A Decision Procedure for Fusion Logic

Antonio Cau, Helge Janicke and Ben Moszkowski

Software Technology Research Laboratory
De Montfort University

2009

STRL Overview (2)

This talk will introduce a decision procedure for Fusion Logic

1. Introduction
2. Motivation
3. Fusion Logic
   - Syntax
   - Semantics
4. Decision Procedure for Fusion Logic
   - Reduction Step
   - Reduction Step v2
   - BDD Step
5. Policy Enforcer
6. Future work

STRL Motivation (3)

Temporal Logic:

- Reasoning about behaviours, i.e., sequences (traces) of system states
- Linear Temporal Logic (LTL):
  ○ (Next), in the next state,
  □ (Always), all states in the behaviour,
  ◇ (Sometimes), exist a state in the behaviour
- Interval Temporal Logic (ITL):
  skip, behaviour of two states,
  ; (Chop), fusion of two behaviours,
  * (chopstar), fusion of a finite number of behaviours
- Propositional Interval Temporal Logic (PITL):
  ITL with only propositional variables, i.e., Boolean values.

STRL Motivation (4)

Verification tools for Propositional Interval Temporal (PITL):

- Tempura, Ben Moszkowski, Roger Hale, 1985. Executable specification, testing
- Lite, Shinji Kono, 1991. Tableau-based
- ITL Library for interactive theorem prover PVS, Antonio Cau, 1997. Axioms via Higher-order Logic
- AnaTempura, Antonio Cau, Shikun Zhou, 1999. Runtime verification, Tempura
- DCVALID, Paritosh Pandya, 2000. MONA, decision procedure for WS1S
- PITL2Mona, Rodolfo Gomez, 2004. MONA, decision procedure for WS1S
- JavaLite, Shinji Kono, 2008. BDD, Tableau-based
- PITL Library for automated theorem prover Prover9, Antonio Cau, 2008. Algebra, proof search

STRL Syntax of Fusion Logic (5)

Syntax of:

State formulae:

$$\boldsymbol{w} ::= \text{true} \mid \boldsymbol{Q} \mid \neg \boldsymbol{w} \mid \boldsymbol{w}_1 \vee \boldsymbol{w}_2$$

Transition formulae:

$$\boldsymbol{t} ::= \text{true} \mid \boldsymbol{Q} \mid \bigcirc \boldsymbol{w} \mid \neg \boldsymbol{t} \mid \boldsymbol{t}_1 \vee \boldsymbol{t}_2$$

Fusion expressions:

$$\boldsymbol{e} ::= \text{test}(\boldsymbol{w}) \mid \text{step}(\boldsymbol{t}) \mid \boldsymbol{e}_1 \vee \boldsymbol{e}_2 \mid \boldsymbol{e}_1 \,;\, \boldsymbol{e}_2 \mid \boldsymbol{e}^*$$

Fusion logic formulae:

$$\boldsymbol{f} ::= \text{true} \mid \boldsymbol{Q} \mid \neg \boldsymbol{f} \mid \boldsymbol{f}_1 \vee \boldsymbol{f}_2 \mid \langle \boldsymbol{e} \rangle \boldsymbol{f}$$

State

A state is a mapping ***State*** from the set of propositional variables $\boldsymbol{Var^b}$ to the set of Boolean values $\boldsymbol{Bool} \mathrel{\widehat{=}} \{\mathrm{tt}, \mathrm{ff}\}$. tt is the semantic 'true' value and ff the semantic 'false' value.

$$\boldsymbol{State} : \boldsymbol{Var^b} \to \boldsymbol{Bool}$$

We will use $\sigma_0, \sigma_1, \sigma_2, \ldots$ to denote states and $\boldsymbol{\Sigma}$ to denote the set of all possible states.

Example

Let σ_0 be a state such that

$$\sigma_0(\boldsymbol{P}) = \mathrm{tt}$$
$$\sigma_0(\boldsymbol{Q}) = \mathrm{ff}$$

Interval and Length (7)

An interval σ is a finite sequence of states

$$\sigma : \sigma_0\sigma_1\sigma_2\ldots$$

Let Σ^+ denote the set of all possible finite intervals with at least one state.
The length of an interval σ is denoted by $|\sigma|$ and is the number of states minus 1.

Example

$$\begin{aligned} \sigma &= \sigma_0 & |\sigma| &= 0 \\ \sigma &= \sigma_0\sigma_1 & |\sigma| &= 1 \\ \sigma &= \sigma_0\sigma_1\ldots\sigma_n & |\sigma| &= n \end{aligned}$$

STRL Prefix, Suffix and Sub Interval (8)

Let $\sigma = \sigma_0\sigma_1\sigma_2\ldots$ be an interval then

- $\sigma_0 \ldots \sigma_k$ (where $0 \leq k \leq |\sigma|$) denotes a prefix interval of σ
- $\sigma_k \ldots \sigma_{|\sigma|}$ (where $0 \leq k \leq |\sigma|$) denotes a suffix interval of σ
- $\sigma_k \ldots \sigma_l$ (where $0 \leq k \leq l \leq |\sigma|$) denotes a sub interval of σ

Example

Let $\sigma = \sigma_0\sigma_1\sigma_2\sigma_3$ be an interval then

$\sigma_0\sigma_1$	is a prefix interval of σ
$\sigma_1\sigma_2\sigma_3$	is a suffix interval of σ
$\sigma_1\sigma_2$	is a sub interval of σ

STRL Semantic Boolean operators (9)

Truth tables for Semantic Boolean operators:

- not :

X	not ***X***
tt	ff
ff	tt

- and :

X	***Y***	***X*** and ***Y***
tt	tt	tt
tt	ff	ff
ff	tt	ff
ff	ff	ff

or :

X	***Y***	***X*** or ***Y***
tt	tt	tt
tt	ff	tt
ff	tt	tt
ff	ff	ff

- implies :

X	***Y***	***X*** implies ***Y***
tt	tt	tt
tt	ff	ff
ff	tt	tt
ff	ff	tt

iff :

X	***Y***	***X*** iff ***Y***
tt	tt	tt
tt	ff	ff
ff	tt	ff
ff	ff	tt

STRL Semantics of w (10)

Let $[\![\ldots]\!]$ be the "meaning" function from 'state formulae' $\times$ Σ^+ to {tt, ff} and let σ be a finite interval ($\sigma \in \Sigma^+$) then

$$
\begin{array}{lcl}
[\![\text{true}]\!]_\sigma & \widehat{=} & \text{tt} \\
[\![Q]\!]_\sigma & \widehat{=} & \sigma_0(Q) \\
[\![\neg w]\!]_\sigma & \widehat{=} & \text{not } ([\![w]\!]_\sigma) \\
[\![w_1 \vee w_2]\!]_\sigma & \widehat{=} & ([\![w_1]\!]_\sigma \text{ or } [\![w_2]\!]_\sigma)
\end{array}
$$

Semantics of w

Example

Let $\sigma_0(P) = \text{tt}$ and $\sigma_0(Q) = \text{ff}$.

$$
\begin{aligned}
& [\![p \vee q]\!]_\sigma \\
= \; & [\![P]\!]_\sigma \text{ or } [\![Q]\!]_\sigma \\
= \; & \sigma_0(P) \text{ or } \sigma_0(Q) \\
= \; & \text{tt or ff} \\
= \; & \text{tt}
\end{aligned}
$$

