On Model-Checking Concurrent Recursive Programs

Tayssir Touili
LIAFA, CNRS & Univ. Paris 7

Sagar Chaki, Ed Clarke
Carnegie Mellon University

Nick Kidd, Tom Reps
University of Wisconsin

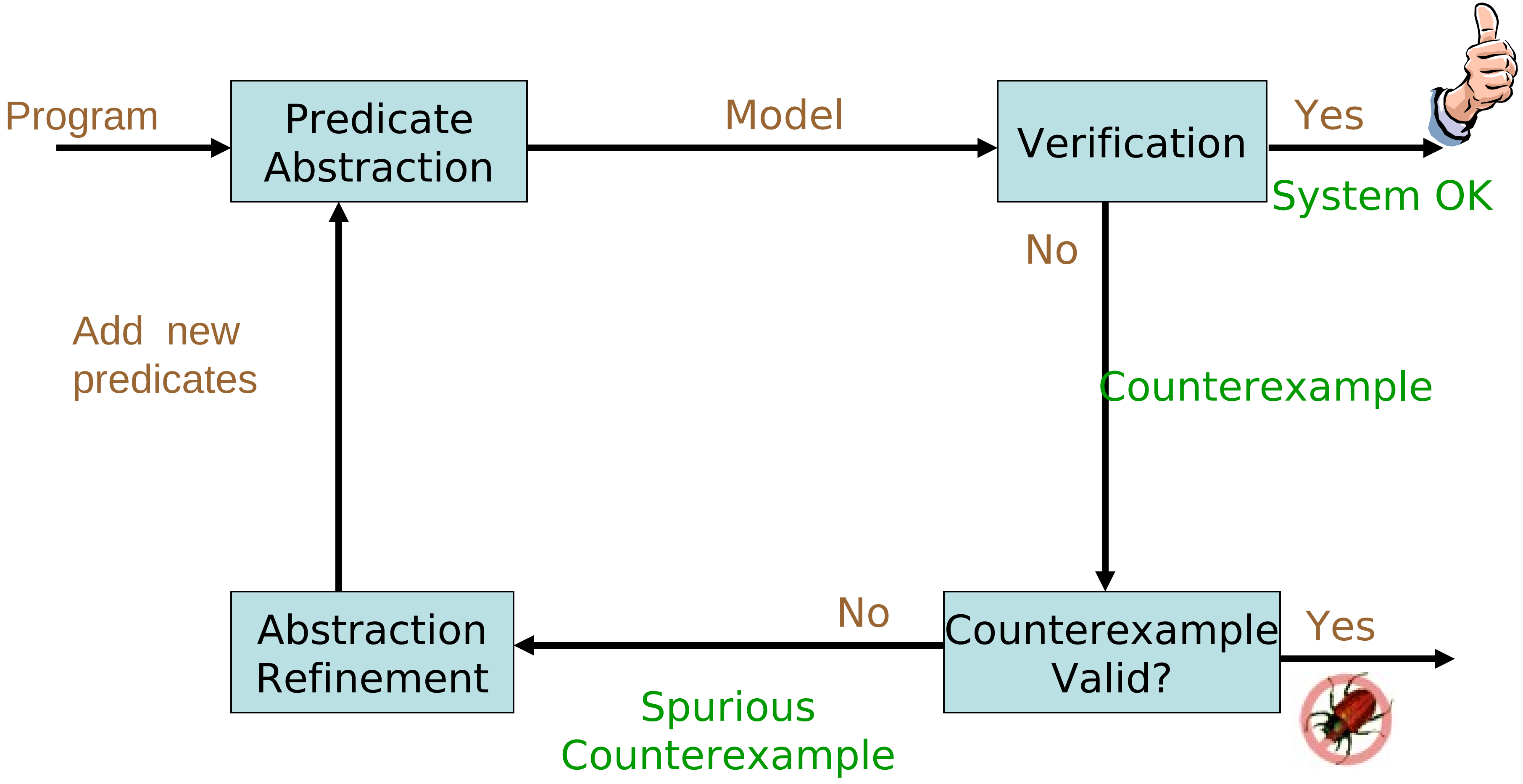
Concurrent Software Model-Checking

- A major challenge in the community
- Various complex features:
  - Data over unbounded (very large) domains
  - Presence of recursive procedure calls
  - Concurrency and synchronisation
  - Dynamism
- Reachability of a control point is undecidable [Ramalingam 2000]
- Any analysis technique is incomplete

Existing Works

- Large data: Predicate abstraction [Graf, Saidi'97]
- *x*:int; (*x*>5) and (*x*<=5)
- Model precision and complexity of the analysis: number of predicates
- Discover a small number of predicates needed to prove the property
- Counter Example Guided Abstraction Refinement (CEGAR)

CEGAR


Program
Predicate Abstraction
Model
Verification
Yes
System OK
No
Counterexample
Add new predicates
Abstraction Refinement
No
Counterexample Valid?
Yes
Spurious Counterexample

Existing Works: CEGAR

- Implemented in several tools:

- SLAM : no concurrency
- MAGIC: no recursion
- BLAST: no recursion+concurrency
- etc

Existing Works (recursion+concurrency)

- Pushdown systems: Sequential recursive programs [Esparza, Knoop'99], [Esparza,Schwoon'01]
- Model Recursion + Represent infinite configurations of recursive programs by regular languages [Bouajjani, Esparza, Maler'97],
- Tool: MOPED

- Compositions of PDSs: Concurrent recursive programs [Bouajjani, Esparza, T. , Qadeer, Rehof,……]

