An Algebraic Approach to the Static Analysis of Concurrent Software

Javier Esparza

Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh

Joint work with Ahmed Bouajjani and Tayssir Touili

Motivation

Interprocedural dataflow analysis

Extension to concurrency badly needed

Object oriented languages

- methods $\longrightarrow$ procedures
- multithreads $\longrightarrow$ concurrency

Extension to concurrency very hard: undecidability results

Former work

Not very much!

[Duesterwald and Soffa, '91]: dataflow equations

- efficient and simple
- approximates the effects of both procedures and concurrency
- no way to trade efficiency for precision

Flanagan,Qadeer,Seshia, '02 : assume-guarantee approach

- relies on specification by programmer

This work

Extends our approach to interprocedural model-checking
[Bouajjani, E., Maler '97], [E., Schwoon '01]

Conservative extension: exact for sequential programs

Abstract interpretation framework for computing abstractions of program paths

Generic computation of commutative abstractions of path languages

An abstraction is commutative if it 'forgets' the order in which actions occur (and maybe more)

The program model: Sequential programs

Sequential programs determined by

control flow of procedures

- assignments, conditionals, loops
- procedure calls with parameter passing / return values

local variables of each procedure

global variables

State space determined by

program pointer

values of global variables

values of local variables (of current procedure)

activation records (return addresses, copies of locals)

The program model: Concurrent programs

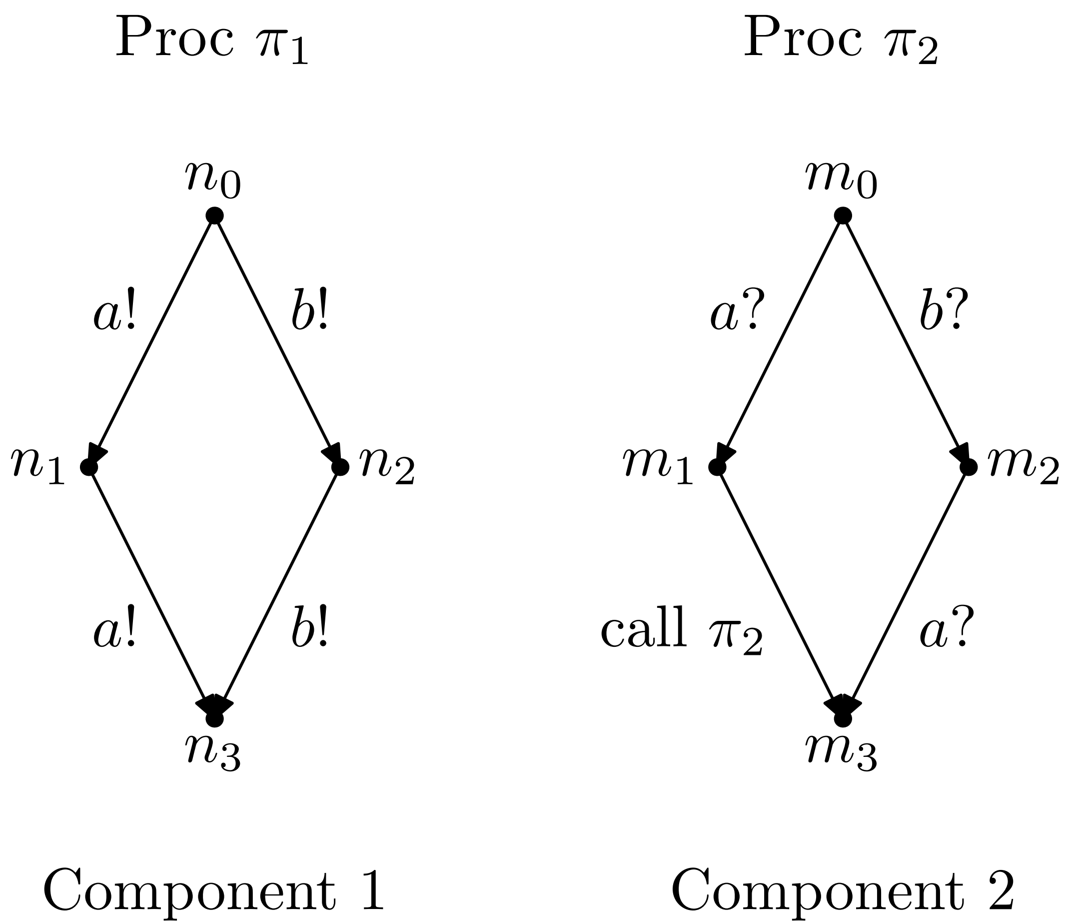
Concurrent program: a tuple of sequential programs

- no process creation in this talk

Communication through rendezvous

- primitives $a!x$ and $a?x$, where a is a channel
- channels are unidirectional and point-to-point
- no dynamic broadcasts (compare with `notifyAll`)

An example (control flow only)



The formal model: Communicating pushdown pystems

A pushdown system (PDS) is a fivetuple $\mathcal{P} = (P, Act, \Gamma, c_0, \Delta)$, where

- P is a finite set of control locations
- Act is a finite set of actions
- Γ is a finite stack alphabet

  A configuration of $\mathcal{P}$ is a pair $c = \langle p, v \rangle$, where $p \in P$, $v \in \Gamma^*$

- c_0 is the initial configuration
- $\Delta \subseteq (P \times Act \cup \{\epsilon\} \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of rules.

If $\langle p, \gamma \rangle \overset{a}{\hookrightarrow} \langle p', v \rangle \in \Delta$ then $\langle p, \gamma w \rangle \xrightarrow{a} \langle p', vw \rangle$ for every $w \in \Gamma^*$

Normalisation: $|v| \leq 2$

A communicating pushdown system is a tuple $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ of pushdown systems

A global configuration is a tuple $g = (c_1, \ldots, c_n)$ of configurations

$g_0 = (c_{10}, \ldots, c_{n0})$ is the initial global configuration

Let $g = (c_1, \ldots, c_n)$ and $g' = (c'_1, \ldots, c'_n)$

$$g \xrightarrow{\epsilon} g' \text{ if } \begin{array}{l} c_i \xrightarrow{\epsilon} c'_i \text{ for some } 1 \leq i \leq n \text{, and} \\ c'_j = c_j \text{ for every } j \neq i \end{array}$$

$$g \xrightarrow{a} g' \text{ if } \begin{array}{l} c_i \xrightarrow{a} c'_i \text{ and } c_j \xrightarrow{a} c'_j \text{ for some } i \neq j \text{, and} \\ c'_k = c_k \text{ for every } i \neq k \neq j \end{array}$$

Semantic mapping

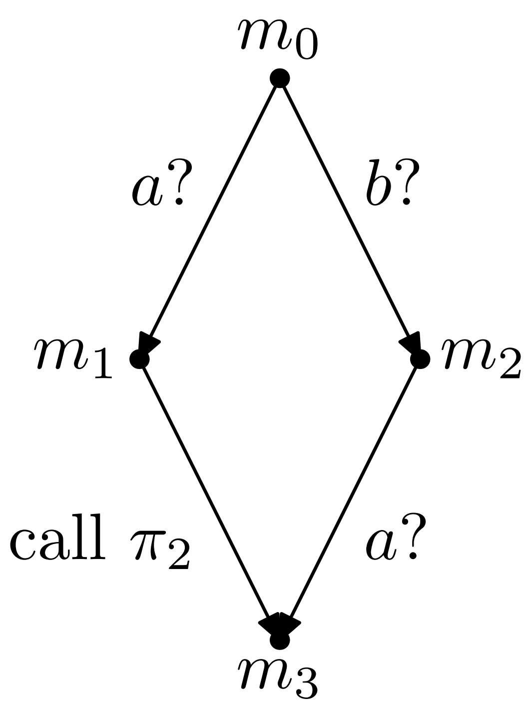
Interpretation of $\langle p, \gamma v \rangle$

- p holds values of global variables of the component
- γ holds (program pointer, values of local variables)
- v holds stack of (return address, saved locals)