Semantics of t (12)

Let $[\![\ldots]\!]$ be the "meaning" function from 'transition formulae' $\times\ \Sigma^+$ to $\{\text{tt}, \text{ff}\}$ and let σ be a finite interval ($\sigma \in \Sigma^+$) then

$$
\begin{array}{lcl}
[\![\text{true}]\!]_\sigma & \widehat{=} & \text{tt} \\
[\![Q]\!]_\sigma & \widehat{=} & \sigma_0(Q) \\
[\![\neg t]\!]_\sigma & \widehat{=} & \text{not } ([\![t]\!]_\sigma) \\
[\![t_1 \vee t_2]\!]_\sigma & \widehat{=} & ([\![t_1]\!]_\sigma \text{ or } [\![t_2]\!]_\sigma)
\end{array}
$$

$$
[\![\bigcirc w]\!]_\sigma = \text{tt} \quad \text{iff} \quad [\![w]\!]_{\sigma_1 \ldots \sigma_{|\sigma|}} \text{ and } |\sigma| > 0
$$

Semantics of t

Example

Let $\sigma_0(Q) = \text{tt}$, $\sigma_1(Q) = \text{ff}$ and $|\sigma| = 1$.

$$
\begin{aligned}
& [\![\neg(\bigcirc Q)]\!]_\sigma \\
= \;& \text{not } ([\![\bigcirc Q]\!]_\sigma) \\
= \;& \text{not } ([\![Q]\!]_{\sigma_1 \ldots \sigma_{|\sigma|}} \text{ and } |\sigma| > 0) \\
= \;& \text{not } (\sigma_1(Q) \text{ and } |\sigma| > 0) \\
= \;& \text{not } (\text{ff and } 1 > 0) \\
= \;& \text{not } (\text{ff and tt}) \\
= \;& \text{tt}
\end{aligned}
$$

Semantics of $\boldsymbol{e}$

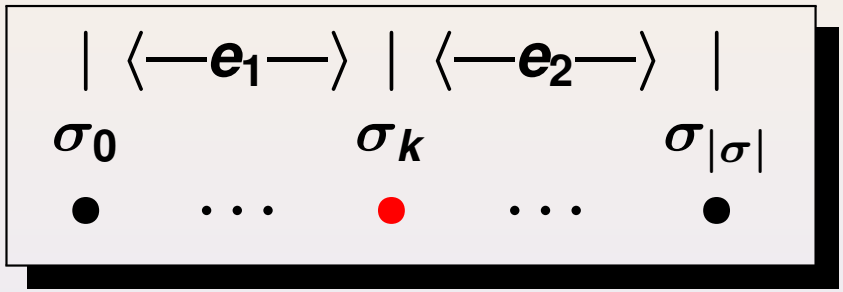
Let $[\![\ldots]\!]$ be the "meaning" function from 'fusion expressions' $\times\ \boldsymbol{\Sigma}^+$ to $\{\mathrm{tt}, \mathrm{ff}\}$ and let σ be a finite interval ($\sigma \in \boldsymbol{\Sigma}^+$) then

$$
\begin{array}{lcl}
[\![\mathrm{test}(\boldsymbol{w})]\!]_\sigma & \widehat{=} & [\![\boldsymbol{w}]\!]_\sigma \text{ and } |\sigma| = \mathbf{0} \\
[\![\mathrm{step}(\boldsymbol{t})]\!]_\sigma & \widehat{=} & [\![\boldsymbol{t}]\!]_\sigma \text{ and } |\sigma| = \mathbf{1} \\
[\![\boldsymbol{e}_1 \vee \boldsymbol{e}_2]\!]_\sigma & \widehat{=} & [\![\boldsymbol{e}_1]\!]_\sigma \text{ or } [\![\boldsymbol{e}_2]\!]_\sigma
\end{array}
$$

Semantics of e

The semantics of 'chop' is as follows $[\![e_1 ; e_2]\!]_\sigma = \text{tt}$ iff

$$(\text{exists } k, \text{ s.t. } [\![e_1]\!]_{\sigma_0 \ldots \sigma_k} = \text{tt} \text{ and } [\![e_2]\!]_{\sigma_k \ldots \sigma_{|\sigma|}} = \text{tt})$$

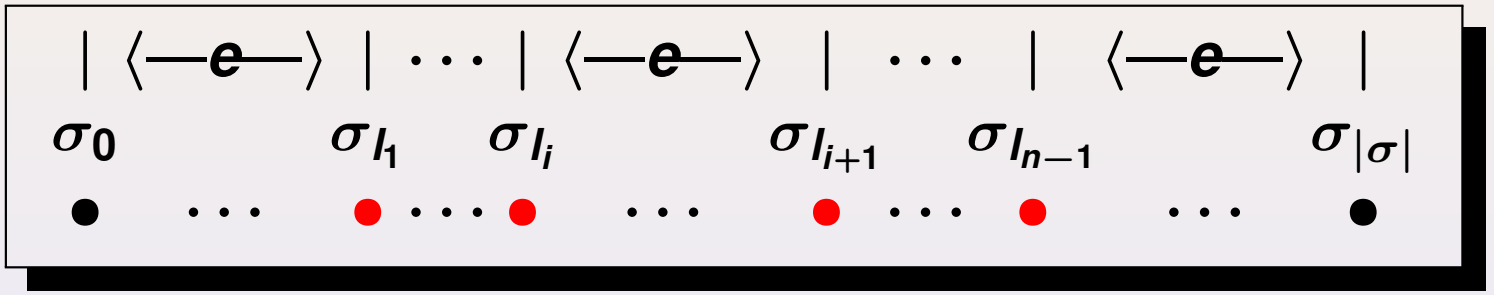


Interval σ is a fusion of prefix interval $\sigma_0 \ldots \sigma_k$ (satisfies e_1) and suffix interval $\sigma_k \ldots \sigma_{|\sigma|}$ (satisfies e_2). State σ_k is shared by both.

Semantics of e

The semantics of 'chopstar' is as follows $[\![e^*]\!]_\sigma =$ tt iff

(exist $l_0, \ldots, l_n$ s.t. $l_0 = 0$ and $l_n = |\sigma|$ and
for all $0 \leq i < n, l_i \leq l_{i+1}$ and $[\![e]\!]_{\sigma_{l_i} \ldots \sigma_{l_{i+1}}} =$ tt)



Interval σ is the fusion of a finite number of sub-intervals each satisfying e.

