Tree Pattern Rewriting Systems

B. Genest[◇] A. Muscholl[♠] O. Serre[♣] M. Zeitoun[♠]

[◇]IRISA, Univ. Rennes 1 & CNRS

[♠]LaBRI, Univ. Bordeaux & CNRS

[♣]LIAFA, Univ. Paris 7 & CNRS. Deserves credit for the slides!

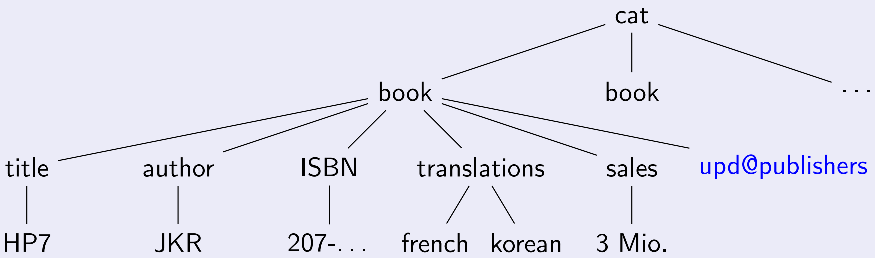
ACTS, Chennai, 2009/31/1

Outline

Document trees

- XML: unranked, unordered, (finitely) labelled finite trees.
- Active XML (AXML): extended by service nodes. Implicit data representation

AXML trees



upd@publishers: update translations and number of sold items.

Capture Service Calls

- Query information on a document tree on given peer (example: service upd on peer publishers),
- Add query result (= forest) to the original tree at a designated node (materialization of service call).

Capture Service Calls

- Query information on a document tree on given peer (example: service upd on peer publishers),
- Add query result (= forest) to the original tree at a designated node (materialization of service call).

Remark

- We consider here wlog. a single peer (i.e., a single document tree).
- Query result may contain itself service nodes (recursion). Order in which services are called can be relevant.

Capture Service Calls

- Query information on a document tree on given peer (example: service upd on peer publishers),
- Add query result (= forest) to the original tree at a designated node (materialization of service call).

Remark

- We consider here wlog. a single peer (i.e., a single document tree).
- Query result may contain itself service nodes (recursion). Order in which services are called can be relevant.

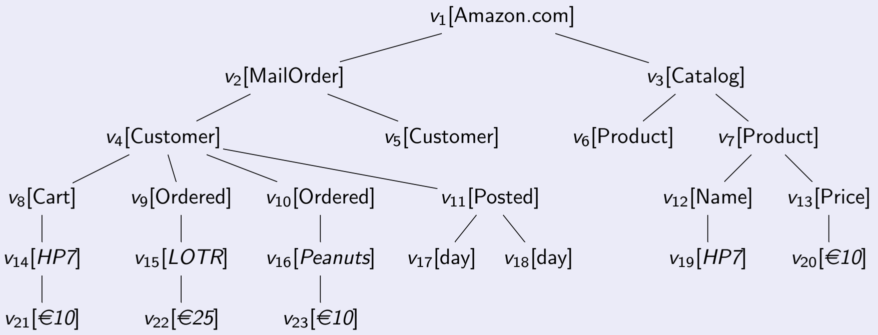
Examples of properties to verify

- Termination: is there an infinite sequence of service calls?
- Reachability: given documents d_1, d_2, can d_2 be reached from d_1?

Abstract Model

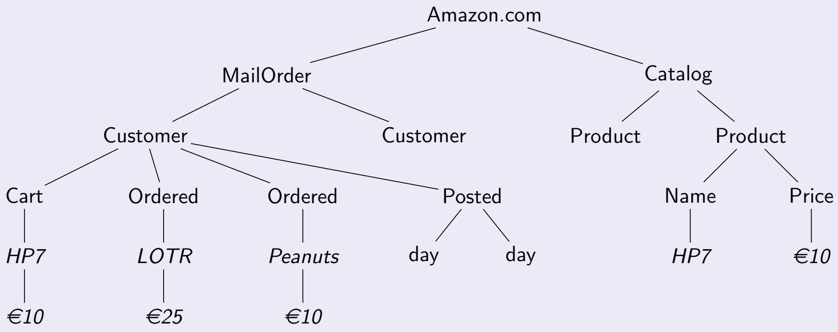
Tree document: unranked, unordered, (finitely) labelled tree.

