

A Decision Procedure for Fusion Logic

Antonio Cau, Helge Janicke and Ben Moszkowski

([Html version of slides](#))

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
 - Time Reversal Step
 - Reduction Step
 - BDD Step
- 5 Future work

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.

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:

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

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:

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

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

$$\mathbf{State} : \mathbf{Var}^b \rightarrow \mathbf{Bool}$$

We will use $\sigma_0, \sigma_1, \sigma_2, \dots$ to denote states and Σ to denote the set of all possible states.

Example

Let σ_0 be a state such that

$$\sigma_0(\mathbf{P}) = \text{tt}$$

$$\sigma_0(\mathbf{Q}) = \text{ff}$$

Interval and Length

(7)

An **interval** σ is a finite sequence of states

$$\sigma : \sigma_0 \sigma_1 \sigma_2 \dots$$

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

$$\sigma = \sigma_0 \quad |\sigma| = 0$$

$$\sigma = \sigma_0 \sigma_1 \quad |\sigma| = 1$$

$$\sigma = \sigma_0 \sigma_1 \dots \sigma_n \quad |\sigma| = n$$

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

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

Let $\llbracket \cdot \rrbracket$ be the “meaning” function from ‘state formulae’ $\times \Sigma^+$ to $\{\text{tt}, \text{ff}\}$ and let σ be a finite interval ($\sigma \in \Sigma^+$) then

$$\begin{aligned}\llbracket \text{true} \rrbracket_{\sigma} &= \text{tt} \\ \llbracket \mathbf{p} \rrbracket_{\sigma} &= \sigma_0(\mathbf{p}) \\ \llbracket \neg \mathbf{W} \rrbracket_{\sigma} &= \text{not } \llbracket \mathbf{W} \rrbracket_{\sigma} \\ \llbracket \mathbf{W}_1 \vee \mathbf{W}_2 \rrbracket_{\sigma} &= \llbracket \mathbf{W}_1 \rrbracket_{\sigma} \text{ or } \llbracket \mathbf{W}_2 \rrbracket_{\sigma}\end{aligned}$$

Let $\llbracket \cdot \rrbracket$ 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

$$\begin{aligned}\llbracket \neg T \rrbracket_{\sigma} &= \text{not } \llbracket T \rrbracket_{\sigma} \\ \llbracket T_1 \vee T_2 \rrbracket_{\sigma} &= \llbracket T_1 \rrbracket_{\sigma} \text{ or } \llbracket T_2 \rrbracket_{\sigma} \\ \llbracket \bigcirc W \rrbracket_{\sigma} &= \llbracket W \rrbracket_{\sigma_1 \dots \sigma_{|\sigma|}} \text{ and } |\sigma| > \mathbf{0}\end{aligned}$$

Let $\llbracket \cdot \rrbracket$ be the “meaning” function from ‘fusion expressions’ $\times \Sigma^+$ to $\{\text{tt}, \text{ff}\}$ and let σ be a finite interval ($\sigma \in \Sigma^+$) then

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

Let $\llbracket \cdot \rrbracket$ be the “meaning” function from ‘right fusion logic formulae’ $\times \Sigma^+$ to $\{\text{tt}, \text{ff}\}$ and let σ be a finite interval ($\sigma \in \Sigma^+$) then

$$\begin{aligned} \llbracket \neg R \rrbracket_{\sigma} &= \text{not } \llbracket R \rrbracket_{\sigma} \\ \llbracket R_1 \vee R_2 \rrbracket_{\sigma} &= \llbracket R_1 \rrbracket_{\sigma} \text{ or } \llbracket R_2 \rrbracket_{\sigma} \\ \llbracket \langle E \rangle R \rrbracket_{\sigma} = \text{tt} &\text{ iff exists a } k, \text{ s.t. } \mathbf{0} \leq k \leq |\sigma| \text{ and} \\ &\llbracket E \rrbracket_{\sigma_0 \dots \sigma_k} = \text{tt} \text{ and } \llbracket R \rrbracket_{\sigma_k \dots \sigma_{|\sigma|}} = \text{tt} \end{aligned}$$