Semantics of f

Let $[\![\ldots]\!]$ be the "meaning" function from 'fusion logic formulae' $\times\ \Sigma^+$ to $\{\text{tt}, \text{ff}\}$ and let σ be a finite interval ($\sigma \in \Sigma^+$) then

$$
\begin{array}{lcl}
[\![\text{true}]\!]_\sigma & \widehat{=} & \text{tt} \\
[\![Q]\!]_\sigma & \widehat{=} & \sigma_0(Q) \\
[\![\neg f]\!]_\sigma & \widehat{=} & \text{not } ([\![f]\!]_\sigma) \\
[\![e_1 \vee e_2]\!]_\sigma & \widehat{=} & [\![e_1]\!]_\sigma \text{ or } [\![e_2]\!]_\sigma \\
[\![\langle e \rangle f]\!]_\sigma = \text{tt} & \text{iff} & \text{exists a } k, \text{ s.t.} \\
 & & [\![e]\!]_{\sigma_0 \ldots \sigma_k} = \text{tt and } [\![f]\!]_{\sigma_k \ldots \sigma_{|\sigma|}} = \text{tt}
\end{array}
$$

Derived operators (18)

Derived Fusion expression operators

$\mathsf{len}(\mathbf{0})$	$\widehat{=}$	$\mathsf{test}(\mathsf{true})$	one state interval
$\mathsf{len}(\boldsymbol{n}+\mathbf{1})$	$\widehat{=}$	$\mathsf{step}(\mathsf{true})\ ;\ \mathsf{len}(\boldsymbol{n})$	interval with $\boldsymbol{n}+\mathbf{2}$ states
finite	$\widehat{=}$	$(\mathsf{len}(\mathbf{1}))^*$	all finite intervals

Derived Fusion logic operators

$\boldsymbol{f_1} \wedge \boldsymbol{f_2}$	$\widehat{=}$	$\neg(\neg \boldsymbol{f_1} \vee \neg \boldsymbol{f_2})$	conjunction
$\bigcirc \boldsymbol{f}$	$\widehat{=}$	$\langle \mathsf{len}(\mathbf{1}) \rangle \boldsymbol{f}$	$\boldsymbol{f}$ in next state
more	$\widehat{=}$	$\bigcirc \mathsf{true}$	all intervals with $\geq \mathbf{2}$ states
empty	$\widehat{=}$	$\neg \mathsf{more}$	all intervals with $\mathbf{1}$ state
skip	$\widehat{=}$	$\bigcirc \mathsf{empty}$	all intervals with $\mathbf{2}$ states
$\Diamond \boldsymbol{f}$	$\widehat{=}$	$\langle \mathsf{finite} \rangle \boldsymbol{f}$	$\boldsymbol{f}$ in some suffix interval
$\Box \boldsymbol{f}$	$\widehat{=}$	$\neg \Diamond \neg \boldsymbol{f}$	$\boldsymbol{f}$ in all suffix intervals
$\mathsf{fin}\ \boldsymbol{f}$	$\widehat{=}$	$\Box(\mathsf{more} \vee \boldsymbol{f})$	$\boldsymbol{f}$ holds in final state
$[\boldsymbol{e}]\boldsymbol{f}$	$\widehat{=}$	$\neg(\langle \boldsymbol{e} \rangle \neg \boldsymbol{f})$	$\boldsymbol{e}$ always followed by $\boldsymbol{f}$

Satisfiable and valid (19)

- A fusion logic formula f is satisfiable if and only if there exists an interval σ such that $[\![f]\!]_\sigma = \mathrm{tt}$
- Decision procedure checks whether f is satisfiable or not, when f is satisfiable a satisfying interval is generated.
- A fusion logic formula f is valid if and only if for all intervals σ, $[\![f]\!]_\sigma = \mathrm{tt}$
- f is not valid if and only if $\neg f$ is satisfiable

  i.e., satisfying interval for $\neg f$ will represent a counter example for f's validity

  f is valid if and only if $\neg f$ is not satisfiable

Decision Procedure

- Reduction Step:

  transform $\mathbf{f}$ *into* $\mathbf{init} \wedge \Box\, \mathbf{l}$

- BDD Step:

  transform $\mathbf{init} \wedge \Box\, \mathbf{l}$ *into a BDD-based satisfiability problem*

Reduction Step (21)

Transform FL formula f into an equivalent reduced form $init \wedge \Box\, l$

- $init$: a state formula r_k
- l: an invariant, $\bigwedge_{i=1}^{i=k}(r_i \equiv t_i)$ (for $k \geq 1$) where

  r_i is a dependent Boolean variable (not appearing in f)

  t_i is a transition formula

Reduction Step

Let $\mathcal{R}(f)$ be a symbolic reduction function that transform any FL formula f into its corresponding invariant of the form $\bigwedge_{i=1}^{i=k}(r_i \equiv t_i)$.

Idea: $\mathcal{R}()$ introduces dependent variables r_i's to 'rewrite' all the FL temporal operators except $\bigcirc$.

Let $|\mathcal{R}(f)|$ denote the number of distinct dependent variables in $\mathcal{R}(f)$.

Reduction Step (23)

Transformation of Fusion Logic Formula:

Let X, X_1 and X_2 denote non state formulae and w a state formula.

f	$\mathcal{R}(f)$
w	$r_1 \equiv w$
$\neg X$	$\mathcal{R}(X) \wedge (r_{k+1} \equiv \neg r_k)$ where $k = \lvert\mathcal{R}(X)\rvert$
$X_1 \vee X_2$	$\mathcal{R}(X_1) \wedge \mathcal{R}(X_2) \uparrow j \wedge r_{j+k+1} \equiv (r_j \vee r_{k+j})$ where $j = \lvert\mathcal{R}(X_1)\rvert$ and $k = \lvert\mathcal{R}(X_2)\rvert$
$\langle e \rangle X$	$\mathcal{R}(X) \wedge (\mathcal{R}(\langle e \rangle q) \uparrow k)[q \leftarrow r_k]$ where q is a fresh variable and $k = \lvert\mathcal{R}(X)\rvert$

where, for any invariant I, the operation $I \uparrow k$ is defined to be the invariant where the subscripts of dependent variables r_i are shifted by k, i.e., r_i becomes r_{i+k}.

Reduction Step

Transformation of Fusion expressions:

We are left with only FL formulae of the form $\langle e \rangle w$.

We will use the following reduction rules to rewrite them

f	$\mathcal{R}(f)$
$\langle \text{test}(w') \rangle w$	$r_1 \equiv (w \wedge w')$
$\langle \text{step}(t) \rangle w$	$r_1 \equiv (t \wedge \bigcirc w)$
$\langle e_1 \vee e_2 \rangle w$	$\mathcal{R}(\langle e_1 \rangle w \vee \langle e_2 \rangle w)$
$\langle e_1 ; e_2 \rangle w$	$\mathcal{R}(\langle e_1 \rangle (\langle e_2 \rangle w))$

Reduction Step

Reduction function for $\langle e^* \rangle w$ is problematic because e could be valid for intervals with only one state.

Solution: a function c is introduced which transforms an arbitrary fusion expression e into another formula $c(e)$ such that $c(e) \equiv (e \wedge \text{len}(n))$ holds ($n \geq 1$).