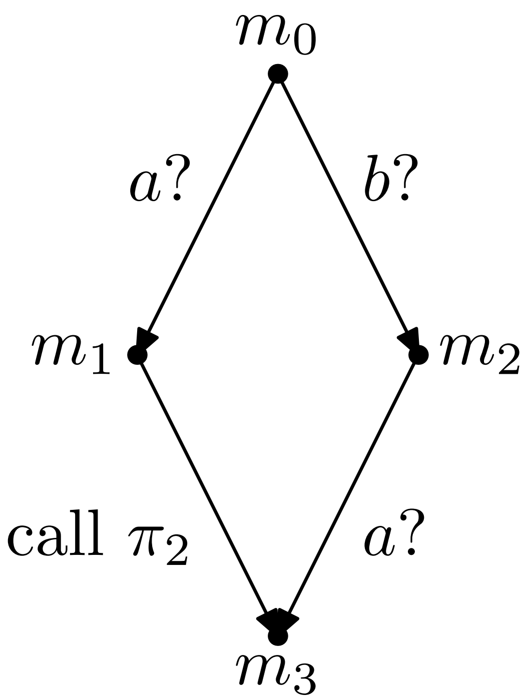
Restriction: finite datatypes

Correspondence between statements and rules

$\langle p, \gamma \rangle \xhookrightarrow{\epsilon} \langle p', \gamma' \rangle$	simple statement
$\langle p, \gamma \rangle \xhookrightarrow{a_v} \langle p', \gamma' \rangle$	communication of value v through channel a
$\langle p, \gamma \rangle \xhookrightarrow{\epsilon} \langle p', \gamma' \gamma'' \rangle$	procedure call
$\langle p, \gamma \rangle \xhookrightarrow{\epsilon} \langle p', \epsilon \rangle$	return statement



$$
\begin{aligned}
r_1 &: \langle p, m_0\rangle \xhookrightarrow{a} \langle p, m_1\rangle \\
r_2 &: \langle p, m_0\rangle \xhookrightarrow{b} \langle p, m_2\rangle \\
r_3 &: \langle p, m_1\rangle \xhookrightarrow{\epsilon} \langle p, m_0 m_3\rangle \\
r_4 &: \langle p, m_2\rangle \xhookrightarrow{a} \langle p, m_3\rangle \\
r_5 &: \langle p, m_3\rangle \xhookrightarrow{\epsilon} \langle p, \epsilon\rangle
\end{aligned}
$$



$$
\begin{aligned}
r_1 &: \quad m_0 \xhookrightarrow{a} m_1 \\
r_2 &: \quad m_0 \xhookrightarrow{b} m_2 \\
r_3 &: \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3 \\
r_4 &: \quad m_2 \xhookrightarrow{a} m_3 \\
r_5 &: \quad m_3 \xhookrightarrow{\epsilon} \epsilon
\end{aligned}
$$

Reachability in communicating pushdown systems

Given:	a communicating pushdown system $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$,
	a set F of final global configurations
To decide:	is F reachable from the initial global configuration g_0,
	i.e., is there $f \in F$ such that $g_0 \to^* f$?

Key to many analysis problems

Unfortunately: undecidable, even for $n = 2$, $F = \{f\}$
[Ramalingam, TOPLAS 2000]

A proof sketch of the undecidability result

Reduction from Given: two context-free grammars G_1, G_2

To decide: is $L(G_1) \cap L(G_2) = \emptyset$?

Let G_1, G_2 be context-free grammars

Construct a communicating pushdown system $(\mathcal{P}_1, \mathcal{P}_2)$ and global configurations $g_0 = (c_{01}, c_{02})$, $f = (f_1, f_2)$ such that

$$\begin{array}{lclcl} L(G_1) & = & L(c_{01}, f_1) & = & \{w \in Act^* \mid c_{01} \xrightarrow{w} f_1\} \\ L(G_2) & = & L(c_{02}, f_2) & = & \{w \in Act^* \mid c_{02} \xrightarrow{w} f_2\} \end{array}$$

$g_0 \rightarrow^* f$ iff $w \in L(c_{01}, f_1) \cap L(c_{02}, f_2)$ for some $w \in Act^*$

So $g_0 \rightarrow^* f$ iff $L(G_1) \cap L(G_2) \neq \emptyset$

Our approach: Abstract path languages

Assume $F = F_1 \times F_2$

Idea: Compute abstract languages $A_1 \supseteq L(c_{01}, F_1)$ and $A_2 \supseteq L(c_{02}, F_2)$ such that $A_1 \cap A_2 \stackrel{?}{=} \emptyset$ is decidable

Then: If $A_1 \cap A_2 = \emptyset$, we have proved $L(c_{01}, F_1) \cap L(c_{02}, F_2) = \emptyset$

A_1, A_2 must be finitely represented through abstract objects d_1 and d_2

$A_1 \cap A_2$ must be replaced by an abstract operation $d_1 \sqcap d_2$

Formal framework: abstract interpretation

Abstract interpretation of path languages

Let $\mathcal{L} = (2^{Act^*}, \subseteq, \cup, \cap, \emptyset, Act^*)$ be the complete lattice of languages over Act

An abstraction consists of

- an abstract lattice $\mathcal{D} = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$, and
- a Galois connection $\mathcal{L} \underset{\gamma}{\overset{\alpha}{\rightleftharpoons}} \mathcal{D}$

$$\begin{array}{rcll} \gamma(\alpha(L)) & \supseteq & L & (A \supseteq L) \\ \gamma(d_1 \sqcap d_2) & = & \gamma(d_1) \cap \gamma(d_2) & (\sqcap \text{ matches } \cap) \end{array}$$

$\alpha(L)$ is the abstract object representing the language $A \supseteq L$

$\gamma(\alpha(L))$ is the language A

An abstraction

$\alpha_1(L)$ is the pair $[F, R]$, $F, R \subseteq Act$, where

F (for forbidden) is the set of actions that do not occur in any word of L

R (for required) is the set of actions that occur in all words of L

Example: for $Act = \{a, b, c\}$, $\alpha_1(ab^*) = [\{c\}, \{a\}]$

$\gamma_1([F, R]) =$ all words containing no letter of F and all letters of R

$[F_1, R_1] \sqcap [F_2, R_2] = [F_1 \cup F_2, R_1 \cup R_2]$ (well, almost true . . .)

Another abstraction

Let $al(w)$ be the alphabet of w (i.e., the set of letters that occur in w)

Define $\alpha_2(L) = \{al(w) \mid w \in L\}$

Example: $\alpha_2(ab^*) = \{ \{a\}, \{a, b\} \}$

$\gamma_2(\{al_1, \ldots, al_n\}) =$ all words over the alphabets $al_1, \ldots, al_n$

$\{al_1, \ldots, al_n\} \sqcap \{al'_1, \ldots, al'_m\} = \{al_1, \ldots, al_n\} \cap \{al'_1, \ldots, al'_m\}$

Yet another abstraction

Let $p(w) : Act \rightarrow \mathbb{N}$ be the Parikh image of w (occurrence count for each letter)

Define $\alpha_3(L) = \{p(w) \mid w \in L\}$

Example: $\alpha_3(ab^*) = \{(1, n) \mid n \geq 0\}$

$\gamma_3(L) =$ all the permutations of the words of L

$\{v_1, v_2, \ldots,\} \sqcap \{v_1', v_2', \ldots,\} = \{v_1, v_2, \ldots,\} \cap \{v_1', v_2', \ldots,\}$

The common feature?

Commutativity: If $w \in A$, then $w' \in A$ for every permutation of w

Commutative abstractions: abstractions in which $\gamma(\alpha(L))$ is a commutative language for all L

We provide a generic algorithm for computing $\alpha(L)$ when α is a commutative abstraction

The approach

In order to compute $\alpha(L(c_0, F))$

- compute $pre^*(F) = \{c \mid \exists f \in F : c \rightarrow^* f\}$
- compute for each $c \in pre^*(F)$ the language $\alpha(L(c, F))$
- return $\alpha(L(c_0, F))$

(It is also possible to use $post^*(c_0)$)

Problem: $pre^*(F)$ can be an infinite set (even F can be infinite)

This can be dealt with when F is regular.

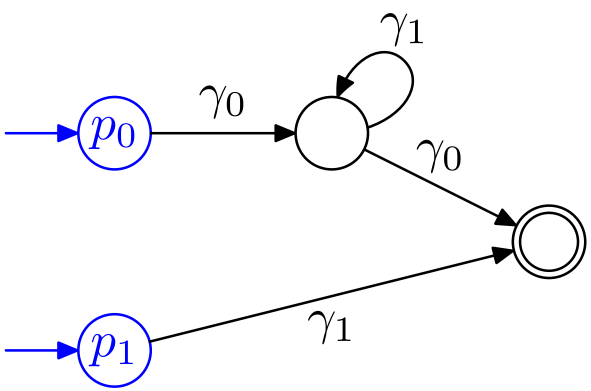
Regular sets and multi-automata

A set of configurations C is regular if for every control point p, the set $\{w \in \Gamma^* \mid \langle p, w \rangle \in C\}$ is regular

Regular sets can be finitely represented by multi-automata:

- P as set of initial states and Γ as alphabet
- $\langle p, v \rangle$ recognized if $p \xrightarrow{v} q$ for some final state q

Example: multi-automaton for the set $\langle p_0, \gamma_0 \gamma_1^* \gamma_0 \rangle \cup \langle p_1, \gamma_1 \rangle$:



The basic theory

Theorem [Büchi '64], [Book and Otto '93], [Caucal '92] . . .

