A Decision Procedure for Fusion Logic

Antonio Cau, Helge Janicke and Ben Moszkowski

(Html version of slides)

Overview (2)

This talk will introduce a decision procedure for Fusion Logic

Introduction

Motivation

Fusion Logic
- Syntax
- Semantics

Decision Procedure for Fusion Logic
- Time Reversal Step
- Reduction Step
- BDD Step

Future work

Motivation (3)

Temporal Logic:

- Reasoning about behaviours, i.e., sequences (traces) of system states
- Linear Temporal Logic (LTL):
  $\bigcirc$ (Next), in the next state,
  $\Box$ (Always), all states in the behaviour,
  $\Diamond$ (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.

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

Syntax of Fusion Logic (5)

State formulae:

$$W ::= \text{true} \mid p \mid W_1 \vee W_2 \mid \neg W$$

Transition formulae:

$$T ::= W \mid \bigcirc W \mid T_1 \vee T_2 \mid \neg T$$

Fusion expressions:

$$E ::= \text{test}(W) \mid \text{step}(T) \mid E_1 \vee E_2 \mid E_1 ; E_2 \mid E^*$$

Right Fusion logic formulae:

$$R ::= \text{true} \mid p \mid \neg R \mid R_1 \vee R_2 \mid \langle E \rangle R$$

Left Fusion logic formulae:

$$L ::= \text{true} \mid \text{fin}(p) \mid \neg L \mid L_1 \vee L_2 \mid L\langle E \rangle$$

Fusion logic formulae:

$$F ::= L \mid R$$

State (6)

A state is a mapping $\boldsymbol{State}$ from the set of propositional variables $\mathbf{Var}^{\boldsymbol{b}}$ to the set of Boolean values $\boldsymbol{Bool} \triangleq \{\mathrm{tt}, \mathrm{ff}\}$.
tt is the semantic 'true' value and ff the semantic 'false' value.
$\boldsymbol{State} : \mathbf{Var}^{\boldsymbol{b}} \to \boldsymbol{Bool}$
We will use $\boldsymbol{\sigma_0, \sigma_1, \sigma_2, \ldots}$ to denote states and $\boldsymbol{\Sigma}$ to denote the set of all possible states.

Example 1

Let $\boldsymbol{\sigma_0}$ be a state such that $\begin{array}{l} \boldsymbol{\sigma_0(P)} = \mathrm{tt} \\ \boldsymbol{\sigma_0(Q)} = \mathrm{ff} \end{array}$

Interval and Length (7)

An interval $\boldsymbol{\sigma}$ is a finite sequence of states

$\boldsymbol{\sigma} : \boldsymbol{\sigma_0\sigma_1\sigma_2\ldots}$

Let $\boldsymbol{\Sigma^+}$ denote the set of all possible finite intervals with at least one state.

The length of an interval $\boldsymbol{\sigma}$ is denoted by $\boldsymbol{|\sigma|}$ and is the number of states minus 1.

Example 2

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

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 3

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

Semantics of State Formula (9)

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

$$
\begin{array}{lcl}
\mathrm{M}[\![\mathrm{true}]\!](\boldsymbol{\sigma}) & = & \mathrm{tt} \\
\mathrm{M}[\![\boldsymbol{p}]\!](\boldsymbol{\sigma}) & = & \boldsymbol{\sigma_0}(\boldsymbol{p}) \\
\mathrm{M}[\![\neg \boldsymbol{W}]\!](\boldsymbol{\sigma}) & = & \text{not } \mathrm{M}[\![\boldsymbol{W}]\!](\boldsymbol{\sigma}) \\
\mathrm{M}[\![\boldsymbol{W_1} \vee \boldsymbol{W_2}]\!](\boldsymbol{\sigma}) & = & \mathrm{M}[\![\boldsymbol{W_1}]\!](\boldsymbol{\sigma}) \text{ or } \mathrm{M}[\![\boldsymbol{W_2}]\!](\boldsymbol{\sigma})
\end{array}
$$

Semantics of Transition formulae (10)

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

$$
\begin{array}{lcl}
\mathrm{M}[\![\neg \boldsymbol{T}]\!](\boldsymbol{\sigma}) & = & \text{not } \mathrm{M}[\![\boldsymbol{T}]\!](\boldsymbol{\sigma}) \\
\mathrm{M}[\![\boldsymbol{T}_1 \vee \boldsymbol{T}_2]\!](\boldsymbol{\sigma}) & = & \mathrm{M}[\![\boldsymbol{T}_1]\!](\boldsymbol{\sigma}) \text{ or } \mathrm{M}[\![\boldsymbol{T}_2]\!](\boldsymbol{\sigma}) \\
\mathrm{M}[\![\bigcirc \boldsymbol{W}]\!](\boldsymbol{\sigma}) & = & \mathrm{M}[\![\boldsymbol{W}]\!](\boldsymbol{\sigma}_1 \ldots \boldsymbol{\sigma}_{|\boldsymbol{\sigma}|}) \text{ and } |\boldsymbol{\sigma}| > \mathbf{0}
\end{array}
$$

