How to Grind JAVA Programs to Extract Full-bodied Infinite-state Models?

Mémoire de D.E.A.

Gilles GEERAERTS (joint work with Laurent VAN BEGIN)
Grind? Full-bodied?


1. To reduce to powder by friction, as in a mill, or with the teeth; to crush into small fragments; to produce as by the action of millstones. [...]

4. To study hard for examination. [College Slang]

**Full-bodied** adj: marked by richness and fullness of flavor; “a rich ruby port”; “full-bodied wines”; “a robust claret”; “the robust flavor of fresh-brewed coffee”

*from: Webster’s Revised Unabridged Dictionary (1913)*
The verification of Java software

Java Software

Local/Global Machine

Multi-Transfer Net

Babylon
JAVA software?

- Multi-threaded JAVA programs...
Multi-threaded JAVA programs... with unbounded instantiation of the threads.
Multi-threaded **JAVA** programs...

...with *unbounded* instantiation of the threads.

...using *communication primitives*: `notify`, `notifyAll`, `wait`...
JAVA software?

- Multi-threaded JAVA programs...
  - ...with unbounded instantiation of the threads.
  - ...using communication primitives: notify, notifyAll, wait...

- Bounded recursion, as we inline the procedure calls
**JAVA software?**

- Multi-threaded JAVA programs...
  - ...with unbounded instantiation of the threads.
  - ...using communication primitives: `notify`, `notifyAll`, `wait`...
- Bounded recursion, as we inline the procedure calls
- Bounded data structures
public class Point {
    private int x = 0;
    private int y = 0;

    public synchronized void incx()
    {
        x = x + 1;
        notifyAll();
    }

    public synchronized void decx()
    {
        while (x == 0)
        {
            wait();
            x = x - 1;
        }
    }

    public synchronized void incy()
    {
        y = y + 1;
        notifyAll();
    }

    public synchronized void decy()
    {
        while (y == 0)
        {
            wait();
            y = y - 1;
        }
    }
}
public class Inc extends Thread {
    private Point p;
    public Inc(Point p) {
        this.p = p;
    }

    private void incpoint() {
        p.incx();
        p.incy();
    }

    public void run() {
        while (true)
            incpoint();
    }
}

public class Dec extends Thread {
    private Point p;
    public Dec(Point p) {
        this.p = p;
    }

    private void decpoint() {
        p.decx();
        p.decy();
    }

    public void run() {
        while (true)
            decpoint();
    }
}
Global/Local Machine(s) ?

- Global Machine =

- A set of Local Machines
- A set of Global Boolean Variables (accessible by every Local Machine)

- Local Machine =
- A finite set of locations
- A finite set of transitions, possibly using communication constructs:
  - Synchronous one-to-one: rendez-vous
  - Asynchronous one-to-one: asynchronous rendez-vous
  - One-to-many: broadcast
Global/Local Machine(s) ?

- Global Machine =
  - A set of Local Machines
Global/Local Machine(s) ?

- Global Machine =
  - A set of Local Machines
  - A set of Global Boolean Variables (accessible by every Local Machine)
Global/Local Machine(s)?

- **Global Machine** =
  - A set of Local Machines
  - A set of Global Boolean Variables (accessible by every Local Machine)
- **Local Machine** =
Global/Local Machine(s) ?

- Global Machine =
  - A set of Local Machines
  - A set of Global Boolean Variables (accessible by every Local Machine)
- Local Machine =
  - A finite set of locations
Global/Local Machine(s) ?

- Global Machine =
  - A set of Local Machines
  - A set of Global Boolean Variables (accessible by every Local Machine)

- Local Machine =
  - A finite set of locations
  - A finite set of transitions, possibly using communication constructs:
Global/Local Machine(s)?

Global Machine =
- A set of Local Machines
- A set of Global Boolean Variables (accessible by every Local Machine)

Local Machine =
- A finite set of locations
- A finite set of transitions, possibly using communication constructs:
  - Synchronous one-to-one: rendez-vous: $m!$ and $m?$
Global/Local Machine(s)?