Data assumed to have a small domain

This Work

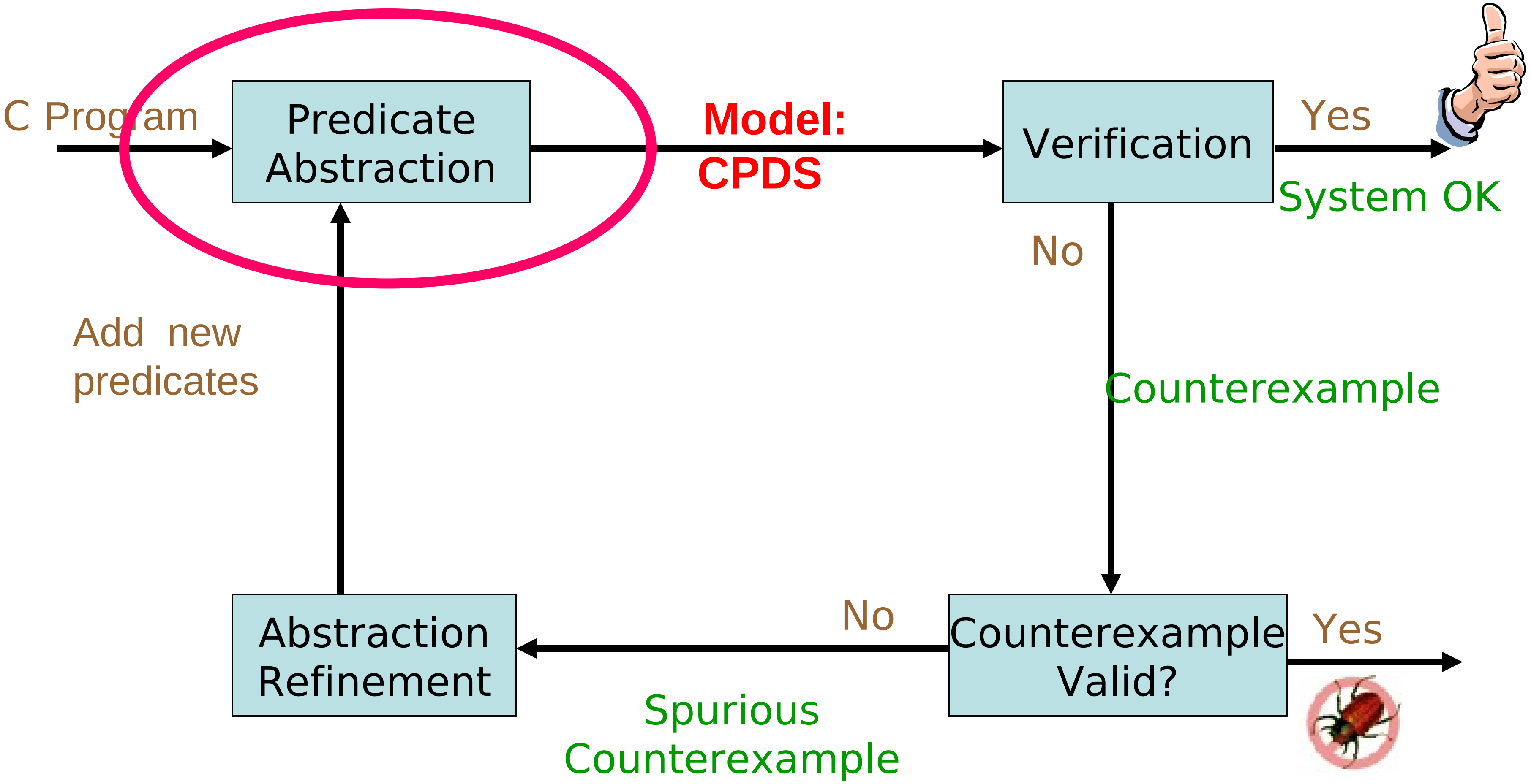
- Handle
  - Large data
  - Recursion
  - Concurrency

Our Approch

- Large Data: CEGAR on predicate abstraction

- Concurrency and recursion: Communicating PDSs

Our Approach



Pushdown System: Definition

Pushdown System : $S = (Q, Act, \Gamma, c_0, \Delta)$:

Q is a finite set of states

Act is a finite set of actions

Γ is a finite stack alphabet

c_0 is the initial configuration $(q, v), q \in Q, v \in \Gamma^*$

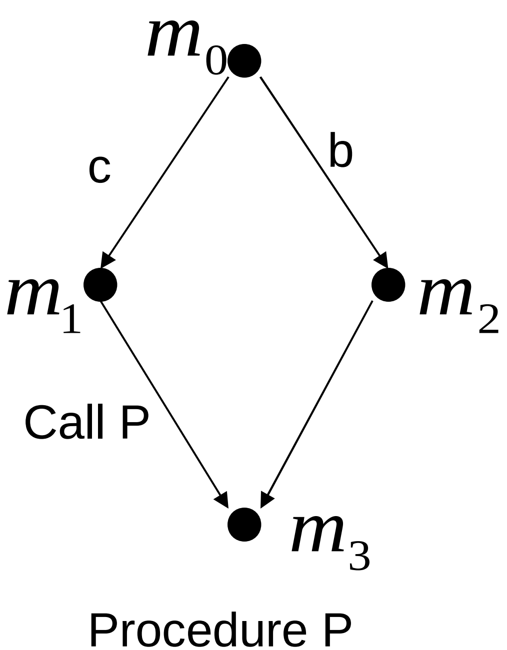
Δ is a finite set of rules of the form

$$(q,\gamma) \xrightarrow{a} (q', w), q, q' \in Q, a \in Act, \gamma \in \Gamma, w \in \Gamma^*$$

Transition relation: $(q, \gamma u) \xrightarrow{a} (q', wu)$

From a Sequential Program to a Pushdown System

[Esparza,Schwoon'01]



$$r_1 : (glob, (m_0, loc)) \xrightarrow{c} (glob', (m_1, loc'))$$

$$r_2 : (glob, (m_0, loc)) \xrightarrow{b} (glob', (m_2, loc'))$$

$$r_3 : (glob, (m_1, loc)) \xrightarrow{\tau} (glob', (m_0, loc')(m_3, loc))$$

$$r_4 : (glob, (m_2, loc)) \xrightarrow{\tau} (glob, (m_3, loc))$$

$$r_5 : (glob, (m_3, loc)) \xrightarrow{\tau} (glob, \varepsilon)$$

Predicates?

- States: Global variables
- Symbols of the stack: Local variables+control points

Predicates?

- Subset *C* of the conditions of the program

- Close it by computing weakest preconditions w.r.t. statements of the program

$$s : x := e$$

$$WP_s(p) : p[x \leftarrow e]$$

Predicates?

- Subset *C* of the conditions of the program
- Close it by computing weakest preconditions w.r.t. statements of the program

Initially *C* empty

Conditions are added using CEGAR

Compute the predicates?

- Associate to each control point *n* a set of predicates $P(n)=P(n)_{loc}+P(n)_{glob}$

- *P(n) :* set of predicates needed at point *n*

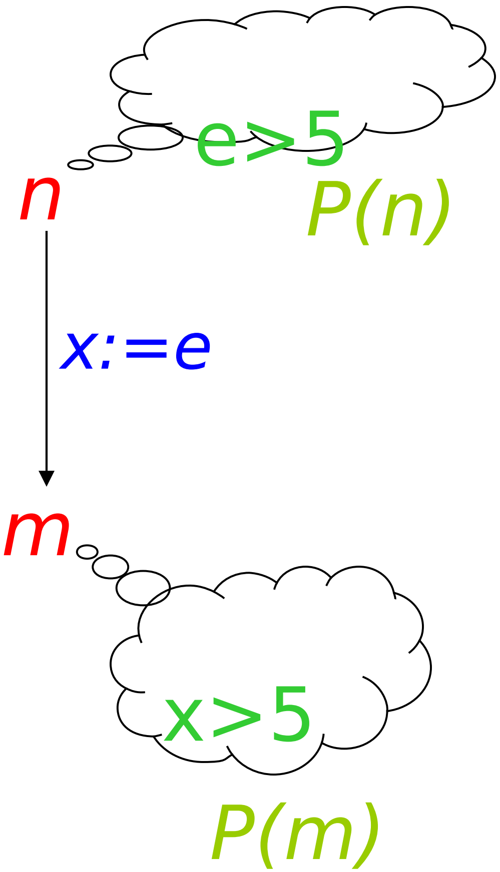
- Initially, $P(n) = \emptyset$ for all *n*

- *P(n) :* updated by computing weakest preconditions

Compute *P(n)*

$s: n \rightarrow m$

- s: assignment; add $WP_s(P(m))$ to $P(n)$



Compute *P(n)*

s: n → m

s: goto or synchronisation statement:

add *P(m)* to *P(n)*

s: (if *c* then) add *P(m)* to *P(n)*

c in *C* : add *c* to *P(n)*

s: call to a procedure *q*

add $P(m)_{loc}$ and $P(init\text{-}q)_{glob}$ to *P(n)*

How to compute the PushDown System?

