Tree Automata and Program Verification

Denis Lugiez

LIF UMR 6166-CNRS-Aix-Marseille Université

Workshop in honour of Hubert COMON

T(ree)A(utomata)
T(heir) A(pplications)

A Collaborative Editing Process issued from a CNRS action

- Lille (LIFL) M.Dauchet, R. Gilleron, S.Tison, M. Tomasi
- Nancy/Marseille (LORIA/LIF) D. Lugiez
- Paris (LRI/LSV) H.Comon, F. Jacquemard
- Aachen (RWTH) C.Loeding

Useful ?

In this talk :

- ➡ Chapter 1 : Basics
- ➡ Chapter 2 : Tree Grammars
- ➡ Chapter 3 : Tree automata and Relations
- ➡ Chapter 8 : Automata on Unranked Trees
- ➡ Chapter 9 : Automata on Unranked Unordered Trees

Useful ?

In this talk :

- ➡ Chapter 1 : Basics
- ➡ Chapter 2 : Tree Grammars
- ➡ Chapter 3 : Tree automata and Relations
- ➡ Chapter 8 : Automata on Unranked Trees
- ➡ Chapter 9 : Automata on Unranked Unordered Trees (Yes not yet available)

The main point :

(Word) Tree automata

are useful for

program analysis

Proviso :

don't do what Hubert likes to do -)

Proviso :

don't do what Hubert likes to do -)

add constraints between variables

Introduction

Word Automata and Pushdown Systems

Tree Automata and Process Algebra

Presburger/Hedge Automata and Multithreaded Programs

Introduction

Word Automata and Pushdown Systems

Tree Automata and Process Algebra

Presburger/Hedge Automata and Multithreaded Programs

Program Analysis

Detect statically bad behaviors of programs

- Model program behavior.
  - Configuration given by some formal object C
  - Transition relation between configurations $\rightarrow$
  - Initial configurations $\mathcal{I}$
  - Bad configurations $\mathcal{B}$
  - Set of successors $Post^*(L) = \{C \mid L \ni C_0 \xrightarrow{*} C\}$
  - Set of predecessors $Pre^*(L) = \{C \mid C \xrightarrow{*} C \in L\}$
- Safety Analysis :
  - $Post^*(\mathcal{I}) \cap \mathcal{B} = \emptyset$ (forward analysis)
  - $Pre^*(\mathcal{B}) \cap \mathcal{I} = \emptyset$ (backward analysis)

The models that we consider in this talk :

- ➡ Pushdown systems
- ➡ Process Algebra
- ➡ Dynamic Networks of Pushdown Systems

Far from real world ?

The models that we consider in this talk :

- ➡ Pushdown systems
- ➡ Process Algebra
- ➡ Dynamic Networks of Pushdown Systems

Far from real world ?

Yes, but close enough for dataflow analysis. (remember Helmut Seidl's talk)

Introduction

Word Automata and Pushdown Systems

Tree Automata and Process Algebra

Presburger/Hedge Automata and Multithreaded Programs

Analysis of Pushdown Systems (PDS)

(Bouajjani-Esparza-Maler, Concur 97)

- Finite set of states Q,
- (Stack) alphabet Σ,
- Transition $\begin{array}{rcl} Q, \Sigma & \rightarrow & Q, \Sigma^* \\ (q, a) & \rightarrow & (q', \gamma) \end{array}$

Configuration (q, s) with $q \in Q, s \in \Sigma^*$)

Regularity Results for Pushdown Systems

Let L be a regular language.

Theorem

$Post^*(L)$ is a regular language.

Theorem

$Pre^*(L)$ is a regular language.

Backward and forward analysis are possible.

Complexity and Proof

Proof :

