On the Development of Specification Languages In the Hardware Domain

Dana Fisman

Dagstuhl 19071

Specification Formalisms for Modern Cyber-Physical Systems
Once upon a time...

There was **no standard** for hardware specification language.

Users of formal verification tools by **different companies** used **different languages**.
It was decided to have a Standard!

A committee consisting of over 30 people was formed.

Including leading people from the academia: Ed Clarke, Allen Emerson, and Moshe Vardi.

70 requirements where collected

74 examples of industry properties were expressed in the 4 languages
It was decided to pick one as a basis

Phase 1: pick 2 out of the 4 by vote.

Phase 2: the 2 selected should modify according to changes. Then final vote.

Phase 3: rename as PSL Refine.

PSL = Property Specification Language
**Landmarks in PSL Standardization**

1994 | Industrial model checkers in use

2000 | Standardization effort begins at Accellera

2001 | Sugar selected as a basis for standardization.

2004 | PSL 1.1 is an official Accellera standard

2005 | Standardization effort moved to IEEE

2006 | IEEE standard PSL was approved
Major changes from Sugar to PSL

Partition to layers:
  - Boolean layer
  - Temporal layer
  - Modeling layer
  - Verification layer

Addition of Flavors:
  - Verilog
  - SystemVerilog
  - SystemC
  - VHDL
  - GDL

Syntax:
- $d = 010 \land \neg b$
- $G(r \rightarrow Fa)$

Move from branching to linear time

---

Vardi
Maidl
Major features of PSL

- Clock / Sampling operator
- Truncated Paths
- Reset / Abort operator
- Local Variables
- Regular expressions & the triggers op
- Strong & Weak REs distinction
Landmarks in PSL Language Development

1994  Syntactic sugaring added to CTL
1995  Regular expr & triggers op
2001  Moved to linear-time semantics
2003  Truncated paths, reset ops, clock op
2005  Weak regular expressions
2008  Local Variables
From LTL to PSL

On the Development of Specification Languages In the Hardware Domain

Dana Fisman
What makes a spec lang good?

Expressiveness

Possible to state what you want!
LTL is as expressive as star-free $\omega$-regular expressions.

The following property cannot be stated in LTL \cite{Wol83}:

\[ p \text{ holds on every even tick} \]
Expressive Power

- $\omega$-regular expr.
- Star-free $\omega$-regular expr.
- Aperiodic languages
- Very weak alternating automata
- Counter-free Buchi Automata
- First-order logic
- Quantified LTL (QPTL) [KP95]
- LTL extended w. aut. connectives [VW94]
- LTL extended w. fix point ops [BB87]
- LTL extended w. triggers op (& reg expr.)
- S1S

References:
- [DG08]
Major features of PSL

The triggers op

Regular expressions
& the triggers op

The triggers op was added to Sugar, when it was still CTL based [BBL98]
Only later Sugar moved to linear time [BBEFGRO1]
The triggers operator (and its dual “follows-by”) are reminiscent of the modalities obtained in the extension of Propositional Dynamic Logic with Regular Programs [FL78]
The triggers operator

[r \vdash \varphi]

Semantics:
\[ w \models r \vdash \varphi \]

\[ \forall j \text{ if } w(0..j) \in \mathcal{L}(r) \text{ then } w(j..) \models \varphi \]
The triggers operator

\[
\{ \text{true}[\ast] \cdot \text{req} \cdot \text{ack} \} \Rightarrow \{ \text{start} \cdot \text{busy}[\ast] \cdot \text{end} \}
\]
The triggers operator

\{ \text{true}[\ast] \cdot \text{req} \cdot \text{ack} \} \Rightarrow \{ \text{start} \cdot \text{busy}[\ast] \cdot \text{end} \}
The triggers operator

Theorem:
LTL extended with the triggers op is as expressive as the $\omega$-regular languages

$\text{p holds on every even tick}$

$((\text{TRUE, TRUE})^*) \rightarrow p$
What makes a spec lang good?

