# On Model Checking Boolean **BI**

Heng Guo Hanpin Wang Zhongyuan Xu Yongzhi Cao

School of Electronic Engineering and Computer Science
Peking university

CSL'09
Coimbra, 07 Sep 2009

# Outline

# Outline

# The logic of Bunched Implication

- A substructural logic with natural resource interpretation, introduced by O'Hearn and Pym '99.
- Additive connectives ($\top$, $\bot$, $\wedge$, $\vee$, $\rightarrow$) along with multiplicative connectives ($\top^*$, $*$, $-\!\!*$).
- Various semantic models: cartesian doubly closed category, preordered commutative monoid, etc.
- The additives are generally interpreted in the *intuitionistic* way.

# Boolean BI

- *Classical* additives: Boolean BI (BBI).
- A typical model: partially defined commutative monoid.
- Most famous application of BBI: Separation Logic.

# The semantics

- Commutative monoid. $\varepsilon$ and $\circ$.
- Additive connectives ($\top$, $\neg$, $\wedge$) are interpreted classically.
- Multiplicative connectives:

$$
\begin{array}{rcl}
m \models \top^* & \Leftrightarrow & m = \varepsilon \\
m \models \varphi_1 * \varphi_2 & \Leftrightarrow & \exists m_1, m_2.\ m = m_1 \circ m_2 \ \text{ s.t.} \\
 & & m_1 \models \varphi_1 \text{ and } m_2 \models \varphi_2 \\
m \models \varphi_1 \mathrel{-\!\!*} \varphi_2 & \Leftrightarrow & \forall m_1.\ m_1 \models \varphi_1. \\
 & & \text{implies } m \circ m_1 \models \varphi_2
\end{array}
$$

# Some Notations

- $\varphi_1 \mathrel{-\!\!*}^{\exists} \varphi_2 = \neg(\varphi_1 \mathrel{-\!\!*} \neg\varphi_2)$. Then $m \models \varphi_1 \mathrel{-\!\!*}^{\exists} \varphi_2$ *iff* $\exists m_1.\ m_1 \models \varphi_1$ and $m_1 \circ m \models \varphi_2$.

- We use $\rho(\varphi)$ to denote the set on which $\varphi$ holds.

- $$\rho(\varphi_1 * \varphi_2) = \rho(\varphi_1) \circ \rho(\varphi_2)$$
  $$\rho(\varphi_1 \mathrel{-\!\!*}^{\exists} \varphi_2) = \rho(\varphi_2) : \rho(\varphi_1)$$

# The model checking problem

- To decide whether $m \models \varphi$ in a given model.
- Some related problems have been resolved:
    - The validity and model checking problems of separation Logic are answered by Calcagno, Yang, O'hearn '01.
    - The validity of BI is decidable using Resource Tableaux. (Galmiche, Méry, Pym '02)

# Our Results

- Generally, the model checking problem is undecidable, even in finitely generated free monoid, somehow the simplest model.

# Our Results

- Generally, the model checking problem is undecidable, even in finitely generated free monoid, somehow the simplest model.
- Generator propositions, analogue of "$\mathtt{x} \mapsto -, -$"in Separation logic.

# Our Results

- Generally, the model checking problem is undecidable, even in finitely generated free monoid, somehow the simplest model.
- Generator propositions, analogue of “$\mathtt{x} \mapsto -, -$”in Separation logic.
- In this setting, we show that for *infinitely related* monoid, the model checking problem is undecidable, and for *finitely related* monoid, decidable.

# Outline

# Semigroup Presentation

- To describe monoids.
- A monoid $M$ is characterized by its generator set $X$, and generation relation $R$. $(X; R)$ is called a presentation of $M$.
- $R = \emptyset$ : Free monoid $X^*$.

# Semigroup Presentation (cont.)

- *Finitely generated* (f.g.) monoid and *finitely related* (f.r.) monoid.
- In the following, we only consider commutative monoid.
- For a *f.g.* monoid $M = (X; R)$, every element $m$ in $M$ is a congruence class in $X^*$, denoted as $[m]$.
- A *f.g.* free monoid $X^*$ is isomorphic to $\mathbb{N}^k$.

# Semigroup Presentation (cont.)

- *Finitely generated* (f.g.) monoid and *finitely related* (f.r.) monoid.
- In the following, we only consider commutative monoid.
- For a *f.g.* monoid $M = (X; R)$, every element $m$ in $M$ is a congruence class in $X^*$, denoted as $[m]$.
- A *f.g.* free monoid $X^*$ is isomorphic to $\mathbb{N}^k$.

