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 (CDPN)

[Bouajjani,Müller-Olm,T. 05]

Procedure calls, Dynamism ☺

Synchronisation not precisely modeled ☹

Communicating PushDown Systems (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 $t ::= 0 \mid X, Y, \ldots \mid t.t \mid t \| t$
- 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: $Act = \{\tau\} \cup \{a!, a? \mid a \in Sync\}$
- **SPAD:** $X \xrightarrow{b} t \;;\; X \cdot Y \xrightarrow{b} t$

Transition Relation)

Basic case:

$$\frac{t_1 \xrightarrow{b} t_2 \in R}{t_1 \overset{b}{\Rightarrow} t_2}$$

Sequential composition: **Prefix rewriting strategy**

$$\frac{t_1 \overset{b}{\Rightarrow} t_2}{t_1 \cdot u \overset{b}{\Rightarrow} t_2 \cdot u} \qquad \frac{t_1 \overset{b}{\Rightarrow} t_2 \ \text{ and } \ u \sim 0}{u \cdot t_1 \overset{b}{\Rightarrow} u \cdot t_2}$$

Transition Relation)

Parallel composition:

$$\frac{t_1 \overset{b}{\Rightarrow} t_2}{u \| t_1 \overset{b}{\Rightarrow} u \| t_2 \text{ and } t_1 \| u \overset{b}{\Rightarrow} t_2 \| u}$$

Synchronisation:

$$\frac{t_1 \overset{a!}{\Rightarrow} t_2 \text{ and } u_1 \overset{a?}{\Rightarrow} u_2}{t_1 \| u_1 \overset{\tau}{\Rightarrow} t_2 \| u_2}$$

Good **Execution:** ***a!*** **matched with** ***a?*** $\Rightarrow$ **only** τ

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: $n \xrightarrow{call(p)} m$ $\quad$ $n \xrightarrow{\tau} e_p.m$

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

Termination: $n \xrightarrow{\tau} 0$

Dynamic creation: $n \xrightarrow{\tau} m_1 \| m_2$

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

$$\mathit{Init} \xrightarrow{?} \mathit{Bad}$$

Init and *Bad*: **Infinite** sets of configurations

(reachability of a control point)

Reachability Problem

$$\mathit{Init} \xrightarrow{?} \mathit{Bad}$$

In our modeling:

Init and *Bad*: **Infinite** sets of terms

$$\underbrace{Good - Executions_{SPAD}(Init, Bad)}_{} = \phi ?$$

$$Executions_{SPAD}(Init, Bad) \cap \tau^{*} = \phi ?$$

Impossible ☹

$$\boxed{A(Executions(Init, Bad)) \cap \tau^{*} = \phi ?}$$

Our Approach

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

Compute over-approximation $A(Executions(Init, Bad))$

$$\boxed{A(Executions(Init, Bad)) \cap \tau^{*} = \phi?}$$

YES

NO

Can we extract a real execution?

YES

NO

Refine approximation

Computing $A(Executions(Init, Bad))$?

• Characterize $Executions(Init, Bad)$ 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)

→ Over-approximation

Prefix k Abstraction Domain

$$L = ababab c^{*}$$

$$\alpha_{3}(L) = aba\,(a + b + c)^{*}$$

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

Refinable abstractions: $\alpha_{1}, \alpha_{2}, \alpha_{3}, \ldots$

$$\alpha_{4}(L) = abab(a + b + c)^{*}$$

Computing $A(Executions(Init, Bad))$?

?

- Characterize $Executions(Init, Bad)$ 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)

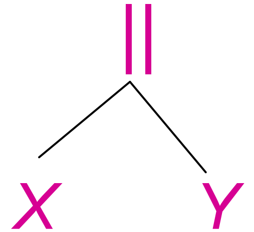
➡ Over-approximation

Characterizing $Executions(Init, Bad)$?