**Expressiveness**
Possible to state what you want!

**Succinctness**
With as few words as possible!
## Derived operators

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>until</td>
<td>next!</td>
<td>next_event!</td>
</tr>
<tr>
<td>until!</td>
<td>next</td>
<td>next_event</td>
</tr>
<tr>
<td>until_</td>
<td>next![n]</td>
<td>next_event![n]</td>
</tr>
<tr>
<td>until!_</td>
<td>next[n]</td>
<td>next_event[n]</td>
</tr>
<tr>
<td>before</td>
<td>next_e![m..n]</td>
<td>next_event_e![m..n]</td>
</tr>
<tr>
<td>before!</td>
<td>next_e[m..n]</td>
<td>next_event_e[m..n]</td>
</tr>
<tr>
<td>before_</td>
<td>next_a![m..n]</td>
<td>next_event_a![m..n]</td>
</tr>
<tr>
<td>before!_</td>
<td>next_a![m..n]</td>
<td>next_event_a![m..n]</td>
</tr>
</tbody>
</table>

- r1 && r2
- r1 & r2
- r1 || r2
- r1 • r2
- r1 ⊕ r2

- b[*]
- b[+]
- b[*i]
- b[*i..j]
- b[*i..inf]

- b[=i]
- b[=i..j]
- b[=i..inf]

- b[->i]
- b[->i..j]
- b[->i..inf]

etc. etc. etc.
A lot of syntactic Sugaring

\{ \text{true}[\,*]\,*\,\text{req}\,*\,\text{ack} \} \Rightarrow \{ \text{start}\,*\,\text{data}[\,=8]\,*\,\text{end} \}
Theorem:
Regular expressions with counting operators are doubly-exponentially more succinct than DFAs and exponentially more succinct than NFAs and regular expressions.

[KT03, G08]
Succinctness is more than Derived Ops

- Most operators of PSL do not add expressive power
- But they are not derived operators
  - SERE Intersection
  - Past operators: ended()
  - Abort operators
  - Clock operator
  - Local variables
Major features of PSL

Strong & Weak REs distinction
Distinguishing Weak and Strong REs

Using only REs?

\[ (\text{busy}^* \cdot \text{done}) \]

Is it weak or strong?

**Syntax ✓**

**Semantics ???**

\[ G ((\text{req} \cdot \text{busy}[^*3..5] \cdot \text{ack}) \rightarrow [\text{busy} \ U \ \text{done}]) \]

\[ G (\text{req} \cdot \text{busy}[^*3..5] \cdot \text{ack}) \rightarrow [\text{busy} \ W \ \text{done}] \]

**Strong Until**

**Weak Until**

**Strong RE**

**Weak RE**
Distinguishing Weak and Strong REs

What should be the semantics?

\[ w \vDash \text{strong}(r) \]

\[ \exists u \preceq w . u \in \mathcal{L}(r) \]

Prefix of

\[ w \vDash \text{weak}(r) \]

allow to

get stuck in

a starred sub-expr.

So that e.g. \( ab^\omega \vDash (a \cdot b^* \cdot c) \)

perhaps:

\[ \forall u \preceq w . \exists v \succeq u . v \in \mathcal{L}(r) \]

extension of

No!
Distinguishing Weak and Strong REs

The problem with

$$\forall u \leq w \cdot \exists v \geq u \cdot v \in \mathcal{L}(r)$$

(busy $W$ done) $\equiv$ (busy$^*$ · done)

(busy $W$ false) $\not\equiv$ (busy$^*$ · false)

(busy . busy)

*does not have an extension satisfying*

(busy$^*$ . false)

Could be replaced with a comp. unsatisfiable formula
Distinguishing Weak and Strong REs

Semantics for weak(r)

Augment the alphabet $\Sigma = 2^{AP}$
with two special letters $\top, \bot$
so that

$\top$ satisfies everything, even false
$\bot$ satisfied nothing, not even true

$w \models \text{weak}(r)$

$\iff$

$\forall u \leq w . \ u \models ^{\omega} \top \models \text{strong}(r)$

[ABKVO3]
Major features of PSL

Hardware Clocks

Clock / Sampling operator

@
Hardware Clocks

G(p → X q)

The next tick with respect to what?
The hardware clock

And if there are multiple clocks?

