Automata Models of Software

Javier Esparza
Technische Universität München

Joint work with Pierre Ganty

Plan

First lecture: Introduction

- From program reachability to non-disjointness
- Complexity analysis

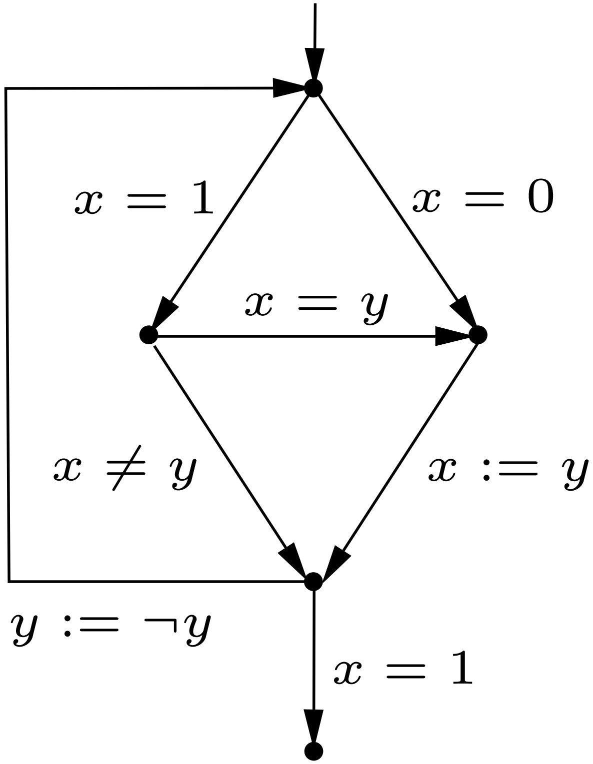
Second lecture: A recent result by E. and Ganty, POPL 2011

While-programs

```
while (ComOK && !EndOfRecord)
  { if (CancelStart) ComOK = false;
    aStr = TempList->Strings[LineNumber];
    PBar1->Position = (LN*100) / NumberOfLines;
    if (aStr.Length() != 0)
    { Data = aStr.c_str();
      if (Data[0] == ':')
      { if ((Data [7] == '0') && (Data [8] == '0'))
        { if (!Communication (WRITE, Data)) ComOK = false;}
          else { if ((Data [7] == '0') && (Data [8] == '1'))
                 EndOfRecord = true;}
        else {MB("Error!", NULL, MB_OK);}
      else {MB(PChar("Error: Empty line"), NULL, MB_OK);}}
```

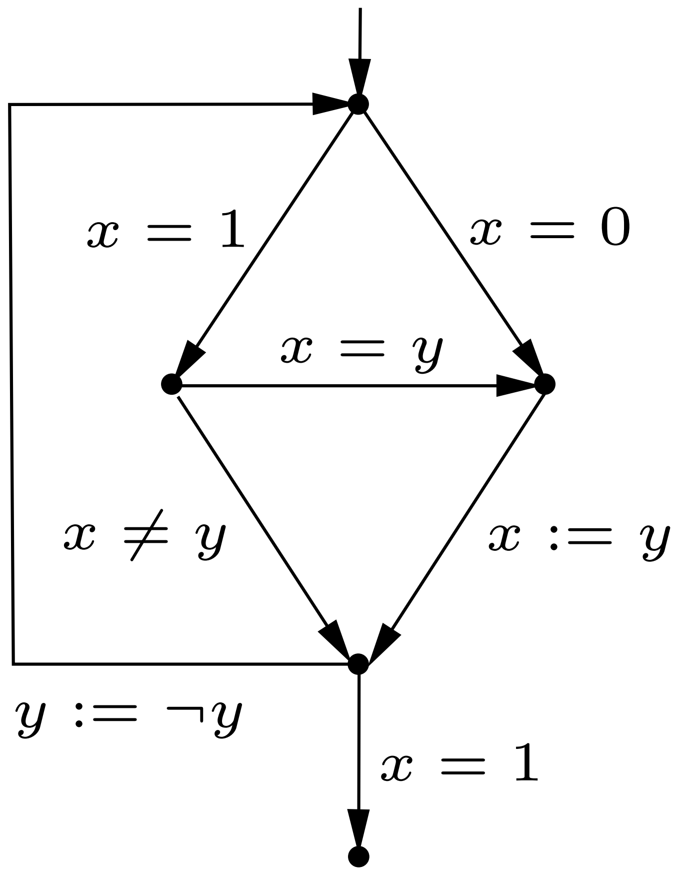
Boolean while-programs

Abstract-check-refine approach to data:



In the following: program $\Rightarrow$ boolean or finite-range program

Automata model of while-programs



Tuple A_x, A_y, A_P of NFAs

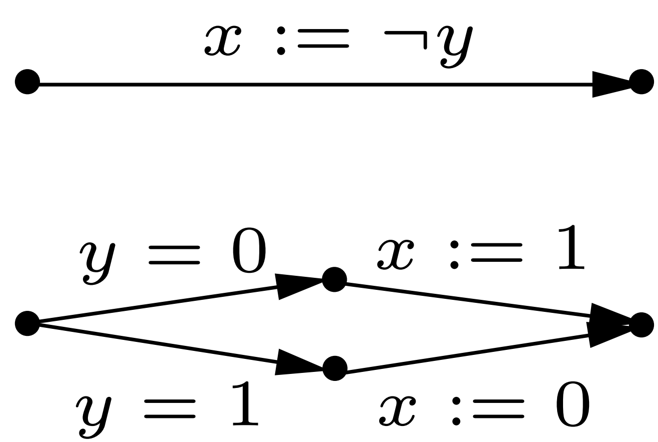
Program point reachable iff

$L(A_x) \cap L(A_y) \cap L(A_P) \neq \emptyset$

Program reachability $\Rightarrow$ non-disjointness

Preprocessing:

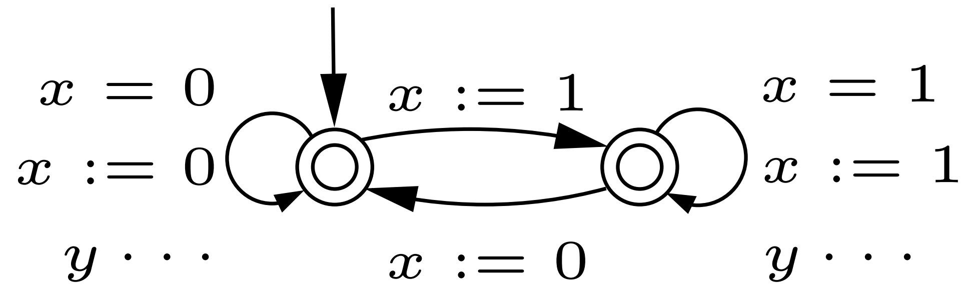
- Construct safety-equivalent program with instructions of the form $x = b$ or $x := b$.