The PDS rules

$s\colon n \to m$: goto

$(glob,(n,loc)) \rightarrow (glob',(m,loc'))$

$loc \in P(n)_{loc}$ $loc' \in P(m)_{loc}$

$glob \in P(n)_{glob}$ $glob' \in P(m)_{glob}$

$(loc \wedge loc')$ satisfiable

$(glob \wedge glob')$ satisfiable

} **Undecidable** for first order formulas over the integers

SIMPLIFY: sound theorem prover that answers true, false, or unknown

The PDS rules

$s: n \rightarrow m$: assignment

$(glob,(n,loc)) \rightarrow (glob',(m,loc'))$

$loc \in P(n)_{loc}$ $loc' \in P(m)_{loc}$

$glob \in P(n)_{glob}$ $glob' \in P(m)_{glob}$

$(loc \wedge WP_s(loc'))$ satisfiable

$(glob \wedge WP_s(glob'))$ satisfiable

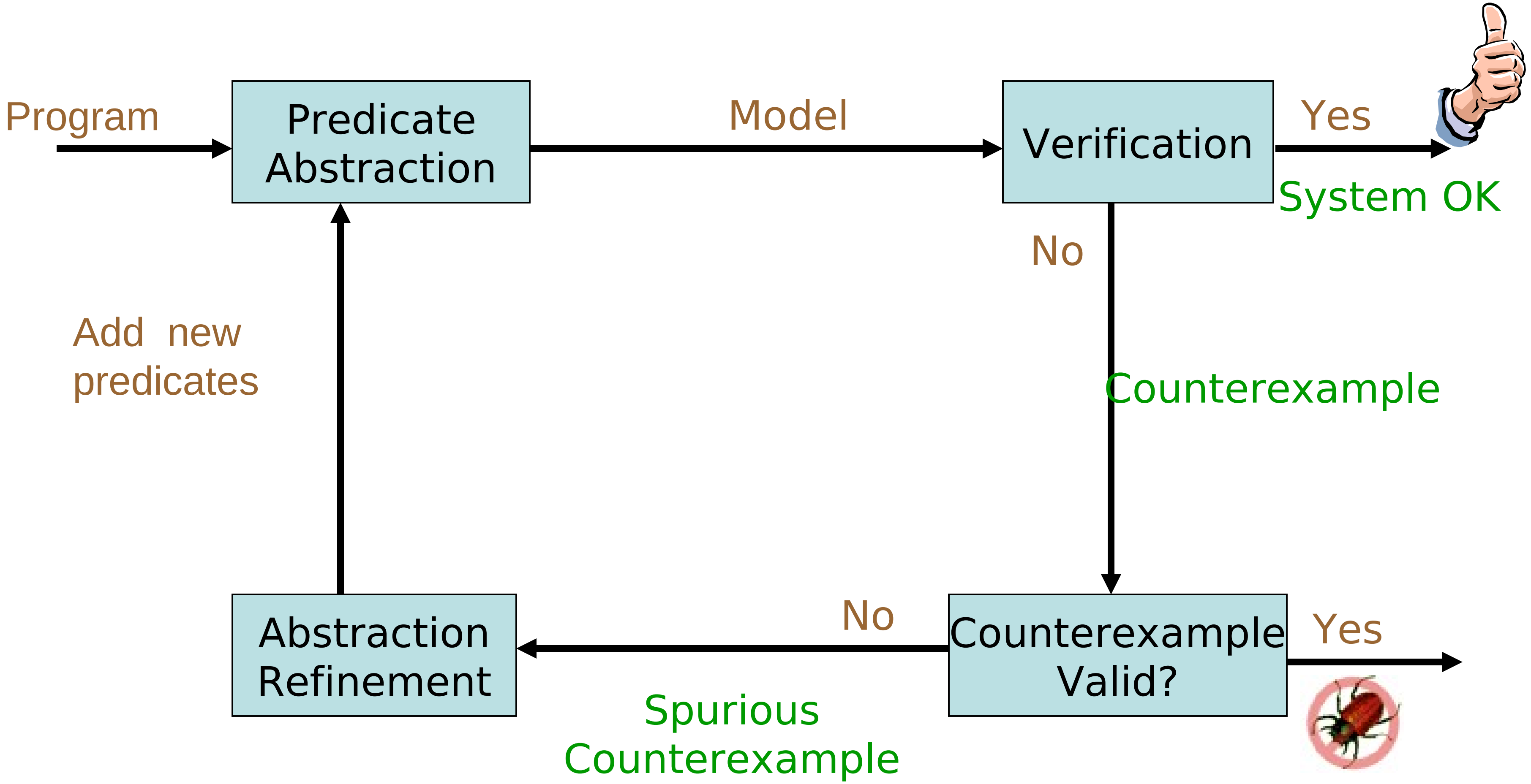
Predicates?

- Subset *C* of the conditions of the program
- Close it by computing weakest preconditions w.r.t. statements of the program

Initially *C* empty

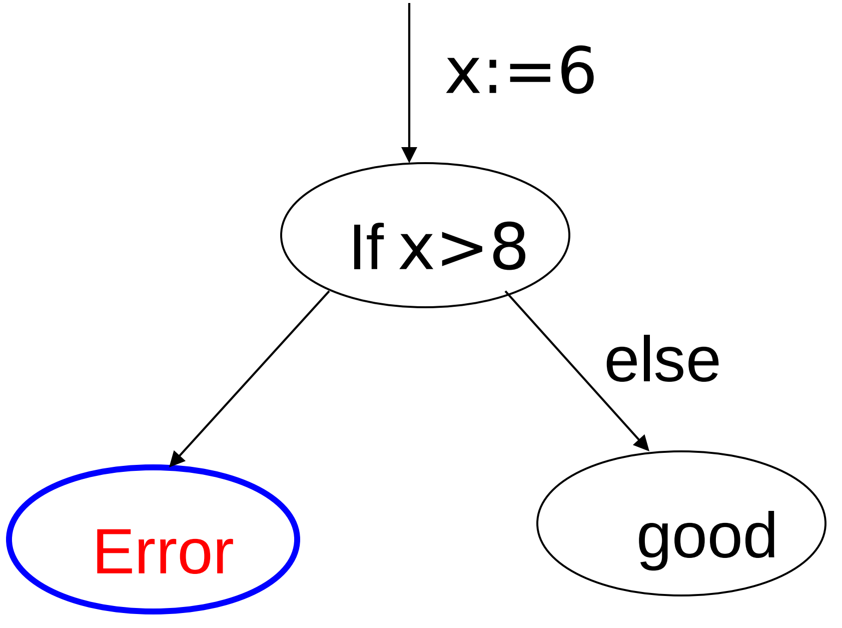
Conditions are added using CEGAR

CEGAR



Program
Predicate Abstraction
Model
Verification
Yes
System OK
No
Counterexample
Add new predicates
Abstraction Refinement
No
Counterexample Valid?
Yes
Spurious Counterexample