**Theorem (Redei's theorem)**

*Every finitely generated commutative monoid is finitely related.*

# Partially defined monoid

- Partial monoid captures some essential property. Like in separation logic, not every two heaps are composable.

# Partially defined monoid

- Partial monoid captures some essential property. Like in separation logic, not every two heaps are composable.
- Simulate partial monoid by total monoid:
  - $m_1 \circ m_2 \ = \ \pi$ if $m_1 \circ m_2$ is undefined.
  - $\pi \circ m \ = \ \pi$
- For simplicity, we only consider total monoid.

# Outline

Introduction

Undecidability Results

- Propositions
- Infinitely related Monoid

Decidability Results

Additional Remarks

# The Hilbert 10th Problem

## Negative Solution of H10 (Matiyasevich '70)

Given a polynomial of several variables $P(x_1 \ldots x_k)$ with integer coefficients, it is undecidable whether there is a vector $(x_1 \ldots x_k) \in \mathbb{N}^k$ that $P(x_1 \ldots x_k) = 0$.

# Undecidability

- Recursively defined propositions lead to undecidability.

# Undecidability

- Recursively defined propositions lead to undecidability.
- In $\mathbb{N}^k$, for any given polynomial $P(x_1 \ldots x_m)$, define

$$\rho(p) \;=\; \{\, (e_1, \ldots, e_m) \mid P(e_1 \ldots e_m) = 0 \,\}$$

# Undecidability

- Recursively defined propositions lead to undecidability.
- In $\mathbb{N}^k$, for any given polynomial $P(x_1 \ldots x_m)$, define

$$\rho(p) \;=\; \{\, (e_1, \ldots, e_m) \mid P(e_1 \ldots e_m) = 0 \,\}$$

Check $\varepsilon \models \top \twoheadrightarrow\!\!*^{\exists} p \Leftrightarrow$ decide whether the equation $P(x_1 \ldots x_m) = 0$ has solutions.

# Outline

Introduction

Undecidability Results

- Propositions
- Infinitely related Monoid

Decidability Results

Additional Remarks

# Generator propositions

- The resource model is often discrete.
- In separation logic, formulae are constructed from atomic assertions like “$\mathtt{x} \mapsto -, -$”.

# Generator propositions

- The resource model is often discrete.
- In separation logic, formulae are constructed from atomic assertions like "$\mathtt{x} \mapsto -, -$".

- Given a monoid $M = (X; R)$, define $p_x$ such that $\rho(p_x) = \{ x \mid x \in X \}$. We call these $p_x$ "generator propositions".

# Undecidability

- Even restricted to generator propositions, the model checking problem in infinitely related monoid is undecidable.

# Undecidability

- Even restricted to generator propositions, the model checking problem in infinitely related monoid is undecidable.
- In comparison, the model checking problem for quantifier-free assertion language of separation logic is decidable. The model is a partially defined infinitely related monoid.

# Minsky Machine

- Deterministic computation model. A series of commands and several counters.

- Two types of commands:
    1. Increase a counter, then jump.
    2. If a counter is zero, then do nothing and jump, else decrease and jump.

# Minsky Machine

- Deterministic computation model. A series of commands and several counters.
- Two types of commands:
  1. Increase a counter, then jump.
  2. If a counter is zero, then do nothing and jump, else decrease and jump.
- Snapshot $(i, m, n)$: current command line $i$, the values of the two counters $m, n$.

# Proof Outline

- Reduce the halting problem of Minsky Machine to the model checking problem.
- Construct a monoid such that Minsky Machine halts iff a special element satisfies a certain formula.

# Generator set

- The generator set contains four parts:
  - $Q = \{q_i\}$ : the command lines;
  - $S = \{s_{i,\lambda_k}\}$ : positions in a command sequence;
  - $A_1 = \{a_{1,i}\}$ and $A_2 = \{a_{2,j}\}$ : the status of the two counters;
  - $halt$.
- $\lambda_k$ is a sequence like $2, 3, 4'1$.

# Generator set

- The generator set contains four parts:
  - $Q = \{q_i\}$ : the command lines;
  - $S = \{s_{i,\lambda_k}\}$ : positions in a command sequence;
  - $A_1 = \{a_{1,i}\}$ and $A_2 = \{a_{2,j}\}$ : the status of the two counters;
  - $halt$.
- $\lambda_k$ is a sequence like $2, 3, 4'1$.
- $q_i \circ a_{1,m} \circ a_{2,n}$ corresponds to the snapshot $(i, m, n)$.

