An introduction to infinite graphs

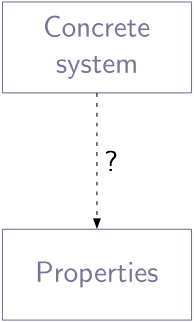
Antoine Meyer

Formal Methods Update 2006
IIT Guwahati

1. Foreword: approach and current issues
2. Pushdown graphs: characterizations
3. Reachability in pushdown graphs
4. Beyond pushdown graphs

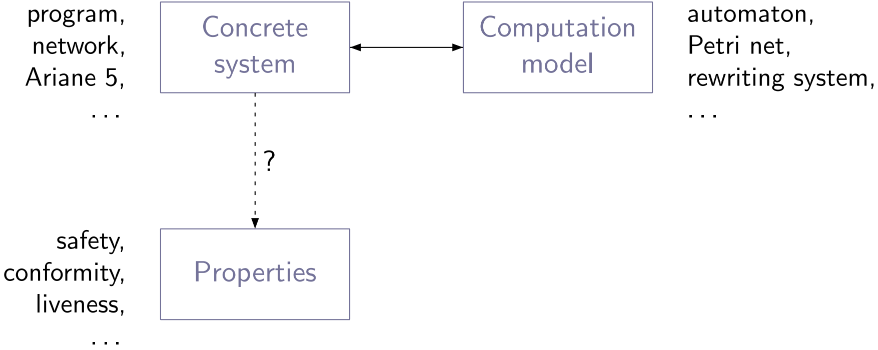
Reasoning about computation

program,
network,
Ariane 5,
...

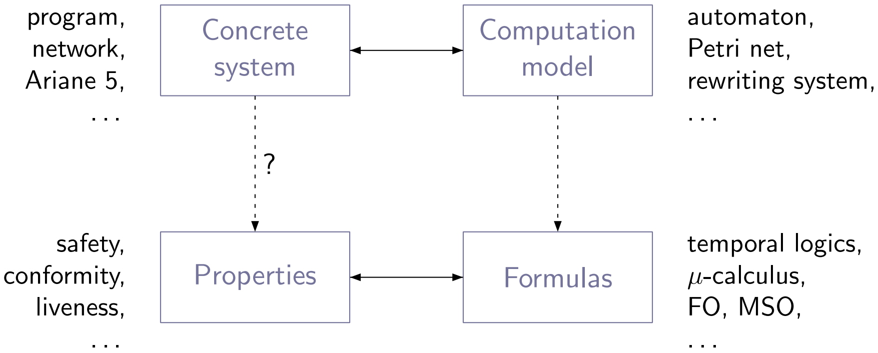


safety,
conformity,
liveness,
...

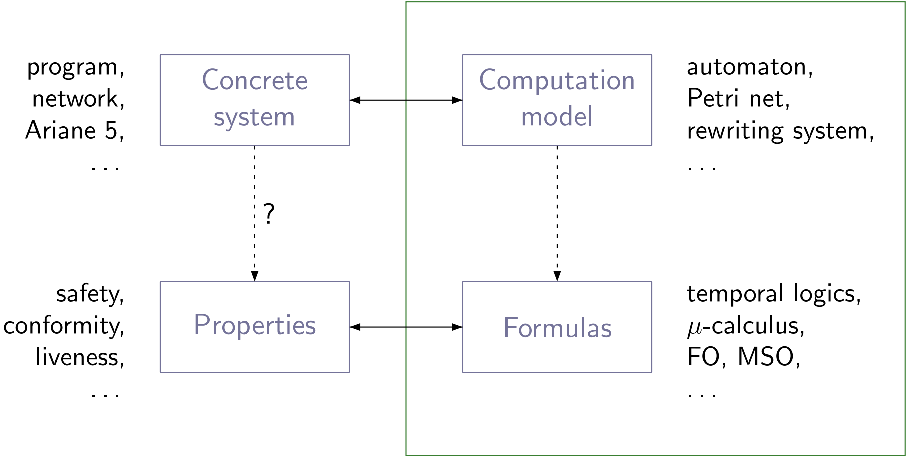
Reasoning about computation



Reasoning about computation



Reasoning about computation



A few current issues

- Successful technique: *finite* models (circuits, protocols)
  - Adopted by the industry (Intel, IBM, Motorola ...)
- New issues: *software* verification
  - Difficulties: data, dynamic evolution ...
  - Specific constraints: reliability, limited resources ...
  - Coexistence of several aspects ("complex" systems)