Example

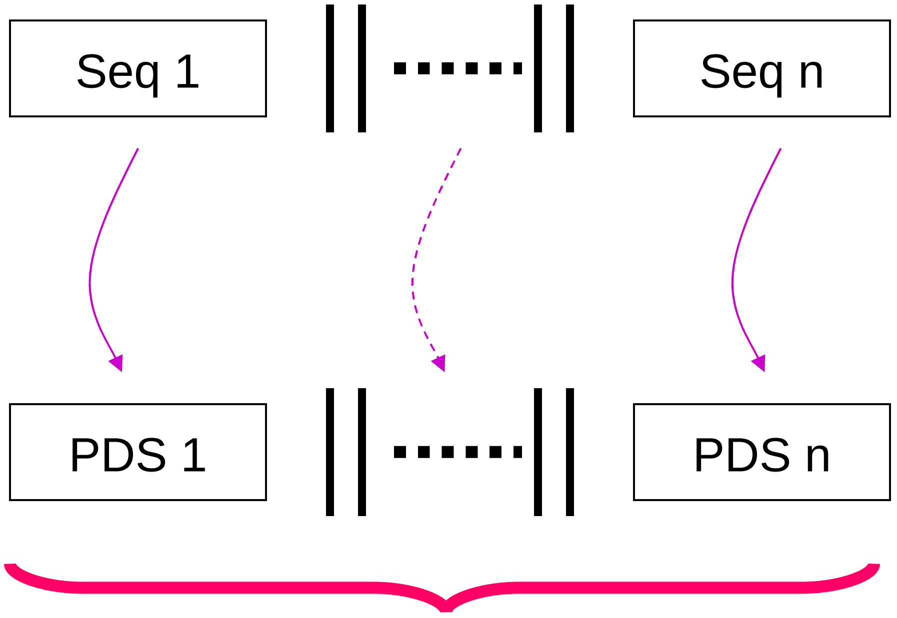


C is empty: Error reachable

(x>8) in *C* : Error non reachable

What about Concurrency?

n sequential components running in parallel, communicating via rendez-vous through blocking synchronizing actions

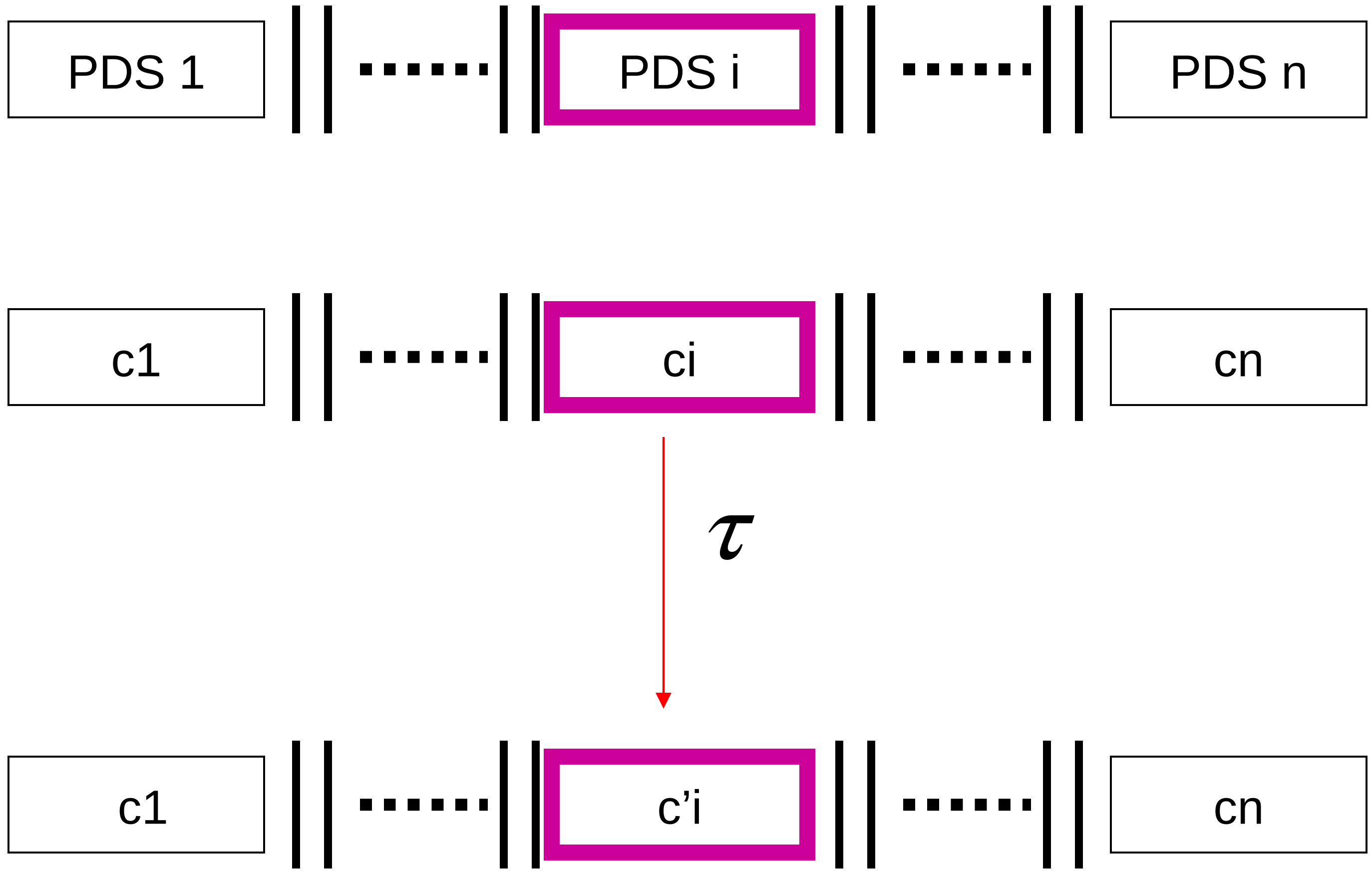


Communicating Pushdown System (CPDS)

[Bouajjani,Esparza,Touili'03]

