SPADE
Verification of Multithreaded Dynamic and Recursive Programs

Gaël Patin, Mihaela Sighireanu, Tayssir Touili
LIAFA, CNRS & Univ. Paris 7, France
Goal

Verify programs with:

• Procedure calls (possibly recursive)
• Dynamic creation of parallel processes
• Communication between parallel processes
  (handshaking by blocking send and receive actions)

Undecidable
  (even with finite-domain variables)

This Work: Approximate analysis techniques
We need to:

Define accurate models:

• Procedure calls,
• Dynamic creation of parallel processes,
• Communication between parallel processes (handshakings)

Find analysis techniques for these models
Existing Work

- No technique that can deal with all the features

- The different models that were considered cannot represent accurately all the features
Previous attempts

Different proposals based on solving sets of constraints  [Müller-Olm, Seidl, Steffen,...]

No Synchronisation 😞

Synchronisation via locks  [Kahloon,...]

Synchronisation via locks 😞
Previous attempts

Constrained Dynamic Pushdown Network \textbf{(CDPN)}
[Bouajjani, Müller-Olm, T. 05]

Procedure calls, Dynamism 😊
Synchronisation not precisely modeled 😞

Communicating PushDown Systems \textbf{(CPDS)}
[Bouajjani, Esparza, T. 03] [Qadeer, Rehof 05][Chaki, Clarke, Kidd, Reps, T. 06]

Procedure calls, Synchronisation 😊
No Dynamism 😞
Previous attempts

Process Rewrite Systems (PRS) [Bouajjani, T. 03-05]
   Procedure calls, Dynamism ☺
   Synchronisation not precisely modeled 😞

Synchronized PA (SPA) [Bouajjani, Esparza, T. 04]
   Synchronisation, Dynamism ☺
   Procedure calls not precisely modeled 😞
This Work

• Define a more general model:

Synchronized PAD (SPAD)

Procedure calls (recursion), Synchronisation, Dynamism 😊

• Define analysis techniques for this model

• Bug found in a Bluetooth driver in Windows
The model: Synchronised PAD
Syntax

- **Term**: \( t ::= 0 | X, Y, \ldots | t.t | t||t \)
- **0 neutral**: \( t.0=0.t=t||0=0||t=t \)
- **. associative**: \( (t.u).v=t.(u.v) \)
- **|| associative**: \( (t||u)||v=t||(u||v) \)
- **|| commutative**: \( t||u=u||t \)

- **Actions**: \( Act = \{ \tau \} \cup \{ a!, a? | a \in Sync \} \)

- **SPAD**: \( X \xrightarrow{b} t ; X \cdot Y \xrightarrow{b} t \)
Transition Relation

Basic case:

\[ t_1 \xrightarrow{b} t_2 \in R \]

\[ t_1 \xrightarrow{b} t_2 \]

Sequential composition: Prefix rewriting strategy

\[ t_1 \xrightarrow{b} t_2 \]

\[ t_1 \cdot u \xrightarrow{b} t_2 \cdot u \]

\[ t_1 \xrightarrow{b} t_2 \text{ and } u \sim 0 \]

\[ u \cdot t_1 \xrightarrow{b} u \cdot t_2 \]
Transition Relation

Parallel composition:

\[
t_1 \xrightarrow{b} t_2 \\
u \parallel t_1 \xrightarrow{b} u \parallel t_2 \quad \text{and} \quad t_1 \parallel u \xrightarrow{b} t_2 \parallel u
\]

Synchronisation:

\[
t_1 \xrightarrow{a!} t_2 \quad \text{and} \quad u_1 \xrightarrow{a?} u_2 \\
t_1 \parallel u_1 \xrightarrow{\tau} t_2 \parallel u_2
\]