    Ex: embedded systems

$\Rightarrow$ Need for more elaborate models and algorithms

The modelling problem

- Wide range of models
  - Mostly from language, automata and rewriting theory
- Tradeoff btw. expressiveness and decidability/complexity

Finite automata		Turing machines
most restricted	$\longleftrightarrow$	*most expressive*
mostly decidable		*mostly undecidable*

- Abstraction/approximation usually required
  - Infinite domains, unbounded recursion, time, etc.

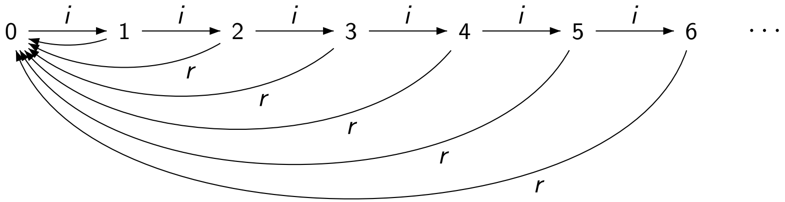
A few simple examples

Configuration: one (unbounded) integer counter
Operations: increment (i)

$$0 \xrightarrow{i} 1 \xrightarrow{i} 2 \xrightarrow{i} 3 \xrightarrow{i} 4 \xrightarrow{i} 5 \xrightarrow{i} 6 \quad \cdots$$

A few simple examples

Configuration: one (unbounded) integer counter
Operations: increment (i) and reset (r)



A few simple examples

Configuration: two (unbounded) integer counters
Operations: increments (i_1, i_2)

$$
\begin{tikzcd}
0,0 \arrow[r, "i_1"] \arrow[d, "i_2"] & 1,0 \arrow[r, "i_1"] \arrow[d, "i_2"] & 2,0 \arrow[r, "i_1"] \arrow[d, "i_2"] & 3,0 \arrow[r, "i_1"] \arrow[d, "i_2"] & 4,0 \arrow[r, "i_1"] \arrow[d, "i_2"] & 5,0 \arrow[r, "i_1"] \arrow[d, "i_2"] & 6,0 \arrow[d, "i_2"] & \cdots \\
0,1 \arrow[r, "i_1"] \arrow[d, "i_2"] & 1,1 \arrow[r, "i_1"] \arrow[d, "i_2"] & 2,1 \arrow[r, "i_1"] \arrow[d, "i_2"] & 3,1 \arrow[r, "i_1"] \arrow[d, "i_2"] & 4,1 \arrow[r, "i_1"] \arrow[d, "i_2"] & 5,1 \arrow[r, "i_1"] \arrow[d, "i_2"] & 6,1 \arrow[d, "i_2"] & \cdots \\
0,2 \arrow[r, "i_1"] & 1,2 \arrow[r, "i_1"] & 2,2 \arrow[r, "i_1"] & 3,2 \arrow[r, "i_1"] & 4,2 \arrow[r, "i_1"] & 5,2 \arrow[r, "i_1"] & 6,2 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots &
\end{tikzcd}
$$

Structural study of infinite graphs

General objective

Systematic structural study of families of infinite graphs

- Infinite graphs induced by classical computation models
- Alternative characterizations of each family
- Focus on closure properties, logics, trace languages and structural and algorithmic properties
- Abstraction from concrete systems ensures reusability

This talk

Overview through the example of pushdown graphs
(references: mostly Büchi, Caucal, Courcelle, Muller & Schupp)