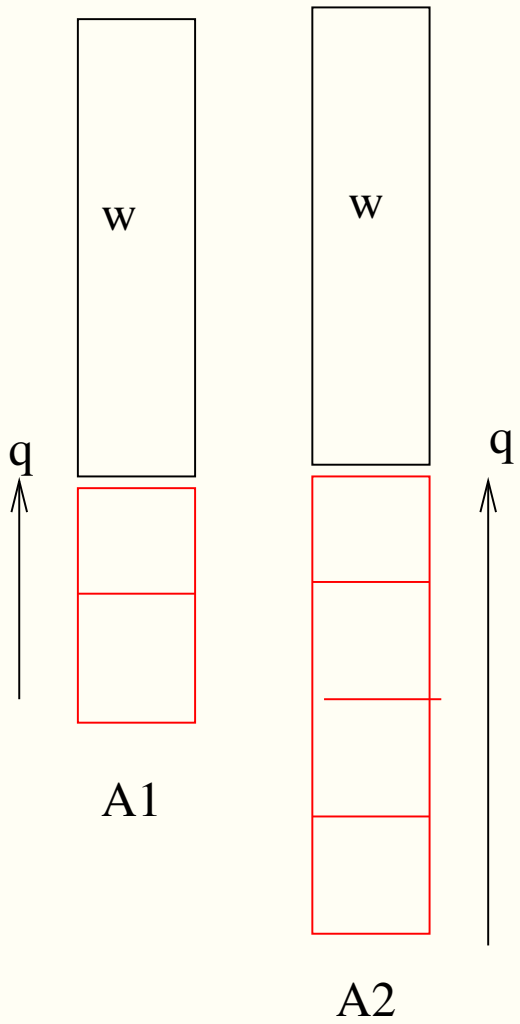
- Add new transitions to an automaton accepting L (only finitely many are possible).
- Difficulty : can't capture $\xrightarrow{n}$ but $\xrightarrow{*}$ at the end.
- Complexity polynomial.

A more general approach

Use :

- ➡ Ground Tree (Word) Transducers (Dauchet-Tison LICS 90)
- ➡ Recognizable relations

Ground Tree (Word) Transducers



GTT G=(A1,A2)