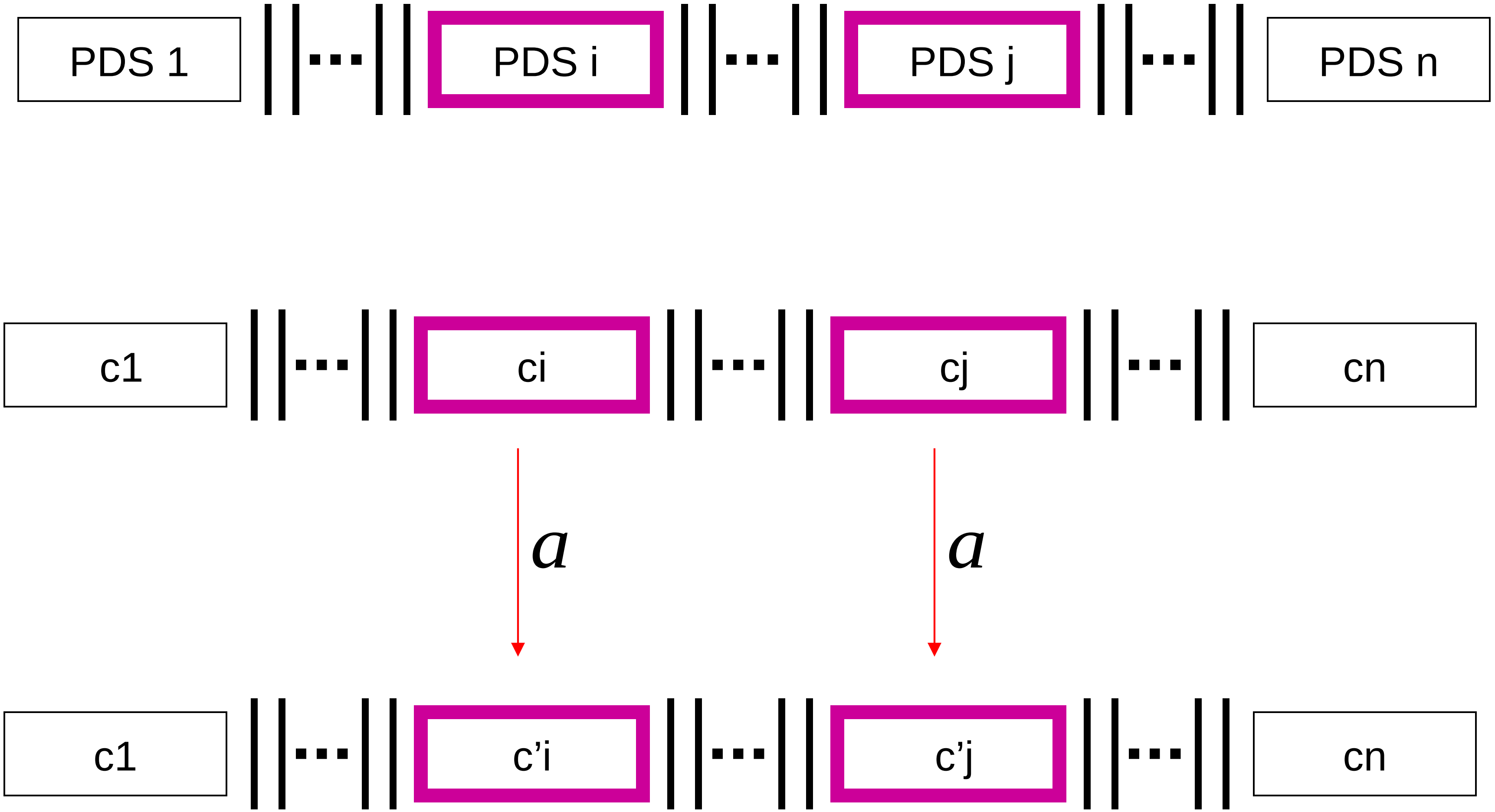
Communicating Pushdown Systems

Internal actions τ



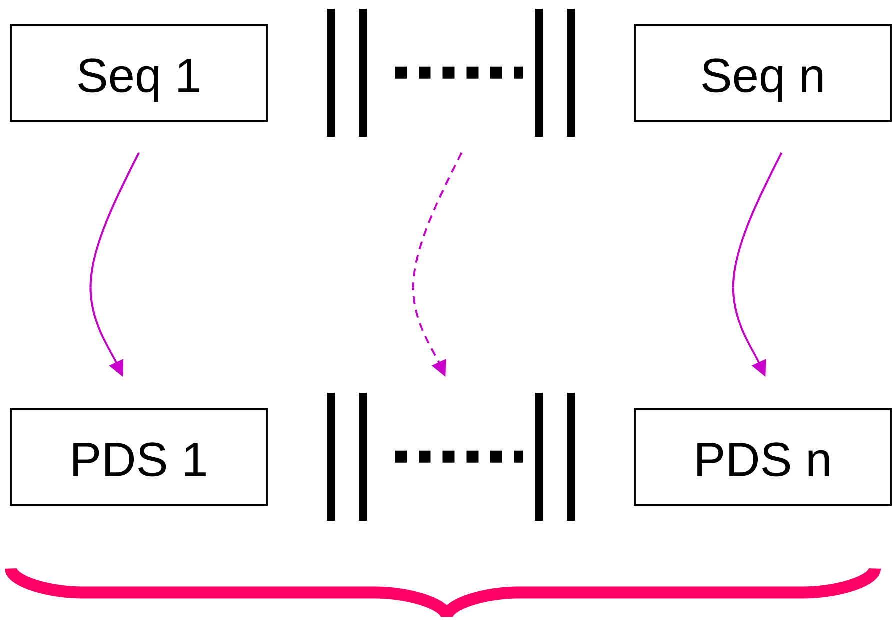
Communicating Pushdown Systems

Synchronizing actions



What about Concurrency?