An introduction to infinite graphs

Antoine Meyer

Formal Methods Update 2006
IIT Guwahati

Characterization 1:
A pushdown system transition graph

Pushdown systems

A pushdown system consists in

- Control states $p, q, \ldots \in Q$
- Stack symbols $A, B, C, \ldots \in \Gamma$
- Label alphabet $a, b, c, \ldots \in \Sigma$
- Transitions of the form

$$p, A \xrightarrow{a} q, \begin{cases} \text{push}B \\ \text{pop} \end{cases}$$

Pushdown systems

A pushdown system consists in

- Control states $p, q, \ldots \in Q$ (global variables, registers)
- Stack symbols $A, B, C, \ldots \in \Gamma$ (local vars., program counter)
- Label alphabet $a, b, c, \ldots \in \Sigma$ (program interactions)
- Transitions of the form

$$p, A \xrightarrow{a} q, \begin{cases} \mathsf{push}\, B & \text{(procedure call)} \\ \mathsf{pop} & \text{(procedure return)} \end{cases}$$

Classical abstract model for recursive sequential programs

Pushdown systems (2)

A configuration is a pair (p, s) with

- p a control state
- s a sequence of stack symbols (top first)

Example

$$p, \emptyset \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{b} q, \text{pop}$$

$$q, A \xrightarrow{b} q, \text{pop}$$

Pushdown systems (2)

A configuration is a pair (p, s) with

- p a control state
- s a sequence of stack symbols (top first)

Example

$p, \emptyset$

$$p, \emptyset \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{b} q, \text{pop}$$

$$q, A \xrightarrow{b} q, \text{pop}$$

Pushdown systems (2)

A configuration is a pair (p, s) with

- p a control state
- s a sequence of stack symbols (top first)

Example

$$p, \emptyset \xrightarrow{a} p, A$$

$p, \emptyset \xrightarrow{a} p, \text{push } A$

$p, A \xrightarrow{a} p, \text{push } A$

$p, A \xrightarrow{b} q, \text{pop}$

$q, A \xrightarrow{b} q, \text{pop}$

Pushdown systems (2)

A configuration is a pair (p, s) with

- p a control state
- s a sequence of stack symbols (top first)

Example

$$p, \emptyset \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{b} q, \text{pop}$$

$$q, A \xrightarrow{b} q, \text{pop}$$

$$\begin{tikzcd}
p, \emptyset \arrow[r, "a"] & p, A \arrow[d, "b"] \\
& q, \emptyset
\end{tikzcd}$$

Pushdown systems (2)

A configuration is a pair (p, s) with

- p a control state
- s a sequence of stack symbols (top first)

Example

$$p, \emptyset \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{b} q, \text{pop}$$

$$q, A \xrightarrow{b} q, \text{pop}$$

$$\begin{tikzcd}
p, \emptyset \arrow[r, "a"] & p, A \arrow[r, "a"] \arrow[d, "b"] & p, AA \\
& q, \emptyset &
\end{tikzcd}$$

Pushdown systems (2)

A configuration is a pair (p, s) with

- p a control state
- s a sequence of stack symbols (top first)

Example

$$p, \emptyset \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{b} q, \text{pop}$$

$$q, A \xrightarrow{b} q, \text{pop}$$

$$\begin{tikzcd}
p, \emptyset \arrow[r, "a"] & p, A \arrow[r, "a"] \arrow[d, "b"] & p, AA \arrow[d, "b"] \\
& q, \emptyset & q, A
\end{tikzcd}$$

Pushdown systems (2)

A configuration is a pair (p, s) with

- p a control state
- s a sequence of stack symbols (top first)

Example

$$p, \emptyset \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{b} q, \text{pop}$$

$$q, A \xrightarrow{b} q, \text{pop}$$

$$\begin{tikzcd}
p, \emptyset \arrow[r, "a"] & p, A \arrow[r, "a"] \arrow[d, "b"] & p, AA \arrow[d, "b"] \\
 & q, \emptyset & q, A \arrow[l, "b"]
\end{tikzcd}$$

Pushdown systems (2)

A configuration is a pair (p, s) with

- p a control state
- s a sequence of stack symbols (top first)