Note: $\langle e^* \rangle w \equiv \langle c(e)^* \rangle w$ holds.

e	$c(e)$
$\text{test}(w)$	$\text{test}(\neg\text{true})$
$\text{step}(t)$	$\text{step}(t)$
$e_1 \vee e_2$	$c(e_1) \vee c(e_2)$
$e_1 \,;\, e_2$	$c(e_1) \,;\, e_2 \vee e_1 \,;\, c(e_2)$
e^*	$c(e) \,;\, e^*$

Reduction of $\langle e^* \rangle w$ is then follows

$$\langle e^* \rangle w \quad \mathcal{R}(w \vee \langle c(e) \rangle q)[q \leftarrow r_k]$$

where q is a fresh variable and

$$k = |\mathcal{R}(w \vee \langle c(e) \rangle q)|$$

Reduction Step

Reduction function for $\langle e^* \rangle w$ is problematic because e could be valid for intervals with only one state.
Solution: a function c is introduced which transforms an arbitrary fusion expression e into another formula $c(e)$ such that $c(e) \equiv (e \wedge \text{len}(n))$ holds ($n \geq 1$).
Note: $\langle e^* \rangle w \equiv \langle c(e)^* \rangle w$ holds.

e	$c(e)$
test(w)	test($\neg$true)
step(t)	step(t)
$e_1 \vee e_2$	$c(e_1) \vee c(e_2)$
$e_1 ; e_2$	$c(e_1) ; e_2 \vee e_1 ; c(e_2)$
e^*	$c(e) ; e^*$

Reduction of $\langle e^* \rangle w$ is then follows

$\langle e^* \rangle w$ $\quad \mathcal{R}(w \vee \langle c(e) \rangle q)[q \leftarrow r_k]$
where q is a fresh variable and
$k = |\mathcal{R}(w \vee \langle c(e) \rangle q)|$

Example

Given FL formula

$$\langle \text{step}(\boldsymbol{A})^* \rangle (\boldsymbol{B} \vee \boldsymbol{C}) \vee \langle \text{step}(\boldsymbol{A}) \,;\, \text{test}(\boldsymbol{B}) \rangle \boldsymbol{D}$$

Then ***init*** is $\boldsymbol{r_6}$.
The corresponding invariant ***I*** is as follows:

$$\begin{aligned}
& \boldsymbol{r_6} \equiv \boldsymbol{r_3} \vee \boldsymbol{r_5} \\
\wedge \quad & \boldsymbol{r_5} \equiv \boldsymbol{A} \wedge \bigcirc \boldsymbol{r_4} \\
\wedge \quad & \boldsymbol{r_4} \equiv \boldsymbol{D} \wedge \boldsymbol{B} \\
\wedge \quad & \boldsymbol{r_3} \equiv \boldsymbol{r_1} \vee \boldsymbol{r_2} \\
\wedge \quad & \boldsymbol{r_2} \equiv \boldsymbol{A} \wedge \bigcirc \boldsymbol{r_3} \\
\wedge \quad & \boldsymbol{r_1} \equiv \boldsymbol{B} \vee \boldsymbol{C}
\end{aligned}$$

Reduction Step v2

Reduction technique that introduces fewer dependent variables.

Transform FL formula f into following equivalent reduced form

$$init \wedge \Box\, l$$

where

$$init \mathrel{\hat{=}} \mathcal{R}'_0(f)$$
$$l \mathrel{\hat{=}} \mathcal{R}_0(f)$$

For $k \in \{0, 1\}$, $\mathcal{R}_k(f)$ is a transition formula, $\mathcal{R}'_0(f)$ is a state formula and $\mathcal{R}'_1(f)$ is a transition formula.

Reduction Step v2 (28)

Let X, X_1 and X_2 denote non state formulae and w a state formula then the definition of Transition formula $\mathcal{R}_k(f)$ is as follows:

f	$\mathcal{R}_k(f)$
w	true
$\langle \text{test}(w) \rangle X$	$\mathcal{R}_k(X)$
$\langle \text{step}(t) \rangle X$	$k = 0 : (r_{\langle \text{step}(t) \rangle X} \equiv (t \wedge \bigcirc \mathcal{R}'_0(X))) \wedge \mathcal{R}_0(X)$
	$k = 1 : \mathcal{R}_0(X)$
$\langle e_1 \vee e_2 \rangle X$	$\mathcal{R}_k(\langle e_1 \rangle X \vee \langle e_2 \rangle X)$
$\langle e_1 ; e_2 \rangle X$	$\mathcal{R}_k(\langle e_1 \rangle \langle e_2 \rangle X)$
$\langle e^* \rangle X$	$(r_{\langle e^* \rangle X} \equiv \mathcal{R}'_1(X_1)) \wedge \mathcal{R}_1(X_1)$, where X_1 is $X \vee \langle c(e) \rangle r_{\langle e^* \rangle X}$
$\neg X$	$\mathcal{R}_k(X)$
$X_1 \vee X_2$	$\mathcal{R}_k(X_1) \wedge \mathcal{R}_k(X_2)$

So only the step(t) and e^* case will introduce a dependent variable.

Reduction Step v2

Let X, X_1 and X_2 denote non state formulae and w a state formula then the definition of state formula $\mathcal{R}'_k(f)$ is as follows:

f	$\mathcal{R}'_k(f)$
w	w
$\langle \text{test}(w) \rangle X$	$w \wedge \mathcal{R}'_k(X)$
$\langle \text{step}(t) \rangle X$	$k = 0 : r_{\langle \text{step}(t) \rangle X}$
	$k = 1 : t \wedge \bigcirc \mathcal{R}'_0(X)$
$\langle e_1 \vee e_2 \rangle X$	$\mathcal{R}'_k(\langle e_1 \rangle X \vee \langle e_2 \rangle X)$
$\langle e_1 ; e_2 \rangle X$	$\mathcal{R}'_k(\langle e_1 \rangle \langle e_2 \rangle X)$
$\langle e^* \rangle X$	$r_{\langle e^* \rangle X}$
$\neg X$	$\neg \mathcal{R}'_k(X)$
$X_1 \vee X_2$	$\mathcal{R}'_k(X_1) \vee \mathcal{R}'_k(X_2)$

Example

Given again FL formula

$$\langle \text{step}(\boldsymbol{A})^* \rangle (\boldsymbol{B} \vee \boldsymbol{C}) \vee \langle \text{step}(\boldsymbol{A}) \,;\, \text{test}(\boldsymbol{B}) \rangle \boldsymbol{D}$$

we now use Reduction Step v2.

The ***init*** is $\boldsymbol{r}_{\boldsymbol{X}_1} \vee \boldsymbol{r}_{\boldsymbol{X}_3}$ where
$\boldsymbol{X}_1 \mathrel{\widehat{=}} \langle \text{step}(\boldsymbol{A})^* \rangle \boldsymbol{B} \vee \boldsymbol{C}$ and $\boldsymbol{X}_3 \mathrel{\widehat{=}} \langle \text{step}(\boldsymbol{A}) \rangle \langle \text{test}(\boldsymbol{B}) \rangle \boldsymbol{D}$.