Semantics of Fusion expressions (11)

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

$$
\begin{array}{lll}
\mathrm{M}[\![\mathrm{test}(W)]\!](\sigma) & = & \mathrm{M}[\![W]\!](\sigma_0) \text{ and } |\sigma| = 0 \\
\mathrm{M}[\![\mathrm{step}(T)]\!](\sigma) & = & \mathrm{M}[\![T]\!](\sigma_0 \ldots \sigma_1) \text{ and } |\sigma| = 1 \\
\mathrm{M}[\![E_1 \vee E_2]\!](\sigma) & = & \mathrm{M}[\![E_1]\!](\sigma) \text{ or } \mathrm{M}[\![E_2]\!](\sigma) \\
\mathrm{M}[\![E_0 ; E_1]\!](\sigma) = \mathrm{tt} & \text{iff} & \text{exists a } k, \text{ s.t. } 0 \leq k \leq |\sigma| \text{ and} \\
 & & \mathrm{M}[\![E_0]\!](\sigma_0 \ldots \sigma_k) = \mathrm{tt} \text{ and } \mathrm{M}[\![E_1]\!](\sigma_k \ldots \sigma_{|\sigma|}) = \mathrm{tt} \\
\mathrm{M}[\![E^*]\!](\sigma) = \mathrm{tt} & \text{iff} & \text{exist } l_0, \ldots, l_n \text{ s.t. } l_0 = 0 \text{ and } l_n = |\sigma| \text{ and} \\
 & & \text{for all } 0 \leq i < n, l_i < l_{i+1} \text{ and} \\
 & & \mathrm{M}[\![E]\!](\sigma_{l_i} \ldots \sigma_{l_{i+1}}) = \mathrm{tt}
\end{array}
$$

Semantics of right Fusion logic (12)

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

$\mathrm{M}[\![\neg \boldsymbol{R}]\!](\boldsymbol{\sigma}) =$ $=$ not $\mathrm{M}[\![\boldsymbol{R}]\!](\boldsymbol{\sigma})$

$\mathrm{M}[\![\boldsymbol{R_1} \vee \boldsymbol{R_2}]\!](\boldsymbol{\sigma})$ $=$ $\mathrm{M}[\![\boldsymbol{R_1}]\!](\boldsymbol{\sigma})$ or $\mathrm{M}[\![\boldsymbol{R_2}]\!](\boldsymbol{\sigma})$

$\mathrm{M}[\![\langle \boldsymbol{E} \rangle \boldsymbol{R}]\!](\boldsymbol{\sigma}) = \mathrm{tt}$ iff exists a $\boldsymbol{k}$, s.t. $\boldsymbol{0} \leq \boldsymbol{k} \leq |\boldsymbol{\sigma}|$ and

$\mathrm{M}[\![\boldsymbol{E}]\!](\boldsymbol{\sigma_0} \ldots \boldsymbol{\sigma_k}) = \mathrm{tt}$ and

$\mathrm{M}[\![\boldsymbol{R}]\!](\boldsymbol{\sigma_k} \ldots \boldsymbol{\sigma_{|\sigma|}}) = \mathrm{tt}$

Semantics of left Fusion logic (13)

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

$$
\begin{array}{lcl}
\mathrm{M}[\![\mathrm{fin}\,(\boldsymbol{p})]\!](\boldsymbol{\sigma}) & = & \boldsymbol{\sigma}_{|\boldsymbol{\sigma}|}(\boldsymbol{p}) \\
\mathrm{M}[\![\neg \boldsymbol{L}]\!](\boldsymbol{\sigma}) & = & \text{not } \mathrm{M}[\![\boldsymbol{L}]\!](\boldsymbol{\sigma}) \\
\mathrm{M}[\![\boldsymbol{L}_1 \vee \boldsymbol{L}_2]\!](\boldsymbol{\sigma}) & = & \mathrm{M}[\![\boldsymbol{L}_1]\!](\boldsymbol{\sigma}) \text{ or } \mathrm{M}[\![\boldsymbol{L}_2]\!](\boldsymbol{\sigma}) \\
\mathrm{M}[\![\boldsymbol{L}\langle \boldsymbol{E} \rangle]\!](\boldsymbol{\sigma}) = \mathrm{tt} & \text{iff} & \text{exists a } \boldsymbol{k}, \text{ s.t. } \boldsymbol{0} \leq \boldsymbol{k} \leq |\boldsymbol{\sigma}| \text{ and} \\
 & & \mathrm{M}[\![\boldsymbol{L}]\!](\boldsymbol{\sigma}_0 \ldots \boldsymbol{\sigma}_k) = \mathrm{tt} \text{ and} \\
 & & \mathrm{M}[\![\boldsymbol{E}]\!](\boldsymbol{\sigma}_k \ldots \boldsymbol{\sigma}_{|\boldsymbol{\sigma}|}) = \mathrm{tt}
\end{array}
$$