Hmmm..... There are....

I meant (posedge clka)
Hardware Clocks

I meant (posedge clka)

G(p → X q)

In this case:

G (¬clka → X((clka ∧ p) →
X[¬(¬clka ∧ X clka)) W (¬clka ∧ X (clka ∧ q))])
Hardware Clocks

\[ G (\neg clka \rightarrow X((clka \land p) \rightarrow \\
X[\neg (\neg clka \land X clka)] \land W (\neg clka \land X (clka \land q))]) \]

Well...

a bit shorter would be better

\[ G(p \rightarrow X q)@ (posedge clka) \]

Syntax ✓

Semantics ???
Hardware Clocks

\[ \{\text{true}[*]; \text{req}; \text{ack}\} \Rightarrow \{\text{start}; \text{data}[\text{*3}]; \text{end}\} \] @clk
Hardware Clocks

Issue 1: @ = projection?

\[
\{ \text{true} [ \ast ] \cdot \text{req} \cdot \text{ack} \} \Rightarrow \{ \text{start} \cdot \text{data} [\ast 3] \cdot \text{end} \}@ clkb \) @ clka
\]
Hardware Clocks

Issue 1: $@$ = projection ?

$\{ \text{true}[\ast] \cdot \text{req} \cdot \text{ack} \} \Rightarrow \{ \text{start} \cdot \text{data}[\ast3] \cdot \text{end} \}$@ clkb ) @ clka

clocks do not accumulate, rather they change the projection
Hardware Clocks

Issue 2

What happens if the clock stops ticking?

\[ \text{busy} \quad U \quad \text{done} \quad @ \quad \text{clk} \]

Should we have two clock ops?

\[ \text{busy} \quad U \quad \text{done} \quad @ \quad \text{clk} \]

\[ \text{busy} \quad U \quad \text{done} \quad @ \quad \text{clk} \]

Strong clk

Weak clk
Hardware Clocks

Issue 2

Problems with two @ ops:

Formula

\( Fp \land Gq \)

cannot be satisfactory clocked on a finite path

A liveness formula may hold on some word \( w \) but not on an extension \( ww' \) of that word

**Solution:** have one @ op and when the clock stops ticking determine satisfaction by the strength of the temporal operator
Let \( f, f_1 \) and \( f_2 \) be formulas in LTL extended with the clock operator \( @ \), let \( b \) and \( c \) be Boolean expressions, and let \( w \) be a word over \( \Sigma = 2^P \). For finite word \( w \), let \( w \models^c \) tick denote that \( w \in \mathcal{L}(\neg c^* \cdot c) \).

- \( w \models^c b \iff \forall j < |w| \text{ s.t. } w(0..j) \models^c \text{ tick }, w(j) \models b \)
- \( w \models^c b! \iff \exists j < |w| \text{ s.t. } w(0..j) \models^c \text{ tick and } w(j) \models b \)
- \( w \models^c \neg f \iff w \not\models^f f \)
- \( w \models^c f_1 \land f_2 \iff w \models^c f_1 \text{ and } w \models^c f_2 \)
- \( w \models^c X! f \iff \exists j < k < |w| \text{ s.t. } w(0..j) \models^c \text{ tick and } w(j + 1..k) \models^c \text{ tick and } w(k..) \models^c f \)
- \( w \models^c [f_1 U f_2] \iff \exists k < |w| \text{ s.t. } w(k) \models c \text{ and } w(k..) \models^c f_2 \text{ and } \forall j < k \text{ s.t. } w(j) \models c, w(j..) \models^c f_1 \)
- \( w \models^c f@c_1 \iff w \models^c_1 f \)
Clocks and Sampling

@ can be used as a sampling abstraction

Consecutive writes cannot be of high priority

\[ G((\text{write} \land \text{high}) \rightarrow X \lnot\text{write} W (\text{write} \land \lnot\text{high})) \]

\[ G(\text{high} \rightarrow X \lnot\text{high})@\text{write} \]
Hardware Clocks

Theorem:
The @ operator does not add expressive power

Theorem:
On singly clocked formulas @ has the effect of projection.