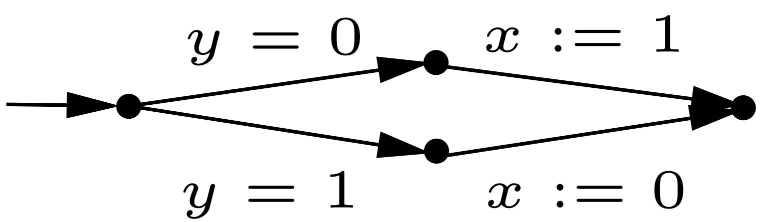


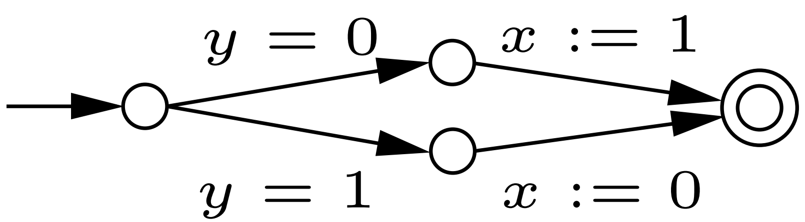
Program reachability $\Rightarrow$ non-disjointness

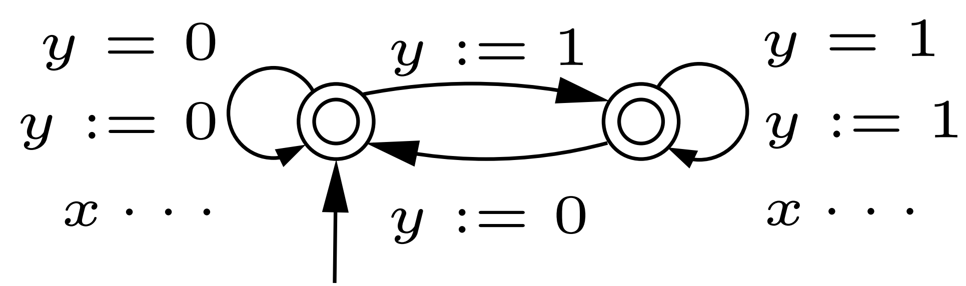
- One NFA A_x with two states for each boolean variable x
- One NFA A_P with the program locations as states
- Alphabet: normalized program instructions, i.e., alphabet letters $x = b$, $x := b$ for every variable x and $b \in \{0, 1\}$
- Transitions correspond to the meaning of the alphabet letters: E.g., there is a transition labeled by `x=0`
  - from state 0 of the NFA for x to itself, and
  - from the control state before an instruction $x = 0$ to the control state after it.
- Additional self-loops on variable NFAs to model idleness:
  - E.g., self-loops labeled by `y=0` on every state of the NFA for variable x.

Program reachability ⇒ non-disjointness









Program reachability $\Rightarrow$ non-disjointness

Theorem: The reachability problem for while-programs is reducible to non-disjointness problem for NFAs, i.e., to the problem

Given:	NFAs $A_1, \dots, A_n$
Decide:	Is $L(A_1) \cap \ldots \cap L(A_n) \neq \emptyset$?

Program reachability $\Rightarrow$ non-disjointness

Theorem: The reachability problem for while-programs is reducible to non-disjointness problem for NFAs, i.e., to the problem

Given: NFAs $A_1, \ldots, A_n$

Decide: Is $L(A_1) \cap \ldots \cap L(A_n) \neq \emptyset$?

Theorem: The non-disjointness problem for NFAs is

(a) P-complete

(b) NP-complete

(c) PSPACE-complete

(d) EXPTIME-complete

Bounded model checking

Examine only computations containing at most k reads/writes of variables.

Bounded model checking

Examine only computations containing at most k reads/writes of variables.

Theorem: The k-non-disjointness problem for NFAs, i.e., the problem

Given: NFAs $A_1, \ldots, A_n$, bound $k \geq 0$

Decide: Is $L(A_1) \cap \ldots \cap L(A_n) \cap \Sigma^{\leq k} = \emptyset$?