Example

$$p, \emptyset \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{a} p, \text{push } A$$

$$p, A \xrightarrow{b} q, \text{pop}$$

$$q, A \xrightarrow{b} q, \text{pop}$$

$$\begin{tikzcd}
p, \emptyset \arrow[r, "a"] & p, A \arrow[r, "a"] \arrow[d, "b"] & p, AA \arrow[r, "a"] \arrow[d, "b"] & p, AAA \arrow[r, "a"] \arrow[d, "b"] & \cdots \\
 & q, \emptyset & q, A \arrow[l, "b"'] & q, AA \arrow[l, "b"'] & \cdots \arrow[l, "b"']
\end{tikzcd}$$

Such graphs are called pushdown graphs

General form: prefix rewriting graphs

- Rewriting system: set of rules $l \xrightarrow{a} r$
- Prefix rewriting: $lu \xrightarrow{a} ru$ whenever $l \xrightarrow{a} r$

Example

$$\begin{aligned} A &\xrightarrow{a} AA \\ AA &\xrightarrow{a} B \\ BA &\xrightarrow{b} B \end{aligned}$$

General form: prefix rewriting graphs

- Rewriting system: set of rules $l \xrightarrow{a} r$
- Prefix rewriting: $lu \xrightarrow{a} ru$ whenever $l \xrightarrow{a} r$

Example

A

$$A \xrightarrow{a} AA$$

$$AA \xrightarrow{a} B$$

$$BA \xrightarrow{b} B$$

General form: prefix rewriting graphs

- Rewriting system: set of rules $l \xrightarrow{a} r$
- Prefix rewriting: $lu \xrightarrow{a} ru$ whenever $l \xrightarrow{a} r$

Example

$$A \xrightarrow{a} AA$$

$$A \xrightarrow{a} AA$$

$$AA \xrightarrow{a} B$$

$$BA \xrightarrow{b} B$$

General form: prefix rewriting graphs

- Rewriting system: set of rules $l \xrightarrow{a} r$
- Prefix rewriting: $lu \xrightarrow{a} ru$ whenever $l \xrightarrow{a} r$

Example

$$A \xrightarrow{a} AA$$

$$AA \xrightarrow{a} B$$

$$BA \xrightarrow{b} B$$

$$\begin{tikzcd} A \arrow[r, "a"] & AA \arrow[d, "b"] \\ & B \end{tikzcd}$$

General form: prefix rewriting graphs

- Rewriting system: set of rules $l \xrightarrow{a} r$
- Prefix rewriting: $lu \xrightarrow{a} ru$ whenever $l \xrightarrow{a} r$

Example

$$A \xrightarrow{a} AA$$
$$AA \xrightarrow{a} B$$
$$BA \xrightarrow{b} B$$

$$\begin{tikzcd}
A \arrow[r, "a"] & AA \arrow[r, "a"] \arrow[d, "b"] & AAA \\
& B &
\end{tikzcd}$$

General form: prefix rewriting graphs

- Rewriting system: set of rules $l \xrightarrow{a} r$
- Prefix rewriting: $lu \xrightarrow{a} ru$ whenever $l \xrightarrow{a} r$

Example

$$A \xrightarrow{a} AA$$
$$AA \xrightarrow{a} B$$
$$BA \xrightarrow{b} B$$

$$\begin{tikzcd}
A \arrow[r, "a"] & AA \arrow[r, "a"] \arrow[d, "b"] & AAA \arrow[d, "b"] \\
& B & BA
\end{tikzcd}$$

General form: prefix rewriting graphs

- Rewriting system: set of rules $l \xrightarrow{a} r$
- Prefix rewriting: $lu \xrightarrow{a} ru$ whenever $l \xrightarrow{a} r$

Example

$$A \xrightarrow{a} AA$$

$$AA \xrightarrow{a} B$$

$$BA \xrightarrow{b} B$$

$$\begin{tikzcd}
A \arrow[r, "a"] & AA \arrow[r, "a"] \arrow[d, "b"] & AAA \arrow[d, "b"] \\
 & B & BA \arrow[l, "b"]
\end{tikzcd}$$

General form: prefix rewriting graphs

- Rewriting system: set of rules $l \xrightarrow{a} r$
- Prefix rewriting: $lu \xrightarrow{a} ru$ whenever $l \xrightarrow{a} r$

