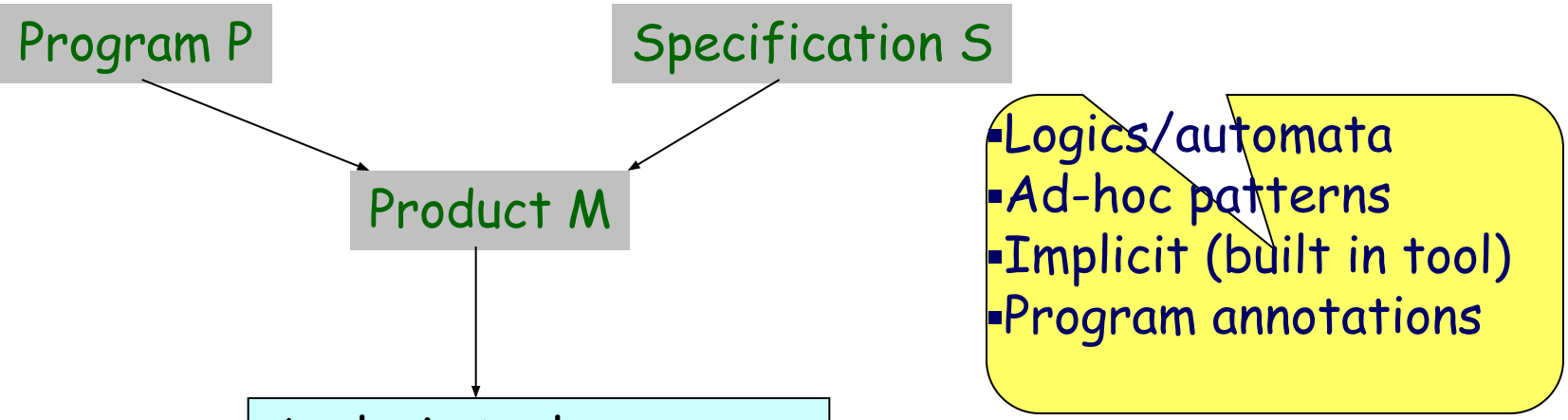
Marrying Words and Trees

Rajeev Alur

University of Pennsylvania

CSR, September 2007

Software Analysis



Analysis tool

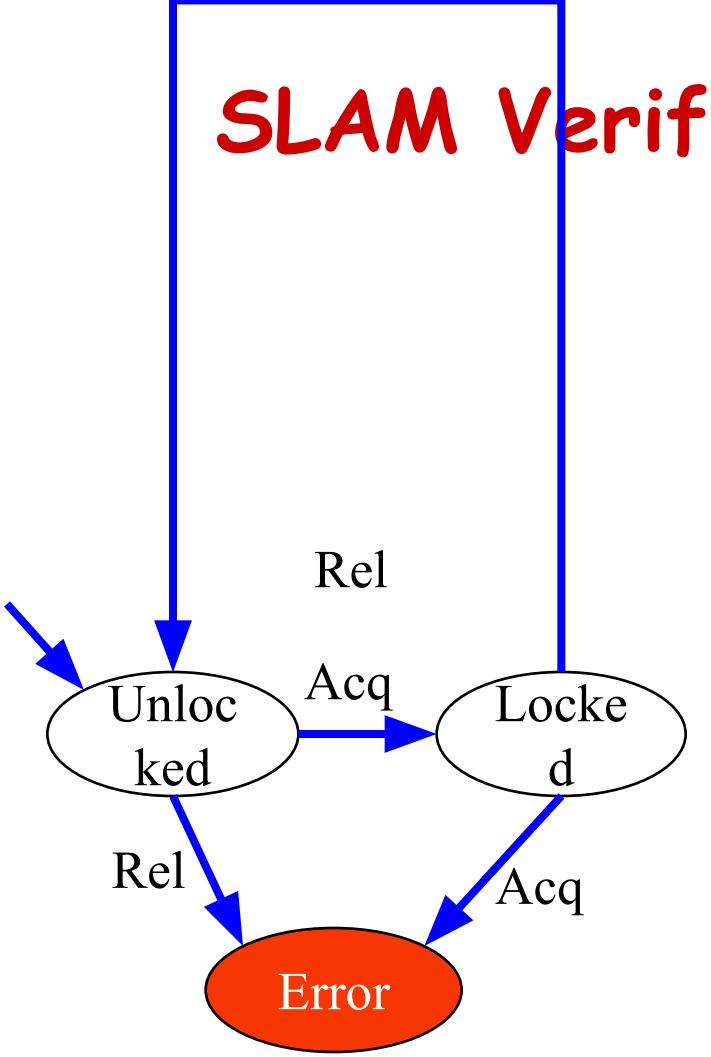
Model checking
Static analysis
Deductive reasoning
Testing
Runtime monitoring

Automata-theoretic Verification

P: Generator for possible executions
S: Acceptor for (in)correct executions
Model checking: Language inclusion
Runtime monitoring: Membership

SLAM Verification Example

Does this code obey the locking spec?