If C is regular, then so is $pre^*(C)$

Theorem [Bouajjani, E., Maler '97], [E.et al '00], [E. and Schwoon '01]

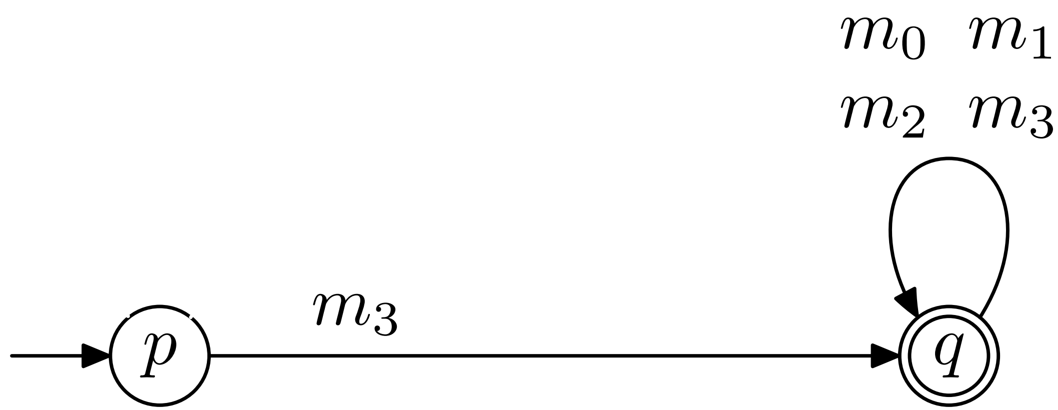
Given a multi-automaton $\mathcal{A}$ recognizing F,

it is possible to effectively (and efficiently) construct

another multi-automaton $\mathcal{A}_{pre^*}$ recognizing $pre^*(F)$

The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$r_1 : \quad m_0 \overset{a}{\hookrightarrow} m_1$$
$$r_2 : \quad m_0 \overset{b}{\hookrightarrow} m_2$$
$$r_3 : \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$$
$$r_4 : \quad m_2 \overset{a}{\hookrightarrow} m_3$$
$$r_5 : \quad m_3 \overset{\epsilon}{\hookrightarrow} p, \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

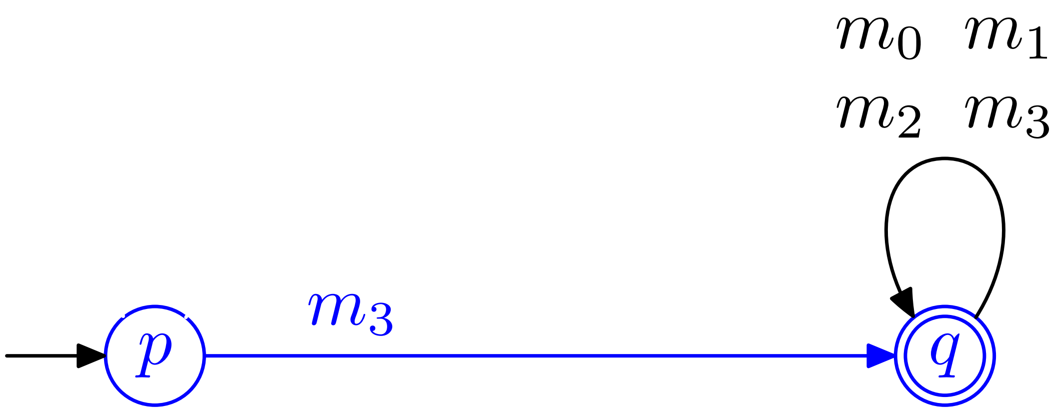
$r_1 : \quad m_0 \xhookrightarrow{a} m_1$

$r_2 : \quad m_0 \xhookrightarrow{b} m_2$

$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$

$r_4 : \quad m_2 \xhookrightarrow{a} m_3$

$r_5 : \quad m_3 \xhookrightarrow{\epsilon} p, \epsilon$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

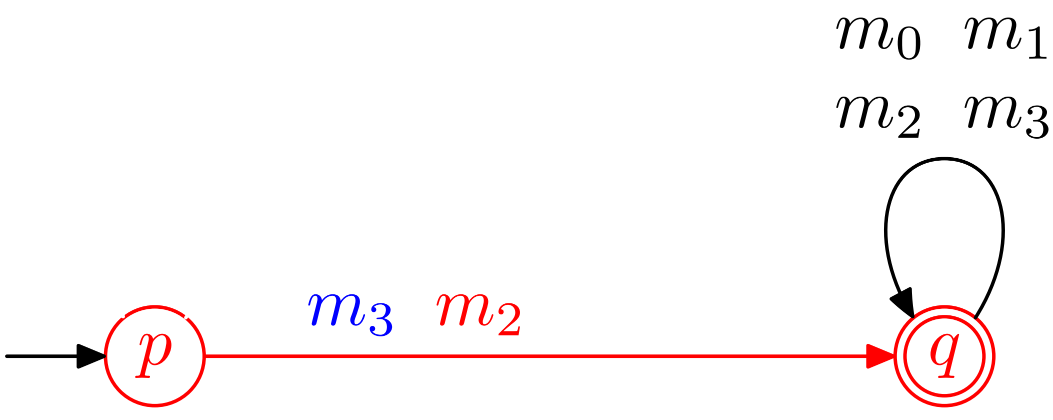
$r_1 : \quad m_0 \xhookrightarrow{a} m_1$

$r_2 : \quad m_0 \xhookrightarrow{b} m_2$

$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$

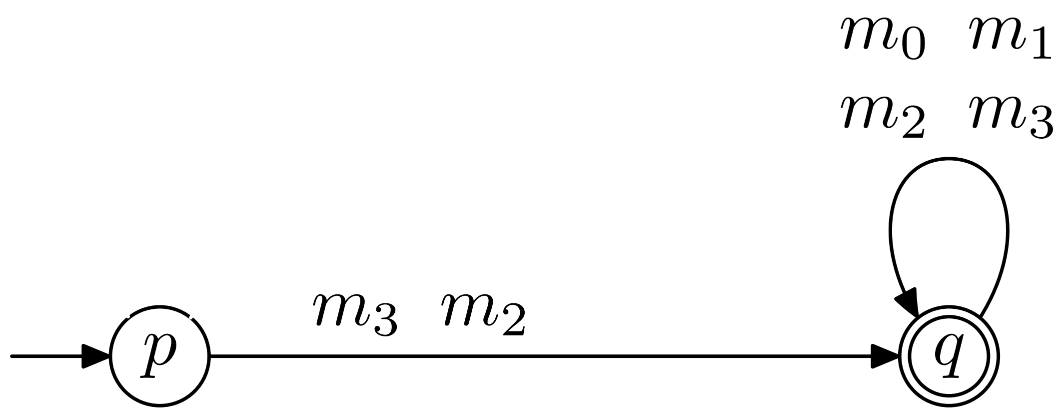
$r_4 : \quad m_2 \xhookrightarrow{a} m_3$

$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$r_1 : \quad m_0 \overset{a}{\hookrightarrow} m_1$$
$$r_2 : \quad m_0 \overset{b}{\hookrightarrow} m_2$$
$$r_3 : \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$$
$$r_4 : \quad m_2 \overset{a}{\hookrightarrow} m_3$$
$$r_5 : \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