**Good** Execution: \( a! \) matched with \( a? \) \( \rightarrow \) only \( \tau \)
This Work

Define a more general model:
*Synchronized PAD (SPAD)*

Procedure calls (recursion), Synchronisation, Dynamism 😊

- Define analysis techniques for this model
- **Bug found** in a **Bluetooth driver in Windows**
From Programs to SPAD

Procedure call:  \[ n \xrightarrow{\text{call}(p)} m \] \[ n \xrightarrow{\tau} e_p.m \]

Result return:  \[ m \xrightarrow{\text{if } p \text{ returns } r_i} m_i \] \[ r_i.m \xrightarrow{\tau} m_i \]

Termination:  \[ n \xrightarrow{\tau} 0 \]

Dynamic creation:  \[ n \xrightarrow{\tau} m_1\parallel m_2 \]

Synchronisation by rendez-vous:  \[ n_1 \xrightarrow{a!} n_2 \] ;   \[ m_1 \xrightarrow{a?} m_2 \]
This Work

• Define a more general model: Synchronized PAD (SPAD)
  Recursion, Synchronisation, Dynamism 😊

• Define analysis techniques for this model

• **Bug found** in a Bluetooth driver in Windows
Reachability Problem

\[ \text{Init} \quad ? \quad \text{Bad} \]

\text{Init} \text{ and } \text{Bad}: \textbf{Infinite} sets of configurations

(reachability of a control point)
Reachability Problem

\[ \text{Init} \quad ? \quad \text{Bad} \]

In our modeling:

- \textit{Init} and \textit{Bad}: \textbf{Infinite} sets of terms

\[
\text{Good - Executions}_{SPAD}(\text{Init}, \text{Bad}) = \emptyset ?
\]

\[
\text{Executions}_{SPAD}(\text{Init}, \text{Bad}) \cap \tau^* = \emptyset ?
\]

Impossible 😞

\[
A(\text{Executions(Init, Bad)}) \cap \tau^* = \emptyset ?
\]
Our Approach

\[ \text{Executions}(\text{Init}, \text{Bad}) \cap \tau^* = \phi \]???

Compute over-approximation \( A(\text{Executions}(\text{Init}, \text{Bad})) \)

\[ A(\text{Executions}(\text{Init}, \text{Bad})) \cap \tau^* = \phi \]?

Can we extract a real execution?

Refine approximation
Computing $A(\text{Executions}(\text{Init}, \text{Bad}))$?

- Characterize $\text{Executions}(\text{Init}, \text{Bad})$ by a set of constraints

- Consider an **abstract finite domain** whose elements represent over-approximations of languages of executions

- Solve the constraints in this **abstract finite domain** (an iterative least fixpoint computation terminates)

  $\Rightarrow$ Over-approximation
Prefix k Abstraction Domain

\[ L = abababbc^* \]

\[ \alpha_3(L) = aba (a + b + c)^* \]

Finite abstract domain: Domain of sets of words of length <= 3

Refinable abstractions: \( \alpha_1, \alpha_2, \alpha_3, \ldots \)

\[ \alpha_4(L) = abab(ab + b + c)^* \]
Computing $A(\text{Executions}(\text{Init},\text{Bad}))$?

- Characterize $\text{Executions}(\text{Init},\text{Bad})$ by a set of constraints.
- Consider an **abstract finite domain** whose elements represent over-approximations of languages of executions.

- Solve the constraints in this **abstract finite domain** (an iterative least fixpoint computation terminates).

→ Over-approximation
Characterizing \( \text{Executions}(\text{Init}, \text{Bad}) \) ?