Specification

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

   if(request){
    request = request->Next;
    KeReleaseSpinLock();
    nPackets++;
   }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

Appeal of Regular Languages

- Well-understood expressiveness: multiple characterizations
  - Deterministic/nondeterministic/alternating finite automata
  - Regular expressions
  - Monadic second order logic of linear order
  - Syntactic congruences

- Regular languages are effectively closed under many operations
  - Union, intersection, complement, conactenation, Kleene-*, homomorphisms…

- Algorithms for decision problems
  - Membership
  - Determinization and minimization
  - Language emptiness (single-source graph reachability)
  - Language inclusion, language equivalence …

Checking Structured Programs

- Control-flow requires stack, so (abstracted) program P defines a context-free language
- Algorithms exist for checking regular specifications against context-free models
  - Emptiness of pushdown automata is solvable
  - Product of a regular language and a context-free language is context-free
- But, checking context-free spec against a context-free model is undecidable!
  - Context-free languages are not closed under intersection
  - Inclusion as well as emptiness of intersection undecidable
- Existing software model checkers: pushdown models (Boolean programs) and regular specifications

Are Context-free Specs Interesting?

- Classical Hoare-style pre/post conditions
  - If p holds when procedure A is invoked, q holds upon return
  - Total correctness: every invocation of A terminates
  - Integral part of emerging standard JML
- Stack inspection properties (security/access control)
  - If setuuid bit is being set, root must be in call stack
- Interprocedural data-flow analysis
- All these need matching of calls with returns, or finding unmatched calls
  - Recall: Language of words over [,] such that brackets are well matched is not regular, but context-free

Checking Context-free Specs

- Many tools exist for checking specific properties
  - Security research on stack inspection properties
  - Annotating programs with asserts and local variables
  - Inter-procedural data-flow analysis algorithms

- What's common to checkable properties?
  - Both program P and spec S have their own stacks, but the two stacks are synchronized

- As a generator, program should expose the matching structure of calls and returns

Solution: Nested words and theory of regular languages over nested words

Program Executions as Nested Words

Program

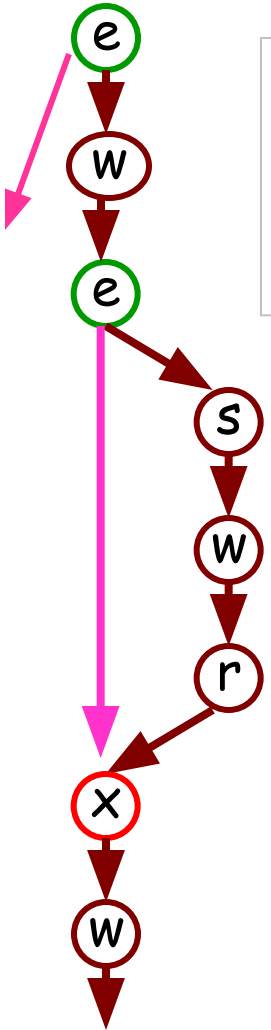
```
global int x;
main() {
  x = 3;
  if P   x = 1 ;
  ....
}

bool P () {
 local int y=0;
  x = y;
  return (x==0);
}
```