- **Global Machine =**
  - A set of Local Machines
  - A set of Global Boolean Variables (accessible by every Local Machine)

- **Local Machine =**
  - A finite set of locations
  - A finite set of transitions, possibly using communication constructs:
    - Synchronous one-to-one: *rendez-vous*: \( m! \) and \( m? \)
    - Asynchronous one-to-one: asynchronous *rendez-vous*: \( m \uparrow \uparrow \) and \( m \downarrow \downarrow \)
Global/Local Machine(s)?

- Global Machine =
  - A set of Local Machines
  - A set of Global Boolean Variables (accessible by every Local Machine)
- Local Machine =
  - A finite set of locations
  - A finite set of transitions, possibly using communication constructs:
    - Synchronous one-to-one: rendez-vous: \( m! \) and \( m? \)
    - Asynchronous one-to-one asynchronous rendez-vous: \( m \uparrow\uparrow \) and \( m \downarrow\downarrow \)
    - One-to-many: broadcast: \( m!! \) and \( m?? \)
Let’s fill the gap!

Java Software

Concurrent Boolean Program

Local/Global Machine
Let’s fill the gap!

Java Software

Concurrent Boolean Program

Local/Global Machine

How do we do this?

How to Grind JAVA Programs to Extract Full-bodied Infinite-state Models?
Concurrent Boolean Programs?

- CBP’s are Abstract multi-threaded programs
Concurrent Boolean Programs?

- CBP’s are **Abstract multi-threaded programs**
- They manipulate boolean variable only (and locks)
Concurrent Boolean Programs?

- CBP's are **Abstract multi-threaded programs**
- They manipulate boolean variable only (and locks)
- The variables can be global or local to the threads
Concurrent Boolean Programs?

- CBP’s are **Abstract multi-threaded programs**
- They manipulate boolean variable only (and locks)
- The variables can be global or local to the threads

Available constructs:
Concurrent Boolean Programs?

- CBP’s are **Abstract multi-threaded programs**
  - They manipulate boolean variable only (and locks)
  - The variables can be global or local to the threads

- Available constructs:
  - Classical flow control instructions:
    - *if, while*
Concurrent Boolean Programs?

- CBP’s are **Abstract multi-threaded programs**
  - They manipulate boolean variable only (and locks)
  - The variables can be global or local to the threads

- Available constructs:
  - Classical flow control instructions:
    - if, while
  - Non-deterministic atomic guarded assignment:
    - choice(c1 : v1, v2 := u1, u2; c2 : v3, v4 := u3, u4; ...)

How to Grind JAVA Programs to Extract Full-bodied Infinite-state Models? – p.9/17
Concurrent Boolean Programs?

- CBP’s are Abstract multi-threaded programs
  - They manipulate boolean variable only (and locks)
  - The variables can be global or local to the threads

Available constructs:
- Classical flow control instructions:
  - if, while
- Non-deterministic atomic guarded assignment:
  - choice( c1 : v1, v2 := u1, u2; c2 : v3, v4 := u3, u4; ...)
- Synchronisation primitives:
  - rendezvous (with value passing), sleep, wakeup, wakeupall, lock, unlock, start...
inc {vars ;
    while(true) {
        lock(lockpoint);
        choice {
            x0 : x0 := false;
            !x0 : x0 := false;
            !x0 : x0 := true;
        }
        wakeupall(msgpoint);
        unlock(lockpoint);
        lock(lockpoint);
        choice {
            y0 : y0 := false;
            !y0 : y0 := false;
            !y0 : y0 := true;
        }
        wakeupall(msgpoint);
        unlock(lockpoint);
    }
}

dec {vars ;
    while(true) {
        lock(lockpoint);
        while(x0) {
            sleep(msgpoint, lockpoint);
        }
        choice {
            x0 : x0 := false;
            !x0 : x0 := false;
            !x0 : x0 := true;
        }
        unlock(lockpoint);
        lock(lockpoint);
        while(y0) {
            sleep(msgpoint, lockpoint);
        }
        choice {
            y0 : y0 := false;
            !y0 : y0 := false;
            !y0 : y0 := true;
        }
        unlock(lockpoint);
    }
}
From the CBP’s to the GM’s