Let $\llbracket \cdot \rrbracket$ be the “meaning” function from ‘left fusion logic formulae’ $\times \Sigma^+$ to $\{\text{tt}, \text{ff}\}$ and let σ be a finite interval ($\sigma \in \Sigma^+$) then

$$\begin{aligned} \llbracket \text{fin } (p) \rrbracket_{\sigma} &= \sigma_{|\sigma|}(p) \\ \llbracket \neg L \rrbracket_{\sigma} &= \text{not } \llbracket L \rrbracket_{\sigma} \\ \llbracket L_1 \vee L_2 \rrbracket_{\sigma} &= \llbracket L_1 \rrbracket_{\sigma} \text{ or } \llbracket L_2 \rrbracket_{\sigma} \\ \llbracket L \langle E \rangle \rrbracket_{\sigma} = \text{tt} &\text{ iff exists a } k, \text{ s.t. } \mathbf{0} \leq k \leq |\sigma| \text{ and} \\ &\llbracket L \rrbracket_{\sigma_0 \dots \sigma_k} = \text{tt} \text{ and } \llbracket E \rrbracket_{\sigma_k \dots \sigma_{|\sigma|}} = \text{tt} \end{aligned}$$

Derived Fusion expression operators

$\text{len}_e(\mathbf{0})$	$\hat{=}$	$\text{test}(\text{true})$
$\text{len}_e(\mathbf{n} + \mathbf{1})$	$\hat{=}$	$\text{step}(\text{true}) ; \text{len}_e(\mathbf{n})$
true_e	$\hat{=}$	$\text{step}(\text{true})^*$
more_e	$\hat{=}$	$\text{step}(\text{true}) ; \text{true}_e$
$\diamond_e \mathbf{W}$	$\hat{=}$	$\text{true}_e ; \text{test}(\mathbf{W}) ; \text{true}_e$
$\square_e \mathbf{W}$	$\hat{=}$	$\text{step}(\mathbf{W})^* ; \text{test}(\mathbf{W})$
$\mathbf{n} : \mathbf{W}$	$\hat{=}$	$\text{true}_e ; \text{test}(\mathbf{W}) ; \text{len}_e(\mathbf{n})$

Derived right Fusion logic operators

$R_1 \wedge R_2$	$\hat{=}$	$\neg(\neg R_1 \vee \neg R_2)$
$R_1 \supset R_2$	$\hat{=}$	$\neg R_1 \vee R_2$
more_r	$\hat{=}$	$\langle \text{step}(\text{true}) \rangle \text{true}$
empty_r	$\hat{=}$	$\neg \text{more}_r$
$\text{len}_r(\mathbf{0})$	$\hat{=}$	empty_r
$\text{len}_r(n + \mathbf{1})$	$\hat{=}$	$\langle \text{step}(\text{true}) \rangle \text{len}_r(n)$
$\diamond_r R$	$\hat{=}$	$\langle \text{true}_e \rangle R$
$\square_r R$	$\hat{=}$	$\neg \diamond_r(\neg R)$
$\square_r T$	$\hat{=}$	$\square_r \langle \text{step}(T) \rangle \text{true}$
$\diamond_r W$	$\hat{=}$	$\square_r \langle \text{true}_e \rangle \langle \text{test}(W) \rangle \text{true}$
$\square_r W$	$\hat{=}$	$\neg \diamond_r(\neg W)$
$n :_r W$	$\hat{=}$	$\langle \text{true}_e \rangle \langle \text{test}(W) \rangle \text{len}_r(n)$
$[E] R$	$\hat{=}$	$\neg(\langle E \rangle \neg R)$