First Problem: **Finitely** represent **infinite** sets of terms

Term = Tree

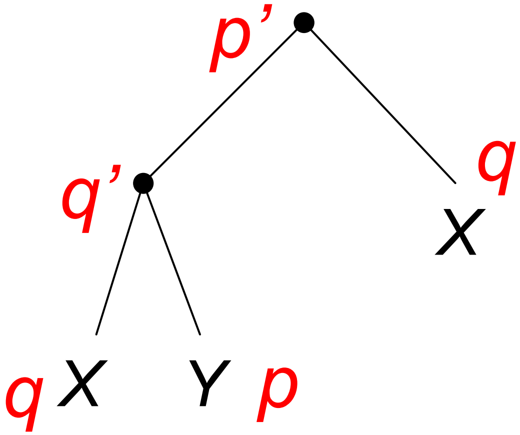
$X \,\|\, Y$



Infinite sets of terms: Tree automata

Tree Automata

- A=(Q,F,δ)
- δ : $X \to q$; $.(q,q') \to q''$; $||(q,q') \to q''$



$X \to q$

$Y \to p$

$\bullet(q,p) \to q'$

$\bullet(q',q) \to p'$

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

Characterizing Executions

$$Executions(L_1, L_2)$$

Theorem :

If L_1 and L_2 compatible then

$$Executions(L_1, L_2) = Executions_{no-equivalence}(L_1, L_2)$$

Characterizing Executions

$$Executions\ (A_1, A_2)$$

$$E(q,q') = Executions(L_q, L_{q'})$$

$$Executions\ (A_1, A_2) = \bigcup_{\substack{q \in F_1 \\ q' \in F_2}} E(q,q')$$ **?**