is (a) P-complete

(b) NP-complete

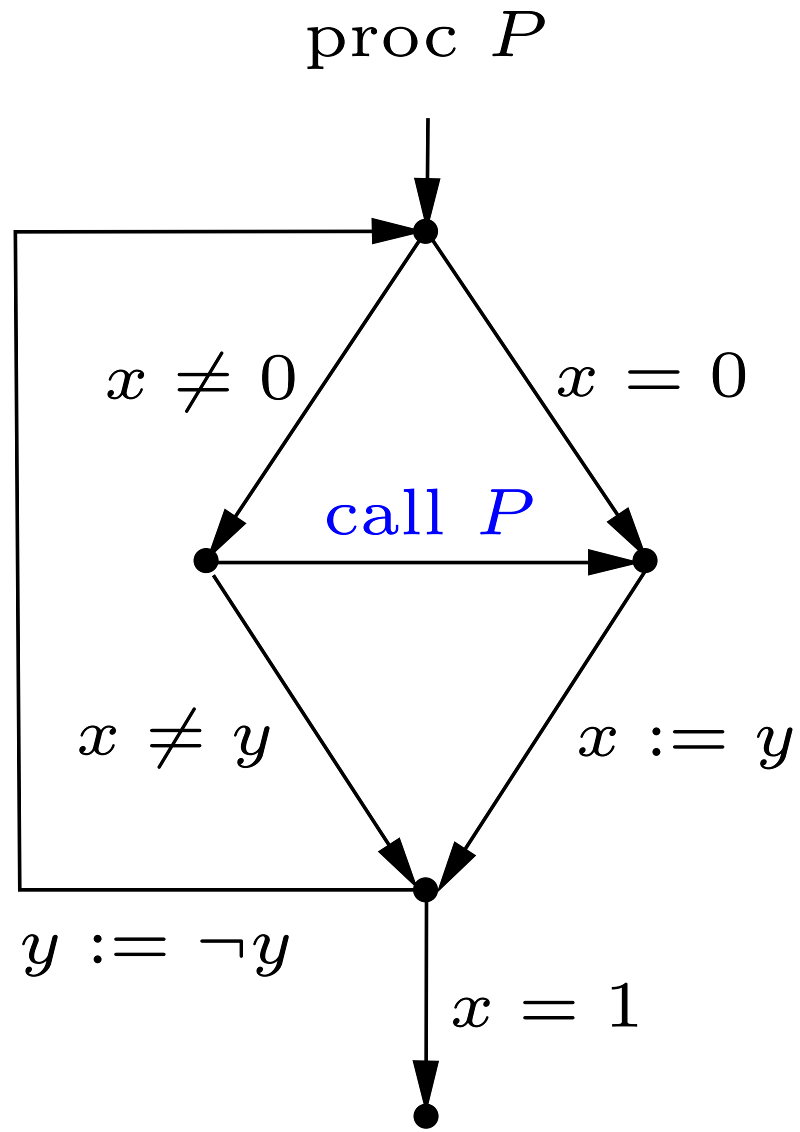
(c) PSPACE-complete

(d) EXPTIME-complete

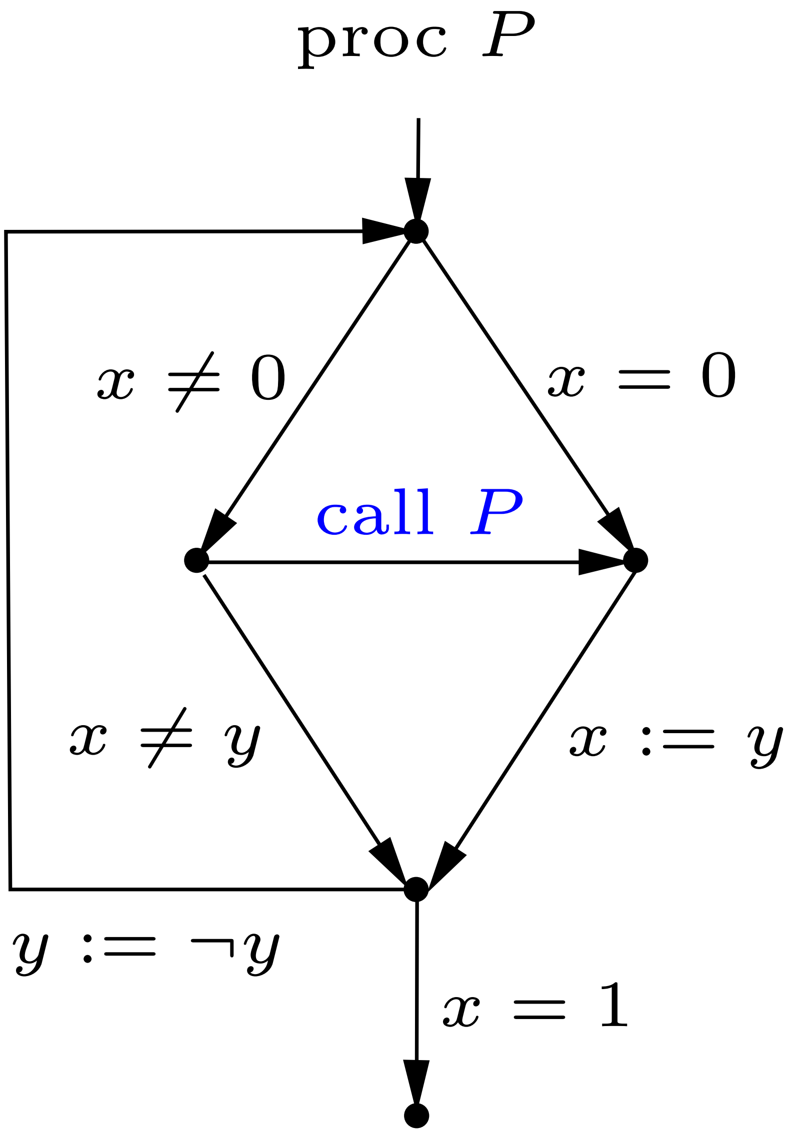
Impact of control structures

- Procedures
- Multithreading
- Parametrization (particular case of process creation)
- Procedures + multithreading