First Problem: \textbf{Finitely} represent \textbf{infinite} sets of terms

\textbf{Term} = \textbf{Tree}

Infinite sets of terms: \textbf{Tree automata}
Tree Automata

- $A = (Q, F, \delta)$
- $\delta : X \rightarrow q$ ; $(q, q') \rightarrow q$ ; $\parallel (q, q') \rightarrow q$

Tree recognized by $p'$ ($\in L_{p'}$)
Characterizing Executions

\[ \text{Executions}(L_1, L_2) \]

Theorem:

If \( L_1 \) and \( L_2 \) compatible then

\[ \text{Executions}(L_1, L_2) = \text{Executions}_{\text{no-equivalence}}(L_1, L_2) \]
Characterizing Executions

*Executions* $(A_1, A_2)$

\[ E(q, q') = \text{Executions}(L_q, L_{q'}) \]

\[ \text{Executions} \ (A_1, A_2) = \bigcup_{q \in F_1} \bigcup_{q' \in F_2} E(q, q') \]
A first constraint

\[ L_q \cap L_s \neq \emptyset \implies \varepsilon \in E(q, s) \]
Another constraint

\[ t_1^b \rightarrow t_2 \in \mathbb{R} \]

\[ E(q, q_{t_1}) \cdot b \cdot E(q_{t_2}, s) \subseteq E(q, s) \]
One more constraint

\((q_1, q_2) \rightarrow q \in A_1 \text{ and } (s_1, s_2) \rightarrow s \in A_2\)

\[L_{q_2} \cap L_{s_2} \neq \emptyset \implies E(q_1, s_1) \subseteq E(q, s)\]
A last constraint

\[(q_1, q_2) \rightarrow q \in A_1 \text{ and } (s_1, s_2) \rightarrow s \in A_2\]

\[E(q_1, s_1^{null}) \cdot E(q_2, s_2) \subseteq E(q, s)\]
4 more constraints …
Characterizing Executions

\[ E(q, q') = \text{Executions}(L_q, L_{q'}) \]

\[ \text{Executions} \ (A_1, A_2) = \bigvee_{q \in F_1, q' \in F_2} E(q, q') \]
Computing $A(\text{Executions}(\text{Init, Bad}))$?

- Characterize $\text{Executions}(\text{Init, Bad})$ by a set of constraints

- Consider an abstract finite domain whose elements represent over-approximations of languages of executions

- Solve the constraints in this abstract finite domain (an iterative least fixpoint computation terminates)

$\rightarrow$ Over-approximation
Our Approach

$$\text{Executions}(\text{Init}, \text{Bad}) \cap \tau^* = \emptyset$$

Compute over-approximation $$A(\text{Executions}(\text{Init}, \text{Bad}))$$

$$A(\text{Executions}(\text{Init}, \text{Bad})) \cap \tau^* = \emptyset$$

- **YES**
  - Can we extract a real execution?
  - **YES**
    - Refine approximation
  - **NO**
- **NO**
  - Refine approximation
SPADE

- Characterize Paths(Init,Bad) by a set of constraints.
- Solve the constraints in the abstract finite domain $\alpha$ (an iterative least fixpoint computation terminates).

Compute $\alpha(\text{Paths}(\text{Init},\text{Bad}))$
Experiments and case studies
The Bluetooth Driver in Windows

• Found **automatically** two bugs in two versions of a **Bluetooth driver in Windows**

• Need to procedure calls, dynamic process creation, and synchronisation

• Previous work **guessed** the number of parallel threads to discover the bugs!!
Java Vector Object

- Programs that concurrently create and remove elements of a Java Vector object present a data race because the constructor of the Java Vector class is not atomic [Wand, Stoller 03]

- SPADE finds this bug for a program with unbounded number of threads

- SPADE proves that a corrected version of this program is correct
Concurrent Insertions in Binary Trees

• A buggy program considered in
  [Chaki, Clarke, Kidd, Reps, Touili’06]

• MAGIC found the bug for programs having
  less than 8 threads

• SPADE finds the bug for arbitrary number of
  threads
Conclusion

• Define a general model:
  Synchronized PAD (SPAD)
  Recursion, Synchronisation, Dynamism 😊

• Define analysis techniques for this model

• Bug found in a Bluetooth driver in Windows (without guessing the number of threads in parallel)
Questions?