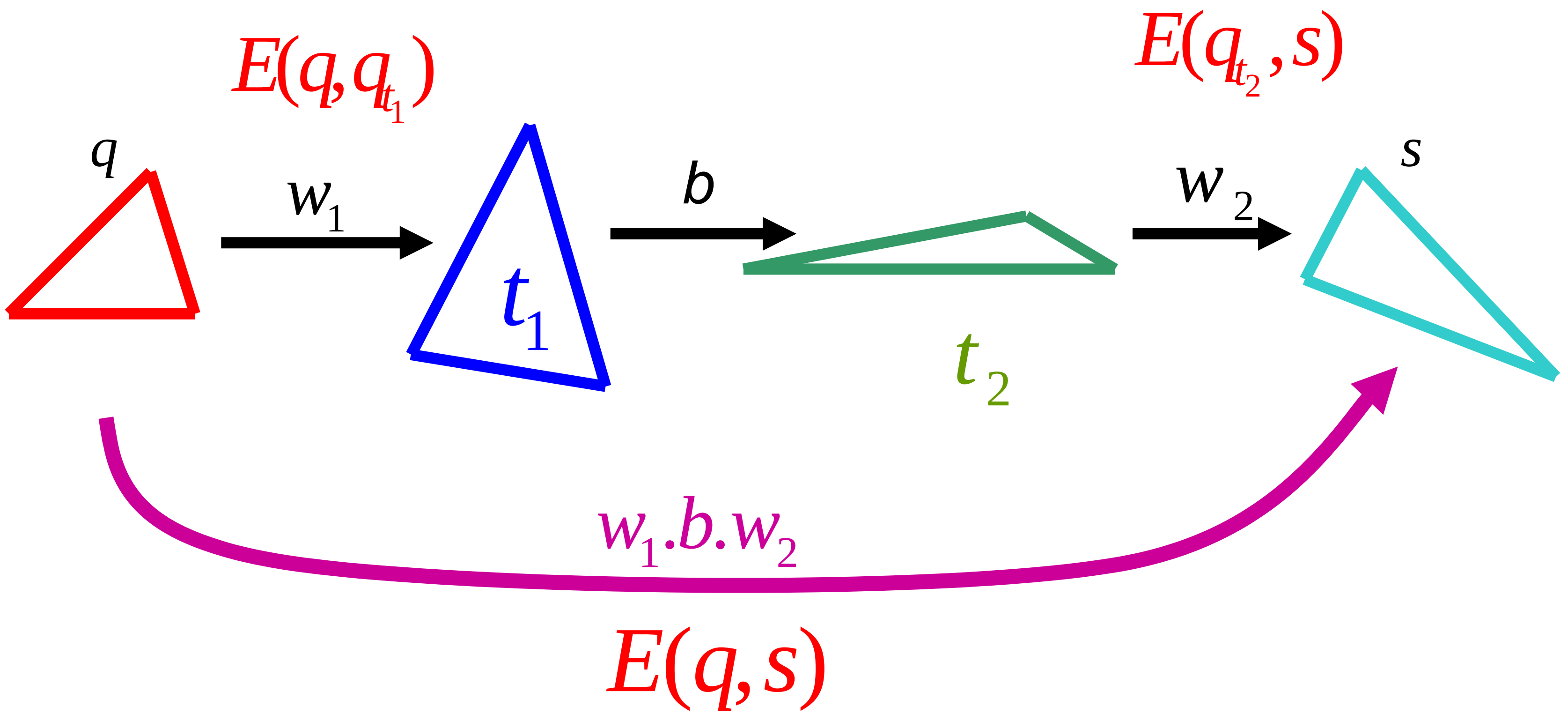
A first constraint

$$L_q \cap L_s \neq \phi \Rightarrow \varepsilon \in E(q,s)$$

Another constraint

$$t_1 \xrightarrow{b} t_2 \in R$$

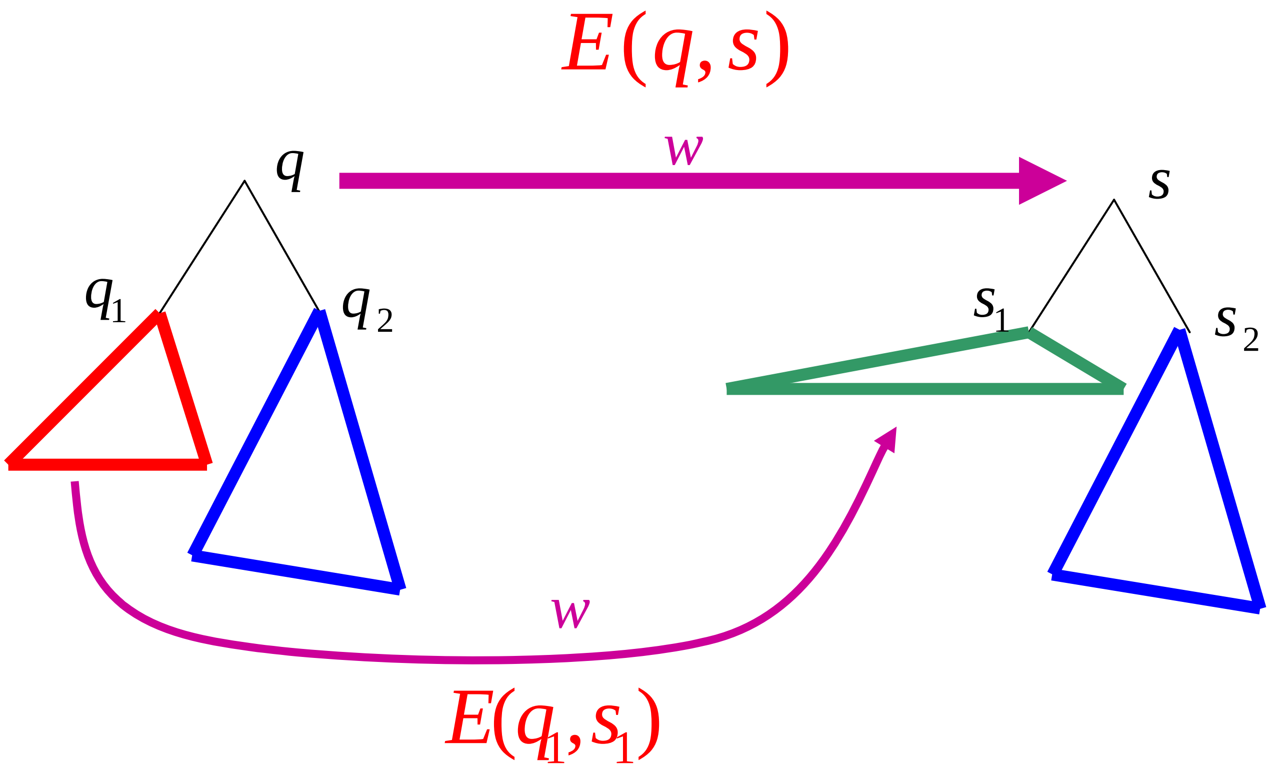
$$E(q, q_{t_1}) \cdot b \cdot E(q_{t_2}, s) \subseteq E(q, s)$$