Derived operators (14)

Derived Fusion expression operators

$$
\begin{array}{ll}
\mathsf{len}_{\boldsymbol{e}}(\mathbf{0}) & \triangleq \quad \mathsf{test}(\mathsf{true}) \\
\mathsf{len}_{\boldsymbol{e}}(\boldsymbol{n}+\mathbf{1}) & \triangleq \quad \mathsf{step}(\mathsf{true}) \,;\, \mathsf{len}_{\boldsymbol{e}}(\boldsymbol{n}) \\
\mathsf{true}_{\boldsymbol{e}} & \triangleq \quad \mathsf{step}(\mathsf{true})^* \\
\mathsf{more}_{\boldsymbol{e}} & \triangleq \quad \mathsf{step}(\mathsf{true}) \,;\, \mathsf{true}_{\boldsymbol{e}} \\
\Diamond_{\boldsymbol{e}}\, \boldsymbol{W} & \triangleq \quad \mathsf{true}_{\boldsymbol{e}} \,;\, \mathsf{test}(\boldsymbol{W}) \,;\, \mathsf{true}_{\boldsymbol{e}} \\
\Box_{\boldsymbol{e}}\, \boldsymbol{W} & \triangleq \quad \mathsf{step}(\boldsymbol{W})^* \,;\, \mathsf{test}(\boldsymbol{W}) \\
\boldsymbol{n} : \boldsymbol{W} & \triangleq \quad \mathsf{true}_{\boldsymbol{e}} \,;\, \mathsf{test}(\boldsymbol{W}) \,;\, \mathsf{len}_{\boldsymbol{e}}(\boldsymbol{n})
\end{array}
$$

Derived operators (15)

Derived right Fusion logic operators

$$
\begin{array}{lcl}
\boldsymbol{R_1 \wedge R_2} & \triangleq & \boldsymbol{\neg(\neg R_1 \vee \neg R_2)} \\
\boldsymbol{R_1 \supset R_2} & \triangleq & \boldsymbol{\neg R_1 \vee R_2} \\
\mathsf{more}_{\boldsymbol{r}} & \triangleq & \langle \mathsf{step}(\mathsf{true}) \rangle \mathsf{true} \\
\mathsf{empty}_{\boldsymbol{r}} & \triangleq & \neg \mathsf{more}_{\boldsymbol{r}} \\
\mathsf{len}_{\boldsymbol{r}}(\boldsymbol{0}) & \triangleq & \mathsf{empty}_{\boldsymbol{r}} \\
\mathsf{len}_{\boldsymbol{r}}(\boldsymbol{n+1}) & \triangleq & \langle \mathsf{step}(\mathsf{true}) \rangle \mathsf{len}_{\boldsymbol{r}}(\boldsymbol{n}) \\
\Diamond_{\boldsymbol{r}}\, \boldsymbol{R} & \triangleq & \langle \mathsf{true}_{\boldsymbol{e}} \rangle \boldsymbol{R} \\
\Box_{\boldsymbol{r}}\, \boldsymbol{R} & \triangleq & \neg \Diamond_{\boldsymbol{r}}(\neg \boldsymbol{R}) \\
\Box_{\boldsymbol{r}}\, \boldsymbol{T} & \triangleq & \Box_{\boldsymbol{r}} \langle \mathsf{step}(\boldsymbol{T}) \rangle \mathsf{true} \\
\diamondsuit_{\boldsymbol{r}}\, \boldsymbol{W} & \triangleq & \Box_{\boldsymbol{r}} \langle \mathsf{true}_{\boldsymbol{e}} \rangle \langle \mathsf{test}(\boldsymbol{W}) \rangle \mathsf{true} \\
\boxdot_{\boldsymbol{r}}\, \boldsymbol{W} & \triangleq & \neg \diamondsuit_{\boldsymbol{r}}(\neg \boldsymbol{W}) \\
\boldsymbol{n} :_{\boldsymbol{r}} \boldsymbol{W} & \triangleq & \langle \mathsf{true}_{\boldsymbol{e}} \rangle \langle \mathsf{test}(\boldsymbol{W}) \rangle \mathsf{len}_{\boldsymbol{r}}(\boldsymbol{n}) \\
\boldsymbol{[E]R} & \triangleq & \boldsymbol{\neg(\langle E \rangle \neg R)}
\end{array}
$$

