sampler}\ f\ \sigma$
  shows $\textit{suffix}\ (f\ n)\ \sigma \approx \textit{suffix}\ n\ (\sigma \circ f)$
$\langle proof \rangle$

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*: $\textit{stutter-sampler}\ f\ \sigma$
    and *lo*: $f\ k \le n$ **and** *hi*: $n < f\ (\textit{Suc}\ k)$
  shows $\sigma[n\ ..] \approx \sigma[f\ k\ ..]$
$\langle proof \rangle$

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** $\forall j<m.\ \exists k<n.\ \sigma[j..] \approx \tau[k..]$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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* φ. *fold-graph Or-ltlp False-ltlp* Φ φ

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

lemma *fold-graph-OR*: *finite* $\Phi \Longrightarrow$ *fold-graph Or-ltlp False-ltlp* Φ (*OR* Φ)
  ⟨*proof*⟩

lemma *fold-graph-AND*: *finite* $\Phi \Longrightarrow$ *fold-graph And-ltlp True-ltlp* Φ (*AND* Φ)
  ⟨*proof*⟩

lemma *holds-of-OR* [*simp*]:
assumes *fin*: *finite* (Φ::'*a pltl set*)
shows ($\sigma \models_p$ *OR* Φ) = ($\exists \varphi \in \Phi$. $\sigma \models_p \varphi$)
⟨*proof*⟩

lemma *holds-of-AND* [*simp*]:
assumes *fin*: *finite* (Φ::'*a pltl set*)
shows ($\sigma \models_p$ *AND* Φ) = ($\forall \varphi \in \Phi$. $\sigma \models_p \varphi$)
⟨*proof*⟩

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* ⇒ *bool*
where
next-free $false_p$ = *True*
| *next-free* ($atom_p(p)$) = *True*
| *next-free* (φ $implies_p$ ψ) = (*next-free* φ ∧ *next-free* ψ)
| *next-free* (X_p φ) = *False*
| *next-free* (φ U_p ψ) = (*next-free* φ ∧ *next-free* ψ)

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

lemma *next-free-true* [*simp*]:
next-free $true_p$
⟨*proof*⟩

lemma *next-free-or* [*simp*]:
next-free (φ or_p ψ) = (*next-free* φ ∧ *next-free* ψ)
⟨*proof*⟩

lemma *next-free-and* [*simp*]: *next-free* (φ and_p ψ) = (*next-free* φ ∧ *next-free* ψ)
⟨*proof*⟩

lemma *next-free-eventually* [*simp*]:
next-free (F_p φ) = *next-free* φ
⟨*proof*⟩

lemma *next-free-always* [*simp*]:
next-free (G_p φ) = *next-free* φ
⟨*proof*⟩

lemma *next-free-release* [*simp*]:
next-free (φ R_p ψ) = (*next-free* φ ∧ *next-free* ψ)
⟨*proof*⟩

lemma *next-free-weak-until* [*simp*]:

$next\text{-}free\ (\varphi\ W_p\ \psi) = (next\text{-}free\ \varphi \wedge next\text{-}free\ \psi)$
⟨*proof*⟩

lemma *next-free-strong-release* [*simp*]:
$next\text{-}free\ (\varphi\ M_p\ \psi) = (next\text{-}free\ \varphi \wedge next\text{-}free\ \psi)$
⟨*proof*⟩

lemma *next-free-OR* [*simp*]:
assumes *fin*: $finite\ (\Phi{::}'a\ pltl\ set)$
shows $next\text{-}free\ (OR\ \Phi) = (\forall\,\varphi{\in}\Phi.\ next\text{-}free\ \varphi)$
⟨*proof*⟩

lemma *next-free-AND* [*simp*]:
assumes *fin*: $finite\ (\Phi{::}'a\ pltl\ set)$
shows $next\text{-}free\ (AND\ \Phi) = (\forall\,\varphi{\in}\Phi.\ next\text{-}free\ \varphi)$
⟨*proof*⟩

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

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*⟩

lemma *stutter-invariantD* [*dest*]:
assumes *stutter-invariant* φ **and** $\sigma \approx \tau$
shows $(\sigma \models_p \varphi) = (\tau \models_p \varphi)$
⟨*proof*⟩