Example

$$A \xrightarrow{a} AA$$
$$AA \xrightarrow{a} B$$
$$BA \xrightarrow{b} B$$

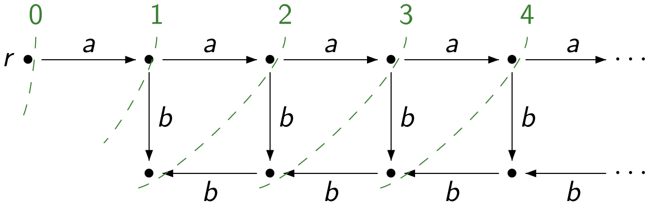
$$\begin{tikzcd}
A \arrow[r, "a"] & AA \arrow[r, "a"] \arrow[d, "b"] & AAA \arrow[r, "a"] \arrow[d, "b"] & AAAA \arrow[r, "a"] \arrow[d, "b"] & \cdots \\
& B & BA \arrow[l, "b"] & BAA \arrow[l, "b"] & \cdots \arrow[l, "b"]
\end{tikzcd}$$

Note: not all vertices are considered,
only vertices of the form $\{A, B\}A^*$
(regular restriction)

Characterization 2:
Building a graph with a grammar

Decomposition by distance

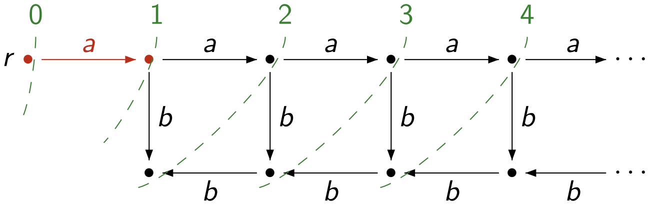
- Consider the distance of any vertex to vertex r:



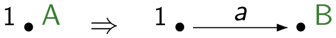
- Build a finite graph grammar using this decomposition

Decomposition by distance

- Consider the distance of any vertex to vertex r:

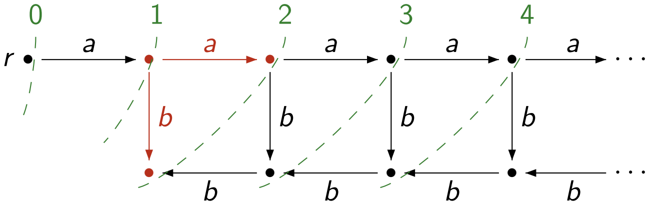


- Build a finite graph grammar using this decomposition

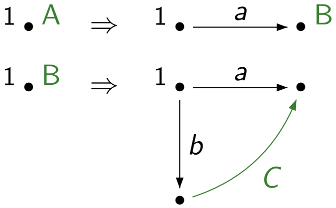


Decomposition by distance

- Consider the distance of any vertex to vertex r:

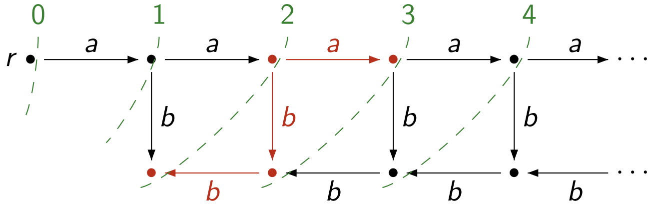


- Build a finite graph grammar using this decomposition

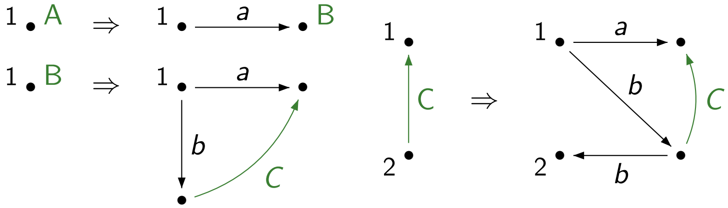


Decomposition by distance

- Consider the distance of any vertex to vertex r:

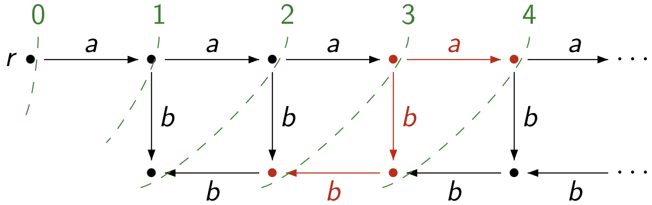


- Build a finite graph grammar using this decomposition

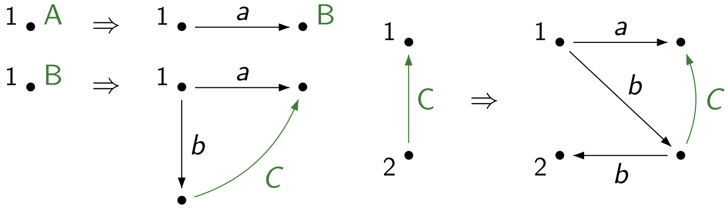


Decomposition by distance

- Consider the distance of any vertex to vertex r:



- Build a finite graph grammar using this decomposition



Characterization 3:
Transforming a simpler graph

Unfold and substitute

Idea

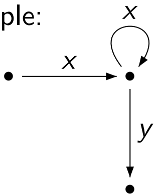
Starting from a family of generators, characterize new graphs by applying transformations

Present case

- Generator: a finite graph
- First transformation: unfold from a vertex,
- Second transformation: substitute paths with edges

Unfold and substitute (2)

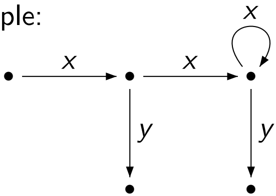
- Example:



- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

Unfold and substitute (2)

- Example:



- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

Unfold and substitute (2)

- Example:

$$\begin{tikzcd}
\bullet \arrow[r, "x"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[d, "y"] \arrow[loop above, "x"] \\
& \bullet & \bullet & \bullet
\end{tikzcd}$$

- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

Unfold and substitute (2)

- Example:

$$\begin{tikzcd}
\bullet \arrow[r, "x"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[d, "y"] \arrow[loop above, "x"] \\
 & \bullet & \bullet & \bullet & \bullet
\end{tikzcd}$$

- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

Unfold and substitute (2)

- Example:

$$\begin{tikzcd}
\bullet \arrow[r, "x"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \cdots \\
& \bullet & \bullet & \bullet & \bullet &
\end{tikzcd}$$

- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

Unfold and substitute (2)

- Example:

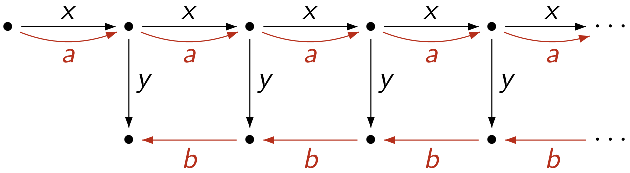
$$\begin{tikzcd}
\bullet \arrow[r, "x"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \bullet \arrow[r, "x"] \arrow[d, "y"] & \cdots \\
& \bullet & \bullet \arrow[l, "b"] & \bullet \arrow[l, "b"] & \bullet \arrow[l, "b"] & \cdots \arrow[l, "b"]
\end{tikzcd}$$

- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

$$\xleftarrow{y}\ \xleftarrow{x}\ \xrightarrow{y} \quad \text{becomes} \quad \xrightarrow{b}$$

Unfold and substitute (2)

- Example:



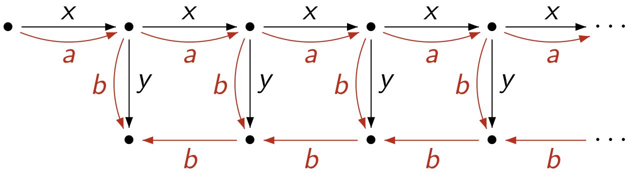
- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

$$\xleftarrow{y}\ \xleftarrow{x}\ \xrightarrow{y} \text{ becomes } \xrightarrow{b}$$

$$\xrightarrow{x} \text{ becomes } \xrightarrow{a}$$

Unfold and substitute (2)

- Example:



- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

$$\xleftarrow{y}\ \xleftarrow{x}\ \xrightarrow{y} \text{ becomes } \xrightarrow{b}$$

$$\xrightarrow{x} \text{ becomes } \xrightarrow{a}$$

$$\xrightarrow{y} \text{ becomes } \xrightarrow{b}$$

Unfold and substitute (2)

- Example:

$$
\begin{tikzcd}
\bullet \arrow[r, "a"] & \bullet \arrow[r, "a"] \arrow[d, "b"'] & \bullet \arrow[r, "a"] \arrow[d, "b"'] & \bullet \arrow[r, "a"] \arrow[d, "b"'] & \bullet \arrow[r, "a"] \arrow[d, "b"'] & \cdots \\
 & \bullet & \bullet \arrow[l, "b"] & \bullet \arrow[l, "b"] & \bullet \arrow[l, "b"] & \cdots \arrow[l, "b"]
\end{tikzcd}
$$

- Start with a finite graph
- Unfold it from its root
- Substitute paths with edges

$$\xleftarrow{y}\ \xleftarrow{x}\ \xrightarrow{y} \quad \text{becomes} \quad \xrightarrow{b}$$

$$\xrightarrow{x} \quad \text{becomes} \quad \xrightarrow{a}$$

$$\xrightarrow{y} \quad \text{becomes} \quad \xrightarrow{b}$$

Equivalence result

Theorem

Let G be a connected graph of finite degree, the following statements are equivalent (up to isomorphism):

- G is the transition graph of a pushdown automaton
- G is the prefix rewriting graph of a finite rewriting system
- G has a finite decomposition by distance from any vertex
- G is generated by a deterministic graph grammar
- G is the result of unfolding a finite graph and applying a finite substitution

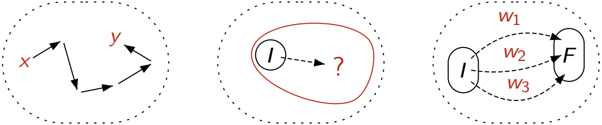
An introduction to infinite graphs

Antoine Meyer

Formal Methods Update 2006
IIT Guwahati

Reachability analysis

- Fundamental questions for most applications
- Several variants:



Reachability relations | Reachable configurations | Path languages

- *Examples :*
  - "Is it possible to reach a deadlock state?"
  - "Is it always possible to reach a target state?"
  - "Is every request eventually answered?"
- Path languages of pushdown graphs: context-free languages

Reachability in pushdown graphs (1)

Notations

- For every symbol A, we write
  - $\bar{A}$ the action of removing a prefix A ("pop" A)
  - A the action of adding a prefix A ("push" A)

  This notation is extended to words: $\overline{Au} = \bar{A}\bar{u}$

  Example: $\overline{AB} = \bar{A}\bar{B}$

- The mirror of a word u is written $\widetilde{u}$ (with $\widetilde{Au} = \widetilde{u}A$ and $\widetilde{\varepsilon} = \varepsilon$)

  Example: $\widetilde{AB} = \widetilde{B}A = BA$

Reachability in pushdown graphs (2)

Representation of rules

- The effect of rewrite rule $u \xrightarrow{a} v$ can be written $\bar{u}\widetilde{v}$
- The effect of any rewriting system R can be represented by the regular language $\{\bar{u}\widetilde{v} \mid u \xrightarrow{a} v \in R\}^*$
- Conversely, any word $\bar{u}\widetilde{v}$ can be seen as a rewrite rule $u \rightarrow v$
- Any regular language $L \subseteq \bar{N}^* N^*$ can be seen as an infinite recognizable rewriting system $R = \{u \rightarrow v \mid \bar{u}\widetilde{v} \in L\}$
- Alternative notation:
  If $L = \bigcup_i \overline{U_i}\,\widetilde{V_i}$, we write $R = \bigcup_i U_i \rightarrow V_i$

Reachability in pushdown graphs (3)