Derived operators (16)

Derived left Fusion logic operators

$$
\begin{array}{lcl}
L_1 \wedge L_2 & \triangleq & \neg(\neg L_1 \vee \neg L_2) \\
L_1 \supset L_2 & \triangleq & \neg L_1 \vee L_2 \\
\mathrm{more}_l & \triangleq & \mathrm{true}\langle \mathrm{step}(\mathrm{true})\rangle \\
\mathrm{empty}_l & \triangleq & \neg \mathrm{more}_l \\
\mathrm{len}_l(0) & \triangleq & \mathrm{empty}_l \\
\mathrm{len}_l(n+1) & \triangleq & \mathrm{len}_l(n)\langle \mathrm{step}(\mathrm{true})\rangle \\
\Diamond_l W & \triangleq & \mathrm{true}\langle \mathrm{test}(W)\rangle\langle \mathrm{true}_e\rangle \\
\Box_l W & \triangleq & \neg \Diamond_l(\neg W) \\
\diamondsuit_l L & \triangleq & L\langle \mathrm{true}_e\rangle \\
\boxdot_l L & \triangleq & \neg \diamondsuit_l(\neg L) \\
n :_l W & \triangleq & \mathrm{true}\langle \mathrm{test}(W)\rangle\langle \mathrm{len}_e(n)\rangle \\
L[E] & \triangleq & \neg(\neg L\langle E\rangle)
\end{array}
$$

Satisfiable and valid (17)

- A fusion logic formula $\boldsymbol{F}$ is satisfiable if and only if there exists an interval $\boldsymbol{\sigma}$ such that $\mathrm{M}[\![\boldsymbol{F}]\!](\boldsymbol{\sigma}) = \mathrm{tt}$
- Decision procedure checks whether $\boldsymbol{F}$ is satisfiable or not, when $\boldsymbol{F}$ is satisfiable a satisfying interval is generated.
- A fusion logic formula $\boldsymbol{F}$ is valid if and only if for all intervals $\boldsymbol{\sigma}$, $\mathrm{M}[\![\boldsymbol{F}]\!](\boldsymbol{\sigma}) = \mathrm{tt}$
- $\boldsymbol{F}$ is not valid if and only if $\neg\boldsymbol{F}$ is satisfiable
  i.e., satisfying interval for $\neg\boldsymbol{F}$ will represent
  a counter example for $\boldsymbol{F}$'s validity
  $\boldsymbol{F}$ is valid if and only if $\neg\boldsymbol{F}$ is not satisfiable

Decision Procedure (18)

- Time Reversal Step:
  transform a left fusion logic formula into a right fusion logic formula
- Reduction Step:
  transform right fusion logic formula $\boldsymbol{R}$ *into* $\boldsymbol{init} \wedge \Box_r \boldsymbol{I}$
- BDD Step:
  transform $\boldsymbol{init} \wedge \Box_r \boldsymbol{I}$ *into a BDD-based satisfiability problem*

Time Reversal Step (19)

Transform a left fusion formula into a right fusion logic formula.

- Let $\boldsymbol{F^r}$ denotes the *time reversed* version of fusion logic formula $\boldsymbol{F}$.
- Let $\boldsymbol{reverse(\sigma)}$ denote the time reversed interval of $\boldsymbol{\sigma}$: $\boldsymbol{reverse(\sigma_0 \ldots \sigma_{|\sigma|})} \triangleq \boldsymbol{\sigma_{|\sigma|} \ldots \sigma_0}$
- $\mathrm{M}[\![\boldsymbol{F^r}]\!](\boldsymbol{\sigma}) = \mathrm{tt}$ iff $\mathrm{M}[\![\boldsymbol{F}]\!](\boldsymbol{reverse(\sigma)}) = \mathrm{tt}$

Time Reversal Step (20)

Rules to rewrite left fusion logic formulae into right fusion logic formulae

left fusion formulae

$((L\langle E\rangle)^r$	$=$	$\langle E^r\rangle L^r$
$(\mathsf{fin}\,(p))^r$	$=$	p
true^r	$=$	true
$(\neg L)^r$	$=$	$\neg(L^r)$
$(L_1 \vee L_2)^r$	$=$	$L_1{}^r \vee L_2{}^r$

fusion expressions

$(\mathsf{test}(W))^r$	$=$	$\mathsf{test}(W)$
$(\mathsf{step}(T))^r$	$=$	$\mathsf{step}(T^r)$
$(E_1 \vee E_2)^r$	$=$	$E_1{}^r \vee E_2{}^r$
$(E_1 \,;\, E_2)^r$	$=$	$E_2{}^r \,;\, E_1{}^r$
$(E^*)^r$	$=$	$(E^r)^*$

transitions

$(\bigcirc W)^r$	$=$	W
W^r	$=$	$\bigcirc W$
$(T_1 \vee T_2)^r$	$=$	$T_1{}^r \vee T_2{}^r$
$(\neg T)^r$	$=$	$\neg(T^r)$