One more constraint

$\cdot (q_1, q_2) \to q \in A_1$ and $\cdot (s_1, s_2) \to s \in A_2$

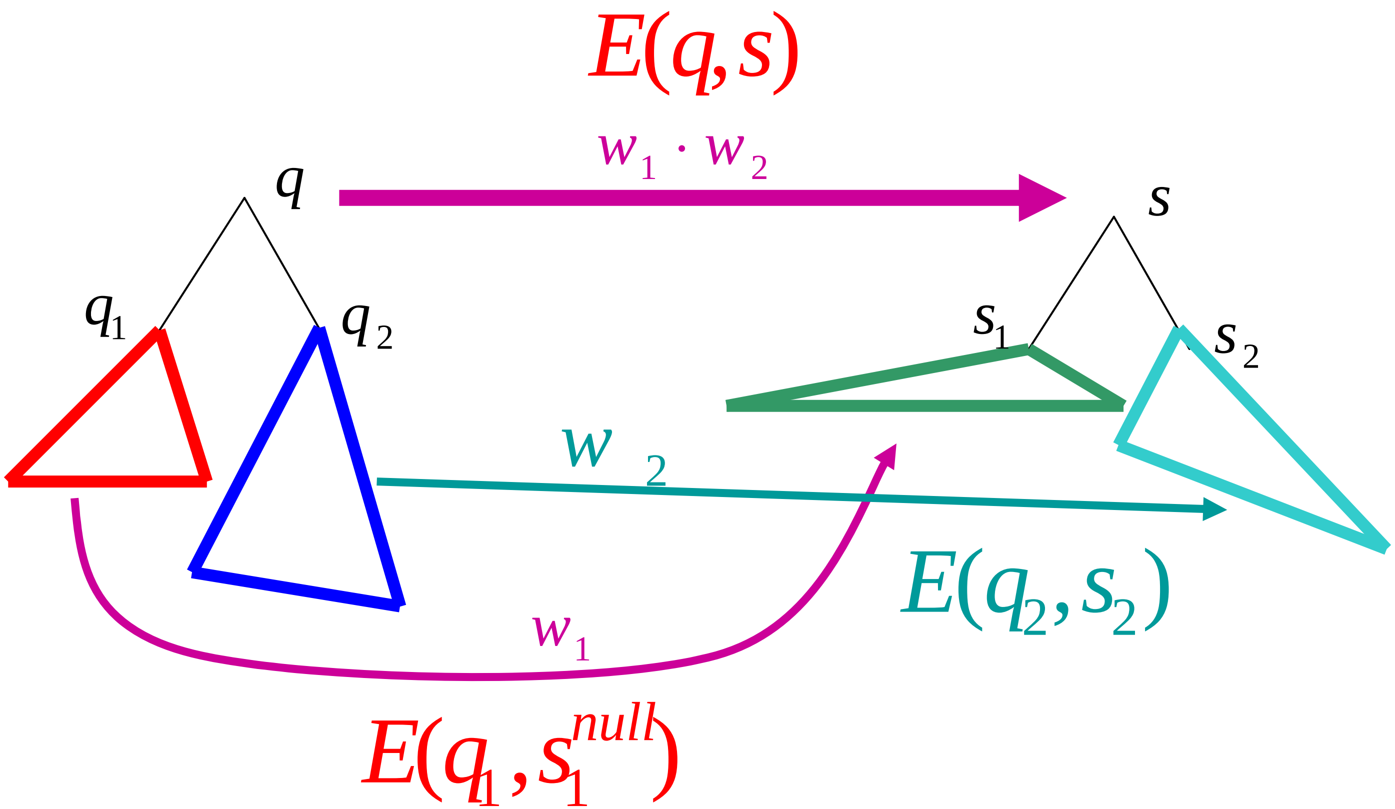
$$L_{q_2} \cap L_{s_2} \neq \phi \Rightarrow E(q_1, s_1) \subseteq E(q, s)$$



A last constraint

$\cdot (q_1, q_2) \to q \in A_1$ and $\cdot (s_1, s_2) \to s \in A_2$

$$E(q_1, s_1^{null}) \cdot E(q_2, s_2) \subseteq E(q, s)$$



4 more constraints …

………

………

………

………

Characterizing Executions

$$Executions\ (A_1, A_2)$$

$$E(q,q') = Executions(L_q, L_{q'})$$

$$Executions\ (A_1, A_2) = \bigcup_{\substack{q \in F_1 \\ q' \in F_2}} E(q,q')$$ ☺

Computing $A(Executions(Init, Bad))$?