The corresponding invariant:

$$\begin{aligned} & \boldsymbol{r}_{\boldsymbol{X}_1} \equiv (\boldsymbol{B} \vee \boldsymbol{C}) \vee (\boldsymbol{A} \wedge \bigcirc \boldsymbol{r}_{\boldsymbol{X}_1}) \\ \wedge \quad & \boldsymbol{r}_{\boldsymbol{X}_3} \equiv \boldsymbol{A} \wedge \bigcirc (\boldsymbol{B} \wedge \boldsymbol{D}) \end{aligned}$$

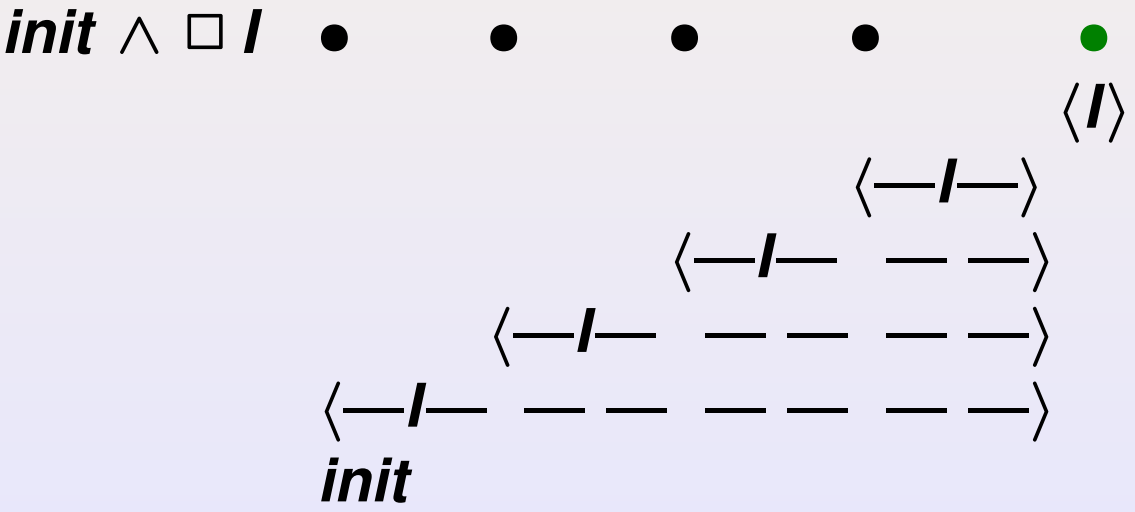
So the number of dependent variables has been reduced from 6 to 2.

BDD Step

Transform $\boldsymbol{init} \wedge \Box\, \boldsymbol{I}$ into BDD based satisfiability problem.

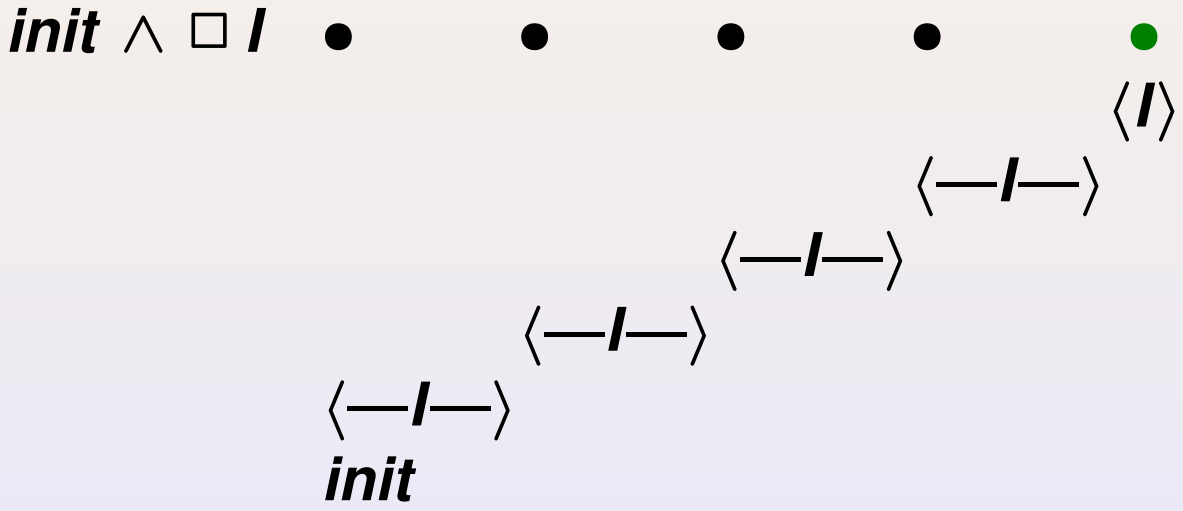
$\boldsymbol{init}$: a state formula $\boldsymbol{r_k}$
$\boldsymbol{I}$: an invariant, $\bigwedge_{i=1}^{i=k}(\boldsymbol{r_i} \equiv \boldsymbol{t_i})$ (for $\boldsymbol{k} \geq \mathbf{1}$) where
$\boldsymbol{r_i}$ is a dependent variable and
$\boldsymbol{t_i}$ is a transition formula

Let us examine the 'Always' operator



BDD Step

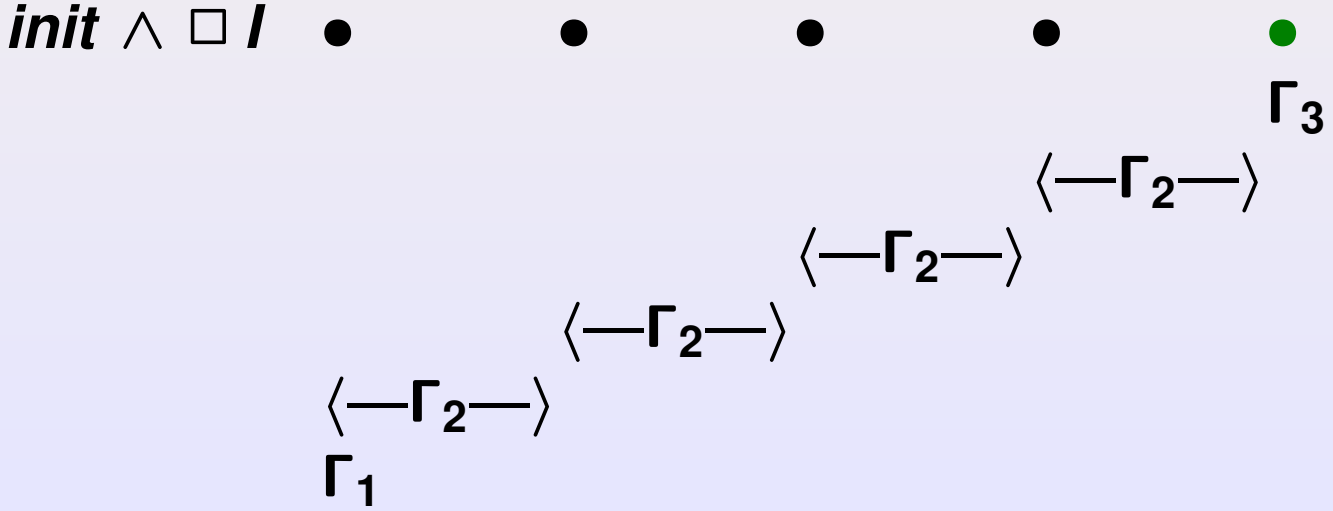
We know that invariant ***I*** only contains ○ (next) as temporal operator. So it can only constrain the first two states of each suffix interval.



BDD Step

Introducing BDDs Γ_i's:

- Γ_1 represents the state formula ***init***.
- Γ_2 captures all pairs of states corresponding to two state intervals satisfying invariant ***I***. This can be done by replacing all variables in the scope of any $\bigcirc$ by a primed version and delete the $\bigcirc$.
- Γ_3 captures the behaviour of invariant ***I*** in an interval with only 1 state. This can be done by replacing each $\bigcirc$ construct by false, i.e., $\neg$true.