Time Reversal Step (21)

The following holds

- Let $\boldsymbol{L}$ be a left fusion logic formula then $\boldsymbol{L^r}$ is a right fusion logic formula.
- Let $\boldsymbol{L}$ be a left fusion logic formula then $\mathrm{M}[\![\boldsymbol{L}]\!](\boldsymbol{\sigma}) = \mathrm{tt}$ iff $\mathrm{M}[\![\boldsymbol{L^r}]\!](\boldsymbol{reverse}(\boldsymbol{\sigma})) = \mathrm{tt}$

Reduction Step (22)

Transform right fusion logic formula $\boldsymbol{R}$ into an equivalent reduced form $\boldsymbol{init} \wedge \Box_r \boldsymbol{I}$ where

- $\boldsymbol{init}$: a state formula
  $\boldsymbol{init} \triangleq \mathcal{R}'_0(f)$
- $\boldsymbol{I}$: an invariant, $\bigwedge_{i=1}^{i=k}(r_{X_i} \equiv t_i)$ (for $k \geq 1$) where
  r_{X_i} is a dependent Boolean variable (not appearing in $\boldsymbol{F}$)
  t_i is a transition formula
  $\boldsymbol{I} \triangleq \mathcal{R}_0(f)$

Reduction Step (23)

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:

For $k \in \{0, 1\}$

R	$\mathcal{R}_k(R)$
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 $X \vee \langle c(E) \rangle r_{\langle E^* \rangle X}$
$\neg X$	$\mathcal{R}_k(X)$
$X_1 \vee X_2$	$\mathcal{R}_k(X_1) \wedge \mathcal{R}_0(X_2)$

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

Reduction Step (24)

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:

For $k \in \{0, 1\}$

R	$\mathcal{R}'_k(R)$
W	W
$\langle \text{test}(W) \rangle X$	$W \wedge \mathcal{R}'_k(X)$
$\langle \text{step}(T) \rangle X$	$k = 0 : r_{\langle \text{step}(T) \rangle X}$
	$k = 1 : T \wedge \bigcirc \mathcal{R}'_0(X)$
$\langle E_1 \vee E_2 \rangle X$	$\mathcal{R}'_k(\langle E_1 \rangle X \vee \langle E_2 \rangle X)$
$\langle E_1 ; E_2 \rangle X$	$\mathcal{R}'_k(\langle E_1 \rangle \langle E_2 \rangle X)$
$\langle E^* \rangle X$	$r_{\langle E^* \rangle X}$
$\neg X$	$\neg \mathcal{R}'_k(X)$
$X_1 \vee X_2$	$\mathcal{R}'_k(X_1) \vee \mathcal{R}'_k(X_2)$

Reduction Step (25)

Reduction function for $\langle E^* \rangle X$ is a bit more involved 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 $c(e) \equiv E \wedge \text{more}_e$ holds.

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 Step (25)

Reduction function for $\langle E^* \rangle X$ is a bit more involved 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
$c(e) \equiv E \wedge \text{more}_e$ holds.
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 Step (26)

The following holds

- Let $\boldsymbol{R}$ be a right fusion Logic formula and $\boldsymbol{dep(R)}$ be the dependent variables introduced by $\boldsymbol{\mathcal{R}'_k(R)}$ and $\boldsymbol{\mathcal{R}_k(R)}$ $(\boldsymbol{k = 0, 1})$ then
  $\boldsymbol{R \equiv \exists dep(R) \bullet (\mathcal{R}'_k(R) \wedge \Box\, \mathcal{R}_k(R))}$

Example

Given a right fusion logic formula
$\langle \text{step}(\boldsymbol{A})^* \rangle (\boldsymbol{B} \vee \boldsymbol{C}) \vee \langle \text{step}(\boldsymbol{A}) \,;\, \text{test}(\boldsymbol{B}) \rangle \boldsymbol{D}$
The Reduction Step yields:

$\boldsymbol{init} = \boldsymbol{r_{X_1}} \vee \boldsymbol{r_{X_3}}$ where
$\boldsymbol{X_1} \triangleq \langle \text{step}(\boldsymbol{A})^* \rangle (\boldsymbol{B} \vee \boldsymbol{C})$ and $\boldsymbol{X_3} \triangleq \langle \text{step}(\boldsymbol{A}) \rangle \langle \text{test}(\boldsymbol{B}) \rangle \boldsymbol{D}$.