Procedural programs



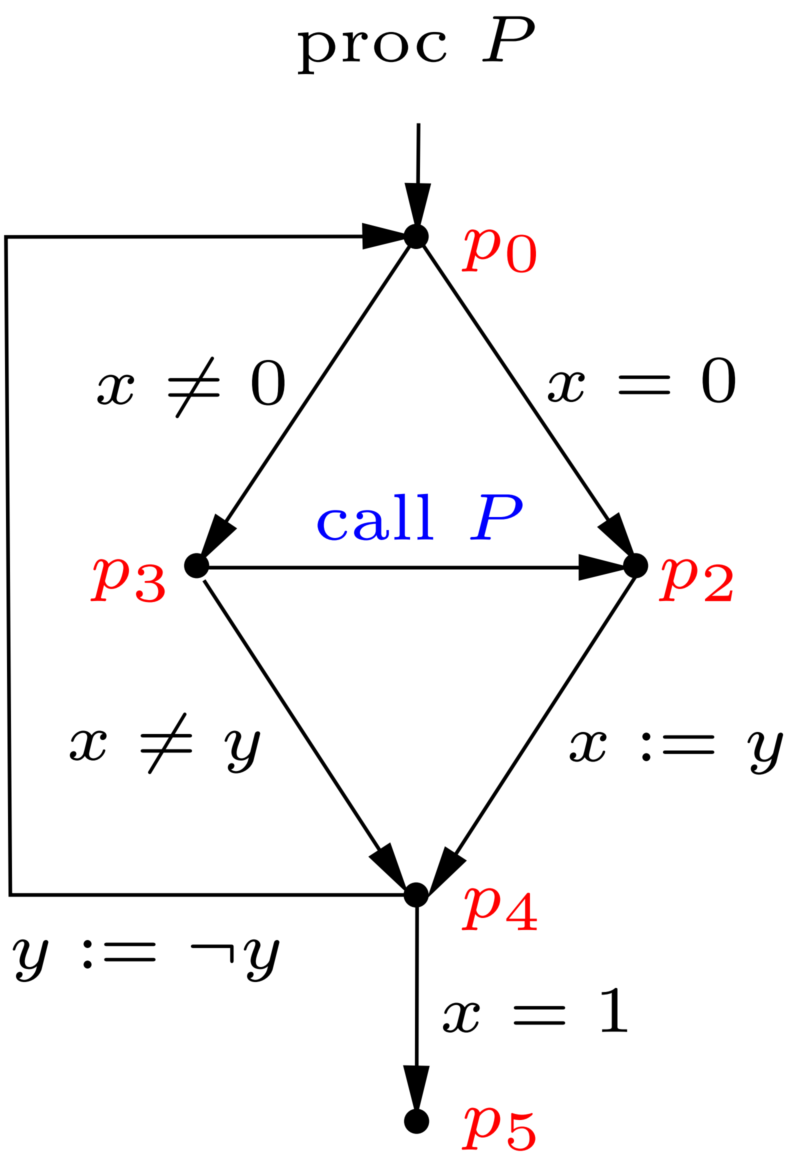
Automata model



One NFA for each boolean variable

A PDA (pushdown automaton) to model control

Automata model



$$q\ p_0 \xrightarrow{x=0} q\ p_2$$

$$q\ p_3 \xrightarrow{call P} q\ p_0\ p_2$$

Program reachability $\longrightarrow$ non-disjointness

Theorem: The reachability problem for boolean procedural programs is reducible to the non-disjointness problem for NFA$^n \times$ PDA, i.e., to the problem

Given: NFAs $A_1, \ldots, A_n$, PDA P

Decide: Is $L(A_1) \cap \ldots \cap L(A_n) \cap L(P) = \emptyset$?

Program reachability $\longrightarrow$ non-disjointness

Theorem: The reachability problem for boolean procedural programs is reducible to the non-disjointness problem for NFA$^n\times$ PDA, i.e., to the problem

Given: NFAs $A_1, \ldots, A_n$, PDA P

Decide: Is $L(A_1) \cap \ldots \cap L(A_n) \cap L(P) = \emptyset$?

Theorem: Non-disjointness of NFA$^n\times$PDA is EXPTIME-complete.

Complexity of efficient algorithms

Global variables: NFAs $G_1, \ldots, G_n$

Local variables: NFAs $L_1, \ldots, L_m$

Control: PDA P

Complexity: $O((|P| \cdot 2^n)^3 \cdot 2^m)$

Bounded model checking

Examine only computations containing at most k reads/writes of variables.

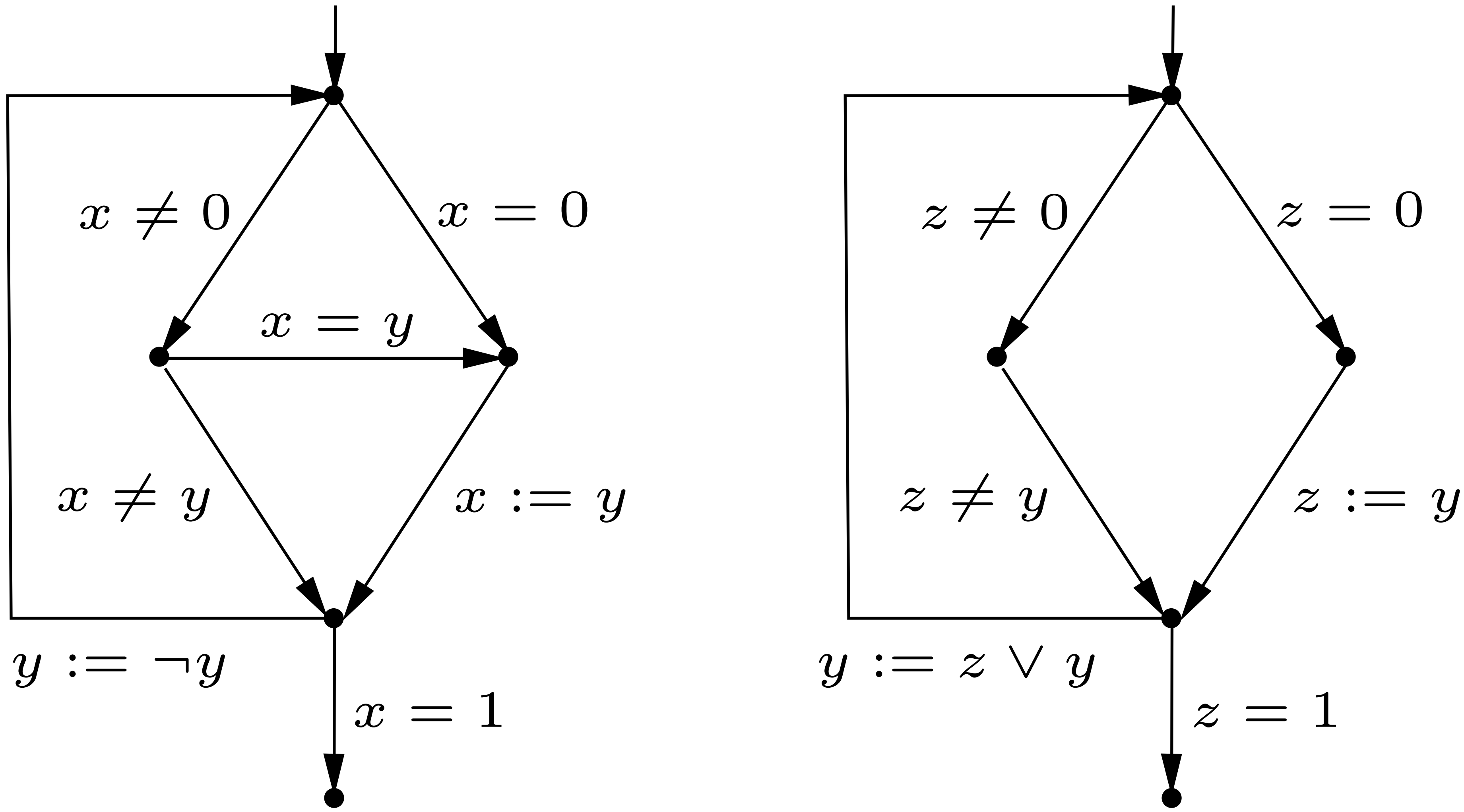
Theorem: k-non-disjointness of NFA$^n \times$ PDA, i.e., the problem

Given: NFAs $A_1, \ldots, A_n$, PDA P, bound $k \geq 0$

Decide: Is $L(A_1) \cap \ldots \cap L(A_n) \cap L(P) \cap \Sigma^{\leq k} = \emptyset$?

is NP-complete.

Multithreaded while-programs



Assume all variables are global (or that local variables are "embedded" into control).

Multithreaded while-programs

Same automata model: tuple of NFAs

No conceptual difference to sequential products.