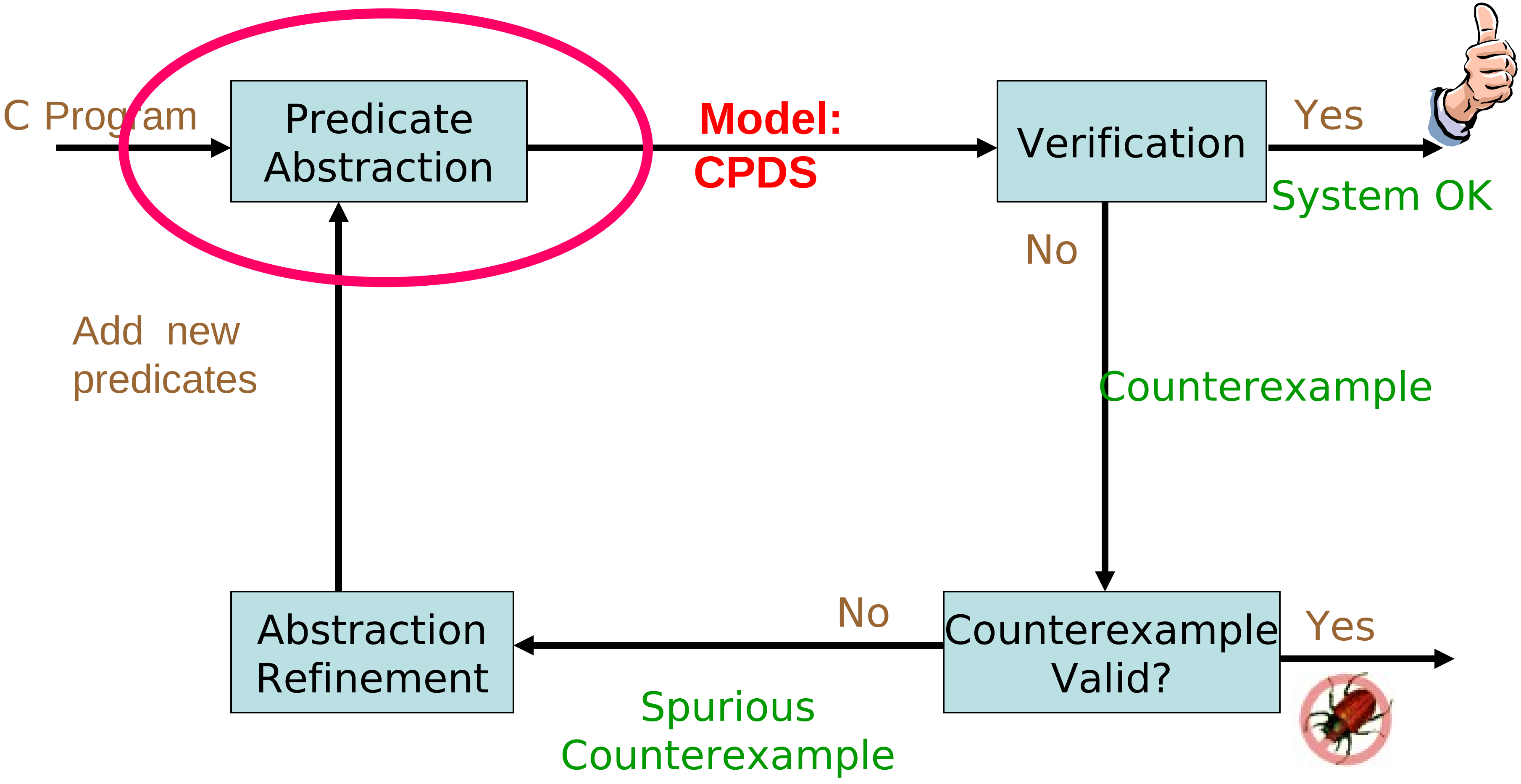
n sequential components running in parallel, communicating via rendez-vous through synchronizing actions



Communicating Pushdown System (CPDS)

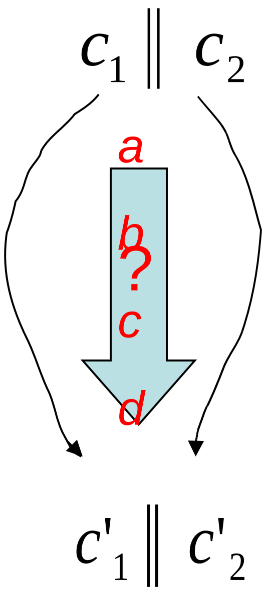
[Bouajjani,Esparza,Touili'03]

Our Approach



Reachability Analysis of CPDSs

PDS 1 || PDS 2



$$L_1 \cap L_2 \neq \phi$$

L_i : paths of PDS_i leading from c_i to c_i'

: Context - free language

Undecidable!

⬇

No exact solution

Solution: A CEGAR Scheme

[Clarke,Chaki,Kidd,Reps,Touili'06]

$$L_1 \cap L_2 = \phi\,??$$

Compute over-approximations $A_1 \supseteq L_1$ and $A_2 \supseteq L_2$

$$I = A_1 \cap A_2 = \phi?$$

YES

NO

Can we extract a real path?

YES

NO

Refine approximation

Compute refinable over-approximations

Computing refinable over-approximations

$$\alpha_k(L) = prefix_k(L) \cdot \Sigma^*$$

Example: $L = ababab c^*$

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

$$\alpha_4(L) = abab(a+b+c)^*$$

Refinable abstractions: $\alpha_1, \alpha_2, \alpha_3, \ldots$

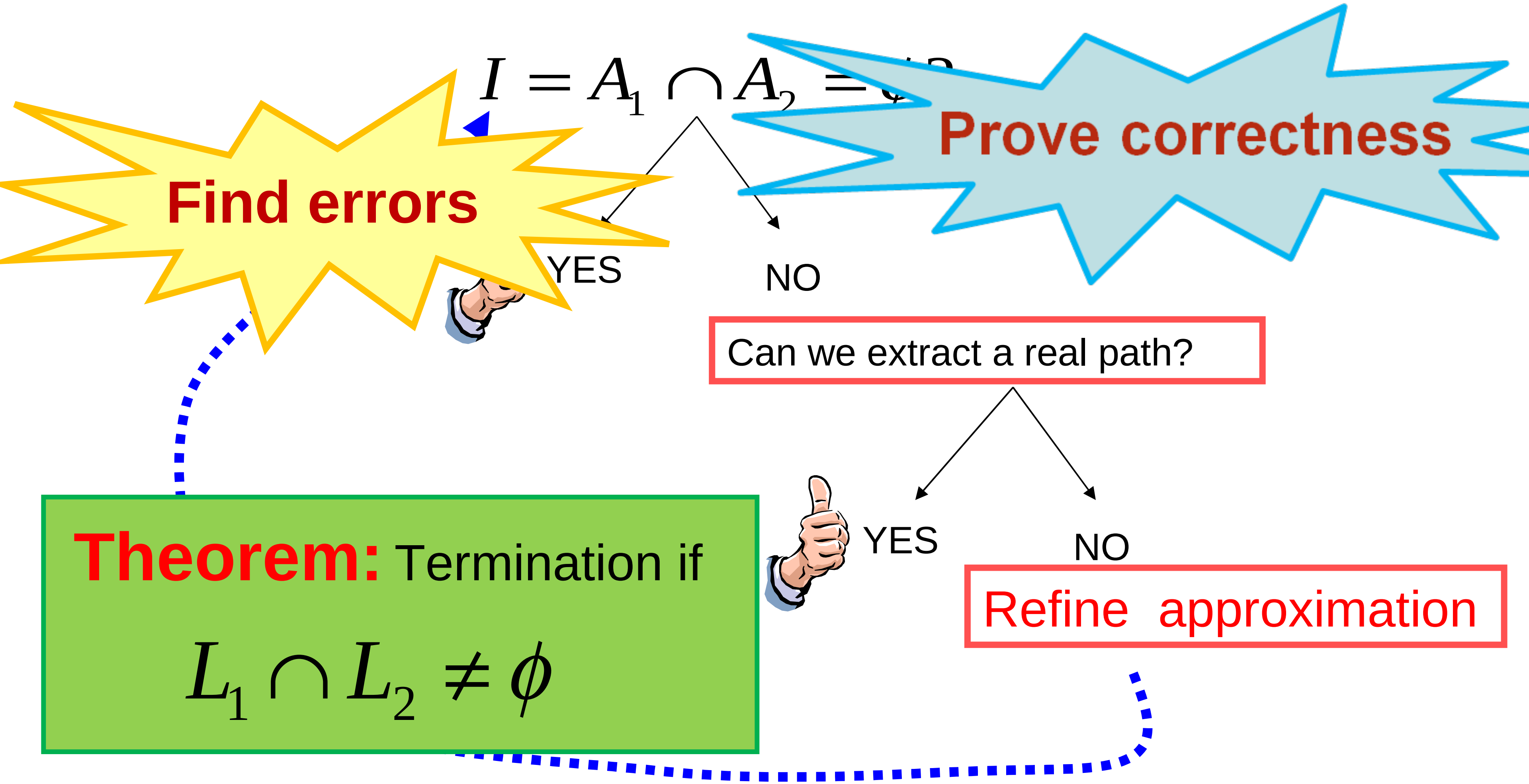
Theorem: **[Bouajjani,Esparza,Touili'03]**