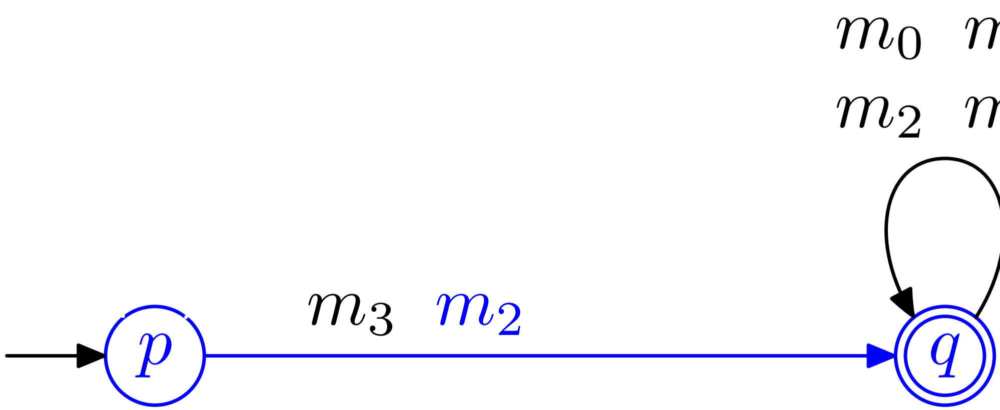
$$r_1 : \quad m_0 \xhookrightarrow{a} m_1$$

$$r_2 : \quad m_0 \xhookrightarrow{b} m_2$$

$$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$$

$$r_4 : \quad m_2 \xhookrightarrow{a} m_3$$

$$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

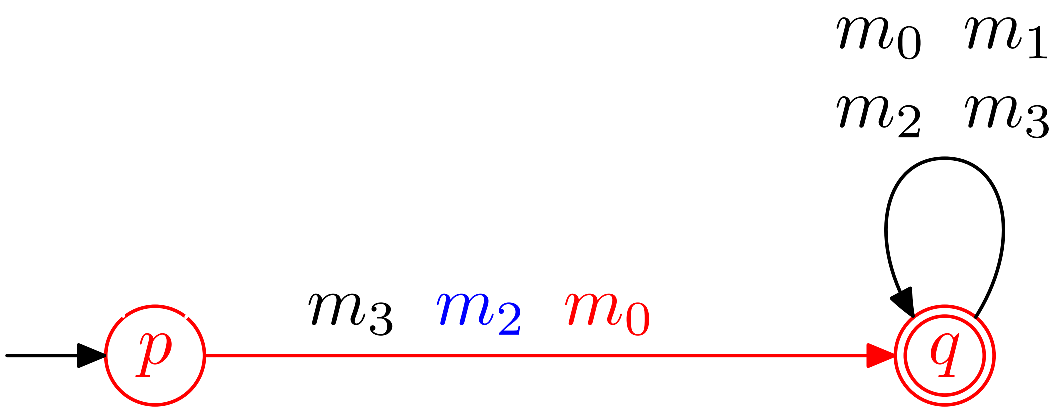
$$r_1 : \quad m_0 \xhookrightarrow{a} m_1$$

$$r_2 : \quad m_0 \xhookrightarrow{b} m_2$$

$$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$$

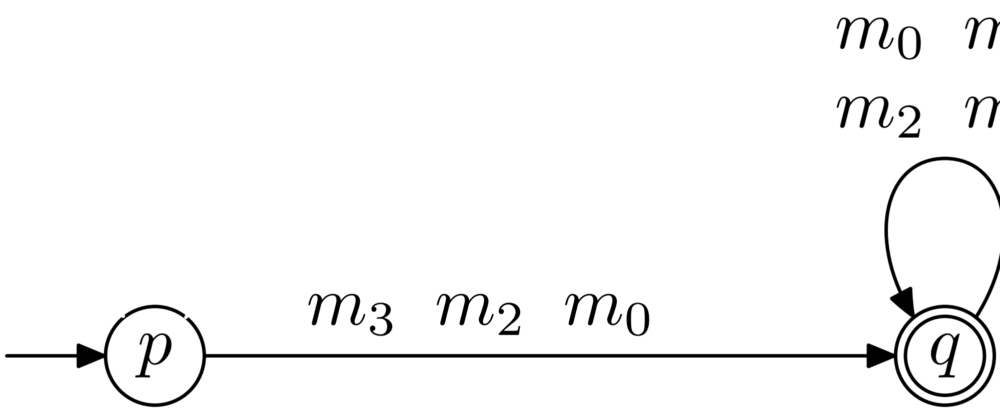
$$r_4 : \quad m_2 \xhookrightarrow{a} m_3$$

$$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$$



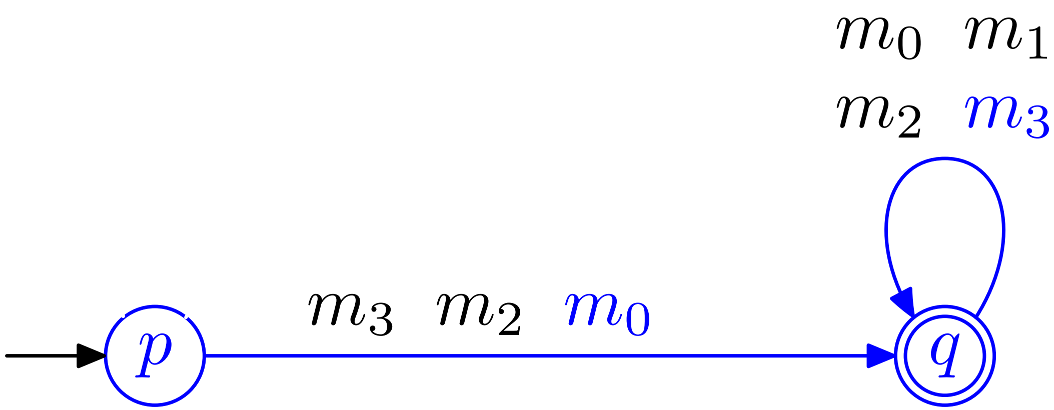
The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$
\begin{aligned}
r_1 &: \quad m_0 \overset{a}{\hookrightarrow} m_1 \\
r_2 &: \quad m_0 \overset{b}{\hookrightarrow} m_2 \\
r_3 &: \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3 \\
r_4 &: \quad m_2 \overset{a}{\hookrightarrow} m_3 \\
r_5 &: \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon
\end{aligned}
$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$r_1 : \quad m_0 \xhookrightarrow{a} m_1$$
$$r_2 : \quad m_0 \xhookrightarrow{b} m_2$$
$$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$$
$$r_4 : \quad m_2 \xhookrightarrow{a} m_3$$
$$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

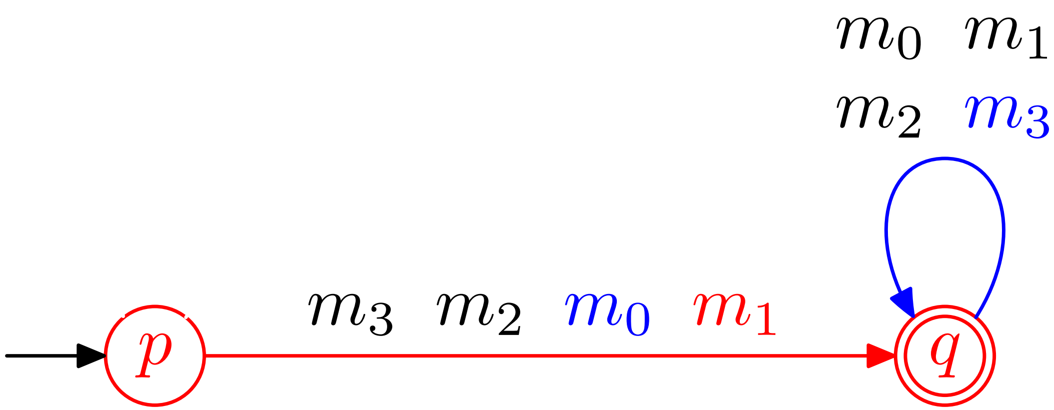
$r_1 : \quad m_0 \overset{a}{\hookrightarrow} m_1$

$r_2 : \quad m_0 \overset{b}{\hookrightarrow} m_2$

$r_3 : \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$

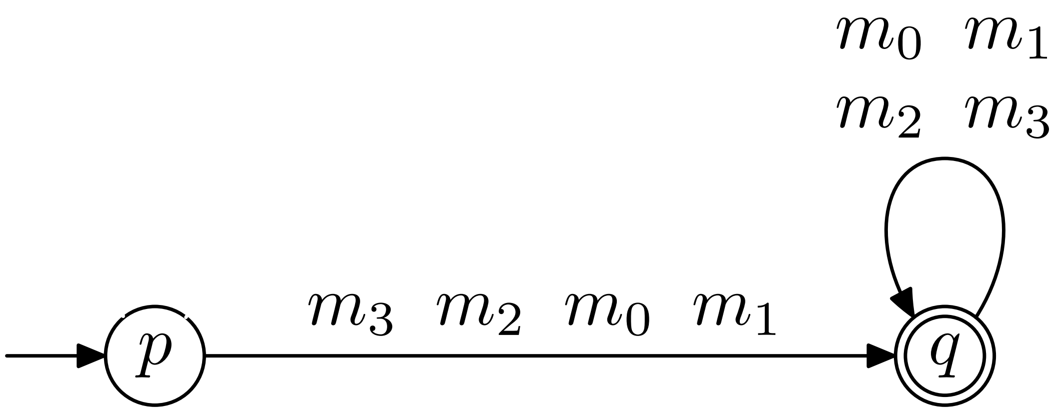
$r_4 : \quad m_2 \overset{a}{\hookrightarrow} m_3$