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*:
$next\text{-}free\ \varphi \Longrightarrow stutter\text{-}invariant\ (\varphi{::}'a\ pltl)$
⟨*proof*⟩

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 $'a \Rightarrow bool$) of a PLTL formula.

lemma *atoms-pltl-OR* [*simp*]:
  assumes *fin*: *finite* (Φ::$'a$ *pltl set*)
  shows *atoms-pltl* (*OR* Φ) = ($\bigcup \varphi \in \Phi$. *atoms-pltl* φ)
⟨*proof*⟩

lemma *atoms-pltl-AND* [*simp*]:
  assumes *fin*: *finite* (Φ::$'a$ *pltl set*)
  shows *atoms-pltl* (*AND* Φ) = ($\bigcup \varphi \in \Phi$. *atoms-pltl* φ)
⟨*proof*⟩

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

definition *state-sim* :: [$'a$, ($'a \Rightarrow bool$) *set*, $'a$] $\Rightarrow$ *bool*
  (‹- ~-~ -› [*70*,*100*,*70*] *50*) **where**
  $s \sim A \sim t = (\forall p \in A.\ p\ s \longleftrightarrow p\ t)$

definition *seq-sim* :: [$nat \Rightarrow 'a$, ($'a \Rightarrow bool$) *set*, $nat \Rightarrow 'a$] $\Rightarrow$ *bool*
  (‹- ≃-≃ -› [*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*]: $s \sim A \sim s$
  ⟨*proof*⟩

lemma *state-sim-sym*: $s \sim A \sim t \Longrightarrow t \sim A \sim s$
  ⟨*proof*⟩

lemma *state-sim-trans*[*trans*]: $s \sim A \sim t \Longrightarrow t \sim A \sim u \Longrightarrow s \sim A \sim u$
  ⟨*proof*⟩

lemma *state-sim-mono*:
  assumes $s \sim A \sim t$ **and** $B \subseteq A$
  shows $s \sim B \sim t$
  ⟨*proof*⟩

lemma *seq-sim-refl* [*simp*]: $\sigma \simeq A \simeq \sigma$
  ⟨*proof*⟩

lemma *seq-sim-sym*: $\sigma \simeq A \simeq \tau \Longrightarrow \tau \simeq A \simeq \sigma$
  ⟨*proof*⟩

lemma *seq-sim-trans*[*trans*]: $\varrho \simeq A \simeq \sigma \Longrightarrow \sigma \simeq A \simeq \tau \Longrightarrow \varrho \simeq A \simeq \tau$
⟨*proof*⟩

lemma *seq-sim-mono*:
assumes $\sigma \simeq A \simeq \tau$ **and** $B \subseteq A$
shows $\sigma \simeq B \simeq \tau$
⟨*proof*⟩

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*: $\sigma \simeq$ *atoms-pltl* $\varphi \simeq \tau \Longrightarrow (\sigma \models_p \varphi) = (\tau \models_p \varphi)$
(**is** *?sim* $\sigma\ \varphi\ \tau \Longrightarrow$ *?P* $\sigma\ \varphi\ \tau$)
⟨*proof*⟩

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* $A\ s \sim A \sim s$
⟨*proof*⟩

lemma *canonize-canonical*:
assumes *st*: $s \sim A \sim t$
shows *canonize* $A\ s =$ *canonize* $A\ t$
⟨*proof*⟩

lemma *canonize-idempotent*:
canonize A (*canonize* $A\ s$) = *canonize* $A\ s$
⟨*proof*⟩

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 $A\ \sigma \Longrightarrow$ *canonical-sequence* $A\ (\sigma[k..])$
⟨*proof*⟩

lemma *canonical-sampled*:
canonical-sequence $A\ \sigma \Longrightarrow$ *canonical-sequence* $A\ (\sigma \circ f)$
⟨*proof*⟩

lemma *canonical-reduced*:

$canonical\text{-}sequence\ A\ \sigma \Longrightarrow canonical\text{-}sequence\ A\ (\natural\sigma)$
⟨*proof*⟩

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** $\tau \simeq A \simeq \sigma$ $canonical\text{-}sequence\ A\ \tau$
⟨*proof*⟩