L : context free language $\Rightarrow \alpha_k(L)$ can be effectively computed

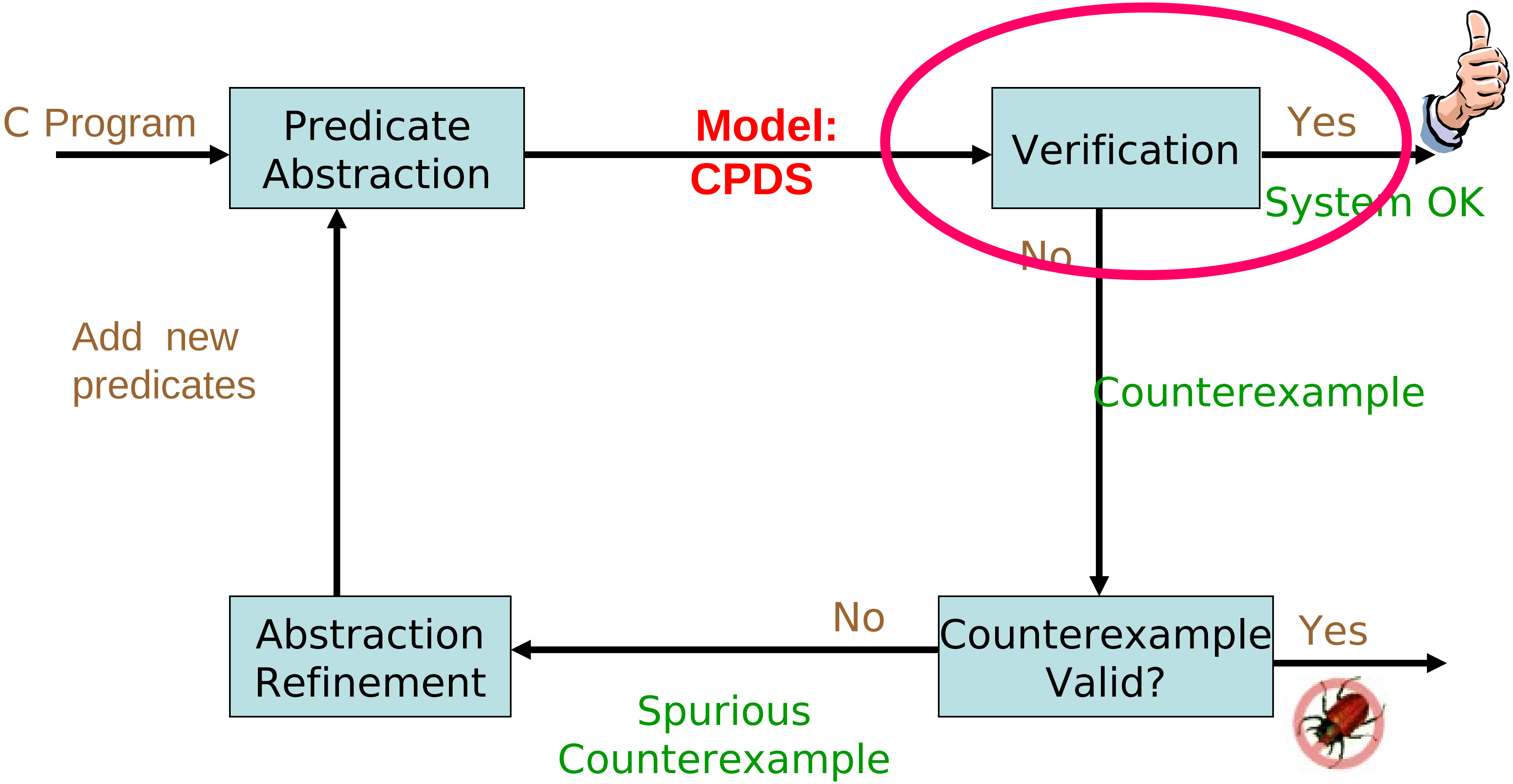
Solution: A CEGAR Scheme

$$L_1 \cap L_2 = \phi \,??$$

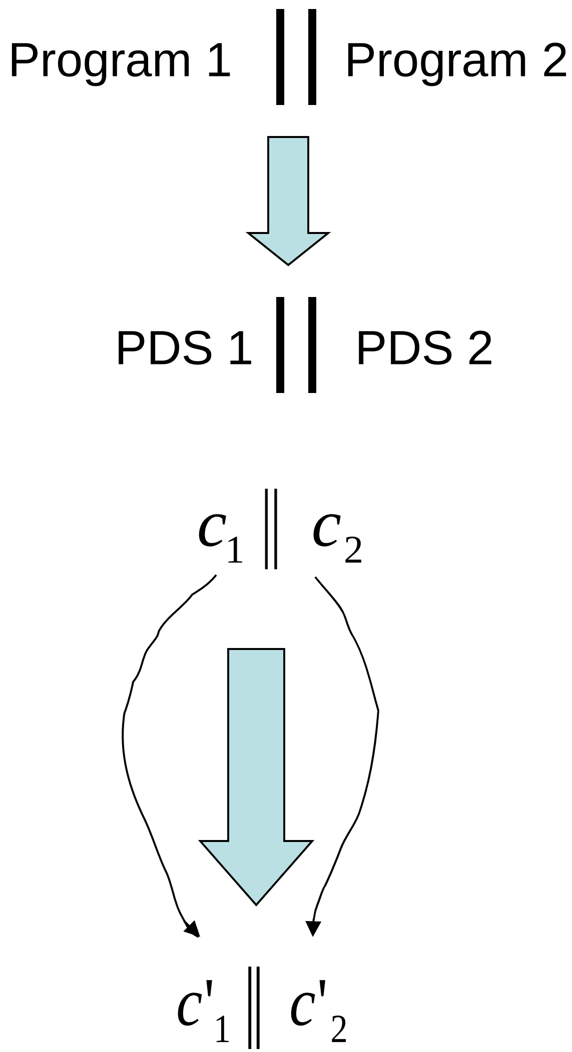
Compute over-approximations $A_1 \supseteq L_1$ and $A_2 \supseteq L_2$



Our Approach



CounterExample Validation



$r_1 \ldots r_n$ in PDS1

$r'_1 \ldots r'_m$ in PDS2

$s_1 \ldots s_n$ in Program1?