Derived left Fusion logic operators

$L_1 \wedge L_2$	$\hat{=}$	$\neg(\neg L_1 \vee \neg L_2)$
$L_1 \supset L_2$	$\hat{=}$	$\neg L_1 \vee L_2$
more_I	$\hat{=}$	$\text{true} \langle \text{step}(\text{true}) \rangle$
empty_I	$\hat{=}$	$\neg \text{more}_I$
$\text{len}_I(\mathbf{0})$	$\hat{=}$	empty_I
$\text{len}_I(\mathbf{n} + \mathbf{1})$	$\hat{=}$	$\text{len}_I(\mathbf{n}) \langle \text{step}(\text{true}) \rangle$
$\diamond_I W$	$\hat{=}$	$\text{true} \langle \text{test}(W) \rangle \langle \text{true}_e \rangle$
$\square_I W$	$\hat{=}$	$\neg \diamond_I(\neg W)$
$\diamond_I L$	$\hat{=}$	$L \langle \text{true}_e \rangle$
$\square_I L$	$\hat{=}$	$\neg \diamond_I(\neg L)$
$n :_I W$	$\hat{=}$	$\text{true} \langle \text{test}(W) \rangle \langle \text{len}_e(n) \rangle$
$L [E]$	$\hat{=}$	$\neg(\neg L \langle E \rangle)$

- A fusion logic formula F is **satisfiable** if and only if there exists an interval σ such that $\llbracket F \rrbracket_{\sigma} = \text{tt}$
- 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 , $\llbracket F \rrbracket_{\sigma} = \text{tt}$
- 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

- Time Reversal Step:

transform a left fusion logic formula into a right fusion logic formula

- Reduction Step:

transform right fusion logic formula R into $init \wedge \square_r I$

- BDD Step:

transform $init \wedge \square_r I$ into a BDD-based satisfiability problem

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

- Let F' denotes the *time reversed* version of fusion logic formula F .
- Let $reverse(\sigma)$ denote the time reversed interval of σ :

$$reverse(\sigma_0 \dots \sigma_{|\sigma|}) \hat{=} \sigma_{|\sigma|} \dots \sigma_0$$

- $\llbracket F' \rrbracket_{\sigma} = \text{tt}$ iff $\llbracket F \rrbracket_{reverse(\sigma)} = \text{tt}$

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

left fusion formulae

$$\overline{((L \langle E \rangle))^r} = \langle E^r \rangle L^r$$

$$\overline{(\text{fin } (p))^r} = p$$

$$\overline{\text{true}^r} = \text{true}$$

$$\overline{(\neg L)^r} = \neg(L^r)$$

$$\overline{(L_1 \vee L_2)^r} = L_1^r \vee L_2^r$$

fusion expressions

$$\overline{(\text{test}(W))^r} = \text{test}(W)$$

$$\overline{(\text{step}(T))^r} = \text{step}(T^r)$$

$$\overline{(E_1 \vee E_2)^r} = E_1^r \vee E_2^r$$

$$\overline{(E_1 ; E_2)^r} = E_2^r ; E_1^r$$

$$\overline{(E^*)^r} = (E^r)^*$$

transitions

$$\overline{(\circ W)^r} = W$$

$$\overline{W^r} = \circ W$$

$$\overline{(T_1 \vee T_2)^r} = T_1^r \vee T_2^r$$

$$\overline{(\neg T)^r} = \neg(T^r)$$

The following holds

- Let L be a left fusion logic formula then L^r is a right fusion logic formula.
- Let L be a left fusion logic formula then $\llbracket L \rrbracket_\sigma = \text{tt}$ iff $\llbracket L^r \rrbracket_{\text{reverse}(\sigma)} = \text{tt}$

Transform right fusion logic formula R into an equivalent reduced form $init \wedge \square_r I$ where

- $init$: a state formula

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

- 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 F)
 t_i is a transition formula

$$I \hat{=} \mathcal{R}_0(f)$$

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 $\mathcal{R}_k(\mathbf{f})$ 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 $\text{step}(t)$ and e^* case will introduce a dependent variable.