# Generation relation

- Every command corresponds to a generation relation pattern.

## Generation relation

- Every command corresponds to a generation relation pattern.
- The both sides of a relation are of the form $s_{j,\lambda_k} \circ q_i \circ a_{1,m} \circ a_{2,n}$, except those containing *halt*.

# Generation relation

- Every command corresponds to a generation relation pattern.
- The both sides of a relation are of the form $s_{j,\lambda_k} \circ q_i \circ a_{1,m} \circ a_{2,n}$, except those containing $halt$.
- Execute $j$th command in $\lambda_k$ in the snapshot $(i, n, m)$, leads to $s_{j+1,\lambda_k}$ multiplies appropriate element.

# Simulation

- Every element whose congruence class is non-trivial is of the form $s_{j,\lambda_k} \circ q_i \circ a_{1,n} \circ a_{2,m}$.

# Simulation

- Every element whose congruence class is non-trivial is of the form $s_{j,\lambda_k} \circ q_i \circ a_{1,n} \circ a_{2,m}$.
- The execution of Minsky machine can be viewed as applying appropriate generation relation from $s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}$.

# Simulation

- Every element whose congruence class is non-trivial is of the form $s_{j,\lambda_k} \circ q_i \circ a_{1,n} \circ a_{2,m}$.
- The execution of Minsky machine can be viewed as applying appropriate generation relation from $s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}$.
- If and only if the Minsky machine halts, there exists a $\lambda_k$ such that $s_{k,\lambda_k} \circ halt \in [s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$.

# Reduction

- Define $\phi_{as} = (\neg(\neg\top^* * \neg\top^*)) \wedge (\bigwedge_i \neg p_{q_i}) \wedge (\neg p_{halt})$.
  Thus $\rho(\varphi_{as}) = S \cup A_1 \cup A_2$.

# Reduction

- Define $\phi_{as} = (\neg(\neg\top^* * \neg\top^*)) \wedge (\bigwedge_i \neg p_{q_i}) \wedge (\neg p_{halt})$.
  Thus $\rho(\varphi_{as}) = S \cup A_1 \cup A_2$.

- Define $\phi = \phi_{as} \mathrel{-\!\!*}^{\exists} (p_{halt} * \phi_{as})$.

# Reduction

- Define $\phi_{as} = (\neg(\neg\top^* * \neg\top^*)) \wedge (\bigwedge_i \neg p_{q_i}) \wedge (\neg p_{halt})$.
  Thus $\rho(\varphi_{as}) = S \cup A_1 \cup A_2$.

- Define $\phi = \phi_{as} \twoheadrightarrow^{\exists} (p_{halt} * \phi_{as})$.

- Minsky machine halts. $\Leftrightarrow q_1 \circ a_{1,0} \circ a_{2,0} \models \varphi$.

# Outline

# Rational sets

## Definition (Rational Sets)

Let $M$ be a monoid (not necessarily be commutative). The class of rational subsets of $M$ is the least class $\mathscr{E}$ of subsets of $M$ satisfying the following conditions:

1. The empty set is in $\mathscr{E}$;
2. Each single element set is in $\mathscr{E}$;
3. If $X, Y \in \mathscr{E}$ then $X \cup Y \in \mathscr{E}$;
4. If $X, Y \in \mathscr{E}$ then $X \circ Y \in \mathscr{E}$;
5. If $X \in \mathscr{E}$ then $X^* \in \mathscr{E}$.

# Semi-linear sets

## Definition (Semi-linear Sets)

A subset $X = \{a\} \circ B^*$ with $a \in M$, $B \subseteq M$, and $B$ finite, is called linear. A finite union of linear sets is called semi-linear.

- Close representation of a semi-linear set :$a_1, \ldots, a_k$ and $B_1, \ldots, B_k$.