- One NFA for each variable
- One NFA for each thread
- Transitions labeled by program instructions.

Multithreaded while-programs

Same automata model: tuple of NFAs

No conceptual difference to sequential products.

- One NFA for each variable
- One NFA for each thread
- Transitions labeled by program instructions.

However: two sources of complexity, threads and variables.

Problem remains PSPACE-complete for programs with fixed number of variables (but arbitrary number of threads).

Bounded-context model checking

Introduced by Qadeer and Rehof.

Context: computation segment without communication between threads

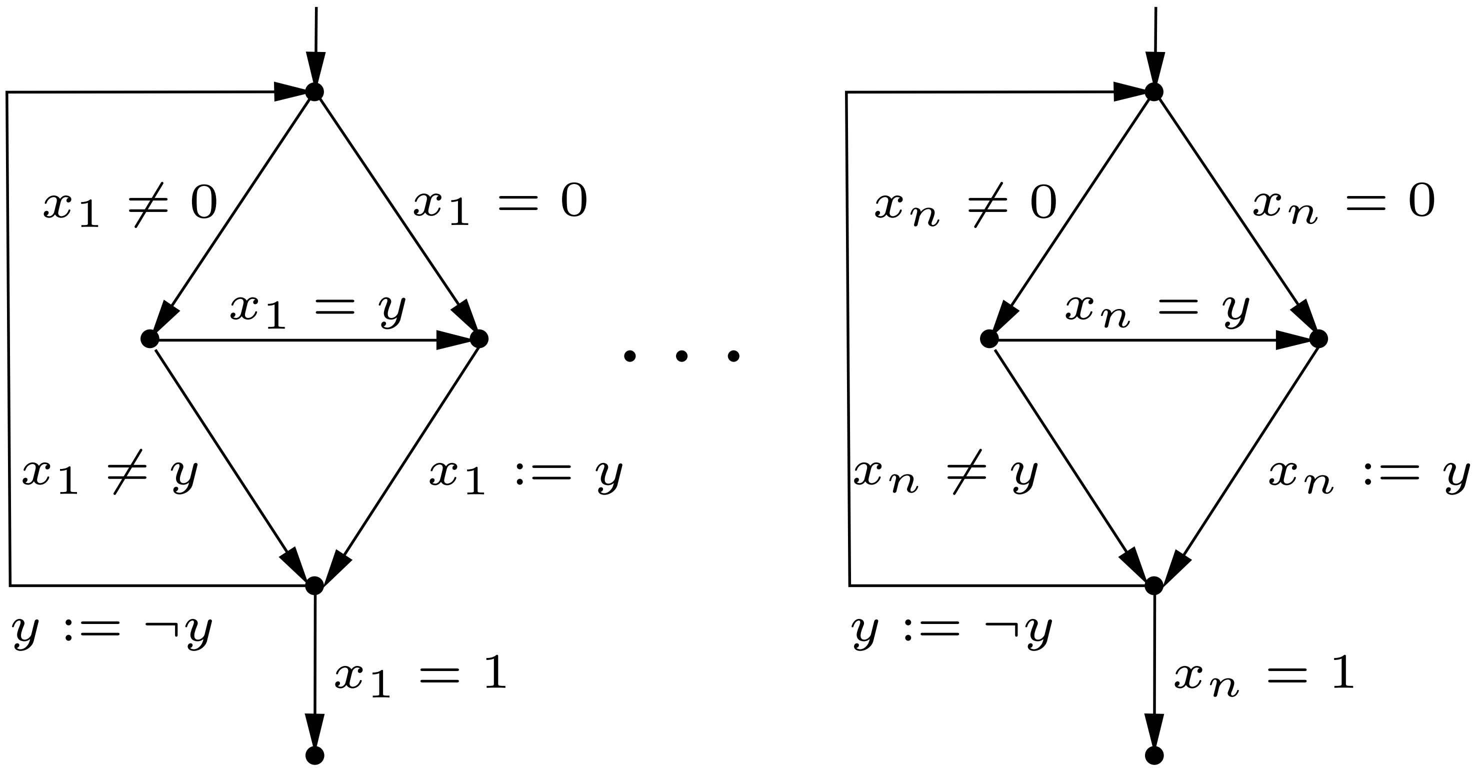
Context-switch: interaction with a global variable (almost).

Reachability within k context-switches: reachability by a computation with up to k reads/writes of global variables
(local variables "embedded " in control)

Fact: Reachability within k context-switches reduces to $c \cdot k$-non-disjointness of NFAs and some constant c.

Theorem: Reachability within k context-switches for multithreaded while-programs is NP-complete.

Parametrized multithreaded while-programs



- Fixed number of (w.l.o.g) shared variables
- n threads executing the same code (instances of a template).

Problem: prove that some safety property holds for all n

Automata model: Petri nets

Automata model: NFA$^n \times$ PN

- One NFA for each variable
- One NFA "with arbitrarily many tokens" for the threads

Automata model: Petri nets

Automata model: NFA$^n \times$ PN

- One NFA for each variable
- One NFA "with arbitrarily many tokens" for the threads

Theorem: Non-disjointness of NFA$^n \times$ PN is EXSPACE-complete
(with appropriate definition)