$r_5 : \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$r_1: \quad m_0 \overset{a}{\hookrightarrow} m_1$$
$$r_2: \quad m_0 \overset{b}{\hookrightarrow} m_2$$
$$r_3: \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$$
$$r_4: \quad m_2 \overset{a}{\hookrightarrow} m_3$$
$$r_5: \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

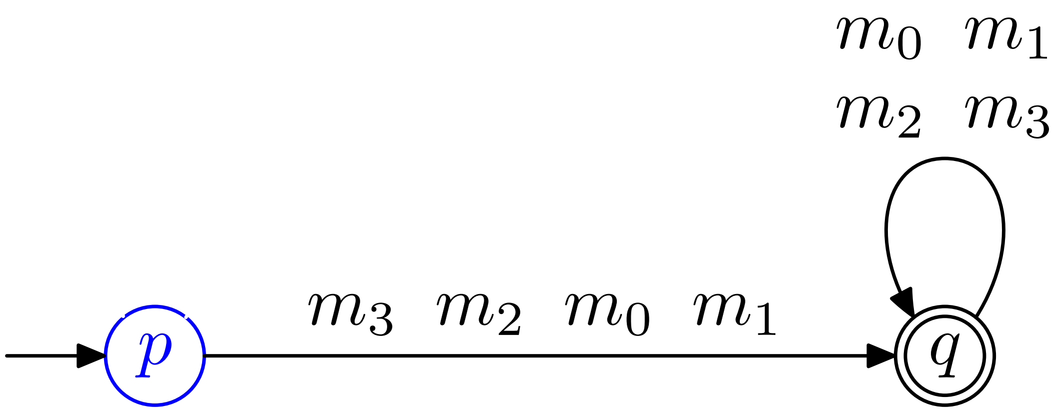
$r_1 : \quad m_0 \xhookrightarrow{a} m_1$

$r_2 : \quad m_0 \xhookrightarrow{b} m_2$

$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$

$r_4 : \quad m_2 \xhookrightarrow{a} m_3$

$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

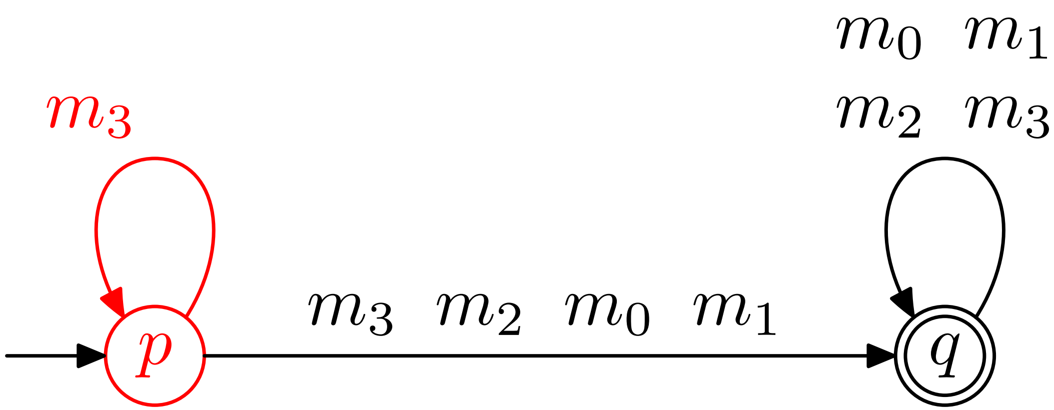
$r_1 : \quad m_0 \overset{a}{\hookrightarrow} m_1$

$r_2 : \quad m_0 \overset{b}{\hookrightarrow} m_2$

$r_3 : \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$

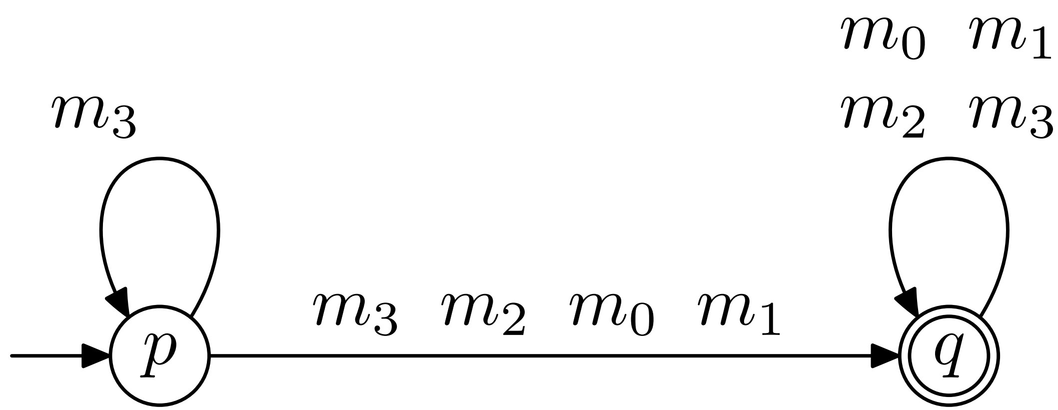
$r_4 : \quad m_2 \overset{a}{\hookrightarrow} m_3$

$r_5 : \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$
\begin{aligned}
r_1 &: \quad m_0 \xhookrightarrow{a} m_1 \\
r_2 &: \quad m_0 \xhookrightarrow{b} m_2 \\
r_3 &: \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3 \\
r_4 &: \quad m_2 \xhookrightarrow{a} m_3 \\
r_5 &: \quad m_3 \xhookrightarrow{\epsilon} \epsilon
\end{aligned}
$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

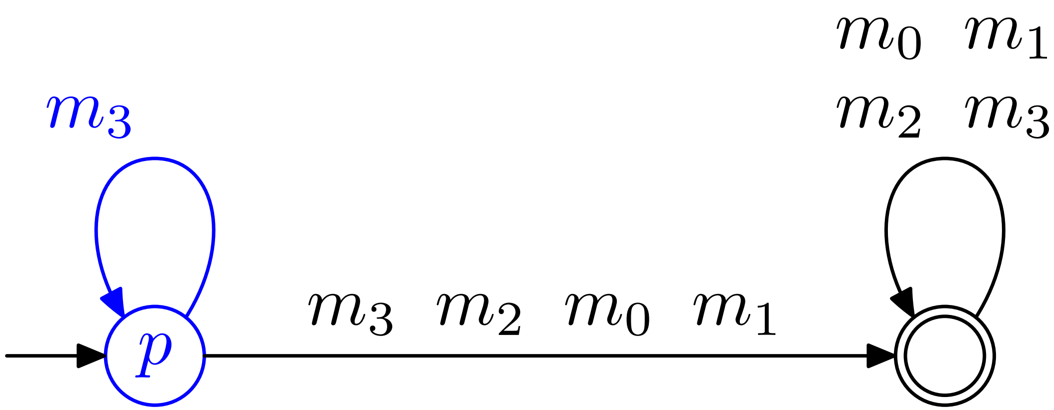
$$r_1 : \quad m_0 \xhookrightarrow{a} m_1$$

$$r_2 : \quad m_0 \xhookrightarrow{b} m_2$$

$$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$$

$$r_4 : \quad m_2 \xhookrightarrow{a} m_3$$

$$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

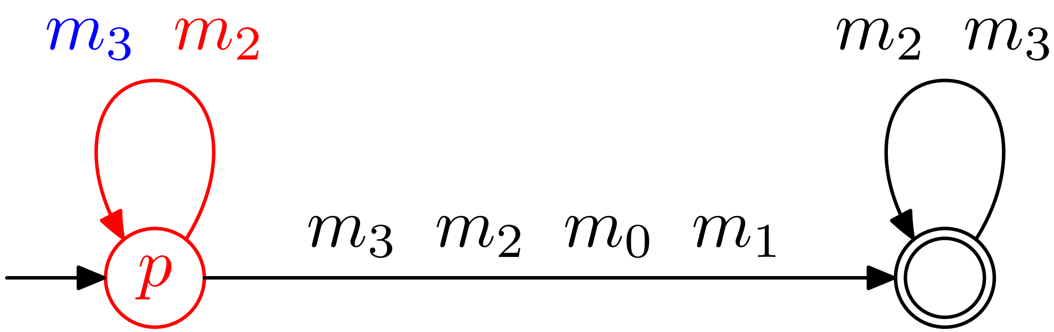
$r_1 : \quad m_0 \xhookrightarrow{a} m_1$