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 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 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^*	$c(E) ; E^*$

The following holds

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

$$R \equiv \exists dep(R) \cdot (\mathcal{R}'_k(R) \wedge \square \mathcal{R}_k(R))$$

Example

(27)

Given a right fusion logic formula

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

The Reduction Step yields:

init = $r_{X_1} \vee r_{X_3}$ where

$X_1 \hat{=} \langle \text{step}(\mathbf{A})^* \rangle (\mathbf{B} \vee \mathbf{C})$ and

$X_3 \hat{=} \langle \text{step}(\mathbf{A}) \rangle \langle \text{test}(\mathbf{B}) \rangle \mathbf{D}$.

The corresponding invariant *I* is

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

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

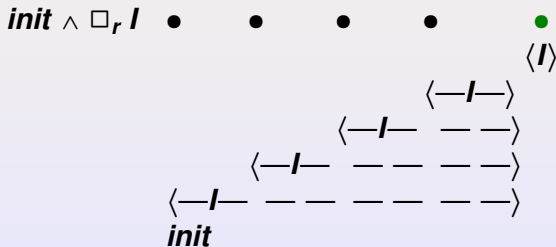
$init$: a state formula

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

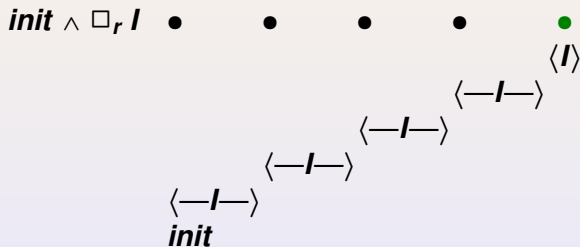
rx_i is a dependent variable and

t_i is a transition formula

Let us examine the 'Always' operator



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



Given right fusion logic formula

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

With $\mathbf{Init} = r_{X_1} \vee r_{X_3}$ and corresponding invariant:

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

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

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

and

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

Encoding as a BDD-based satisfiability problem:

- We use Γ_2 and Γ_1 to **iteratively** calculate a sequence of BDDs $\Delta_0, \dots, \Delta_n$, 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 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**.

Encoding as a BDD-based satisfiability problem:

- We use Γ_2 and Γ_1 to **iteratively** calculate a sequence of BDDs $\Delta_0, \dots, \Delta_n$, 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 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**.

Encoding as a BDD-based satisfiability problem:

- We use Γ_2 and Γ_1 to **iteratively** calculate a sequence of BDDs $\Delta_0, \dots, \Delta_n$, 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 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**.

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 $\sigma_0 \dots \sigma_{m-1} \sigma_m$ will then represent a (minimal) **satisfying interval σ** for F .

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 $\sigma_0 \dots \sigma_{m-1} \sigma_m$ will then represent a (minimal) **satisfying interval σ** for F .

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 $\sigma_0 \dots \sigma_{m-1} \sigma_m$ will then represent a (minimal) **satisfying interval σ** for F .

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.

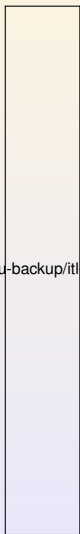
- If there are **no independent** variables (only r_i variables) then **any interval of length m** will satisfy F .
- If there are independent variables then
Find a value assignment σ_m for the **independent** variables for BDD Δ_m , i.e., choose **one** state σ_m of Δ_m .
Compute Pr_{m-1} denoting those 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 $\sigma_0 \dots \sigma_{m-1} \sigma_m$ will then represent a (minimal) **satisfying interval σ** for F .

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 $\sigma_0 \dots \sigma_{m-1} \sigma_m$ will then represent a (**minimal**) **satisfying interval** σ for F .

serenity/acau-backup/itl-talks/figures/flcheck.png



Implementation of decision procedure

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