Theorem:
Strong until (U) and weak until (W) maintain their fix point characterization under the @ semantics
Major features of PSL

Hardware Resets

Reset op
Hardware Resets

req should always be answered by ack after 3 cycles unless rst was asserted

Reset cancels future obligations!

G (req → XXX ack) reset rst

(req → XXX ack) W rst

Syntax ✔

Semantics ???
Hardware Resets

Perhaps

\[ w \not\models \varphi \text{ reset } b \]

Either \[ w \models \varphi \]

Or \[ \exists i. \ w(i) \models b \] and \[ \exists w'. \ w(0..i)w' \models \varphi \]

No!

For reasons similar to weak REs

(busy U (p \land \neg p)) \text{ reset } b

This formula should hold if busy holds up until b
Hardware Resets

Semantics

Let \( a \) be an accept signal and \( r \) a reject signal

Let \( \varphi \) and \( \psi \) be formulas of LTL extended with the \textsc{reset} operator, let \( a \) and \( r \) be Boolean expressions, and let \( w \) be a word over \( \Sigma = 2^P \).

- \( \langle w, a, r \rangle = p \) \iff \( w(0) \models a \lor (p \land \neg r) \)
- \( \langle w, a, r \rangle = \neg \varphi \) \iff \( \langle w, r, a \rangle \not\models \varphi \)
- \( \langle w, a, r \rangle = \varphi \land \psi \) \iff \( \langle w, a, r \rangle \models \varphi \) and \( \langle w, a, r \rangle \models \psi \)
- \( \langle w, a, r \rangle = X \varphi \) \iff \( w(0) \models a \) or both \( w(0) \not\models r \) and \( \langle w(1..), a, r \rangle \models \varphi \)
- \( \langle w, a, r \rangle = [\varphi \mathcal{U} \psi] \) \iff \( \exists k < |w| \text{ s.t. } \langle w(k..), a, r \rangle \models \psi, \text{ and } \forall j < k, \langle w(j..), a, r \rangle \models \varphi \)
- \( \langle w, a, r \rangle = \varphi \text{ \textsc{reset} } b \) \iff \( \langle w, a \lor (b \land \neg r), r \rangle \models \varphi \)

\( \langle w, \text{FALSE}, \text{FALSE} \rangle \models \varphi \)
Major features of PSL

Truncated Paths

- Simulation
- Dynamic Verification
- Reset op
- Clock op
- Bounded MC
- Infinite Paths
- Finite Paths (maximal)
- Truncated Paths
every request must receive a grant, and once asserted, the request signal must stay asserted until it receives its grant.

\[ G(request \rightarrow X [request \ U \ grant]) \]

\[ G(request \rightarrow X [request \ W \ grant]) \]

But sometimes they have “no opinion” on the correct length of the test.

If using BMC should we change to: ???

Sometimes test are designed to continue until correct output can be confirmed.
Truncated Paths

The Dilemma

Should the verifier change the property when switching tools?

No!
Properties should state the desired behavior!

So???
Use Views!

The truncated semantics evaluates a formula \( \text{wrt. (i) a path} \)
\( \text{(ii) a view} \)

A view is one of the three:

\{weak, neutral, strong\}

- **Optimistic**: accepts if nothing bad happened so far
- **Realistic**: considers the truncated path as if it was maximal
- **Pessimistic**: rejects if eventualities are not fulfilled
Views for Truncated Paths

Dynamic Verification

(p p p p p p p p , weak) \not\models pUq
(p p p p p p p p , neutral) \models pUq
(p p p p p p p p , strong) \not\models pUq
The Truncated Semantics