Automata model: Petri nets

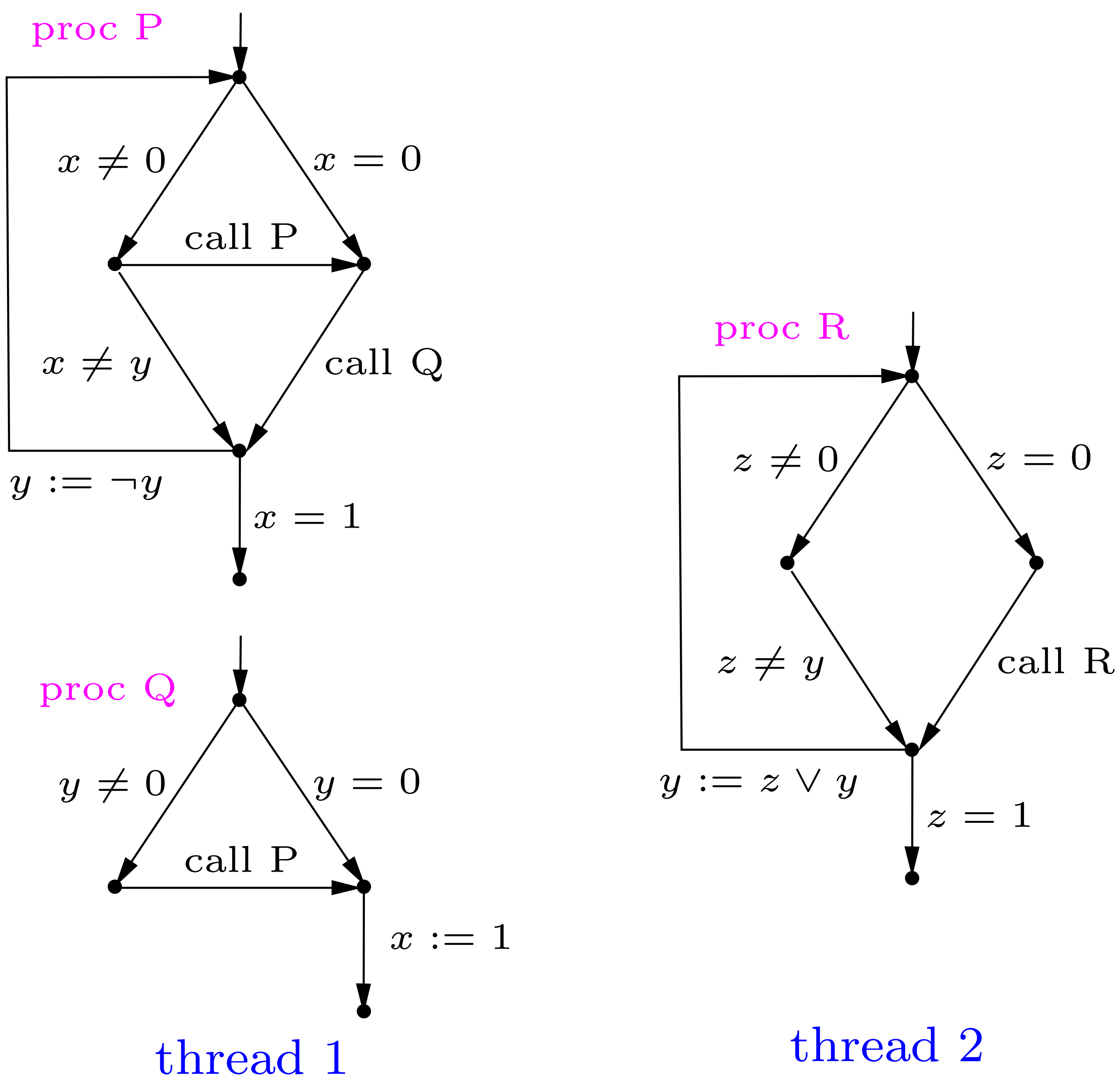
Automata model: NFA$^n\times$ PN

- One NFA for each variable
- One NFA "with arbitrarily many tokens" for the threads

Theorem: Non-disjointness of NFA$^n\times$ PN is EXSPACE-complete (with appropriate definition)

Theorem: k-non-disjointness of NFA$^n\times$ PN is NP-complete.

Multithreaded procedural programs



Multithreaded procedural programs

Automata model: NFA$^{n}\times$ PDA^{m}

- One NFA for each variable
- One PDA for each thread (no thread creation)

Multithreaded procedural programs

Automata model: $\text{NFA}^n \times\ \text{PDA}^m$

- One NFA for each variable
- One PDA for each thread (no thread creation)

Theorem: Non-disjointness of PDA^2 is undecidable.

Multithreaded procedural programs

Automata model: NFA$^n \times$ PDA^m

- One NFA for each variable
- One PDA for each thread (no thread creation)

Theorem: Non-disjointness of PDA^2 is undecidable.

Theorem: k-non-disjointness of NFA$^n \times$PDA^m is NP-complete.

Summary

Programs	Reachability	k-reachability
flat	PSPACE-complete	NP-complete
procedural	EXPTIME-complete	NP-complete
multithreaded	PSPACE-complete	NP-complete
parametrized multithreaded	EXSPACE-complete	NP-complete
multithreaded procedural	Undecidable	NP-complete

Question

Can we go beyond k-reachability

while keeping NP-completeness?

Patterns

Introduced by Kahlon, further studied by Ganty, Majumdar, and Monmege, and then by E. and Ganty, POPL '11.

A pattern is a regular expression of the form $w_1^* w_2^* \dots w_n^*$.

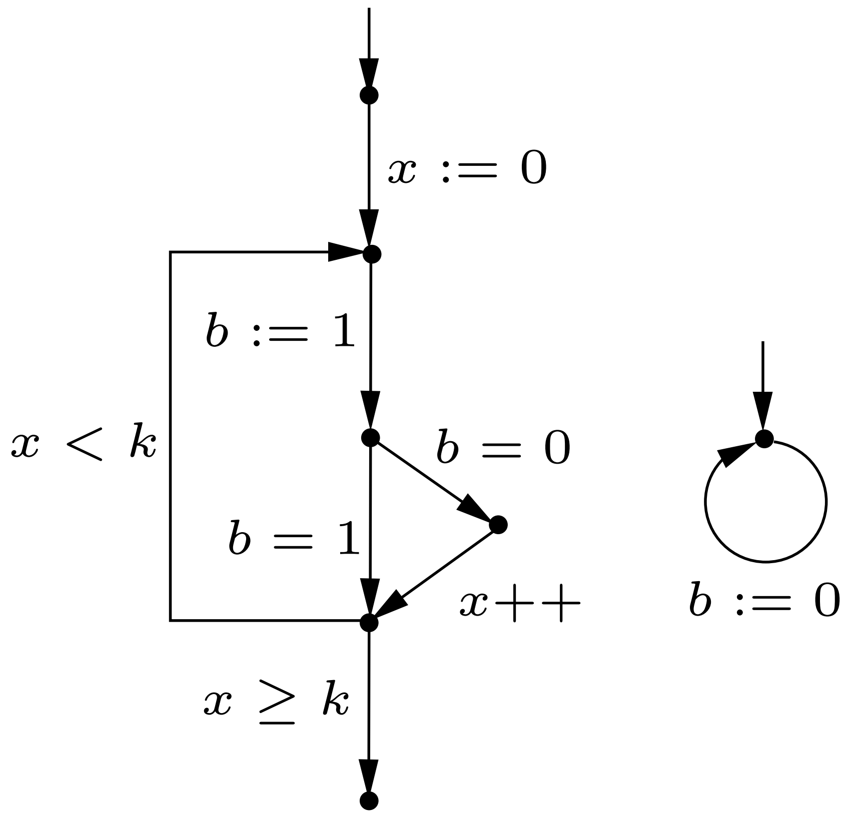
Intuition: explore "in depth" the computations that obey the pattern.