Example: `Amazon.com` database



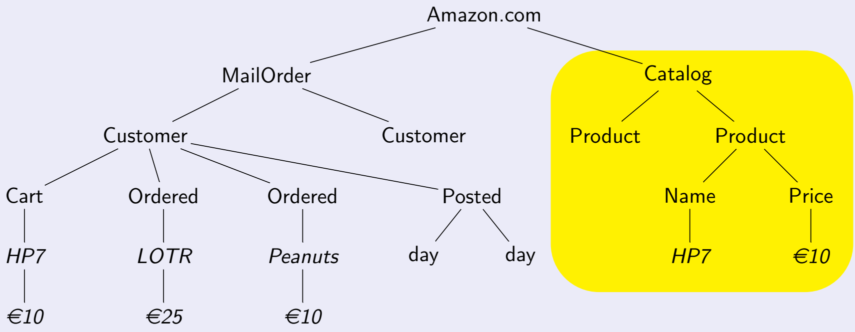
Tree document: unranked, unordered, (finitely) labelled tree.

Example: `Amazon.com` database



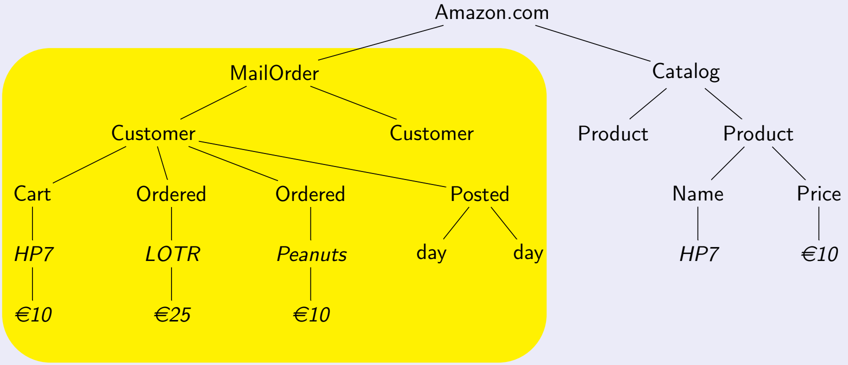
Tree document: unranked, unordered, (finitely) labelled tree.

Example: `Amazon.com` database



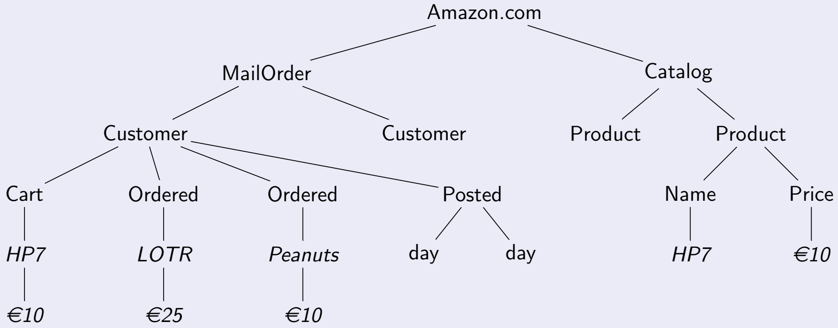
Tree document: unranked, unordered, (finitely) labelled tree.

Example: `Amazon.com` database



Tree document: unranked, unordered, (finitely) labelled tree.

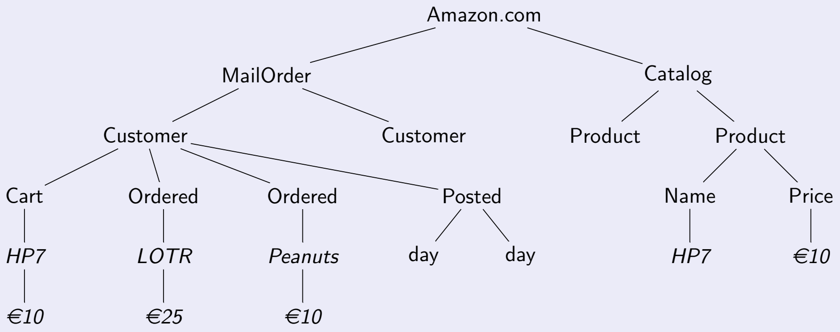
Example: `Amazon.com` database



Order service on `Amazon.com`: (add-product + delete-product)*checkout.

Tree document: unranked, unordered, (finitely) labelled tree.