Example

Given FL formula

$$\langle \text{step}(\boldsymbol{A})^* \rangle (\boldsymbol{B} \vee \boldsymbol{C}) \vee \langle \text{step}(\boldsymbol{A}) \,;\, \text{test}(\boldsymbol{B}) \rangle \boldsymbol{D}$$

With corresponding invariant:

$$\begin{array}{llll} r_6 \equiv r_3 \vee r_5 & \wedge & r_5 \equiv \boldsymbol{A} \wedge \bigcirc r_4 & \wedge \\ r_4 \equiv \boldsymbol{D} \wedge \boldsymbol{B} & \wedge & r_3 \equiv r_1 \vee r_2 & \wedge \\ r_2 \equiv \boldsymbol{A} \wedge \bigcirc r_3 & \wedge & r_1 \equiv \boldsymbol{B} \vee \boldsymbol{C} & \end{array}$$

Then $\Gamma_1 \mathrel{\widehat{=}} r_6$

$\Gamma_2 \mathrel{\widehat{=}}$

$$\begin{array}{llll} r_6 \equiv r_3 \vee r_5 & \wedge & r_5 \equiv \boldsymbol{A} \wedge r_4' & \wedge \\ r_4 \equiv \boldsymbol{D} \wedge \boldsymbol{B} & \wedge & r_3 \equiv r_1 \vee r_2 & \wedge \\ r_2 \equiv \boldsymbol{A} \wedge r_3' & \wedge & r_1 \equiv \boldsymbol{B} \vee \boldsymbol{C} & \end{array}$$

$\Gamma_3 \mathrel{\widehat{=}}$

$$\begin{array}{llll} r_6 \equiv r_3 \vee r_5 & \wedge & r_5 \equiv \boldsymbol{A} \wedge \text{false} & \wedge \\ r_4 \equiv \boldsymbol{D} \wedge \boldsymbol{B} & \wedge & r_3 \equiv r_1 \vee r_2 & \wedge \\ r_2 \equiv \boldsymbol{A} \wedge \text{false} & \wedge & r_1 \equiv \boldsymbol{B} \vee \boldsymbol{C} & \end{array}$$

BDD Step

Encoding as a BDD-based satisfiability problem:

- We use Γ_2 and Γ_1 to iteratively calculate a sequence of BDDs $\Delta_0, \ldots, \Delta_n$, so that for any I, Δ_n described all states which can be reached from Γ_1 in exactly n steps using Γ_2.
- We determine at each iteration whether BDD $\Gamma_3 \wedge \Delta_n$ is true or not. If false we must continue to iterate and if true then there is exists some state satisfying Γ_3 which can be reached in n steps from Γ_1, so we can stop the iteration, i.e., original f is satisfiable.
- During the iteration process we maintain a BDD $\bigvee_{0 \leq i \leq n} \Delta_i$ representing the set of all states so far reachable from Γ_1. If $(\bigvee_{0 \leq i \leq n} \Delta_i) \equiv (\bigvee_{0 \leq i \leq n+1} \Delta_i)$, i.e., no new states are found, then we stop the iteration and if we can't find a state that satisfies Γ_3 then original f is not satisfiable.

BDD Step

Encoding as a BDD-based satisfiability problem:

- We use Γ_2 and Γ_1 to iteratively calculate a sequence of BDDs $\Delta_0, \ldots, \Delta_n$, so that for any I, Δ_n described all states which can be reached from Γ_1 in exactly n steps using Γ_2.
- We determine at each iteration whether BDD $\Gamma_3 \wedge \Delta_n$ is true or not. If false we must continue to iterate and if true then there is exists some state satisfying Γ_3 which can be reached in n steps from Γ_1, so we can stop the iteration, i.e., original f is satisfiable.
- During the iteration process we maintain a BDD $\bigvee_{0 \le i \le n} \Delta_i$ representing the set of all states so far reachable from Γ_1. If $(\bigvee_{0 \le i \le n} \Delta_i) \equiv (\bigvee_{0 \le i \le n+1} \Delta_i)$, i.e., no new states are found, then we stop the iteration and if we can't find a state that satisfies Γ_3 then original f is not satisfiable.

BDD Step

Encoding as a BDD-based satisfiability problem:

- We use Γ_2 and Γ_1 to iteratively calculate a sequence of BDDs $\Delta_0, \ldots, \Delta_n$, so that for any I, Δ_n described all states which can be reached from Γ_1 in exactly n steps using Γ_2.
- We determine at each iteration whether BDD $\Gamma_3 \wedge \Delta_n$ is true or not. If false we must continue to iterate and if true then there is exists some state satisfying Γ_3 which can be reached in n steps from Γ_1, so we can stop the iteration, i.e., original f is satisfiable.
- During the iteration process we maintain a BDD $\bigvee_{0 \leq i \leq n} \Delta_i$ representing the set of all states so far reachable from Γ_1. If $(\bigvee_{0 \leq i \leq n} \Delta_i) \equiv (\bigvee_{0 \leq i \leq n+1} \Delta_i)$, i.e., no new states are found, then we stop the iteration and if we can't find a state that satisfies Γ_3 then original f is not satisfiable.

BDD Step

To construct a satisfying interval (in case f is satisfiable) we proceed as follows.
Let Δ_m be that set of states for which $\Gamma_3 \wedge \Delta_m$ is true.

- If there are no independent variables (only r_i variables) then any interval of length m will satisfy f.
- If there are independent variables then
  Find a value assignment σ_m for the independent variables for BDD Δ_m, i.e., choose one state σ_m of Δ_m.

  Compute Pre_{m-1} denoting those states of Δ_{m-1} that lead via Γ_2 to state σ_m (weakest precondition of Γ_2 and σ_m). Again choose one state σ_{m-1} of Pre_{m-1}.

  Continue until we reach Pre_0 and then choose state σ_0.

  The states $\sigma_0 \ldots \sigma_{m-1}\sigma_m$ will then represent a (minimal) satisfying interval σ for f.

BDD Step

To construct a satisfying interval (in case f is satisfiable) we proceed as follows.
Let Δ_m be that set of states for which $\Gamma_3 \wedge \Delta_m$ is true.

- If there are no independent variables (only r_i variables) then any interval of length m will satisfy f.
- If there are independent variables then
  Find a value assignment σ_m for the independent variables for BDD Δ_m, i.e., choose one state σ_m of Δ_m.

  Compute Pre_{m-1} denoting those states of Δ_{m-1} that lead via Γ_2 to state σ_m (weakest precondition of Γ_2 and σ_m). Again choose one state σ_{m-1} of Pre_{m-1}.

  Continue until we reach Pre_0 and then choose state σ_0.

  The states $\sigma_0 \ldots \sigma_{m-1}\sigma_m$ will then represent a (minimal) satisfying interval σ for f.

BDD Step

To construct a satisfying interval (in case f is satisfiable) we proceed as follows.
Let Δ_m be that set of states for which $\Gamma_3 \wedge \Delta_m$ is true.

- If there are no independent variables (only r_i variables) then any interval of length m will satisfy f.
- If there are independent variables then
  Find a value assignment σ_m for the independent variables for BDD Δ_m, i.e., choose one state σ_m of Δ_m.

  Compute Pre_{m-1} denoting those states of Δ_{m-1} that lead via Γ_2 to state σ_m (weakest precondition of Γ_2 and σ_m). Again choose one state σ_{m-1} of Pre_{m-1}.