The corresponding invariant $\boldsymbol{I}$ is
$\boldsymbol{I} = (\boldsymbol{r_{X_1}} \equiv (\boldsymbol{B} \vee \boldsymbol{C}) \vee (\boldsymbol{A} \wedge \bigcirc \boldsymbol{r_{X_1}})) \wedge (\boldsymbol{r_{X_3}} \equiv \boldsymbol{A} \wedge \bigcirc (\boldsymbol{B} \wedge \boldsymbol{D}))$

BDD Step (28)

Transform $\mathit{init} \wedge \Box_r I$ into BDD based satisfiability problem.

init: a state formula

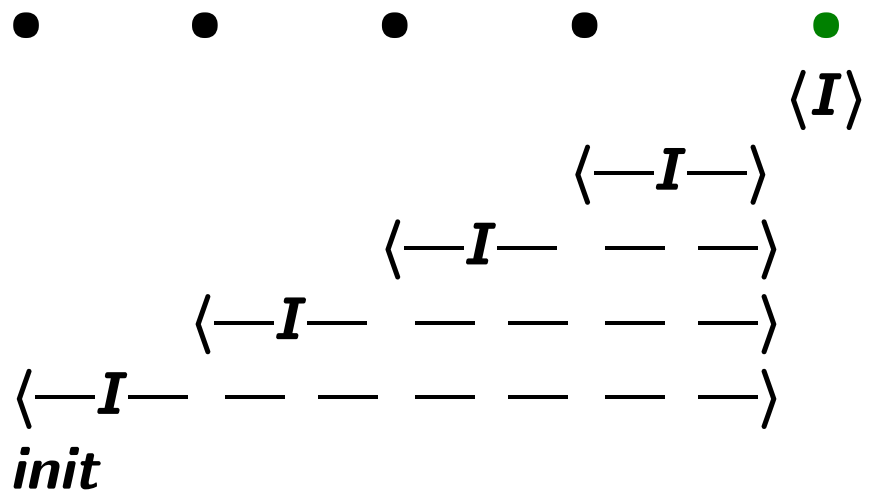
I: an invariant, $\bigwedge_{i=1}^{i=k}(r_{X_i} \equiv t_i)$ (for $k \geq 1$) where

r_{X_i} is a dependent variable and

t_i is a transition formula

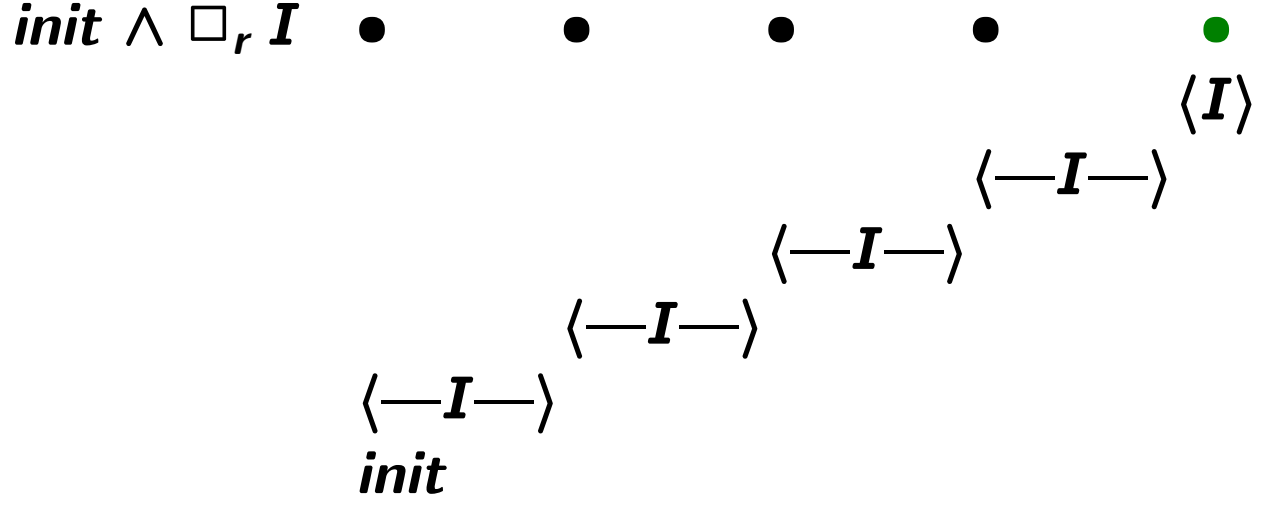
Let us examine the 'Always' operator

$\mathit{init} \wedge \Box_r I$



BDD Step (29)