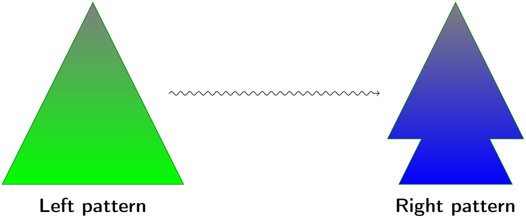
Example: `Amazon.com` database



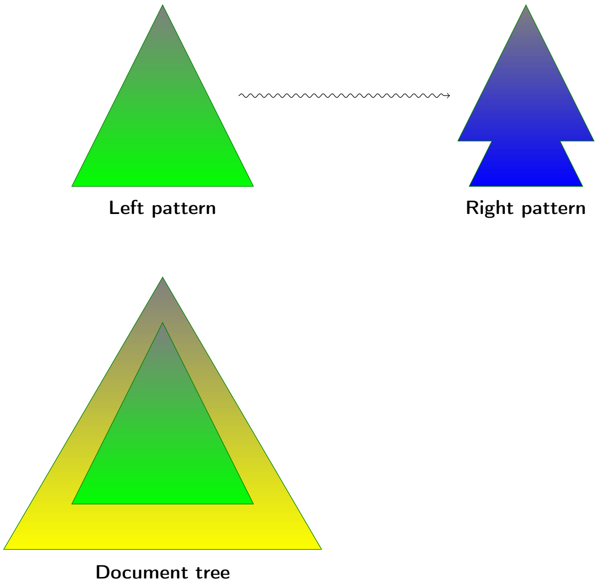
Examples of actions to model: add a new customer, add a product to the cart of a customer, delete a product from the cart…

Outline

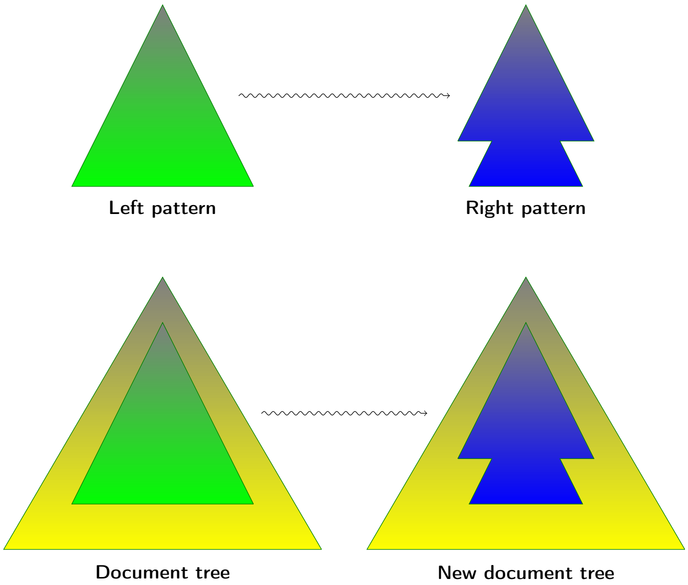
Tree Rewriting Rules: Informal Description



Tree Rewriting Rules: Informal Description

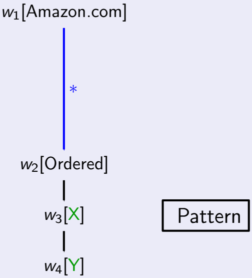


Tree Rewriting Rules: Informal Description



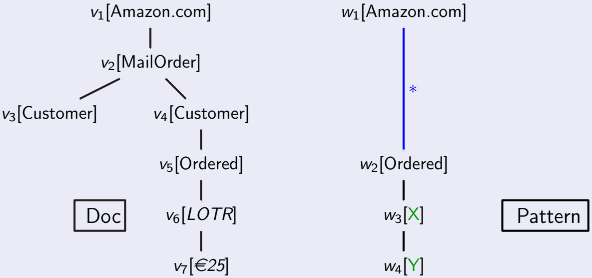
Definition + Example

- Pattern: Tree P with
  - node labels from $\mathit{Tags} \cup \mathit{Var}$,
  - child edges
  - descendant edges (marked $*$).
- Match a pattern P against a document T: injective mapping from P into T, from the root, label-preserving on Tags.



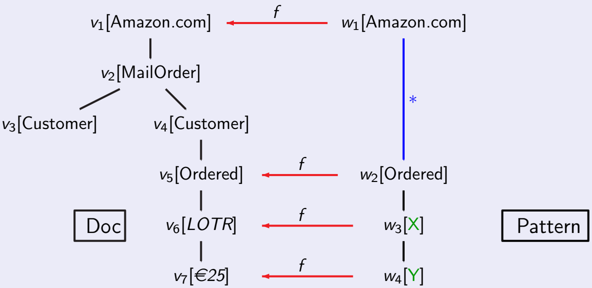
Definition + Example

- Pattern: Tree P with
  - node labels from $\mathit{Tags} \cup \mathit{Var}$,
  - child edges
  - descendant edges (marked $*$).
- Match a pattern P against a document T: injective mapping from P into T, from the root, label-preserving on Tags.



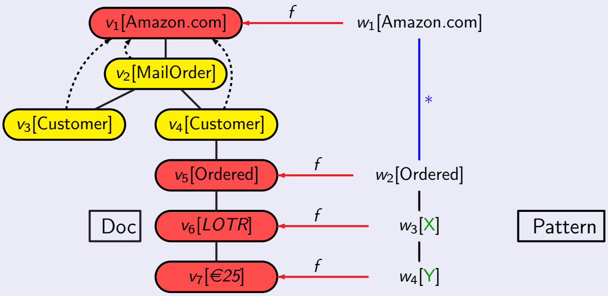
Definition + Example

- Pattern: Tree P with
  - node labels from $Tags \cup Var$,
  - child edges
  - descendant edges (marked $*$).
- Match a pattern P against a document T: injective mapping from P into T, from the root, label-preserving on Tags.