  Continue until we reach Pre_0 and then choose state σ_0.

  The states $\sigma_0 \ldots \sigma_{m-1}\sigma_m$ will then represent a (minimal) satisfying interval σ for f.

BDD Step

To construct a satisfying interval (in case f is satisfiable) we proceed as follows.
Let Δ_m be that set of states for which $\Gamma_3 \wedge \Delta_m$ is true.

- If there are no independent variables (only r_i variables) then any interval of length m will satisfy f.
- If there are independent variables then
  Find a value assignment σ_m for the independent variables for BDD Δ_m, i.e., choose one state σ_m of Δ_m.

  Compute Pre_{m-1} denoting those states of Δ_{m-1} that lead via Γ_2 to state σ_m (weakest precondition of Γ_2 and σ_m). Again choose one state σ_{m-1} of Pre_{m-1}.

  Continue until we reach Pre_0 and then choose state σ_0.

  The states $\sigma_0 \ldots \sigma_{m-1}\sigma_m$ will then represent a (minimal) satisfying interval σ for f.

BDD Step

To construct a satisfying interval (in case f is satisfiable) we proceed as follows.
Let Δ_m be that set of states for which $\Gamma_3 \wedge \Delta_m$ is true.

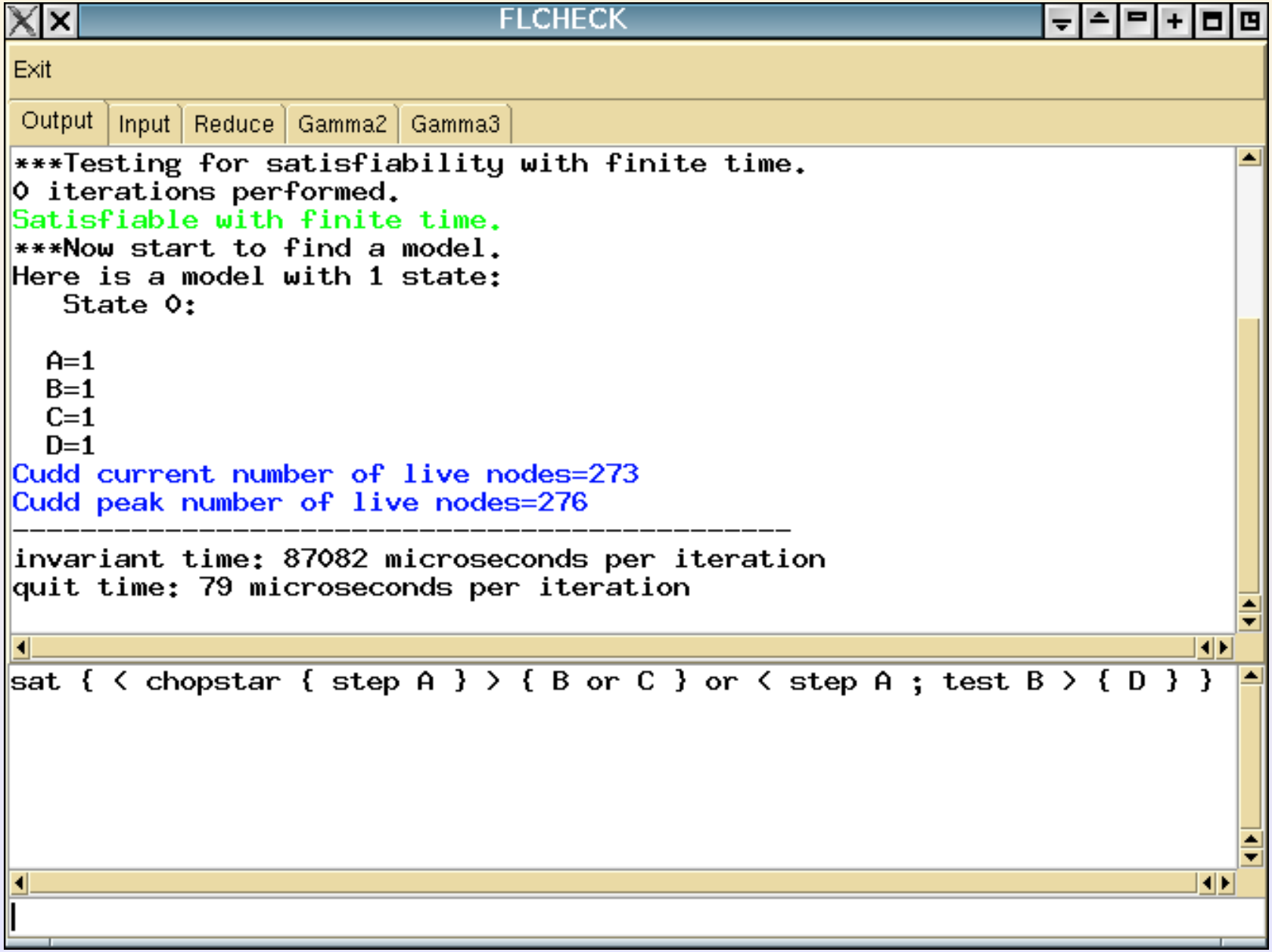
- If there are no independent variables (only r_i variables) then any interval of length m will satisfy f.
- If there are independent variables then
  Find a value assignment σ_m for the independent variables for BDD Δ_m, i.e., choose one state σ_m of Δ_m.

  Compute Pre_{m-1} denoting those states of Δ_{m-1} that lead via Γ_2 to state σ_m (weakest precondition of Γ_2 and σ_m). Again choose one state σ_{m-1} of Pre_{m-1}.

  Continue until we reach Pre_0 and then choose state σ_0.

  The states $\sigma_0 \ldots \sigma_{m-1}\sigma_m$ will then represent a (minimal) satisfying interval σ for f.

STRL Implementation (37)



Policy Enforcer

Access control policy rules are of the form **[*e*]*w***

$$[\![[e]w]\!]_\sigma = \text{tt} \quad \text{iff} \quad \text{for all } k, \text{ s.t. } [\![e]\!]_{\sigma_0 \ldots \sigma_k} = \text{tt implies } [\![w]\!]_{\sigma_k \ldots \sigma_{|\sigma|}} = \text{tt}$$

- ***w***: state formula, access control authorisations: positive, negative and decision.
- ***e***: fusion expression, premise: describing history behaviour
- Use Decision Procedure to check policy rules properties, conflicts, safety, etc.
- Enforcement:

  Policy rule is never violated

  Access control policy rule does not constrain the interval length and are 'for all'-type of formulae

Policy Enforcer

Access control policy rules are of the form $[e]w$

$$[\![[e]w]\!]_\sigma = \text{tt} \quad \text{iff} \quad \text{for all } k, \text{ s.t.}$$
$$[\![e]\!]_{\sigma_0 \ldots \sigma_k} = \text{tt implies } [\![w]\!]_{\sigma_k \ldots \sigma_{|\sigma|}} = \text{tt}$$