- Characterize $Executions(Init, Bad)$ 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)

➡ Over-approximation

Our Approach

$$Executions(Init, Bad) \cap \tau^* = \phi???$$

Compute over-approximation $A(Executions(Init, Bad))$

$$\boxed{A(Executions(Init, Bad)) \cap \tau^* = \phi?}$$

YES

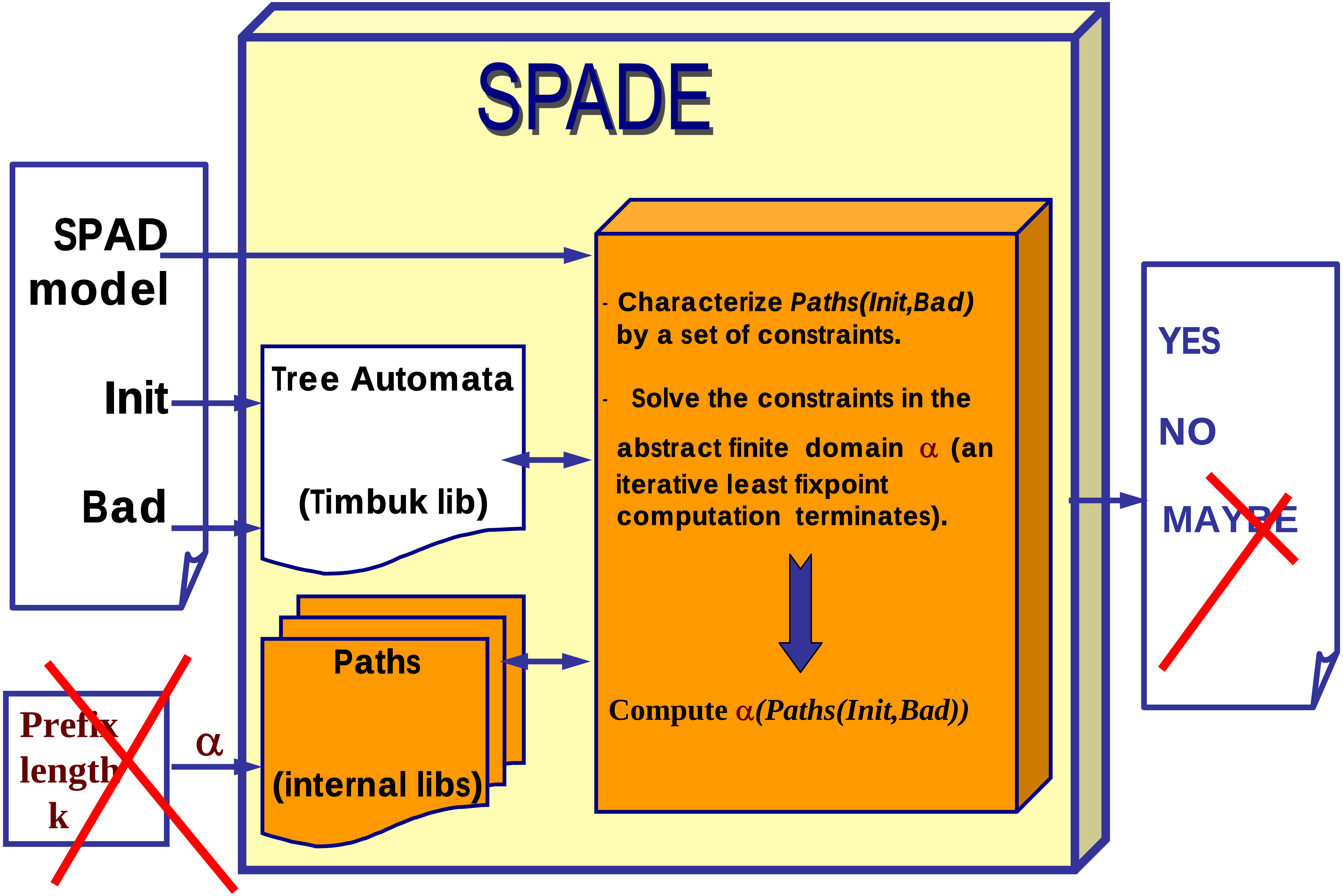
NO

Can we extract a real execution?

YES

NO

Refine approximation


SPADE
SPAD model
Init
Bad
Tree Automata
(Timbuk lib)
Prefix length k
α
Paths
(internal libs)
- Characterize *Paths(Init,Bad)* by a set of constraints.
- Solve the constraints in the abstract finite domain α (an iterative least fixpoint computation terminates).
Compute α(*Paths(Init,Bad)*)
YES
NO
MAYBE

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 progam 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?