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**
$characteristic\text{-}formula\ A\ s \equiv$
$((AND\ \{\ atom_p(p)\ |\ p\ .\ p \in A \wedge p\ s\ \})\ and_p\ (AND\ \{\ not_p\ (atom_p(p))\ |\ p\ .\ p \in A \wedge \neg(p\ s)\ \}))$

lemma *characteristic-holds*:
$finite\ A \Longrightarrow \sigma \models_p characteristic\text{-}formula\ A\ (\sigma\ 0)$
⟨*proof*⟩

lemma *characteristic-state-sim*:
assumes *fin*: $finite\ A$
shows $(\sigma\ 0\ {}^{\sim}A^{\sim}\ \tau\ 0) = (\tau \models_p characteristic\text{-}formula\ A\ (\sigma\ (0{::}nat)))$
⟨*proof*⟩

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 $A \supseteq atoms\text{-}pltl\ \varphi$).

lemma *ex-next-free-stutter-free-canonical*:
assumes *A*: $atoms\text{-}pltl\ \varphi \subseteq A$ **and** *fin*: $finite\ A$
shows $\exists\psi.\ next\text{-}free\ \psi \wedge atoms\text{-}pltl\ \psi \subseteq A \wedge$
$(\forall\sigma.\ stutter\text{-}free\ \sigma \wedge canonical\text{-}sequence\ A\ \sigma \longrightarrow (\sigma \models_p \psi) = (\sigma \models_p \varphi))$
(**is** $\exists\psi.\ ?P\ \varphi\ \psi$)
⟨*proof*⟩

Comparing the definition of the next-free formula in the case of formulas X_p φ 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*⟩

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 $\varphi \longleftrightarrow$
  $(\exists \psi.$ *next-free* $\psi \wedge$ *atoms-pltl* $\psi \subseteq$ *atoms-pltl* $\varphi \wedge (\forall \sigma.\ \sigma \models_p \psi \longleftrightarrow \sigma \models_p \varphi))$
⟨*proof*⟩

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* ⇒ *bool*
  where
  ltlc-next-free $true_c$ = *True*
| *ltlc-next-free* $false_c$ = *True*
| *ltlc-next-free* $(prop_c(q))$ = *True*
| *ltlc-next-free* $(not_c\ \varphi)$ = *ltlc-next-free* φ
| *ltlc-next-free* $(\varphi\ and_c\ \psi)$ = (*ltlc-next-free* $\varphi \wedge$ *ltlc-next-free* ψ)
| *ltlc-next-free* $(\varphi\ or_c\ \psi)$ = (*ltlc-next-free* $\varphi \wedge$ *ltlc-next-free* ψ)
| *ltlc-next-free* $(\varphi\ implies_c\ \psi)$ = (*ltlc-next-free* $\varphi \wedge$ *ltlc-next-free* ψ)
| *ltlc-next-free* $(X_c\ \varphi)$ = *False*
| *ltlc-next-free* $(F_c\ \varphi)$ = *ltlc-next-free* φ
| *ltlc-next-free* $(G_c\ \varphi)$ = *ltlc-next-free* φ
| *ltlc-next-free* $(\varphi\ U_c\ \psi)$ = (*ltlc-next-free* $\varphi \wedge$ *ltlc-next-free* ψ)
| *ltlc-next-free* $(\varphi\ R_c\ \psi)$ = (*ltlc-next-free* $\varphi \wedge$ *ltlc-next-free* ψ)
| *ltlc-next-free* $(\varphi\ W_c\ \psi)$ = (*ltlc-next-free* $\varphi \wedge$ *ltlc-next-free* ψ)
| *ltlc-next-free* $(\varphi\ M_c\ \psi)$ = (*ltlc-next-free* $\varphi \wedge$ *ltlc-next-free* ψ)

lemma *ltlc-next-free-iff*[*simp*]: *next-free* (*ltlc-to-pltl* φ) $\longleftrightarrow$ *ltlc-next-free* φ
  ⟨*proof*⟩

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 $r \models_c \varphi \longleftrightarrow r' \models_c \varphi$
$\langle proof \rangle$

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.