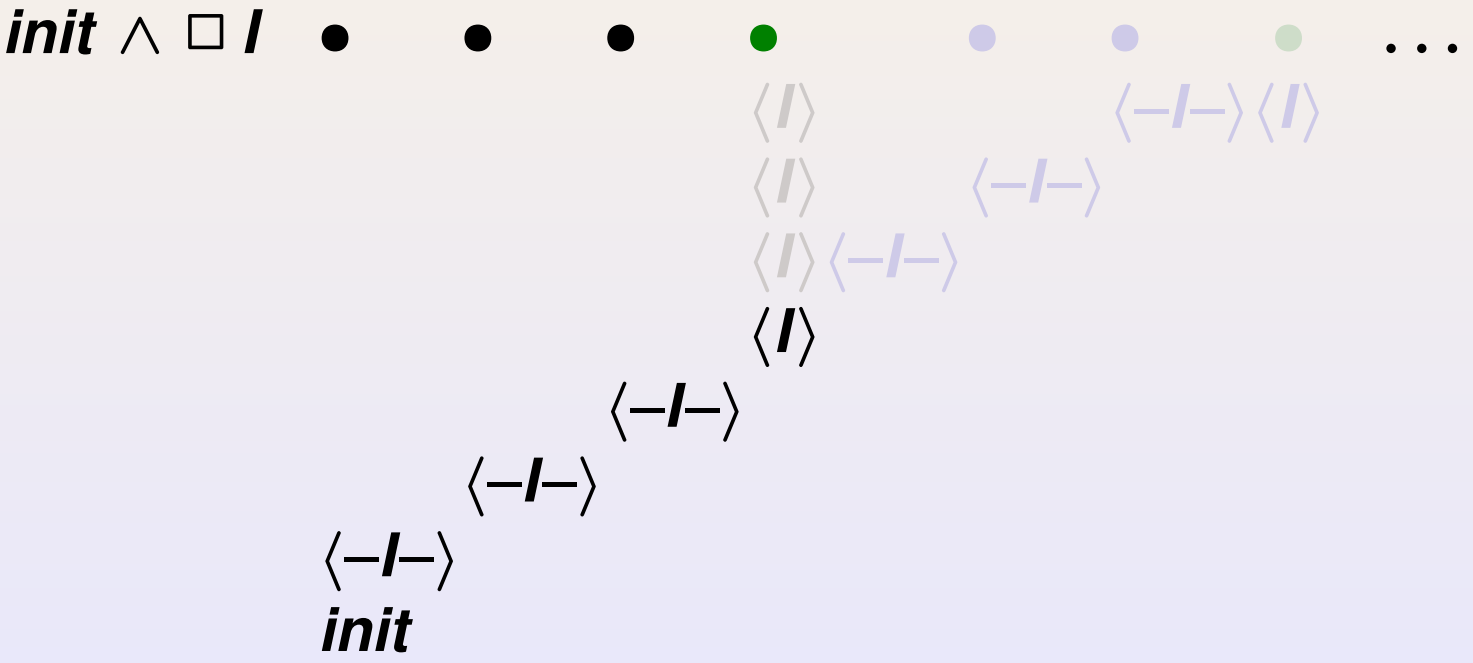
- w: state formula, access control authorisations: positive, negative and decision.
- e: fusion expression, premise: describing history behaviour
- Use Decision Procedure to check policy rules properties, conflicts, safety, etc.
- Enforcement:

  Policy rule is never violated

  Access control policy rule does not constrain the interval length and are 'for all'-type of formulae

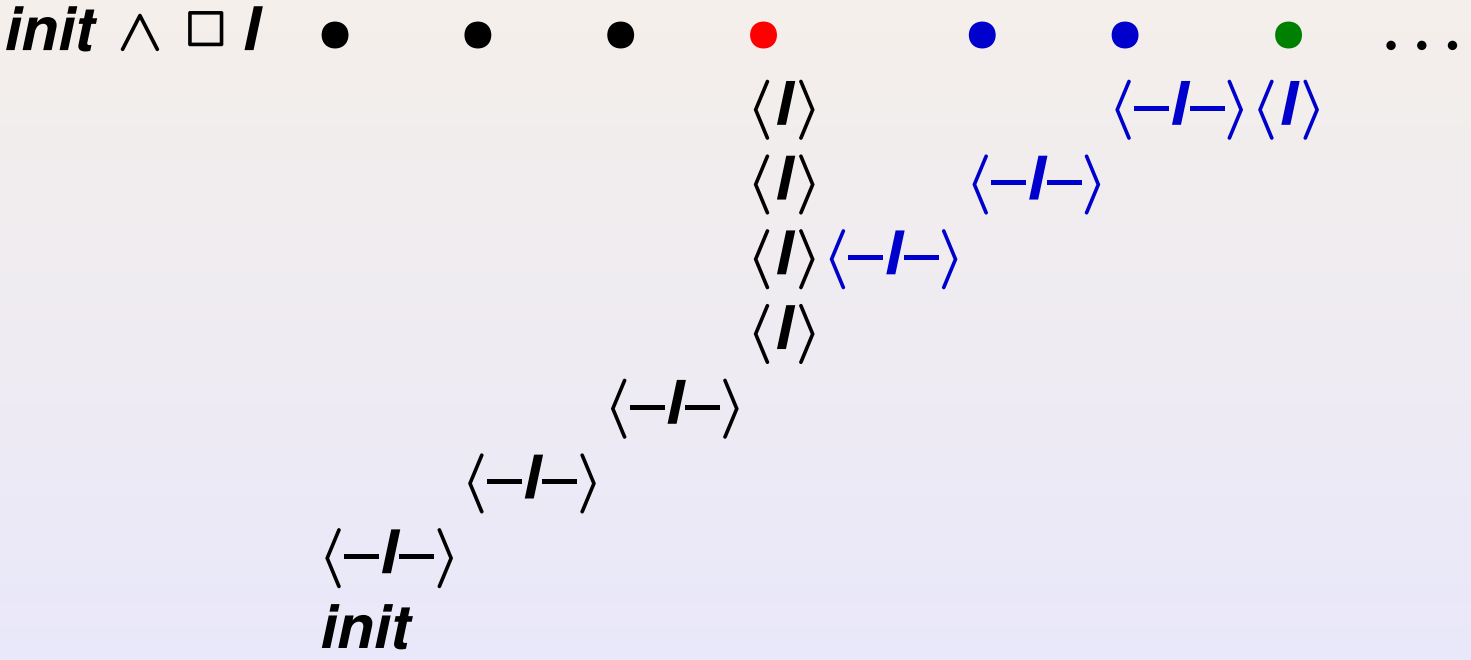
Policy Enforcer

Enforcing policy f means once a satisfying interval σ has been found continue to find the 'next' satisfying interval $\sigma \cdot \sigma'$, i.e., don't stop at the first interval σ but try to extend it with σ'. Repeat this extension step.



Policy Enforcer

Enforcing policy ***f*** means once a satisfying interval σ has been found continue to find the 'next' satisfying interval $\sigma \cdot \sigma'$, i.e., don't stop at the first interval σ but try to extend it with σ'. Repeat this extension step.



Future work

- Compare with JavaLite and PITL2MONA
- Introduce all PITL derived operators in FL
- Extend to infinite time
- Extend to integer variables
- Combine Decision Procedure with theorem prover Prover9
- Compare FL enforcement with other enforcement frameworks
- Optimise FL enforcement
- Compare with tools for mu-calculus

Any...

QUESTIONS?