Global State of a CBP:

Instructions that remain to be executed by thread 1

Valuation of the locks

Local states of the threads

Valuation of the Global variables

Valuation of thread 1’s local variables

\langle \Gamma, \Lambda, [\langle P_1, \rho_1 \rangle, \langle P_2, \rho_2 \rangle, \ldots] \rangle
From the CBP’s to the GM’s

Global State of a CBP:

- Intructions that remain to be executed by thread 1
- Valuation of thread 1’s local variables
- Valuation of the locks
- Local states of the threads
- Valuation of the Global variables

Global State of a GM:

- Local states of the threads
- Valuation of the boolean variables

\[ \langle G, \{s_1, s_2, \ldots \} \rangle \]

\[ \langle \Gamma, \Lambda, \{\langle P_1, \rho_1 \rangle, \langle P_2, \rho_2 \rangle, \ldots \} \rangle \]
From the CBP’s to the GM’s cont’d

- We can easily find a correspondence
From the CBP’s to the GM’s cont’d

- We can easily find a correspondence
- between $\Gamma \cup \Lambda$ and $G$,
From the CBP’s to the GM’s cont’d

- We can easily find a correspondence between $\Gamma \cup \Lambda$ and $G$,
- between $\langle P_i, \rho_i \rangle$ and $s_i$ (for all $i$).
From the CBP’s to the GM’s cont’d

- We can easily find a correspondence between $\Gamma \cup \Lambda$ and $G$,
- between $\langle P_i, \rho_i \rangle$ and $s_i$ (for all $i$).
- We first relabel the program to ensure the unity of the labels.
We can easily find a correspondence
between $\Gamma \cup \Lambda$ and $G$,
between $\langle P_i, \rho_i \rangle$ and $s_i$ (for all $i$).
We first relabel the program to ensure the unity of the labels.
The valuations of the CBP local variables are encoded into the GM local states.
We can easily find a correspondence
between $\Gamma \cup \Lambda$ and $G$,
between $\langle P_i, \rho_i \rangle$ and $s_i$ (for all $i$).
We first relabel the program to ensure the unity
of the labels.
The valuations of the CBP local variables are
encoded into the GM local states.
Thus, if $P_i \equiv [\ell] I \cdot P'$ then $s_i \equiv \langle \ell, \rho_i \rangle$. 

How to Grind JAVA Programs to Extract Full-bodied Infinite-state Models ? – p.12/17
From the CBP’s to the GM’s cont’d

- We can easily find a correspondence between $\Gamma \cup \Lambda$ and $G$,
- between $\langle P_i, \rho_i \rangle$ and $s_i$ (for all $i$).
- We first relabel the program to ensure the unity of the labels.
- The valuations of the CBP local variables are encoded into the GM local states.
- Thus, if $P_i \equiv [\ell] I \cdot P'$ then $s_i \equiv \langle \ell, \rho_i \rangle$.
- We handle the creation of threads as follows:
From the CBP’s to the GM’s cont’d

- We can easily find a correspondence between $\Gamma \cup \Lambda$ and $G$,
- between $\langle P_i, \rho_i \rangle$ and $s_i$ (for all $i$).
- We first relabel the program to ensure the unity of the labels.
- The valuations of the CBP local variables are encoded into the GM local states.
- Thus, if $P_i \equiv [\ell] I \cdot P'$ then $s_i \equiv \langle \ell, \rho_i \rangle$.