Synchronization states: common states

G accepts pairs (w1,w2) : relation $\rightarrow$

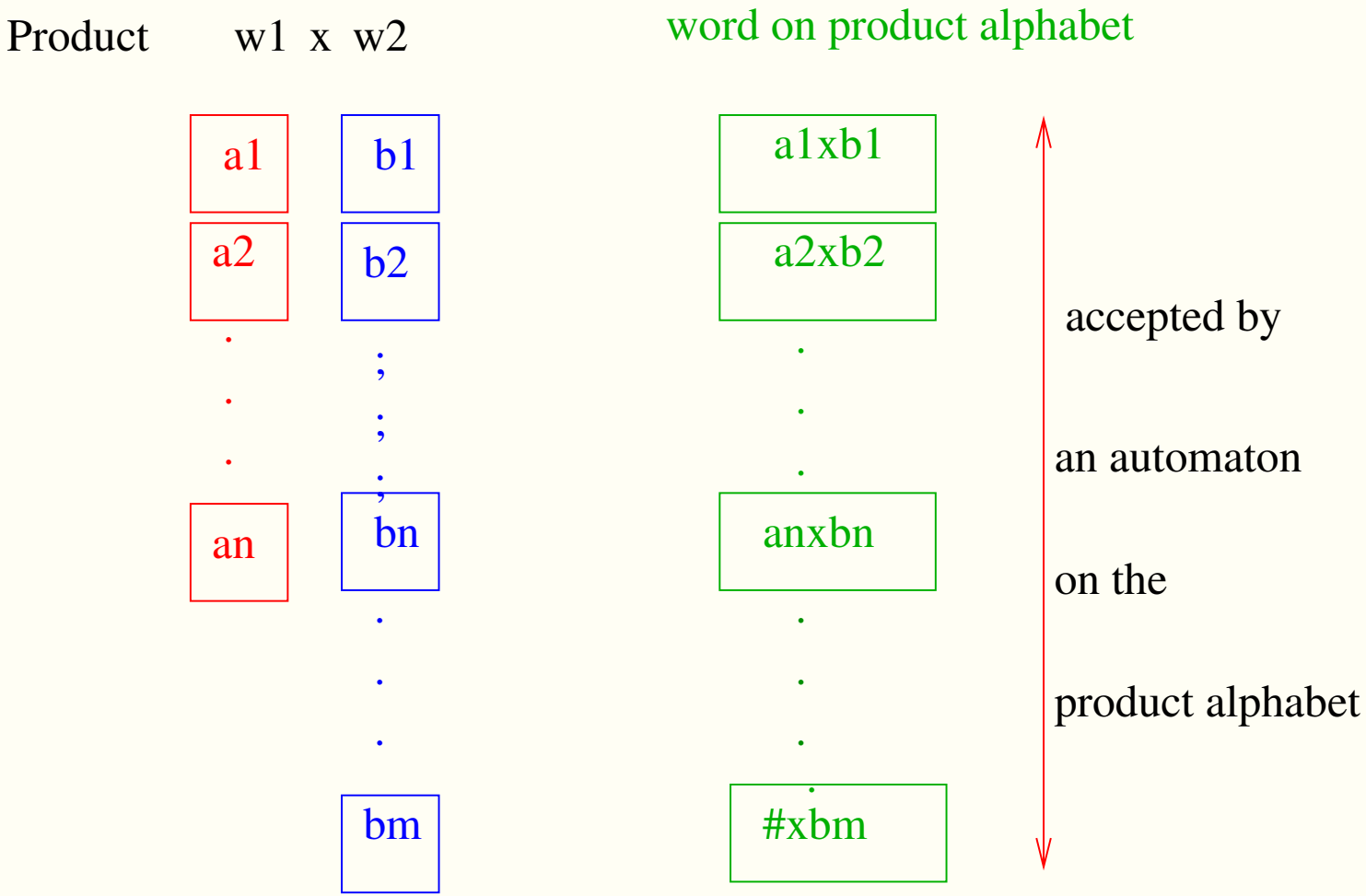
GTT closed under

composition

transitive closure $\xrightarrow{*}$

(Dauchet–Tison)

Recognizable Relation



Summing up

- GTT relation are recognizable
- One-step $\rightarrow$ suffix rewriting is GTT
- GTT closed under transitive closure hence $\stackrel{*}{\rightarrow}$ recognizable
- If L is a regular language, R recognizable relation then $R(L)$ and $R^{-1}(L)$ are regular.

Theorem

If L is regular then $Pre^*(L) = \stackrel{*}{\rightarrow}(L)$ and $Post^*(L) = \stackrel{*}{\rightarrow}^{-1}(L)$ regular.

(see also works of Buchi, Nivat, Caucal on word rewriting)

For free..

Theorem

The first order theory of $\xrightarrow{*}$ is decidable.

Properties like confluence are decidable.

Introduction

Word Automata and Pushdown Systems

Tree Automata and Process Algebra

Presburger/Hedge Automata and Multithreaded Programs

Analysis of Process Algebra

(Lugiez-Schnoebelen, Concur 98, Icalp 00)

Process Algebra : Process variables X,Y,...
Sequencing $\cdot$ and parallel $\|$ operations
Term process ex : $X \| (Y \cdot Z)$
Transitions rules $X \to t \in \Delta$ (finite)

Configuration : term process

Transition relation : $s \not\to, s' \stackrel{*}{\to} t'$ then $s \cdot s' \stackrel{*}{\to} s \cdot t'$
$s \to s', t \stackrel{*}{\to} t'$ then $s \| t \stackrel{*}{\to} s' \cdot t'$

Regularity theorems

- Rules $X \longrightarrow t$ are ground rewrite rules on trees not words
- GTT works for trees (not only for words)
- Same case as pushdown systems ?

Regularity theorems

- Rules $X \to t$ are ground rewrite rules on trees not words
- GTT works for trees (not only for words)
- Same case as pushdown systems ?

NO !

Why ?

Regularity theorems

- Rules $X \longrightarrow t$ are ground rewrite rules on trees not words
- GTT works for trees (not only for words)
- Same case as pushdown systems ?

NO !

Why ?

Transition relation for $\cdot$:
$s \not\rightarrow, s' \xrightarrow{*} t'$ then $s \cdot s' \xrightarrow{*} s \cdot t'$

Regularity theorems

Not ground rewriting but close enough to prove

Theorem

$\xrightarrow{*}$ *is recognizable.*

(direct proof about 40 cases to consider).

Theorem

If L is a regular tree language then $Post^*(L)$ and $Pre^*(L)$ are regular tree languages.

Complexity : polynomial.

For free..

Theorem

The first order theory of $\xrightarrow{*}$ is decidable.

Properties like confluence are decidable.