Definition + Example

- Pattern: Tree P with
  - node labels from $\mathit{Tags} \cup \mathit{Var}$,
  - child edges
  - descendant edges (marked $*$).
- Match a pattern P against a document T: injective mapping from P into T, from the root, label-preserving on Tags.

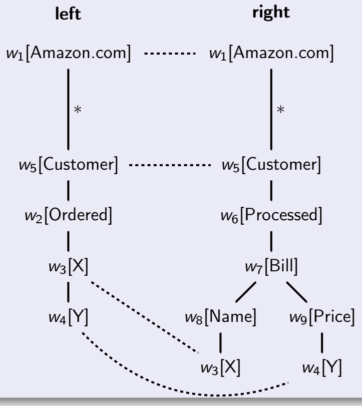


Tree Pattern Rewriting Rule

Rewriting rule (**left**, **right**):

- **left**, **right**: tree patterns + nodes ids w_1,w_2,...

Example: processing an order

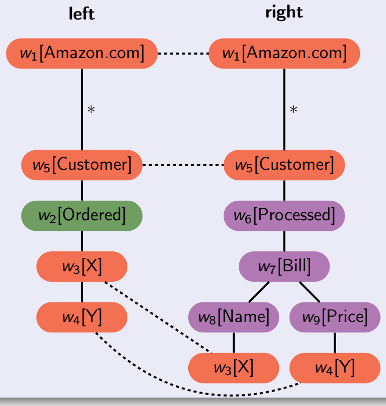


Tree Pattern Rewriting Rule

Rewriting rule (**left**, **right**):

- **left**, **right**: tree patterns + nodes ids $w_1, w_2, \ldots$

Example: processing an order



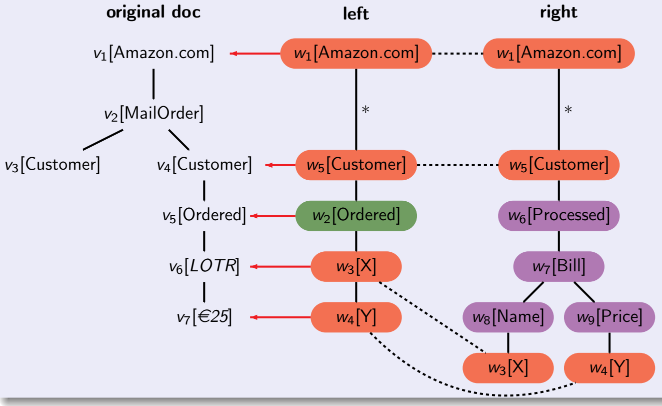
- nodes both in **left** and **right**: will be **kept**.
- nodes in **left** but not in **right**: will be **deleted**.
- nodes in **right** but not in **left**: will be **created**.

Tree Pattern Rewriting Rule

Rewriting rule (**left**, **right**):

- **left**, **right**: tree patterns + nodes ids $w_1, w_2, \ldots$

Example: processing an order

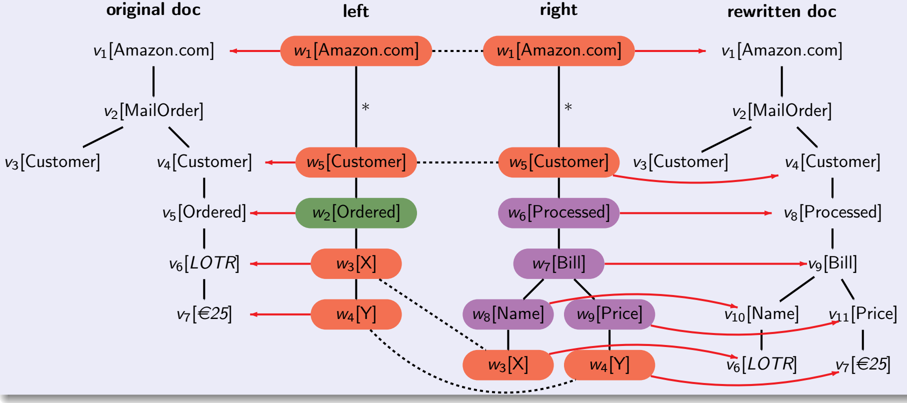


Tree Pattern Rewriting Rule

Rewriting rule (**left**, **right**):