$r_2 : \quad m_0 \xhookrightarrow{b} m_2$

$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$

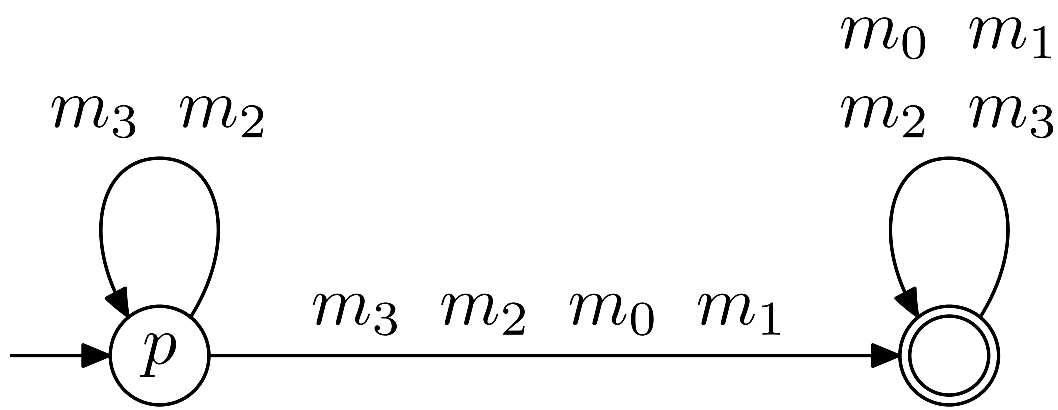
$r_4 : \quad m_2 \xhookrightarrow{a} m_3$

$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$r_1 : \quad m_0 \overset{a}{\hookrightarrow} m_1$$
$$r_2 : \quad m_0 \overset{b}{\hookrightarrow} m_2$$
$$r_3 : \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$$
$$r_4 : \quad m_2 \overset{a}{\hookrightarrow} m_3$$
$$r_5 : \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

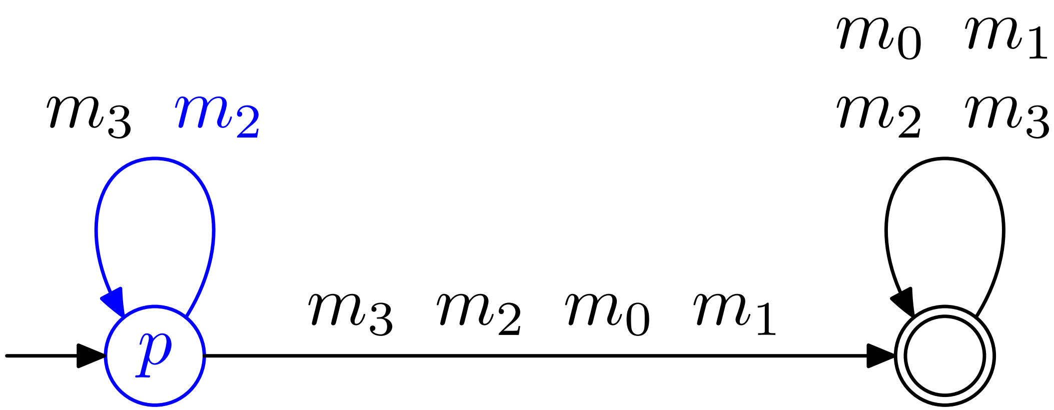
$r_1 : \quad m_0 \overset{a}{\hookrightarrow} m_1$

$r_2 : \quad m_0 \overset{b}{\hookrightarrow} m_2$

$r_3 : \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$

$r_4 : \quad m_2 \overset{a}{\hookrightarrow} m_3$

$r_5 : \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

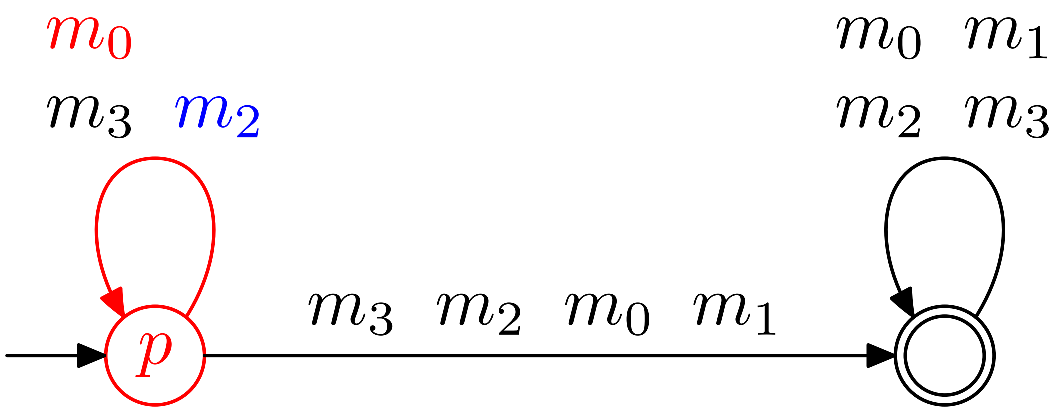
$$r_1: \quad m_0 \overset{a}{\hookrightarrow} m_1$$

$$r_2: \quad m_0 \overset{b}{\hookrightarrow} m_2$$

$$r_3: \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$$

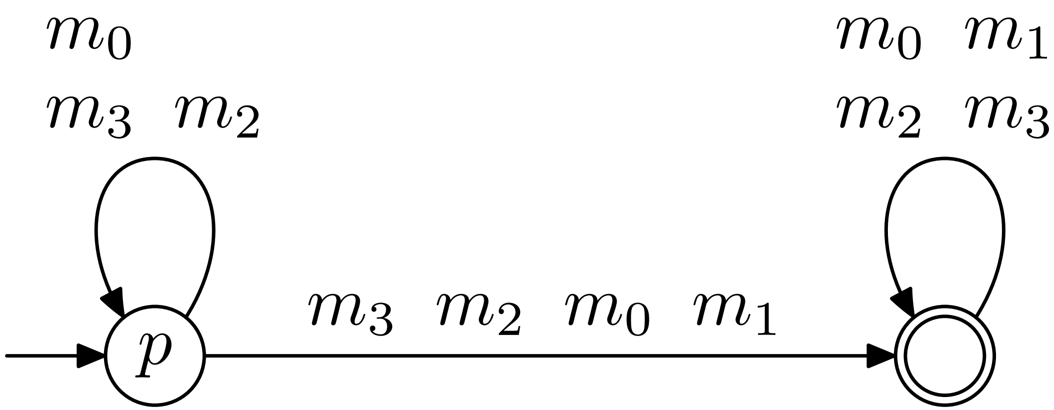
$$r_4: \quad m_2 \overset{a}{\hookrightarrow} m_3$$

$$r_5: \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

$$r_1 : \quad m_0 \xhookrightarrow{a} m_1$$
$$r_2 : \quad m_0 \xhookrightarrow{b} m_2$$
$$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$$
$$r_4 : \quad m_2 \xhookrightarrow{a} m_3$$
$$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