Non-disjointness modulo a pattern:

Given: Machines $M_1, \dots, M_n$, pattern p

Decide: Is $L(M_1) \cap \dots \cap L(M_n) \cap L(p) \neq \emptyset$?

Patterns



Terminating computations require k context-switches.
But always one satisfying the pattern

$$(x := 0)^* \ (b := 1 \ b := 0 \ b = 0 \ x{+}{+} \ x < k)^* \ (b := 1 \ b = 1 \ x \geq k)^*$$

Complexity

Theorem: Non-disjointness modulo a pattern for NFA$^n \times$PDA^m is NP-complete

Proof sketch: For the case $n = 0$, $m = 2$, and $p = a_1^* a_2^* \ldots a_n^*$.
Let $\mathcal{C}(L)$ denote the commutative image of a language L

$$\mathcal{C}(\ \{aab, a, aba, \epsilon\}\ ) = \{\ (2,1), (1,0), (0,0)\ \}$$

Complexity

Theorem: Non-disjointness modulo a pattern for NFA$^n \times$PDA^m is NP-complete

Proof sketch: For the case $n = 0$, $m = 2$, and $p = a_1^* a_2^* \ldots a_n^*$.
Let $\mathcal{C}(L)$ denote the commutative image of a language L

$$\mathcal{C}(\ \{aab, a, aba, \epsilon\}\ ) = \{\ (2,1), (1,0), (0,0)\ \}$$

$$L(P_1) \cap L(P_2) \cap L(p) = \emptyset$$

Complexity

Theorem: Non-disjointness modulo a pattern for NFA$^n \times$PDA^m is NP-complete

Proof sketch: For the case $n = 0$, $m = 2$, and $p = a_1^* a_2^* \ldots a_n^*$.
Let $\mathcal{C}(L)$ denote the commutative image of a language L

$$\mathcal{C}(\ \{aab, a, aba, \epsilon\}\ ) = \{\ (2,1), (1,0), (0,0)\ \}$$

$$\begin{aligned} & L(P_1) \cap L(P_2) \cap L(p) = \emptyset \\ \Leftrightarrow \quad & L(P_1 \times p) \cap L(P_2 \times p) = \emptyset \end{aligned}$$

Complexity

Theorem: Non-disjointness modulo a pattern for NFA$^n \times$PDA^m is NP-complete

Proof sketch: For the case $n = 0$, $m = 2$, and $p = a_1^* a_2^* \ldots a_n^*$.
Let $\mathcal{C}(L)$ denote the commutative image of a language L

$$\mathcal{C}(\ \{aab, a, aba, \epsilon\}\ ) = \{\ (2,1), (1,0), (0,0)\ \}$$

$$\begin{aligned} & L(P_1) \cap L(P_2) \cap L(p) = \emptyset \\ \Leftrightarrow\quad & L(P_1 \times p) \cap L(P_2 \times p) = \emptyset \\ \Leftrightarrow\quad & \mathcal{C}(L(P_1 \times p)) \cap \mathcal{C}(L(P_2 \times p)) = \emptyset \quad \text{(GS66)} \end{aligned}$$

Complexity

Theorem: Non-disjointness modulo a pattern for $\mathsf{NFA}^n \times \mathsf{PDA}^m$ is NP-complete

Proof sketch: For the case $n = 0$, $m = 2$, and $p = a_1^* a_2^* \ldots a_n^*$.
Let $\mathcal{C}(L)$ denote the commutative image of a language L

$$\mathcal{C}(\ \{aab, a, aba, \epsilon\}\ ) = \{\ (2,1), (1,0), (0,0)\ \}$$

$$\begin{array}{ll} & L(P_1) \cap L(P_2) \cap L(p) = \emptyset \\ \Leftrightarrow & L(P_1 \times p) \cap L(P_2 \times p) = \emptyset \\ \Leftrightarrow & \mathcal{C}(L(P_1 \times p)) \cap \mathcal{C}(L(P_2 \times p)) = \emptyset \quad \text{(GS66)} \\ \Leftrightarrow & F_1 \wedge F_2 \text{ is satisfiable} \end{array}$$

where F_1, F_2 formulas of existential Presburger arithmetic (NSS05).

Complexity

Theorem: Non-disjointness modulo a pattern for NFA$^n \times$PDA^m is NP-complete

Proof sketch: For the case $n = 0$, $m = 2$, and $p = a_1^* a_2^* \dots a_n^*$.
Let $\mathcal{C}(L)$ denote the commutative image of a language L

$$\mathcal{C}(\ \{aab, a, aba, \epsilon\}\ ) = \{\ (2,1), (1,0), (0,0)\ \}$$

$$\begin{array}{ll} & L(P_1) \cap L(P_2) \cap L(p) = \emptyset \\ \Leftrightarrow & L(P_1 \times p) \cap L(P_2 \times p) = \emptyset \\ \Leftrightarrow & \mathcal{C}(L(P_1 \times p)) \cap \mathcal{C}(L(P_2 \times p)) = \emptyset \quad \text{(GS66)} \\ \Leftrightarrow & F_1 \wedge F_2 \text{ is satisfiable} \end{array}$$

where F_1, F_2 formulas of existential Presburger arithmetic (NSS05).

Satisfiability of existential Presburger arithmetic is NP complete.

Multiparameter analysis

Size of the input may depend on parameters.

Complexity may be polynomial in one parameter and “necessarily exponential” in another.

Relevant instances may have small values for the “critical” parameters

Multiparameter analysis

We analyze the dependence of the complexity on four parameters:

Multiparameter analysis

We analyze the dependence of the complexity on four parameters:

size of pattern $\Rightarrow$ size of pattern

Multiparameter analysis

We analyze the dependence of the complexity on four parameters:

size of pattern	$\Rightarrow$	size of pattern
number of threads	$\Rightarrow$	number of NFAs and PDAs

Multiparameter analysis

We analyze the dependence of the complexity on four parameters:

size of pattern	$\Rightarrow$	size of pattern
number of threads	$\Rightarrow$	number of NFAs and PDAs
size of threads	$\Rightarrow$	maximum size of a NFA/PDA

Multiparameter analysis

We analyze the dependence of the complexity on four parameters:

size of pattern	$\Rightarrow$	size of pattern
number of threads	$\Rightarrow$	number of NFAs and PDAs
size of threads	$\Rightarrow$	maximum size of a NFA/PDA
height of call graph	$\Rightarrow$	longest simple stack content

Multiparameter analysis

We analyze the dependence of the complexity on four parameters:

size of pattern	$\Rightarrow$	size of pattern
number of threads	$\Rightarrow$	number of NFAs and PDAs
size of threads	$\Rightarrow$	maximum size of a NFA/PDA
height of call graph	$\Rightarrow$	longest simple stack content

Observe: Recursion allowed. Height of the call graph $=$ maximal length of a simple path!