- **left**, **right**: tree patterns + nodes ids $w_1, w_2, \ldots$

Example: processing an order



Tree Pattern Rewriting Rule

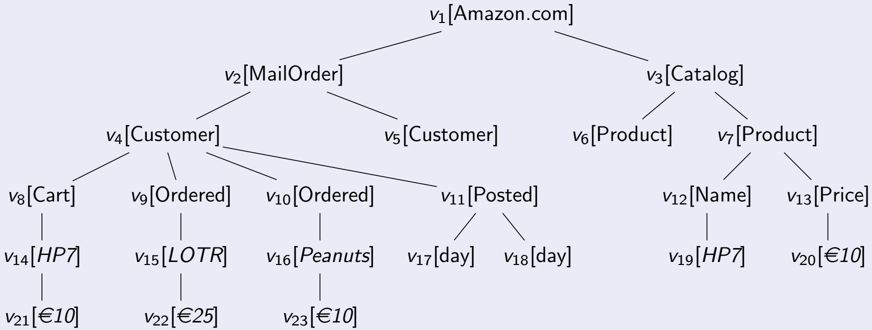
Rewriting rule (**left**, **right**):

- **left**, **right**: tree patterns + nodes ids $w_1, w_2, \ldots$

Application of a Rule

1. Match document with **left**.
2. Keep those nodes (and related ones) matched with nodes in **left** $\cap$ **right**.
3. Delete those nodes (and related ones) matched with nodes in **left** $\setminus$ **right**.
4. Create nodes induced by **right** $\setminus$ **left**.

Back to the `Amazon.com` Database: What Can you Model?



Add a new customer.

Back to the `Amazon.com` Database: What Can you Model?

w_1[Amazon.com]

w_2[MailOrder]

left

w_1[Amazon.com]

w_2[MailOrder]

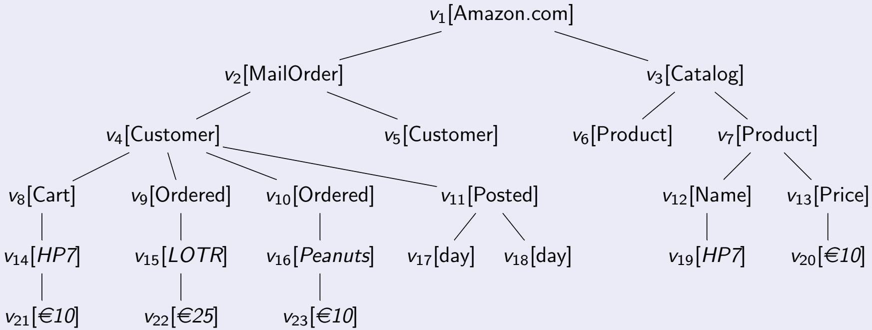
w_3[Customer]

w_4[Cart]

right

Add a new customer.

Back to the Amazon.com Database: What Can you Model?



Delete a product from the cart.

Back to the `Amazon.com` Database: What Can you Model?

$$\begin{array}{c} w_1[\text{Amazon.com}] \\ \Big|{*} \\ w_2[\text{Cart}] \\ \Big| \\ w_3[\text{X}] \\ \Big| \\ w_4[\text{Y}] \end{array}$$

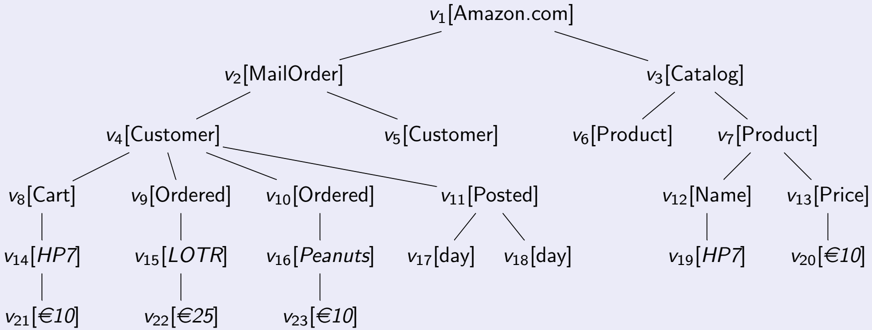
left

$$\begin{array}{c} w_1[\text{Amazon.com}] \\ \Big|{*} \\ w_2[\text{Cart}] \end{array}$$

right

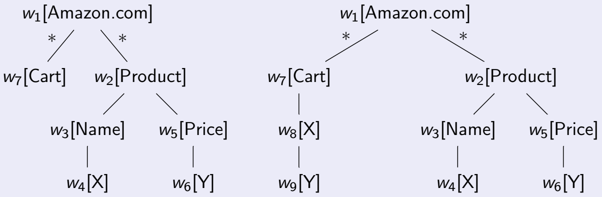
Delete a product from the cart.

Back to the `Amazon.com` Database: What Can you Model?



Add a product to the cart of a customer.