Given a finite/infinite word \( w \in \Sigma^\infty \)

a view \( v \in \{ +, \cdot, - \} \)

and a formula \( \varphi \)

\[ \langle w, v \rangle \models \varphi \]  \( \text{formula } \varphi \text{ holds on path } w \text{ under the view } v \)

\[ \langle w, - \rangle \models \varphi \]  \( \varphi \text{ holds weakly on } w \)

\[ \langle w, \cdot \rangle \models \varphi \]  \( \varphi \text{ holds neutrally on } w \)

\[ \langle w, + \rangle \models \varphi \]  \( \varphi \text{ holds strongly on } w \)
Semantics

\[ \bar{v} = \begin{cases} + & \text{if } v = - \\ \cdot & \text{if } v = \cdot \\ - & \text{if } v = + \end{cases} \]

Semantics:

\[ \langle w, v \rangle \models X^i p \iff \text{either } |w| > i \text{ and } w[i+1] \models p \]

or \( v = - \) and if \( |w| > i \) then \( w[i+1] \models p \)

\[ \langle w, v \rangle \models \neg \varphi \iff \langle w, \bar{v} \rangle \not\models \varphi \]

\[ \langle w, v \rangle \models \varphi_1 \lor \varphi_2 \iff \langle w, v \rangle \models \varphi_1 \text{ or } \langle w, v \rangle \models \varphi_2 \]

\[ \langle w, v \rangle \models [\varphi_1 \mathbf{U} \varphi_2] \iff \exists k. \langle w[k..] , v \rangle \models \varphi_2 \text{ and } \forall j < k. \langle w[j..] , v \rangle \models \varphi_1 \]

\[ \langle w, v \rangle \models [\varphi \mathbf{T} b] \iff \text{either } \langle w, v \rangle \models \varphi \]

or \( \exists j. \langle w[j] , v \rangle \models b \text{ and } \langle w[..j-1] , \cdot \rangle \models \varphi \)
Semantics

**Complementary view**

\[ \bar{v} = \begin{cases} 
+ & \text{if } v = - \\
\cdot & \text{if } v = \cdot \\
- & \text{if } v = + 
\end{cases} \]

**Semantics:**

\[
\langle w, v \rangle \models \text{X}_i p \iff \text{either } |w| > i \text{ and } w[i + 1] \models p \text{ or } v = - \text{ and if } |w| > i \text{ then } w[i + 1] \models p
\]

\[
\langle w, v \rangle \models \neg \varphi \iff \langle w, \bar{v} \rangle \not\models \varphi
\]

\[
\langle w, v \rangle \models \varphi_1 \lor \varphi_2 \iff \langle w, v \rangle \models \varphi_1 \text{ or } \langle w, v \rangle \models \varphi_2
\]

\[
\langle w, v \rangle \models [\varphi_1 \mathbf{U} \varphi_2] \iff \exists k. \langle w[k..], v \rangle \models \varphi_2 \text{ and } \forall j < k. \langle w[j..], v \rangle \models \varphi_1
\]

\[
\langle w, v \rangle \models [\varphi \mathbf{T} b] \iff \text{either } \langle w, v \rangle \models \varphi \text{ or } \exists j. \langle w[j], v \rangle \models b \text{ and } \langle w[..j-1], \bar{v} \rangle \models \varphi
\]
Relation to Hardware resets

Theorem:

\[
\langle w, \text{false, false} \rangle \models \varphi \text{ reset } b \\
\iff \\
\langle w, \cdot \rangle \models \varphi \text{ reset } b
\]

[EFHLMV03]

The reset semantics and the truncated semantics are equivalent!!!