$s'_1 \ldots s'_m$ in Program2?

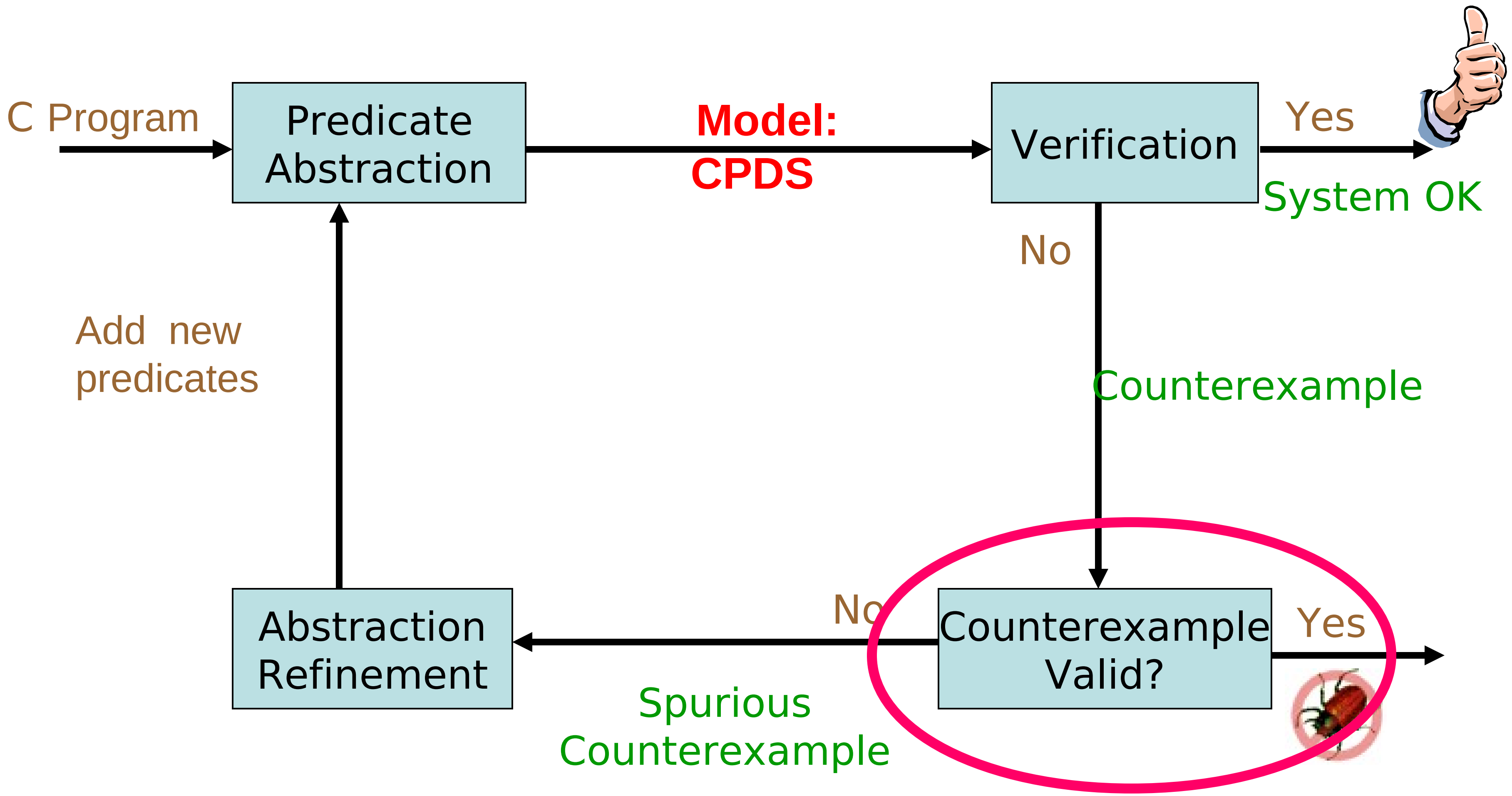
CounterExample Validation

$s_1 \ldots s_n$ in Program1?

- Initially, $\varphi = glob_0 \wedge loc_0$
- For $i=1$ to n do:
- If s_i assignment, compute the strongest postcondition of φ w.r.t. s_i

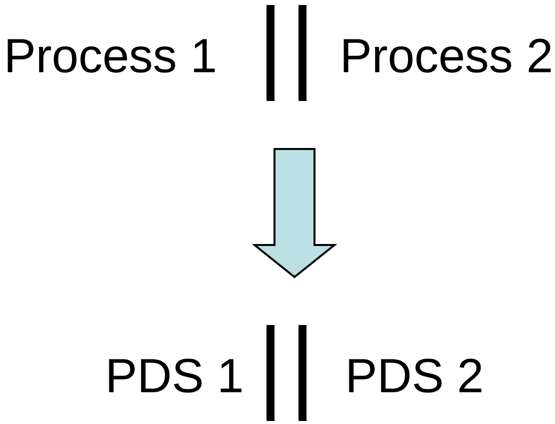
- s_i : $x:=x+5;$ $\varphi : 1<x<4;$ $\varphi' : 6<x<9$
- If s_i : *if* statement with condition $c;$ $\varphi' : \varphi \wedge c$
- If φ satisfiable, the counterexample is valid

Our Approach



Abstraction Refinement

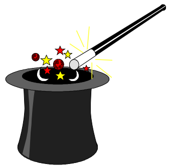
$s_1 \ldots s_n$ not in Program1



Refine PDS1

Add new conditions in *C*

Implementation

MAGIC 

- Modular Analysis of proGrams In C
  http://www.cs.cmu.edu/~chaki/magic

- A Model checker for Concurrent C-programs developed at CMU

- Counter-Example Guided Abstraction Refinement (CEGAR)

Experiments:

Windows NT Bluetooth driver

- Discovery of a new unknown bug in a corrected version (20 seconds)

- Re-discovery of a bug in an old version (5 seconds)

Experiments:

Concurrent Insertions in Binary trees

- Find a bug in an uncorrect version

#processes	time(s)
2	0.8
3	0.8
4	0.8
5	1.1
6	2.7
7	12.9

Experiments
Non-recursive Programs

- Our technique behaves better than inlining

OpenSSL Server1	16.2	2.82
OpenSSl Server2	16.1	3.83
OpenSSL Server3	14.0	19.2
OpenSSL Server4	14.2	2.76
OpenSSl Server5	14.0	3.02
OpenSSL Server6	14.0	2.93
OpenSSL Server7	15.0	3.34
2MicroC	578.0	1.324
3MicroC	*	2.144
	INLINING	CPDS

Conclusion

- Techniques+tool for the verification of concurrent recursive programs with unbounded data
- CEGAR in 2 levels: Abstraction level and the verification level
- Technique compositional: scalable to large programs
- Encouraging experimental results (discovery of an unknown bug in Windows NT driver+ better execution times)

Questions?