Back to the `Amazon.com` Database: What Can you Model?



left **right**

Add a product to the cart of a customer.

Tree Pattern Queries (TPQ)

TPQ query : $Q \rightsquigarrow P$:

- Q: tree pattern.
- P: tree possibly using variables appearing in Q.

Tree Pattern Queries (TPQ)

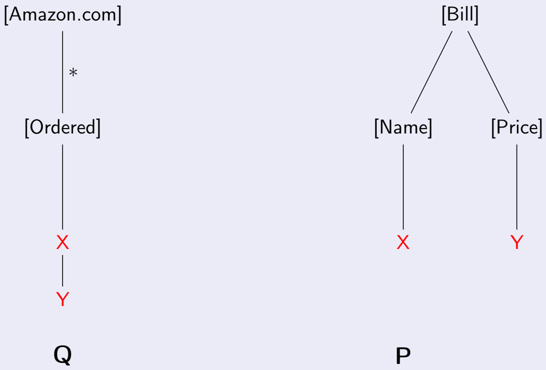
TPQ $\mathrm{query} : Q \rightsquigarrow P$:

- Q: tree pattern.
- P: tree possibly using variables appearing in Q.

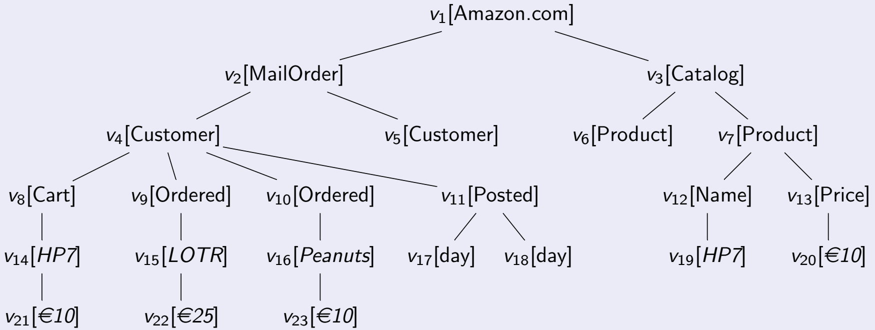
Each matching of a tree T with Q leads an instance of P in which variables are replaced by the tag implied by the matching.

Result of a TPQ $\mathrm{query} : Q \rightsquigarrow P$ on a tree T: forest $\mathrm{query}(T)$ of all instantiations of P by matching between Q and T.

Example query : $Q \rightsquigarrow P$

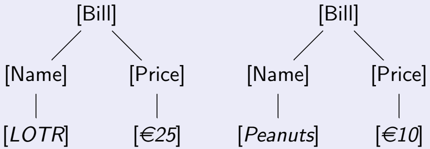


Result of query : $Q \rightsquigarrow P$ on Amazon.com:



Tree Pattern Queries

Result of query : $Q \rightsquigarrow P$ on Amazon.com:



Actually Rewriting Rules Might Be Richer...

Rewriting rule (**left**, **right**, **query**, **guard**):

- **left**, **right**: tree patterns. TP **right** might contain special nodes marked by $.
- **query**: tree pattern query.
- **guard**: set of forests.

Actually Rewriting Rules Might Be Richer...

Rewriting rule (**left**, **right**, **query**, **guard**):

- **left**, **right**: tree patterns. TP **right** might contain special nodes marked by $.
- **query**: tree pattern query.
- **guard**: set of forests.

Application of a rule to a tree T

1. Match T with **left** via some embedding f.
2. The rule is enabled for f iff $\mathbf{query}_f(T) \in \mathbf{guard}$.
3. Then everything is as before except that one attach to any node marked $ a copy of $\mathbf{query}_f(T)$

Guards

- If after 21 days a posted parcel is still not received the customer can require a payback.
- Cancel some approvisioning from the manufacturer for some product when the stock is greater than some threshold.

Plug results from TPQ

- Produce a bill.
- Give the list of all articles in all carts.

Outline

Tree Pattern Rewriting Systems (TPRS)

TPRS $(T, \mathcal{R})$: initial tree T, finite set $\mathcal{R}$ of rewriting rules.
In general, a TPRS is an infinite-state system.

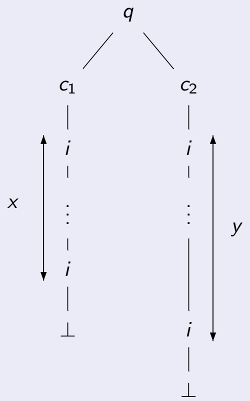
Questions on input $(T, \mathcal{R})$:

- Termination.
- Finite-state property.
- Reachability.
- Pattern reachability (or coverability).
- Confluence from reachable T_1 and T_2.
- Weak confluence: for any reachable T_1, T_2, do some T_1', T_2' exist with $T_1 \xrightarrow{*} T_1'$, $T_2 \xrightarrow{*} T_2'$ and T_1' subsumed by T_2'?