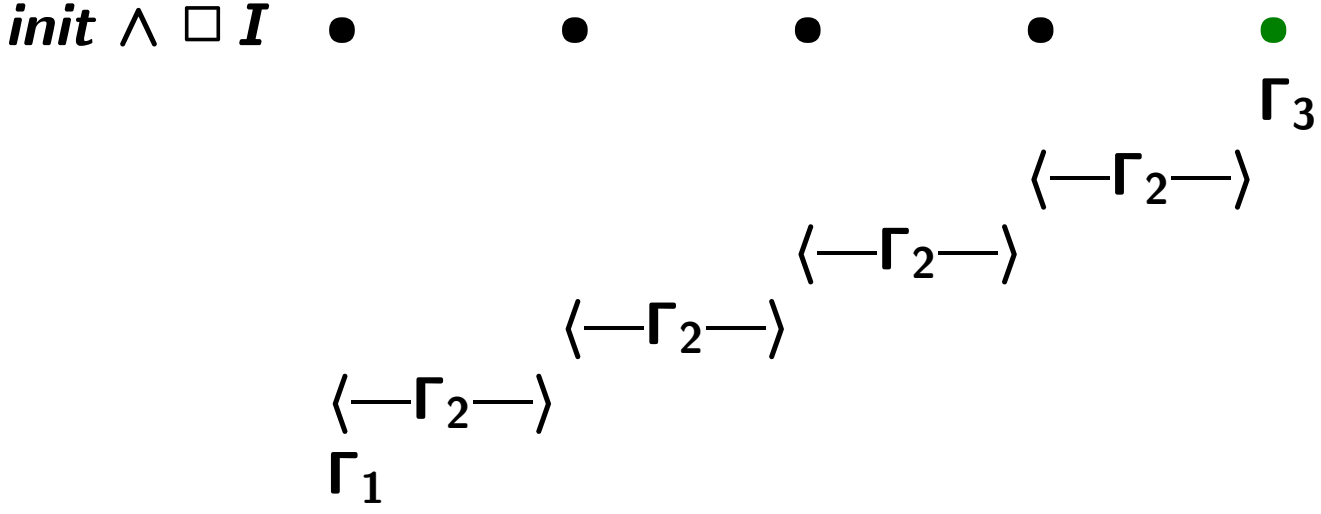
We know that invariant $\boldsymbol{I}$ only contains $\bigcirc$ (next) as temporal operator. So it can only constrain the first two states of each suffix interval.



BDD Step (30)

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 $\boldsymbol{I}$. 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 $\boldsymbol{I}$ in an interval with only 1 state. This can be done by replacing each $\bigcirc$ construct by false, i.e., $\neg$true.



Example

Given right fusion logic 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 $\boldsymbol{Init} = \boldsymbol{r_{X_1}} \vee \boldsymbol{r_{X_3}}$ and corresponding invariant:

$\boldsymbol{I} = (\boldsymbol{r_{X_1}} \equiv (\boldsymbol{B} \vee \boldsymbol{C}) \vee (\boldsymbol{A} \wedge \bigcirc \boldsymbol{r_{X_1}})) \wedge (\boldsymbol{r_{X_3}} \equiv \boldsymbol{A} \wedge \bigcirc(\boldsymbol{B} \wedge \boldsymbol{D}))$

Then $\boldsymbol{\Gamma_1} = \boldsymbol{r_{X_1}} \vee \boldsymbol{r_{X_3}}$ and

$\boldsymbol{\Gamma_2} = (\boldsymbol{r_{X_1}} \equiv (\boldsymbol{B} \vee \boldsymbol{C}) \vee (\boldsymbol{A} \wedge \boldsymbol{r'_{X_1}})) \wedge (\boldsymbol{r_{X_3}} \equiv \boldsymbol{A} \wedge (\boldsymbol{B'} \wedge \boldsymbol{D'}))$

and

$\boldsymbol{\Gamma_3} = (\boldsymbol{r_{X_1}} \equiv (\boldsymbol{B} \vee \boldsymbol{C}) \vee (\boldsymbol{A} \wedge \text{false})) \wedge (\boldsymbol{r_{X_3}} \equiv \boldsymbol{A} \wedge \text{false})$

BDD Step (32)

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 n, 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 (32)

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 n, 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 (32)

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 n, 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 (33)

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 Pr_{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 Pr_{m-1}.
  Continue until we reach Pr_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 (33)

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 Pr_{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 Pr_{m-1}.
  Continue until we reach Pr_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 (33)

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 Pr_{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 Pr_{m-1}.
  Continue until we reach Pr_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 (33)

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.