- We handle the creation of threads as follows:
  - Each LM has a initial state representing the ‘not-yet-created’ state of the CBP thread.
We can easily find a correspondence between $\Gamma \cup \Lambda$ and $G$,

between $\langle P_i, \rho_i \rangle$ and $s_i$ (for all $i$).

We first relabel the program to ensure the unity of the labels.

The valuations of the CBP local variables are encoded into the GM local states.

Thus, if $P_i \equiv [\ell]I \cdot P'$ then $s_i \equiv \langle \ell, \rho_i \rangle$.

We handle the creation of threads as follows:

Each LM has a initial state representing the ‘not-yet-created’ state of the CBP thread.

The start is modelled by a rendez-vous.
To cope with the possibly unbounded creation of threads, we translate a CBP \( \langle \mathcal{G}, \mathcal{L}, \mathcal{T} \rangle \) into a Family of GM's:

\[
\mathcal{F}(B) = \{(\mathcal{G} \cup \mathcal{L}, \{(\mathcal{L}_1, k_1), \ldots, (\mathcal{L}_n, k_n)\}) \mid \forall 1 \leq i \leq n : k_i \geq 1\}
\]

where \( \{\mathcal{L}_1, \ldots, \mathcal{L}_k\} \) is the set of Local Machines we have obtained by translating each CBP thread.
From the CBP’s to the GM’s cont’d

To cope with the possibly unbounded creation of threads, we translate a CBP \( \langle G, L, T \rangle \) into a Family of GM’s:

\[
\mathcal{F}(B) = \{ (G \cup L, \{ (L_1, k_1), \ldots, (L_n, k_n) \}) \mid \forall 1 \leq i \leq n : k_i \geq 1 \}
\]

where \( \{L_1, \ldots, L_k\} \) is the set of Local Machines we have obtained by translating each CBP thread.

**Theorem**

Given \( B = \langle G, L, \{ T_1, \ldots, T_n \} \rangle \), a CBP, and \( \mathcal{F}(B) \), its corresponding family of GM’s: \( e \) is an execution of \( B \) iff there exists a GM \( G \in \mathcal{F}(B) \), and a run \( f \) of \( G \) such that:

\[
\Phi'(f) = e
\]
From the CBP’s to the GM’s – example
Some last remarks:

- We need to reduce the size of the models to avoid intractability
Some last remarks:

- We need to reduce the size of the models to avoid intractability.
- Some lock-based reduction techniques by Corbett and Stoller already exist.
Some last remarks:

- We need to reduce the size of the models to avoid intractability.
- Some lock-based reduction techniques by Corbett and Stoller already exist.
- We need to exploit static analysis technique.
Some last remarks:

- We need to reduce the size of the models to avoid intractability.
- Some lock-based reduction techniques by Corbett and Stoller already exist.
- We need to exploit static analysis technique.

We have implemented a tool (CBP2GM) to translate the CBP’s into GM’s.
From Java to the CBP’s

This is *really* the *funny* part of the problem...
From Java to the CBP’s

- This is *really* the funny part of the problem...
- Existing techniques work by iteratively refining the models (*SLAM* tool by Ball and Rajamani)
From Java to the CBP’s

- This is *really* the *funny* part of the problem...
- Existing techniques work by iteratively refining the models (*SLAM* tool by Ball and Rajamani)
  - One begins with a *coarse skeleton* of the program.
This is *really* the funny part of the problem...

Existing techniques work by iteratively refining the models (*SLAM* tool by Ball and Rajamani)

One begins with a coarse skeleton of the program.

It is model-checked.
From Java to the CBP’s

- This is *really* the funny part of the problem . . .
- Existing techniques work by iteratively refining the models (*SLAM* tool by Ball and Rajamani)
  - One begins with a coarse skeleton of the program.
  - It is model-checked.
  - If the error trace is spurious, the model is adapted to avoid it . . .
From Java to the CBP’s

- This is *really* the funny part of the problem . . .
- Existing techniques work by iteratively refining the models (*SLAM* tool by Ball and Rajamani)
  - One begins with a coarse skeleton of the program.
  - It is model-checked.
  - If the error trace is spurious, the model is adapted to avoid it . . .
- But . . .
From Java to the CBP’s

- This is *really* the **funny** part of the problem... 