Theorem:

And these two are equivalent to the semantics with \( \top \) and \( \bot \)

[EFHMO5]
The Strength Relation Theorem

[EFHLMV03]

Holds strongly

\[ w \models^{\pm} \varphi \implies w \models \varphi \implies w \models^{=} \varphi \]

Holds neutrally

Holds weakly

Degrees of Satisfaction

Holds strongly

Holds neutrally

Holds weakly (pending)

Fails

\[ H_{\text{strong}} \rightarrow H_{\text{neutral}} \rightarrow H_{\text{weak}} \rightarrow \text{Fails} \]
The Prefix/Extension Theorem

\[ \text{Holds weakly} \quad \begin{align*}
\bullet \ w & \models \varphi \\
& \iff \forall w' \leq w. \ w' \models \varphi
\end{align*} \]

\[ \text{Holds strongly} \quad \begin{align*}
\bullet \ w & \models^+ \varphi \\
& \iff \forall w'' \geq w. \ w'' \models^\mp \varphi
\end{align*} \]

\[ \text{Holds weakly on all prefixes} \]

\[ \text{Holds strongly on all extensions} \]

[EFHLMV03]
What makes a spec lang good?

- Expressiveness
- Succinctness
- Good Tools
- Readability
- Reasonable Complexity for Required Algs.
Complexity

Lesson 14 - outline

Automata and Logic on Infinite Objects

2.1 Summary of complexity results of satisfiability for declarative formalisms

Formalisms expressive as star-free \( \omega \)-regular languages:
- LTL: PSPACE-complete \([BH06]\)

Formalisms expressive as \( \omega \)-regular languages:
- S1S: non-elementary \([BH06]\)
- QPTL: non-elementary \([BH06]\)
- corePSL: PSPACE-complete \([BH06]\)
- PSL: EXPSPACE-complete \([BH06]\)
- \( \omega \)-regular: EXPSPACE-complete
- star-free \( \omega \)-regular: PSPACE-complete

Thus, by augmenting LTL with the operator \( r \)' we get (a) a logic which is as expressive as the full \( \omega \)-regular languages (rather than just the star-free \( \omega \)-regular languages as LTL is) (b) a logic that doesn't use explicit quantifiers as do QPTL and S1S (and with the same expressive power as QPTL and S1S) and (c) a logic whose satisfiability can be checked in PSPACE, which is a tremendous improvement compared to the non-elementary complexity of S1S and QPTL.

3 Muller Automata

3.1 Definition and Example

1. The last acceptance criteria for \( \omega \)-automata we'll see is Muller automata.
2. It is the most general one, and the sharpest.
3. It will help us complete the picture of expressive power.
4. DEF. A Muller automaton is a tuple \( M = (\mathcal{A}, Q, q_0, \mathcal{F}) \) where the acceptance condition \( \mathcal{F} \) is a set of set of states \( F_1, \ldots, F_k \) where \( F_i \subseteq Q \) for all \( 1 \leq i \leq k \).
5. A run \( r \) of a Muller automaton is accepting if \( \text{Inf}(r) \subseteq \mathcal{F} \). That is, if the set of states visited infinitely often by the run \( r \) is exactly one of the sets \( F_i \) specified in \( \mathcal{F} \).
6. For example, consider the following DMWs \( M_1 \) and \( M_2 \) defined over the same structure.

\[
M_1 = \{ \{0\}, \{1\} \}
\]

\[
M_2 = \{ \{0, 1\} \}
\]

We have that \( \models M_1 = (a + b)^* (a! + b!) \) and \( \models M_2 = b^* (a + b + b!) \).
Looking for subsets with better complexity

General LTL formula \( \varphi \)

Buchi Automaton

\( M \times B_{\neg \varphi} \)

\( 2^{O(n)} \)

Emptiness

Check for fair cycles

Safety LTL formula \( \varphi \)

NFA recognizing bad prefixes

\( M \times N_{\neg \varphi} \)

\( 2^{2^{O(n)}} \)

\( 2^{2^{\Omega(\sqrt{n})}} \)

Emptiness

Reachability

[Wy86]

[KV99]
Let $\varphi$ be safety formula.

A bad prefix for $\varphi$ is a finite word $w$ all of whose extensions violate.

An informative bad prefix for $\varphi$ is bad prefix $w$ that provides enough information to explain violation of $\varphi$.