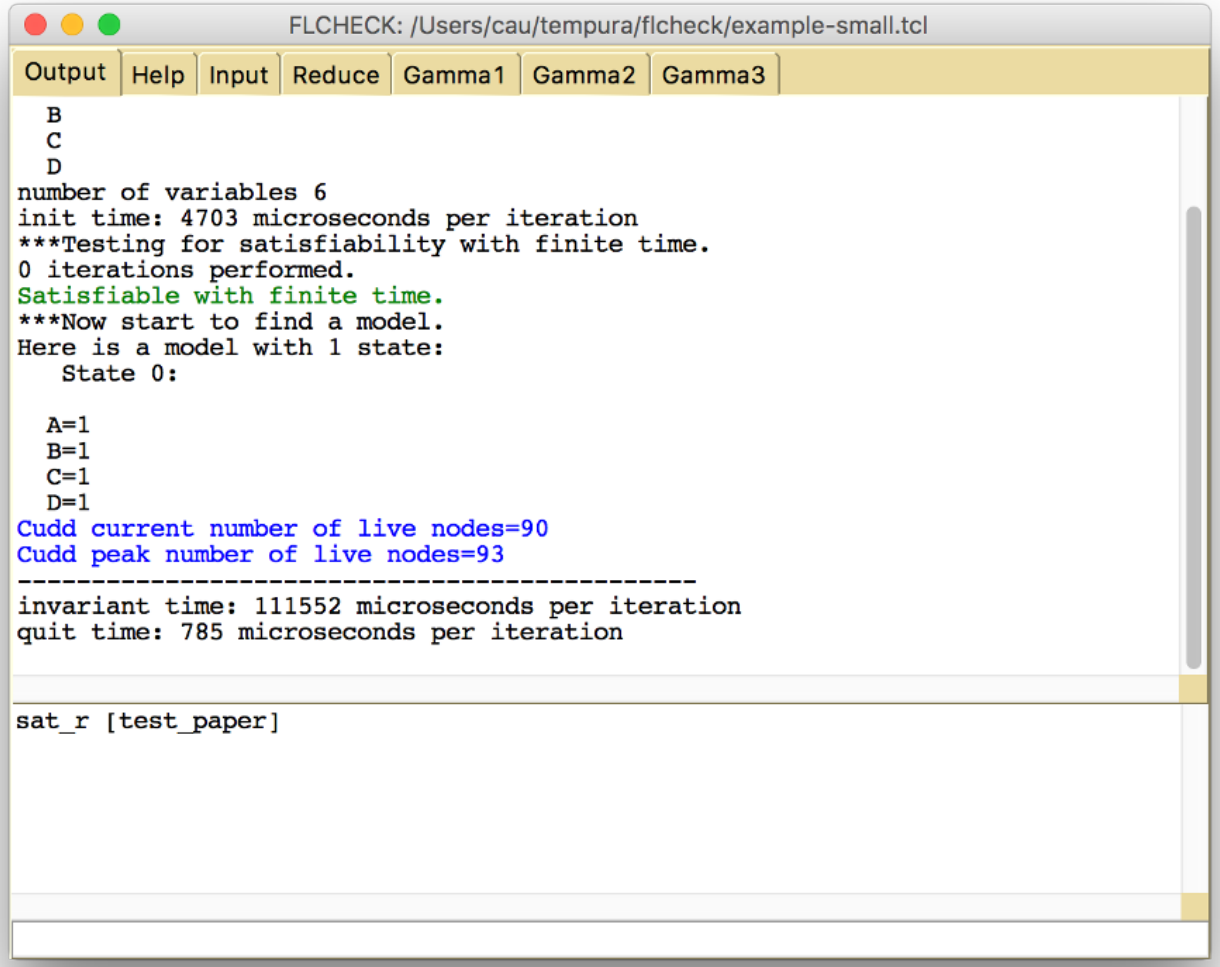
- If there are no independent variables (only r_i variables) then any interval of length m will satisfy F.
- If there are independent variables then
  Find a value assignment σ_m for the independent variables for BDD Δ_m, i.e., choose one state σ_m of Δ_m.
  Compute Pr_{m-1} denoting those states of 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 Pr_{m-1}.
  Continue until we reach Pr_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 (33)

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 Pr_{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 Pr_{m-1}.
  Continue until we reach Pr_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.

Implementation



Implementation of decision procedure

Future work (35)

- Compare with JavaLite and PITL2MONA
- Combine Decision Procedure with theorem prover Prover9
- Compare with tools for mu-calculus