Introduction

Word Automata and Pushdown Systems

Tree Automata and Process Algebra

Presburger/Hedge Automata and Multithreaded Programs

Analysis of Dynamic Networks of PDS

(Bouajjani-Muller-Olm-Touili Concur 2005)

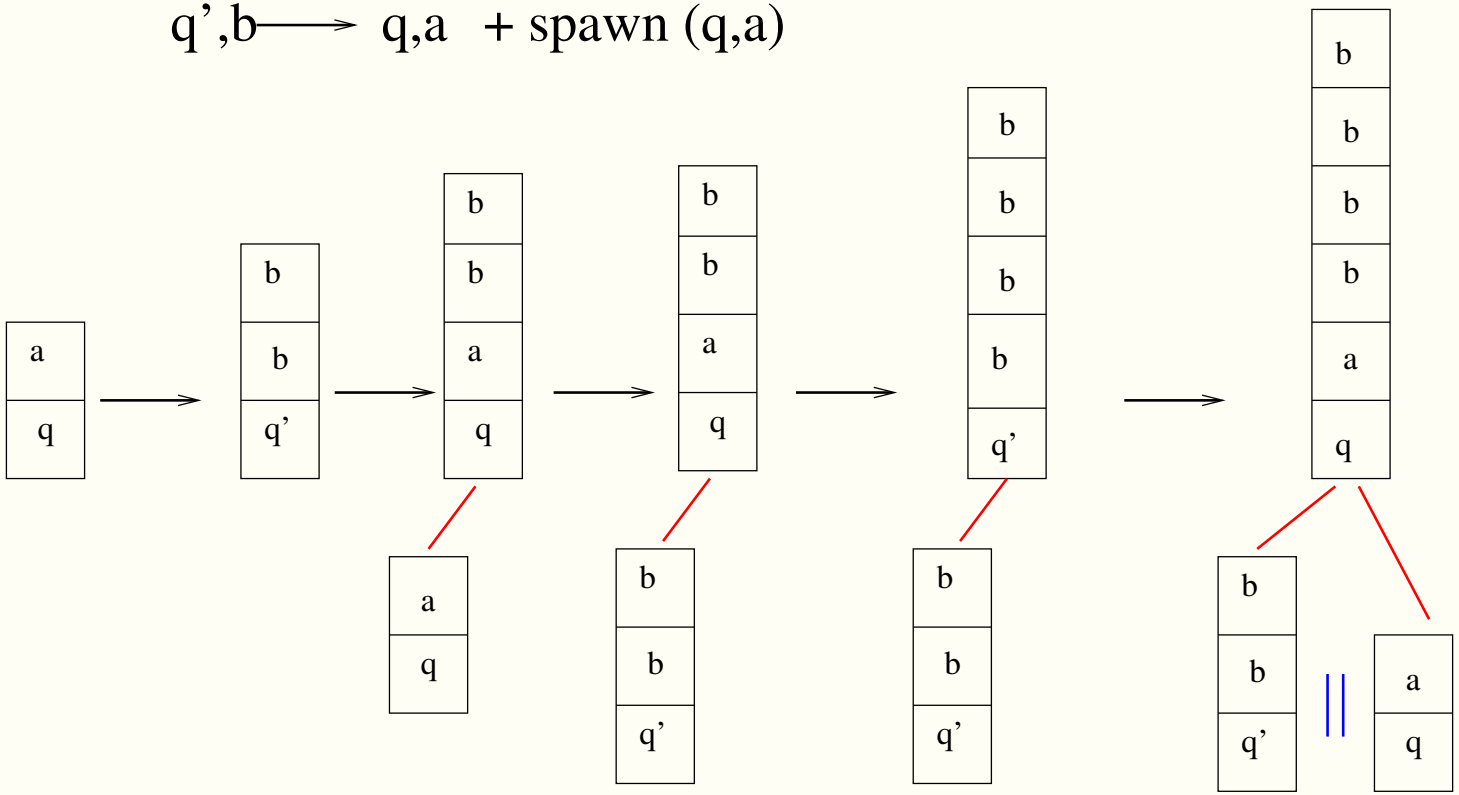
Networks of Pushdown systems.
Motivation : a thread can spawn news threads and live independently of his subprocesses.

- ➡ Process : pushdown systems
- ➡ Threads : running in parallel

More general that Process Algebra.

Configurations of a dynamic pushdown network

Rules: q,a ⟶ q',bb
q',b ⟶ q,a + spawn (q,a)



Expected Results

Let L a regular tree language.

Theorem

$Post^*(L)$ is a regular language.