$XXp$  $\quad FGp \land FG\neg p$

all its bad prefixes are informative  None of its bad prefixes is informative
## Complexity of Safety Formulas

<table>
<thead>
<tr>
<th>General LTL formula $\phi$</th>
<th>Safety LTL formula $\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buchi Automaton $B_{\neg\phi}$</td>
<td>NFA recognizing informative prefixes $N_{\neg\phi}$</td>
</tr>
<tr>
<td>$2^{O(n)}$</td>
<td>$2^{\Omega(\sqrt{n})}$</td>
</tr>
</tbody>
</table>

- $\forall n \in \mathbb{N}$, $\lceil \log_2(n) \rceil = O(n)$

**Complexity Analysis:**

- NFA recognizing informative prefixes $N_{\neg\phi}$ with $2^{\Omega(\sqrt{n})}$
- NFA recognizing bad prefixes $N'_{\neg\phi}$ with $2^{O(n)}$

**Question:** Can we suffice with recognizing informative prefixes?
A formula is intentionally safe if all its bad prefixes are informative.

A formula is accidentally safe if every computation that violates it has an informative prefix.

A formula is pathologically safe if there is a computation that violates it and has no informative prefixes.
Classification to Safety Formulas
and the truncated semantics

Theorem:
Let $\varphi$ be an LTL formula
and $w$ a finite non-non-empty word.
Then $w$ is an informative prefix for
iff $w \models \neg \varphi$

[EFHMVO3]
A set of safety formulas defined syntactically

Let $b$ be a Boolean expression and $r$ a regular expression. Then $\text{RLTL}_{lv}$ consists of the formulas defined by the following grammar:

$$\varphi ::= b \mid \varphi \land \varphi \mid X \varphi \mid (b \land \varphi) \lor (\neg b \land \varphi) \mid [(b \land \varphi) \mathbf{W} (\neg b \land \varphi)] \mid r \rightarrow \varphi$$

The analogous result for $\text{RLTL}_{lv}$ provides the promised efficiency for both simulation and model checking:

**Theorem 21** ([10]). Let $\varphi$ be an $\text{RLTL}_{lv}$ formula of size $n$. There exists an NFA of size $O(n)$ recognizing $\varphi$.
The Simple Subset

General LTL formula $\varphi$

- Buchi Automaton $B_{\neg \varphi}$
- $2^{O(n)}$

Safety LTL formula $\varphi$

- NFA recognizing bad prefixes $N_{\neg \varphi}$
- $2^{2^{\Omega(\sqrt{n})}}$

Simple Subset formula $\varphi$

- NFA recognizing informative prefixes $N'_{\neg \varphi}$
- $2^{O(n)}$

- NFA recognizing informative prefixes $N'_{\neg \varphi}$
- $O(n)$

References:

- [MVW86]
- [KV99]
What makes a spec lang good?

- Expressiveness
- Succinctness
- Good Tools
- Rigorous
  - Abstractions
  - Complexity
  - Relations to other formalisms
  - Theorems
  - Sanity Checks
  - Algebraic Properties
  - Topological relations
  - Translation to HOL

Reasonable Complexity for Required Algs.
Open Issues

- One-to-one correspondence
- Hyper Properties
- Quantitative properties
- Triggering procedural code from within a formula
- Separation of Concerns
References


F07  D. Fisman. On the characterization of until as a fixed point under clocked semantics. 2007
## References

<table>
<thead>
<tr>
<th>FL78</th>
<th>M.J. Fischer and R. E. Ladner: Propositional Dynamic Logic of Regular Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>G08</td>
<td>W. Gelade. Succinctness of regular expressions with interleaving, intersection and counting.</td>
</tr>
<tr>
<td>Pnu77</td>
<td>A. Pnueli. The Temporal Logic of Programs. FOCS 1977: 46-57</td>
</tr>
<tr>
<td>Wol83</td>
<td>P. Wolper. Temporal logic can be more expressive. 1983</td>
</tr>
</tbody>
</table>
Now let’s define a spec lang for modern CPS!

Thank you for listening!

Comments / Questions?