- Existing techniques work by iteratively refining the models (*SLAM* tool by Ball and Rajamani)
  - One begins with a **coarse skeleton** of the program.
  - It is model-checked.
  - If the error trace is spurious, the model is adapted to avoid it...

- But...
  - These techniques are for sequential programs.
From Java to the CBP’s

- This is *really* the **funny** part of the problem. . .
- Existing techniques work by iteratively refining the models (*SLAM* tool by Ball and Rajamani)
  - One begins with a **coarse skeleton** of the program.
  - It is model-checked.
  - If the error trace is spurious, the model is adapted to avoid it. . .
- But...
  - These techniques are for sequential programs.
  - What if we have **unbounded intricate data structures** (lists, a.s.o.) ?
From Java to the CBP’s

- This is really the funny part of the problem...
- Existing techniques work by iteratively refining the models (SLAM tool by Ball and Rajamani)
  - One begins with a coarse skeleton of the program.
  - It is model-checked.
  - If the error trace is spurious, the model is adapted to avoid it...
- But...
  - These techniques are for sequential programs.
  - What if we have unbounded intricate data structures (lists, a.s.o.)?
  - Finer static analysis structures (like Sagiv’s) seem worth looking into.
Conclusion and future works

- By extending Rajamani’s and Ball’s Boolean Programs, we now have the theoretical basis to investigate the problems of model extraction of Java programs.

To extend our work, we could try to cope with two dimensions of infinity: unbounded control through unbounded recursion. Unbounded data through unbounded data structures.

Conclusion: There is still a lot of work to do!!
Conclusion and future works

- By extending Rajamani’s and Ball’s Boolean Programs, we now have the theoretical basis to investigate the problems of *model extraction* of Java programs.
- We still have to investigate the model reduction techniques.
Conclusion and future works

- By extending Rajamani’s an Ball’s Boolean Programs, we now have the theoretical basis to investigate the problems of model extraction of Java programs.
- We still have to investigate the model reduction techniques.
- To extend our work, we could try to cope with two dimensions of infinity: unbounded control through unbounded recursion, unbounded data’s through unbounded data structure.

Conclusion: There is still a lot of work to do!!
Conclusion and future works

- By extending Rajamani’s and Ball’s Boolean Programs, we now have the theoretical basis to investigate the problems of model extraction of Java programs.
- We still have to investigate the model reduction techniques.
- To extend our work, we could try to cope with two dimensions of infinity:
  - Unbounded control through unbounded recursion.
Conclusion and future works

- By extending Rajamani’s and Ball’s Boolean Programs, we now have the theoretical basis to investigate the problems of model extraction of Java programs.
- We still have to investigate the model reduction techniques.
- To extend our work, we could try to cope with two dimensions of infinity:
  - Unbounded control through unbounded recursion.
  - Unbounded data’s through unbounded data structure.
Conclusion and future works

- By extending Rajamani’s and Ball’s Boolean Programs, we now have the theoretical basis to investigate the problems of model extraction of Java programs.
- We still have to investigate the model reduction techniques.
- To extend our work, we could try to cope with two dimensions of infinity:
  - Unbounded control through unbounded recursion.
  - Unbounded data’s through unbounded data structure.
- Conclusion:

There is still a lot of work to do !!
Conclusion and future works

- By extending Rajamani’s and Ball’s Boolean Programs, we now have the theoretical basis to investigate the problems of model extraction of Java programs.
- We still have to investigate the model reduction techniques.
- To extend our work, we could try to cope with two dimensions of infinity:
  - Unbounded control through unbounded recursion.
  - Unbounded data’s through unbounded data structure.
- Conclusion: There is still a lot of work to do!!