Observation: Sequences of the form $A\bar{A}$ can be discarded

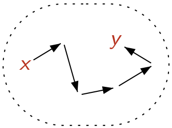
Algorithm

1. Start with automaton accepting $L_R = \{\bar{u}\widetilde{v} \mid u \xrightarrow{a} v \in R\}^*$
2. Aim: remove all "unnecessary steps" of the form $A\bar{A}$
   $\hookrightarrow$ For all path $p \xrightarrow{A\bar{A}} q$ in $\mathcal{A}$, add ε-transition $p \xrightarrow{\varepsilon} q$
3. Iterate the previous step until saturation
4. Intersect the obtained language with $\bar{N}^* N^*$
5. Interpret as a union of relations $U \to V$

Reachability in pushdown graphs (4)

Theorem (Caucal, Dauchet&Tison)

The reachability relation in a prefix rewriting graph can always be written as a finite union of relations $(U \to V) \cdot \mathsf{Id}$, with U, V regular



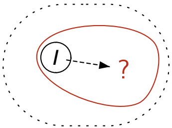
Note:

- When using regular restrictions, more general form needed: finite union of relations $(U \to V) \cdot \mathsf{Id}_W$
- Such relations are called prefix-recognizable

Reachability in pushdown graphs (5)

Corollary

The set of vertices F reachable from any regular set I in a pushdown graph is regular



Other possible interpretation:

Corollary (Büchi)

The set of stack contents reachable from the initial configuration in a pushdown automaton is a regular language

An introduction to infinite graphs
Antoine Meyer

Formal Methods Update 2006
IIT Guwahati

Prefix-recognizable graphs

Q: Prefix-recognizable relations express reachability in prefix rewriting graphs. What about graphs whose edges are P-R?

A: Extension of (nearly) *all* previous characterizations

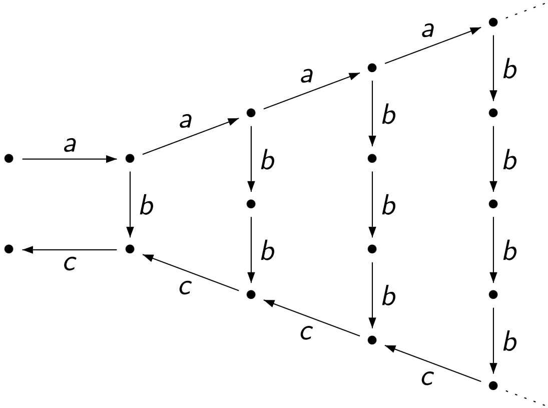
Theorem

Let G be a graph, the following statements are equivalent:

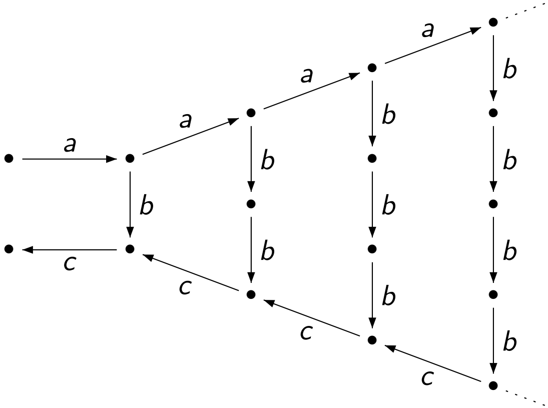
- *G is the transition graph of a pushdown automaton with ε-transitions*
- *G is the prefix rewriting graph of a recognizable rewriting system (+ regular restriction)*
- *G is the result of unfolding a finite graph and applying a regular substitution*

Additionally, *all* previous reachability results remain true

A graph



Not a pushdown graph!



This graph “looks” regular though, how can we characterize it?

Extensions and variants

- More general computation models
  (e.g. linearly bounded machines, petri nets)
- Arbitrary finitely presented binary relations
  (e.g. automatic or rational relations)
- More powerful or iterated transformations
- New operators in graph equations (or grammars)
- Restrictions or specialization of existing families
  (e.g. degree, tree-width, connectedness)

Conclusion

- Open topic with numerous variants and extensions
- Links with other theoretical topics
  Language theory, automata, rewriting, logics . . .
- (Prospective) applications in computer science
  Modeling (notion of structural richness)
  Verification (through logics and algorithms)