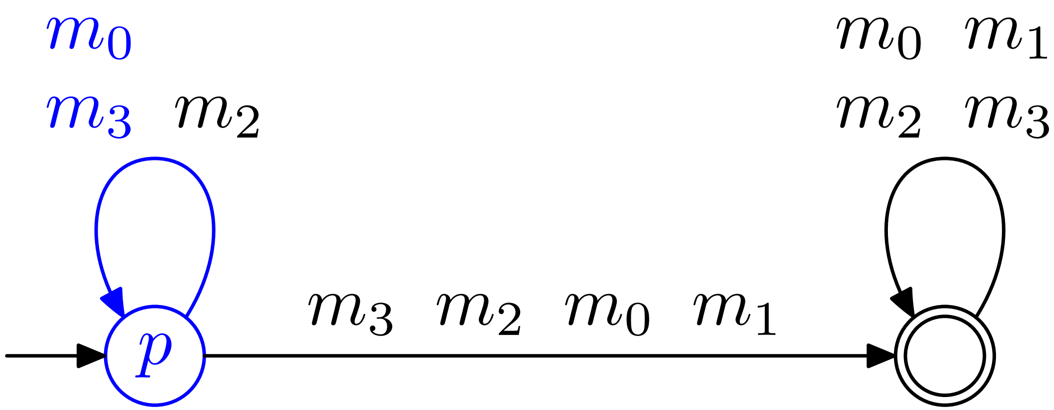
$$r_1 : \quad m_0 \overset{a}{\hookrightarrow} m_1$$

$$r_2 : \quad m_0 \overset{b}{\hookrightarrow} m_2$$

$$r_3 : \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$$

$$r_4 : \quad m_2 \overset{a}{\hookrightarrow} m_3$$

$$r_5 : \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

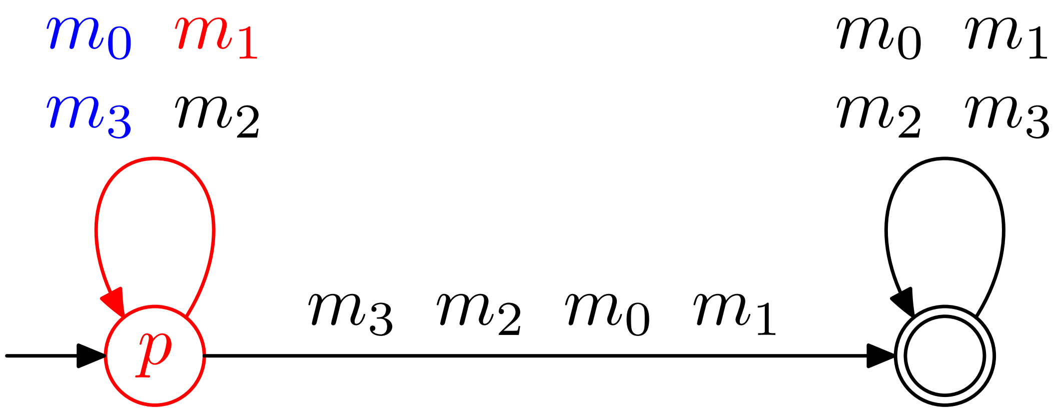
$$r_1 : \quad m_0 \xhookrightarrow{a} m_1$$

$$r_2 : \quad m_0 \xhookrightarrow{b} m_2$$

$$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$$

$$r_4 : \quad m_2 \xhookrightarrow{a} m_3$$

$$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$$



The saturation algorithm for computing $\mathcal{A}_{pre^*}$

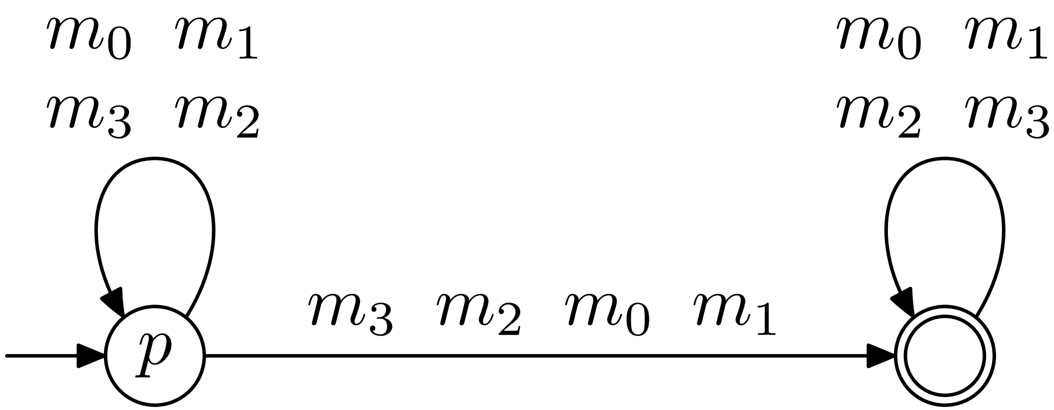
$r_1: \quad m_0 \overset{a}{\hookrightarrow} m_1$

$r_2: \quad m_0 \overset{b}{\hookrightarrow} m_2$

$r_3: \quad m_1 \overset{\epsilon}{\hookrightarrow} m_0 m_3$

$r_4: \quad m_2 \overset{a}{\hookrightarrow} m_3$

$r_5: \quad m_3 \overset{\epsilon}{\hookrightarrow} \epsilon$



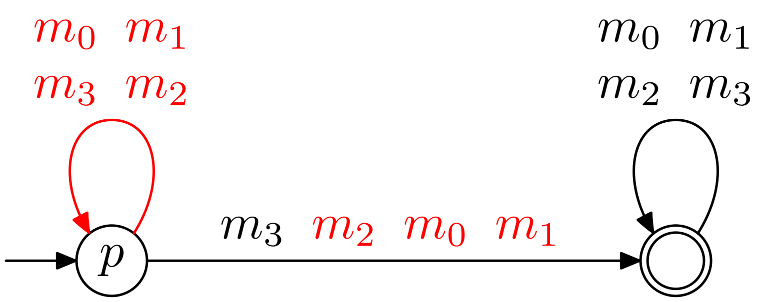
The approach once more . . .

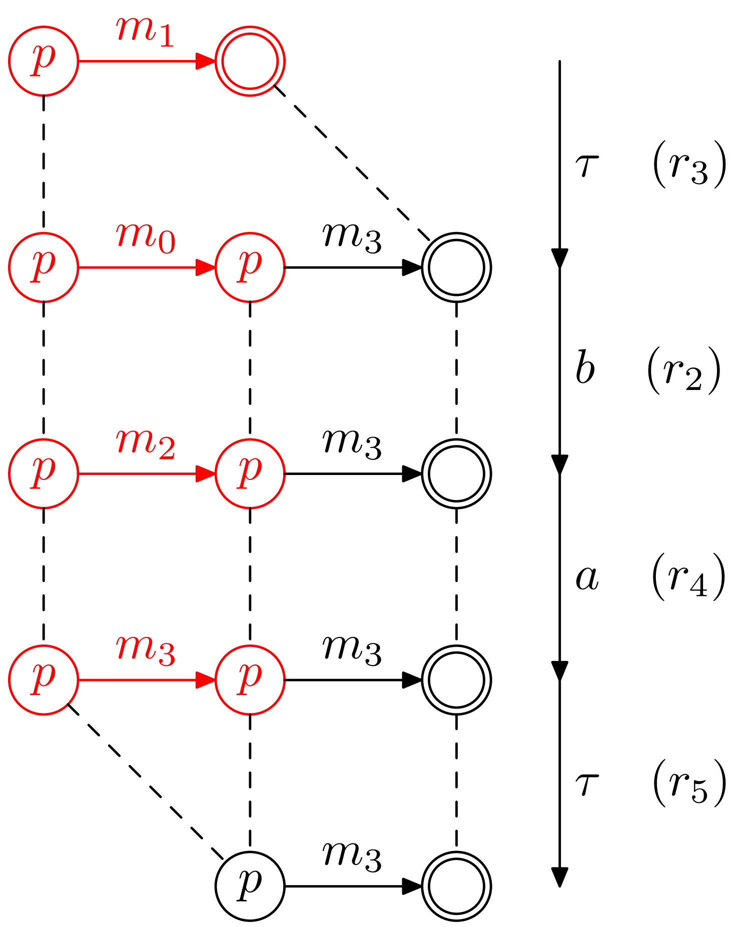
In order to compute $\alpha(L(c_0, F))$

- compute $pre^*(F) = \{c \mid \exists f \in F : c \rightarrow^* f\}$ $\surd$
- compute for each $c \in pre^*(F)$ the language $\alpha(L(c, F))$
- return $\alpha(L(c_0, F))$

How to compute $\alpha(L(c, F))$?

$$r_1 : \quad m_0 \xhookrightarrow{a} m_1$$
$$r_2 : \quad m_0 \xhookrightarrow{b} m_2$$
$$r_3 : \quad m_1 \xhookrightarrow{\epsilon} m_0 m_3$$
$$r_4 : \quad m_2 \xhookrightarrow{a} m_3$$
$$r_5 : \quad m_3 \xhookrightarrow{\epsilon} \epsilon$$