Theorem

Any two-counter machine can be simulated by a TPRS such that the machine stops iff the TPRS terminates.

Sketch.

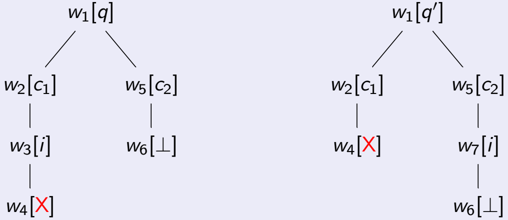


Coding configuration (q, x, y).

Theorem

Any two-counter machine can be simulated by a TPRS such that the machine stops iff the TPRS terminates.

Sketch.



left **right**

In state q with $c_1 \neq 0$ and $c_2 = 0$ go to q', decrement c_1 and increment c_2.

Theorem

Any two-counter machine can be simulated by a TPRS such that the machine stops iff the TPRS terminates.

Undecidability causes

- Deletion?
- Ability to copy subtrees?
- Unbounded depth?

Well-quasi-ordering

Well-quasi-ordering on a set X: quasi-ordering $\preceq$ such that every infinite sequence of elements from X contains an infinite increasing subsequence.

WSTS [Finkel/Schnoebelen, Abdulla/Jonsson, '96]

- Well-quasi-ordering $\preceq$ on the set of configurations.
- Upwards compatibility.

$$\begin{array}{ccc} T_1 & \preceq & T_1' \\ {\scriptstyle *}\downarrow & & \\ T_2 & & \end{array}$$

A Tool for Decidability: Well-Structured Transition Systems

Well-quasi-ordering

Well-quasi-ordering on a set X: quasi-ordering $\preceq$ such that every infinite sequence of elements from X contains an infinite increasing subsequence.

WSTS [Finkel/Schnoebelen, Abdulla/Jonsson, '96]

- Well-quasi-ordering $\preceq$ on the set of configurations.
- Upwards compatibility.

$$\begin{tikzcd} T_1 \arrow[r, phantom, "\preceq"] \arrow[d, "*"'] & T_1' \arrow[d, "*"] \\ T_2 \arrow[r, phantom, "\preceq"] & T_2' \quad \exists \end{tikzcd}$$

A Tool for Decidability: Well-Structured Transition Systems

Well-quasi-ordering

Well-quasi-ordering on a set X: quasi-ordering $\preceq$ such that every infinite sequence of elements from X contains an infinite increasing subsequence.

WSTS [Finkel/Schnoebelen, Abdulla/Jonsson, '96]

- Well-quasi-ordering $\preceq$ on the set of configurations.
- Upwards compatibility.

$$\begin{tikzcd} T_1 \arrow[d, "*"'] & \preceq & T_1' \arrow[d, "*"] & \\ T_2 & \preceq & T_2' & \exists \end{tikzcd}$$

Theorem [Finkel/Schnoebelen, Abdulla/Jonsson, '96]

Termination and coverability are decidable for WSTS (requires some additional effectiveness properties).

Three restrictions

- Strict TPRS: no deletion allowed (all node in **left** are in **right**).
- Depth-bounded TPRS: for some constant K, every T' with $T \xrightarrow{*} T'$ is of depth at most K.
- Guards are upward closed.

Three restrictions

- Strict TPRS: no deletion allowed (all node in **left** are in **right**).
- Depth-bounded TPRS: for some constant K, every T' with $T \xrightarrow{*} T'$ is of depth at most K.
- Guards are upward closed.

	Term.	FS	Reach.	P-reach.	Confl.	W-confl.
Strict	U	U	D	U	U	U

Strictness is not enough

Simulating a 2-counter machine still works (instead of deleting, move to some garbage node).

Decidability Issues for Depth-bounded TPRS

Three restrictions

- Strict TPRS: no deletion allowed (all node in **left** are in **right**).
- Depth-bounded TPRS: for some constant K, every T' with $T \xrightarrow{*} T'$ is of depth at most K.
- Guards are upward closed.

	Term.	FS	Reach.	P-reach.	Confl.	W-confl.
Strict	U	U	D	U	U	U
Depth-Bounded	D	U	U	D	U	U

Decidability

- **Def.** A tree T' subsumes a tree T iff there is an injective embedding of T into T' preserving the root, the labelling and the parent relation.
- **Lemma.** For any $K \geq 0$, the subsumed relation is a well-quasi order over unordered trees of depth at most K.
- Techniques from WSTS yield decidability.

Three restrictions

- Strict TPRS: no deletion allowed (all node in **left** are in **right**).
- Depth-bounded TPRS: for some constant K, every T' with $T \xrightarrow{*} T'$ is of depth at most K.
- Guards are upward closed.