# Some facts

- For a f.g. commutative monoid $M$, A subset $X \subseteq M$ is rational iff it is semi-linear. (Eilenberg and Schutzenberger '69)
- If $X$ and $Y$ are rational subsets of a commutative monoid $M$, then their intersection $X \cap Y$, difference $Y \backslash X$ (hence $\overline{X} = M \backslash X$) and $Y : X$ are rational. (E, S '69)

# Some facts

- For a f.g. commutative monoid $M$, A subset $X \subseteq M$ is rational iff it is semi-linear. (Eilenberg and Schutzenberger '69)
- If $X$ and $Y$ are rational subsets of a commutative monoid $M$, then their intersection $X \cap Y$, difference $Y \backslash X$ (hence $\overline{X} = M \backslash X$) and $Y : X$ are rational. (E, S '69)

Recall that $\rho(\varphi_1 * \varphi_2) = \rho(\varphi_1) \circ \rho(\varphi_2)$, $\rho(\varphi_1 \mathrel{-\!\!*}^{\exists} \varphi_2) = \rho(\varphi_2) : \rho(\varphi_1)$. By induction, it follows that all $\rho(\varphi)$ are rational sets, and hence semi-linear sets.

# Compute semi-linear sets

- Indeed, all $[m]$ are also semi-linear sets.

# Compute semi-linear sets

- Indeed, all $[m]$ are also semi-linear sets.
- Koppenhagen and Mayr have developed an algorithm to compute the closed representation of a congruence class within exponential space.

# Back to the model checking problem

- Consider the canonical surjective morphism $\alpha : X^* \mapsto M$, $\alpha^{-1}(m) = [m]$. We have:

$$
\begin{aligned}
m \in \rho(\varphi) \quad &\Leftrightarrow \quad [m] \subseteq \alpha^{-1}(\rho(\varphi)) \\
&\Leftrightarrow \quad [m] \cap \alpha^{-1}(\rho(\varphi)) \neq \emptyset.
\end{aligned}
$$

# Back to the model checking problem

- Consider the canonical surjective morphism $\alpha : X^* \mapsto M$, $\alpha^{-1}(m) = [m]$. We have:

$$\begin{array}{rcl} m \in \rho(\varphi) & \Leftrightarrow & [m] \subseteq \alpha^{-1}(\rho(\varphi)) \\ & \Leftrightarrow & [m] \cap \alpha^{-1}(\rho(\varphi)) \neq \emptyset. \end{array}$$

- We already can compute the closed representation of $[m]$. In the following we show how to compute that of $\alpha^{-1}(\rho(\varphi))$.
- In fact, we compute it inductively, and hence the following lemma is needed.

# From connectives to set operations

## Lemma

For a f.g. monoid $M = (X; R)$ and BI formulae $\varphi$, $\varphi_1$, and $\varphi_2$, the following holds:

- $\alpha^{-1}(\rho(p_x)) = [x]$
- $\alpha^{-1}(\rho(\top)) = X^*$
- $\alpha^{-1}(\rho(\neg\varphi)) = \overline{\alpha^{-1}(\rho(\varphi))}$
- $\alpha^{-1}(\rho(\varphi_1 \wedge \varphi_2)) = \alpha^{-1}(\rho(\varphi_1)) \cap \alpha^{-1}(\rho(\varphi_2))$
- $\alpha^{-1}(\rho(\top^*)) = [\varepsilon]$
- $\alpha^{-1}(\rho(\varphi_1 * \varphi_2)) = \alpha^{-1}(\rho(\varphi_1)) \circ \alpha^{-1}(\rho(\varphi_2))$
- $\alpha^{-1}(\rho(\varphi_1 \mathrel{-\!\!*^{\exists}} \varphi_2)) = \alpha^{-1}(\rho(\varphi_2)) : \alpha^{-1}(\rho(\varphi_1))$

# Compute semi-linear sets

- Since $\alpha^{-1}(\rho(p_x)) = [x]$, Koppenhagen-Mayr algorithm also builds up our induction basis. What we left to do is to compute the closed representations of $\overline{X}$, $X \cap Y$, $X \circ Y$, and $X : Y$.

# Compute semi-linear sets

- Since $\alpha^{-1}(\rho(p_x)) = [x]$, Koppenhagen-Mayr algorithm also builds up our induction basis. What we left to do is to compute the closed representations of $\overline{X}$, $X \cap Y$, $X \circ Y$, and $X : Y$.
- Since $X^* \cong \mathbb{N}^k$, we consider these semi-linear sets in $\mathbb{N}^k$.