Idea: annotate each transition $t = q \xrightarrow{a} q'$ of $\mathcal{A}_{pre^*}$ with the language

$$L(t) = \text{ language transforming } t \text{ into a run of } \mathcal{A} \text{ ('black run')}$$

In our example: $ab \in L(p \xrightarrow{a} q)$

Given a run $\rho = t_1 t_2 \ldots t_n$, define $L(\rho) = L(t_1) \cdot \ldots \cdot L(t_n)$

It holds

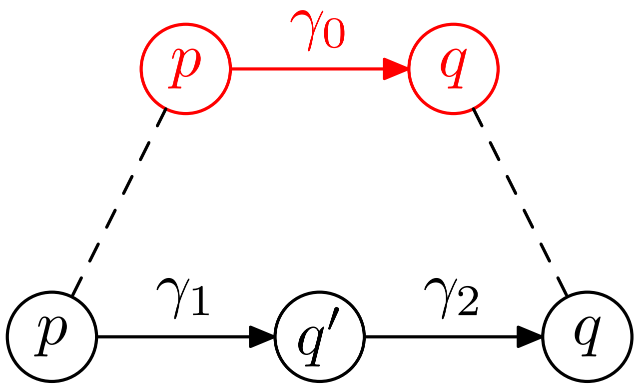
$$\begin{aligned} L(c, F) &= \text{union of } L(\rho) \text{ for all runs } \rho \text{ that recognize } c \\ \alpha(L(c, F)) &= \text{union of } \alpha(L(\rho)) \text{ for all transitions runs } \rho \text{ that recognize } c \end{aligned}$$

So it remains to show how to compute $\alpha(L(t))$

How to compute the languages $\alpha(L(t))$?

Assume we have

$$\gamma_0 \overset{a}{\hookrightarrow} \gamma_1\gamma_2$$



Then it holds

$$L(p \xrightarrow{\gamma_0} q) \;=\; a \;\cdot\; L(p \xrightarrow{\gamma_1} q') \;\cdot\; L(q' \xrightarrow{\gamma_2} q)$$

$$\alpha(L(p \xrightarrow{\gamma} q)) \;=\; \alpha(a) \;\odot\; \alpha(L(p \xrightarrow{\gamma_1} q')) \;\odot\; \alpha(L(q' \xrightarrow{\gamma_2} q))$$

where $\odot$ is defined by $\alpha(L_1 \cdot L_2) = \alpha(L_1) \odot \alpha(L_2)$
and $\oplus$ is defined by $\alpha(L_1 + L_2) = \alpha(L_1) \oplus \alpha(L_2)$

If we denote $d_i = \alpha(L(t_i))$, this yields equations of the form

$$f_i(d_1, \ldots, d_n) = d_i \quad 1 \leq i \leq n$$

where the $f_i's$ are polynomials constructed out of $d_1, \ldots, d_n$, $\odot$, and $\oplus$

$(d_1, \ldots d_n)$ is the least solution of

$$f_i(x_1, \ldots, x_n) = x_i \quad 1 \leq i \leq n$$

In our example

$$
\begin{array}{rclcl}
(x_4 \odot x_9) \oplus (x_8 \odot x_3) & = & x_1 & \qquad & p \xrightarrow{m_1} q \\
\alpha(a) \odot x_3 & = & x_2 & & p \xrightarrow{m_2} q \\
\alpha(\epsilon) & = & x_3 & & p \xrightarrow{m_3} q \\
(\alpha(a) \odot x_1) \oplus (\alpha(b) \odot x_2) & = & x_4 & & p \xrightarrow{m_0} q \\
x_8 \odot x_6 & = & x_5 & & p \xrightarrow{m_1} p \\
\alpha(\epsilon) & = & x_6 & & p \xrightarrow{m_3} p \\
\alpha(a) \odot x_6 & = & x_7 & & p \xrightarrow{m_2} p \\
(\alpha(b) \odot x_7) \oplus (\alpha(a) \odot x_5) & = & x_8 & & p \xrightarrow{m_0} p \\
\alpha(\epsilon) & = & x_9 & & q \xrightarrow{m_0, m_1, m_2, m_3} q
\end{array}
$$

How to solve the system of equations

I’m not proud of the books I’ve written,
but of the books I’ve read.
Jorge Luis Borges

In the non-commutative case: No closed form solution

In the commutative case: Beautiful solution by Hopkins and Kozen, LICS ’99

Hopkins and Kozen's procedure

$\mathcal{L}$ is a Kleene algebra: $\cdot$, $+$, $*$, $\overline{0} = \emptyset$, $\overline{1} = \{\epsilon\}$

$\mathcal{D}$ is a commutative Kleene algebra: $\odot$, $\oplus$, $\star$, $\overline{0} = \bot$, $\overline{1} = \alpha(\epsilon)$,
where $d^* = \bigoplus_{n \geq 0} d^n$

Define the partial differential operator $\frac{\partial}{\partial x_i}$ by

- $\frac{\partial x_i}{\partial x_i} = \overline{1}$, $\frac{\partial x_j}{\partial x_i} = \overline{0}$ for $i \neq j$, and $\frac{\partial a}{\partial x_i} = \overline{0}$ for $a \in D$.
- $\frac{\partial}{\partial x_i}(f \oplus g) = \frac{\partial f}{\partial x_i} \oplus \frac{\partial g}{\partial x_i}$
- $\frac{\partial}{\partial x_i}(f \odot g) = (f \odot \frac{\partial g}{\partial x_i}) \oplus (\frac{\partial f}{\partial x_i} \odot g)$
- $\frac{\partial}{\partial x_i}(f^\star) = f^\star \odot \frac{\partial f}{\partial x_i}$

Rejoice in this . . .

The least solution of

$$f_i(x_1, \ldots, x_n) \leq x_i \quad 1 \leq i \leq n$$

is the fixpoint of the chain

$$\mathbf{d_0} \leq \mathbf{d_1} \leq \mathbf{d_2} \ldots$$

given by

$$\begin{aligned} \mathbf{d_0} &= \mathbf{f}(\overline{\mathbf{0}}) \\ \mathbf{d_{k+1}} &= \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{d_k})^\star \odot \mathbf{d_k}, \end{aligned}$$

The solution

$$\begin{aligned} d_1 = d_5 = d_8 &= \alpha(a)^\star \odot \alpha(a) \odot \alpha(b) \\ d_3 = d_6 = d_9 &= \alpha(\epsilon) \\ d_2 = d_7 &= \alpha(a) \\ d_4 &= (\alpha(a) \odot \alpha(a)^\star \odot \alpha(a) \odot \alpha(b)) \oplus (\alpha(a) \odot \alpha(b)) \end{aligned}$$

Instantiating the solution

$$\begin{array}{lllllllll}
\alpha_1(a) & = & [\{b\},\{a\}] & \alpha_1(b) & = & [\{b\},\{a\}] & \alpha_1(\epsilon) & = & [Act,\emptyset] \\
\alpha_2(a) & = & \{\{a\}\} & \alpha_2(b) & = & \{\{b\}\} & \alpha_2(\epsilon) & = & \{\emptyset\} \\
\alpha_3(a) & = & \{(1,0)\} & \alpha_3(b) & = & \{(0,1)\} & \alpha_3(\epsilon) & = & \{(0,0)\}
\end{array}$$

Let $c_0 = m_0$ and $F = m_3\Gamma^*$. Then $\alpha(L(c_0, F)) = d_4$

$$\begin{array}{llll}
d_{41} & = & [\emptyset, \{a,b\}] & \text{(a and b required to reach m_3)} \\
d_{42} & = & \{\{a,b\}\} & \text{($\{a,b\}$ only possible alphabet to reach m_3)} \\
d_{43} & = & \{(k,1) \mid k \geq 1\} & \text{(only one b, otherwise not possible)}
\end{array}$$

Complexity of the fixpoint algorithm

The complexity is $O(r^3 \cdot t \cdot c)$, where

- r is the number of rules of the pushdown system
- t is the number of iterations of the fixpoint algorithm
- c is the maximal cost of an $\oplus, \odot, \star$ operation

Forbidden and required sets: $O(Act \cdot r^3)$

Alphabets: $2^{O(Act)} \cdot r^3$

Parikh images (worst case): double exponential in r

Conclusions

Extension of our automata-theoretic approach to interprocedural analysis

Automatic, possible to trade efficiency for precision

Generic fixpoint algorithm, which can be also generically implemented

To be implemented as an extension of Moped (Schwoon)