Multiparameter analysis

We analyze the dependence of the complexity on four parameters:

size of pattern	$\Rightarrow$	size of pattern
number of threads	$\Rightarrow$	number of NFAs and PDAs
size of threads	$\Rightarrow$	maximum size of a NFA/PDA
height of call graph	$\Rightarrow$	longest simple stack content

Observe: Recursion allowed. Height of the call graph = maximal length of a simple path!

Does the problem remain NP-complete if we fix the values of a subset of the parameters?

Standard NP-completeness theory . . .

Theorem: The following problems are NP-hard:

Instance: NFAs $P_1, \ldots, P_n$.
Question: Is $\bigcap_{i=1}^{n} L(P_i) \cap L(p) \neq \varnothing$ for the pattern $p = a^*$?

Instance: PDAs $P_1, \ldots, P_n$ of fixed size and a pattern p.
Question: Is $\bigcap_{i=1}^{n} L(P_i) \cap L(p) \neq \varnothing$?

Instance: Two PDAs P_1, P_2 over the alphabet $\{a\}$.
Question: Is $L(P_1) \cap L(P_2) \neq \varnothing$?

The remaining case

The fixed-parameter problem is NP-complete for all cases but two:

- fixed values for all parameters.
  Solvable in linear time, only finitely many instances!
- fixed number of threads and variables,
  fixed height of call graph,
  fixed size of pattern,
  threads of arbitrary size

We show that this second case is polynomial.

A polynomial algorithm

Input (say): PDAs P_1, P_2 with call graph of height at most 3,
pattern of size at most 5.

A polynomial algorithm

Input (say): PDAs P_1, P_2 with call graph of height at most 3,
pattern of size at most 5.

$$L(P_1) \cap L(P_2) \cap L(p) = \emptyset$$

A polynomial algorithm

Input (say): PDAs P_1, P_2 with call graph of height at most 3,
pattern of size at most 5.

$$
\begin{aligned}
& L(P_1) \cap L(P_2) \cap L(p) = \emptyset \\
\Leftrightarrow \quad & L(P_1 \times p) \cap L(P_2 \times p) = \emptyset
\end{aligned}
$$

A polynomial algorithm

Input (say): PDAs P_1, P_2 with call graph of height at most 3,
pattern of size at most 5.

$$
\begin{aligned}
& L(P_1) \cap L(P_2) \cap L(p) = \emptyset \\
\Leftrightarrow\quad & L(P_1 \times p) \cap L(P_2 \times p) = \emptyset \\
\Leftrightarrow\quad & \mathcal{C}(L(P_1 \times p)) \cap \mathcal{C}(L(P_2 \times p)) = \emptyset \quad \text{(GS66)}
\end{aligned}
$$

A polynomial algorithm

Input (say): PDAs P_1, P_2 with call graph of height at most 3,
pattern of size at most 5.

$$
\begin{aligned}
& L(P_1) \cap L(P_2) \cap L(p) = \emptyset \\
\Leftrightarrow \quad & L(P_1 \times p) \cap L(P_2 \times p) = \emptyset \\
\Leftrightarrow \quad & \mathcal{C}(L(P_1 \times p)) \cap \mathcal{C}(L(P_2 \times p)) = \emptyset \quad \text{(GS66)}
\end{aligned}
$$

How do we check this in polynomial time?

Parikh's theorem

Theorem (Par66): For every PDA P there is a NFA A such that $\mathcal{C}(L(P)) = \mathcal{C}(L(A))$ (a Parikh-NFA for P)

Theorem (EG11): PDAs with call stack of fixed height have Parikh-NFAs of polynomial size.

Corollary: $P_1 \times p$ and $P_2 \times p$ have Parikh-NFAs A_1, A_2 of polynomial size.

It remains to check in polynomial time whether $\mathcal{C}(L(A_1)) \cap \mathcal{C}(L(A_2)) \neq \emptyset$ holds for two given NFAs.

Two-way counter machines

Two-way NFAs whose transitions are labeled with actions on a set of counters:

- increase counter c_i by 1
- decrease counter c_i by 1 (blocks if $c_j = 0$)
- check if $c_i = 0$

Two-way counter machines

Two-way counter machine M recognizing $\mathcal{C}(L(A_1)) \cap \mathcal{C}(L(A_2))$:

- M uses one counter for each alphabet letter of A_1 and A_2.
- M checks first if input belongs to $\mathcal{C}(L(A_1))$, then to $\mathcal{C}(L(A_2))$.
- For the check:
  - M reads the input, counting the number of occurrences of each letter, and then
  - guesses a path of A_i in which each letter occurs the number of times given by the counter.

Great, but ...

Theorem: The emptiness problem for 2-counter machines is undecidable.

Great, but . . .

Theorem: The emptiness problem for 2-counter machines is undecidable.

Consider the following parameters:

- number of counters,
- number of times a boundary between input symbols is crossed, and
- number of counter reversals.

Great, but ...

Theorem: The emptiness problem for 2-counter machines is undecidable.

Consider the following parameters:

- number of counters,
- number of times a boundary between input symbols is crossed, and
- number of counter reversals.

Theorem (GI 81): Emptiness of two-way counter machines with a fixed number of counters, boundary crosses, and counter reversals can be decided in polynomial time.

In our case ...

- fixed number of counters: alphabet size bounded size of pattern;

In our case ...

- fixed number of counters: alphabet size bounded size of pattern;
- fixed number of boundary crossings: 2
  (more generally: bounded by number of threads)

In our case ...

- fixed number of counters: alphabet size bounded size of pattern;
- fixed number of boundary crossings: 2
  (more generally: bounded by number of threads)
- fixed number of counter reversals: 2
  (more generally: bounded by number of threads)

Conclusions

Programs with finite-range variables can be modeled by automata

Reachability reduces naturally to non-disjointness

Automata theory helps to determine the asymptotic complexity and to identify tractable cases

Interesting question: identify interesting subsets of computations that can be explored with reduced complexity.