	Term.	FS	Reach.	P-reach.	Confl.	W-confl.
Strict	U	U	D	U	U	U
Depth-Bounded	D	U	U	D	U	U

Undecidability

- Simulate a reset Petri net (depth 2 is enough).
- FS property and reachability are undecidable for reset Petri nets [Dufour/Finkel/Schnoebelen '98].
- One can reduce reachability to (weak) confluence.

Three restrictions

- Strict TPRS: no deletion allowed (all node in **left** are in **right**).
- Depth-bounded TPRS: for some constant K, every T' with $T \xrightarrow{*} T'$ is of depth at most K.
- Guards are upward closed.

	Term.	FS	Reach.	P-reach.	Confl.	W-confl.
Strict	U	U	D	U	U	U
Depth-Bounded	D	U	U	D	U	U
Depth-B. Strict	D	D	D	D	U	U

Decidability

- Reachability is easy: exploration of a finite set.
- FS property comes from the fact that one has a strict WSTS.

Remark. Decidability is implicitly based on non-constructive proofs coming from Higman's Lemma.

Remark. Decidability is implicitly based on non-constructive proofs coming from Higman's Lemma.

Theorem

The following problems have at least non-elementary complexity:

- Input: A pattern P, a TPRS $(T, \mathcal{R})$ and an integer k such that the depth of $(T, \mathcal{R})$ is bounded by k.
- Problem 1: Is the pattern P reachable in $(T, \mathcal{R})$?
- Problem 2: Does $(T, \mathcal{R})$ terminate?

Theorem

Pattern reachability and termination are non-elementary decidable.

Sketch.

- Simulate a run of M, a $n \mapsto \text{tower}(k, n)$-space bounded TM on input x by a linear size depth-bounded TPRS.
- Encode each configuration of M by a tree.
- Build TPRS to enforce transitions of M.
- Use counters to distinguish tape positions.
- At depth K, one can count up to $\text{tower}(K, 2)$.

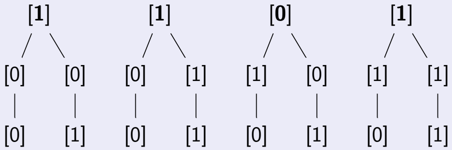
Theorem

Pattern reachability and termination are non-elementary decidable.

Sketch.

Encoding of counters as in [Walukiewicz '98]

A level 2 counter encoding 13:

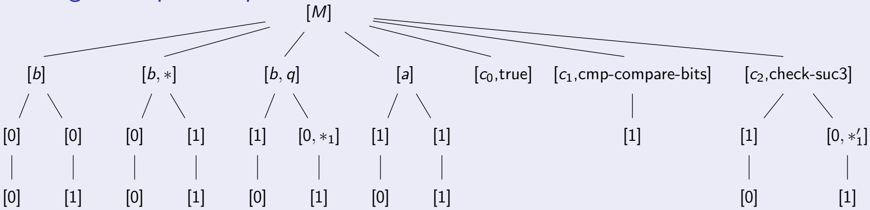


Theorem

Pattern reachability and termination are non-elementary decidable.

Sketch.

Encoding the tape $b\ b\ q\ b\ a$:



Undirected TPRS

- No more restriction on the depth.
- TP used in **left** and in **query** cannot use the parent relation (only ancestor relation).

Undirected TPRS

- No more restriction on the depth.
- TP used in **left** and in **query** cannot use the parent relation (only ancestor relation).

Theorem

For undirected TPRS one gets the same decidability results as for depth-bounded TPRS.

Theorem

Termination and pattern reachability have at least non primitive recursive complexity for undirected TPRS.

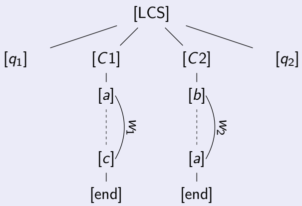
Theorem

For undirected TPRS one gets the same decidability results as for depth-bounded TPRS.

Theorem

Termination and pattern reachability have at least non primitive recursive complexity for undirected TPRS.

Proof: encode a LCS



AC term rewriting

Essential difference: term rewriting is about ranked trees. Unclear how to simulate TPRS rewriting on the ranked version of a tree.

Regular ground tree rewriting systems [Löding '02]

- Rules $L \longrightarrow R$, with L, R regular sets of trees (subtrees from L can be replaced by any element in R).
- Decidability: Reachability (pre*-operator preserves regularity).
- Extension to unranked, ordered trees [Löding/Spelten '07].

Guarded AXML [Abiteboul/Segoufin/Vianu '08]

- Infinite data allowed. Uses Boolean combinations of tree patterns as guards and temporal logics over tree patterns for property specification.
  Decidable case: no recursion in calls.

Thank you!