Theorem

$Pre^*(L)$ is a regular language.

The Harsh Reality

Theorem

$Post^*(L)$ can be a non-regular language.

The Harsh Reality

Theorem

$Post^*(L)$ can be a non-regular language.

Rule : $(q, a) \longrightarrow (q, aa) + spawn(q, b)$

generates $q, a \cdot a^n + (q, b)^n$

$n + 1$ vertical a's, n horizontal b's.

The Harsh Reality

Theorem

$Post^*(L)$ can be a non-regular language.

Rule : $(q, a) \longrightarrow (q, aa) + spawn(q, b)$

generates $q, a \cdot a^n + (q, b)^n$
$n + 1$ vertical a's, n horizontal b's.

By the way

what does regular mean here ?

Hedge Tree Automata

1. Signature

   $\cdot$ an associative symbol $\approx$ catenation in words

   $\mathcal{F}$ free symbols (here monadic + constant)

2. Terms

$$T \ni t ::= t_1 \cdot \ldots \cdot t_p \mid f(t_1, \ldots, t_n)$$

3. Hedge automata

$$\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_F, \mathcal{F}, \Delta)$$
$$\Delta = \; f(q_1, \ldots, q_n) \to q$$
$$\quad\; Reg \to q \quad Reg \text{ regular expression on } \mathcal{Q}$$

The Actual Results

Theorem

$Pre^*(L)$ is a regular language.

Why backyard is easier than forward ?

Tree homomorphism $h : x \longrightarrow f(x, x)$

- $h(L) = \{f(s,t) \mid L \ni s = t \in L\}$ non-regular
- $h^{-1}(f(L_1, L_2)) = L_1 \cap L_2$ regular

Forward analysis ?

The transition relation preserves context-free languages.

- ➡ $Post^*(L)$ context-free if L context-free
- ➡ Emptiness of intersection of regular and context-free is decidable

Not formally stated but forward analysis should work for regular sets of initial and bad configurations...
Rk : More complex rules allowed (regularity constraints) in backward analysis.

Parallel Processes

Another point of view : Parallel processes are unordered.

- ➡ $\cdot$ is associative-commutative
- ➡ What are regular languages ?

The Unordered case : Presburger Tree Automata

1. Signature

   $\cdot$ an associative-commutative symbol

   $\mathcal{F}$ free symbols (here monadic + constant ϵ)

2. Terms

$$T ::= t_1 \cdot \ldots \cdot t_p \mid f(t_1, \ldots, t_n)$$

3. Presburger automata

$$\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_F, \mathcal{F}, \Delta)$$

$$f(q_1, \ldots, q_n) \to q$$

$$\phi(\#w_{\mathcal{Q}}) \to q \quad \begin{cases} \phi \text{ Presburger arithmetic formula} \\ \#w_{\mathcal{Q}} \text{ Parikh mapping of } w_{\mathcal{Q}} \in \mathcal{Q}^* \end{cases}$$

Parikh mapping by example :

- $\Sigma = \{q_1, q_2\}$
- $w = q_1 \cdot q_2 \cdot q_2 \cdot q_1 \cdot q_1$ then $\#w = (3, 2)$

Backward Analysis

Theorem

$Pre^*(L)$ is a regular language.

Proof : The Parikh mapping of a regular (context-free) language is defined by a Presburger arithmetic formula.

Remark : Closure of regular tree language are Presburger languages (see Osahki's talk and work on equational tree automata).

Forward Analysis

Configurations combine vertical words and horizontal words

$$\begin{array}{ccc} & a_p & \\ & \vdots & \\ & a_1 & \\ & q & \\ q_1 & q_i & q_n \end{array}$$

Presburger automata too weak : no comparison between *vertical* and *horizontal* parts.

Needed for describing : n a's and n q_i's.

Conclusion(s)

Automata and Recognizable relations

- ➡ Powerful tool for analysis
- ➡ Good complexity
- ➡ Customizable approach

Conclusion(s)

Automata and Recognizable relations

- ➡ Powerful tool for analysis
- ➡ Good complexity
- ➡ Customizable approach

Drawback

Can't deal with synchronization (accessibility becomes undecidable)

Other applications for system analysis ?

- Analysis of lossy channel systems,
- Analysis of XML documents,
- Analysis of cryptographic protocols,
- ...