# Compute semi-linear sets

- Since $\alpha^{-1}(\rho(p_x)) = [x]$, Koppenhagen-Mayr algorithm also builds up our induction basis. What we left to do is to compute the closed representations of $\overline{X}$, $X \cap Y$, $X \circ Y$, and $X : Y$.
- Since $X^* \cong \mathbb{N}^k$, we consider these semi-linear sets in $\mathbb{N}^k$.
- For two semi-linear sets $X = \bigcup_i (a_i + B_i^*)$ and $Y = \bigcup_j (a_j + B_j^*)$, it is easy to see:

$$
\begin{aligned}
X + Y &= \bigcup_{i,j}((a_i + B_i^*) + (a_j + B_j^*)) \\
X \cap Y &= \bigcup_{i,j}((a_i + B_i^*) \cap (a_j + B_j^*)) \\
Y - X &= \bigcup_{i,j}((a_j + B_j^*) - (a_i + B_i^*)) \\
\overline{X} &= \bigcap_i (\overline{a_i + B_i^*})
\end{aligned}
$$

Hence we only need to deal with linear sets.

# The case of $X + Y$ and $X \cap Y$

$X + Y$ For two linear sets $a + B^*$ and $a' + B'^*$, it is easy to see their summation is:

$$(a + a') + (B \cup B')^*$$

# The case of $X + Y$ and $X \cap Y$

$X + Y$ For two linear sets $a + B^*$ and $a' + B'^*$, it is easy to see their summation is:

$$(a + a') + (B \cup B')^*$$

$X \cap Y$ For two linear sets $a + B^*, a' + B'^* \subseteq \mathbb{N}^k$. Assume $B = \{b_1, \ldots, b_n\}$ and $B' = \{b'_1, \ldots, b'_{n'}\}$, then every element in $X \cap Y$ corresponds to two vectors $\{x_i\}$, $\{x'_i\}$, which satisfies the following system of linear Diophantine equations:

$$\sum_{i=1}^{n} b_i x_i - \sum_{j=1}^{n'} b'_j x'_j = a' - a$$

## Solving the system of linear Diophantine equations

- The solution of a system of linear Diophantine equations, in fact, constitutes a semi-linear set.

# Solving the system of linear Diophantine equations

- The solution of a system of linear Diophantine equations, in fact, constitutes a semi-linear set.
- There are many algorithms to solve this problem.

# The case of $Y - X$

For two linear sets $X = a + B^*$ and $Y = a' + B'^*$, assume $B = \{b_1, \ldots, b_n\}$ and $B' = \{b'_1, \ldots, b'_{n'}\}$. It is easy to see that

$$Y - X = \{(a' - a) + \sum_{i=1}^{n'}(t'_i b'_i) - \sum_{j=1}^{n}(t_j b_j) | t'_i, t_j \in \mathbb{N}\} \cap \mathbb{N}^k$$

Then it is similar to the $X \cap Y$ case. We can get the representation after solving the system of linear Diophantine equations:

$$(a' - a) + \sum_{i=1}^{n'}(t'_i b'_i) - \sum_{j=1}^{n}(t_j b_j) = \sum_{i=1}^{k} x_i e_i$$

in which $t'_i$, $t_i$, $x_i$ are variables.

# The case of $\overline{X}$

- Assume $X = a + B^*$. Divide $\mathbb{N}^k$ into a series of semi-linear sets $\{a_j + B_j^* + B^*\}$.
- $X$ must lie in some of these sets. It is easy to express the subtraction.

# Check $m \models \varphi$

Procedure:

1. Generate the representation of $[m]$ and $\alpha^{-1}(\rho(\varphi))$.
2. Decide whether them overlap.

# Outline

# Finitely Related Monoid

- For *infinitely generated finitely related* monoid, the model checking problem can be reduced to the finitely generated case.

# Finitely Related Monoid

- For *infinitely generated finitely related* monoid, the model checking problem can be reduced to the finitely generated case.
- There are only finitely many generators that will be involved in the process of model checking.
- Map all the generator of no interest to one of them. The truth of the satisfaction relation will not change.
- The model checking problem for all finitely related monoid is decidable.

# Automata theory

- We may add a new connective corresponds to $X^*$. Thus every rational set has the the form of $\rho(\varphi)$.
- Kleene theorem : In a free commutative monoid, a set is rational iff it is recognizable by finite automata.
- It is shown that in the case of finitely generated commutative monoid, a monoid is kleene iff it is rational. (Rupert '91)
  $\Rightarrow$ The set $\rho(\varphi)$ is recognizable by finite automata, iff the monoid is rational, .

# Model checking BI and CBI

BI Preorder.
Chain condition.

# Model checking BI and CBI

BI Preorder.
Chain condition.

CBI Similar to inverse monoid or cancellative monoid.
Weaker decidable condition.

# Thanks!