If a procedure writes to x, it must later read it

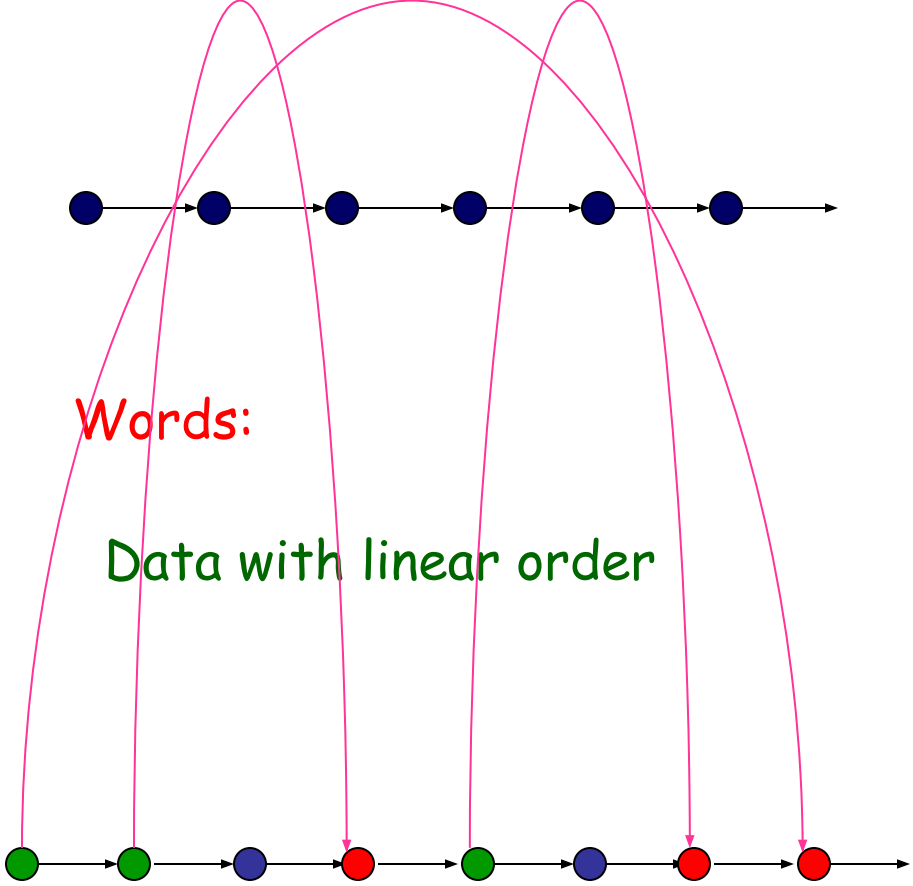
An execution as a word

Symbols:
w : write x
r : read x
e: enter
x: exit
s : other

An execution as a nested word

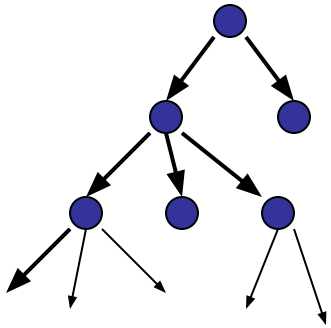


Summary edges from calls to returns



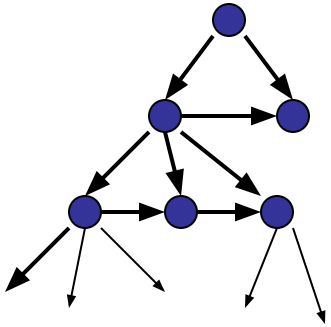
Words:

Data with linear order



(Unordered) Trees:

Data with hierarchical order



Nested Words (AM06):

Data with linear order +

Nesting edges

Ordered Trees/Hedges:

Data with hierarchical order +

Linear order on siblings

Document Processing

HTML Document

Query Processing

Query 1: Find documents that contain "Ekaterinburg" followed by "Google" (refers to linear/word structure)

Query 2: Find documents related to conferences sponsored by Google in Ekaterinburg (refers to hierarchical/tree structure)

Model a document *d* as a nested word
Nesting edges from <tag> to </tag>

Compile query into automata over nested words

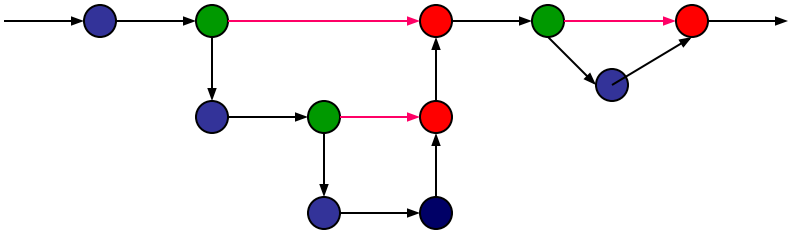
Analysis: Membership question
Does document *d* satisfy query *L* ?

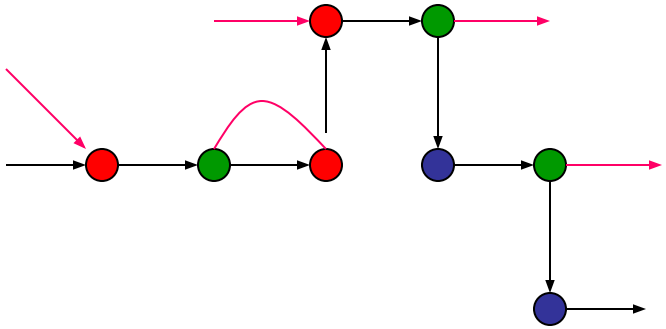
Talk Overview

- Introduction to Nested Words
- Regular Languages of Nested Words
- Relation to Pushdown Automata and Tree Automata
- Conclusions and Future Work

Nested Shape:

- Linear sequence + Non-crossing nesting edges
- Nesting edges can be pending, Sequence can be infinite





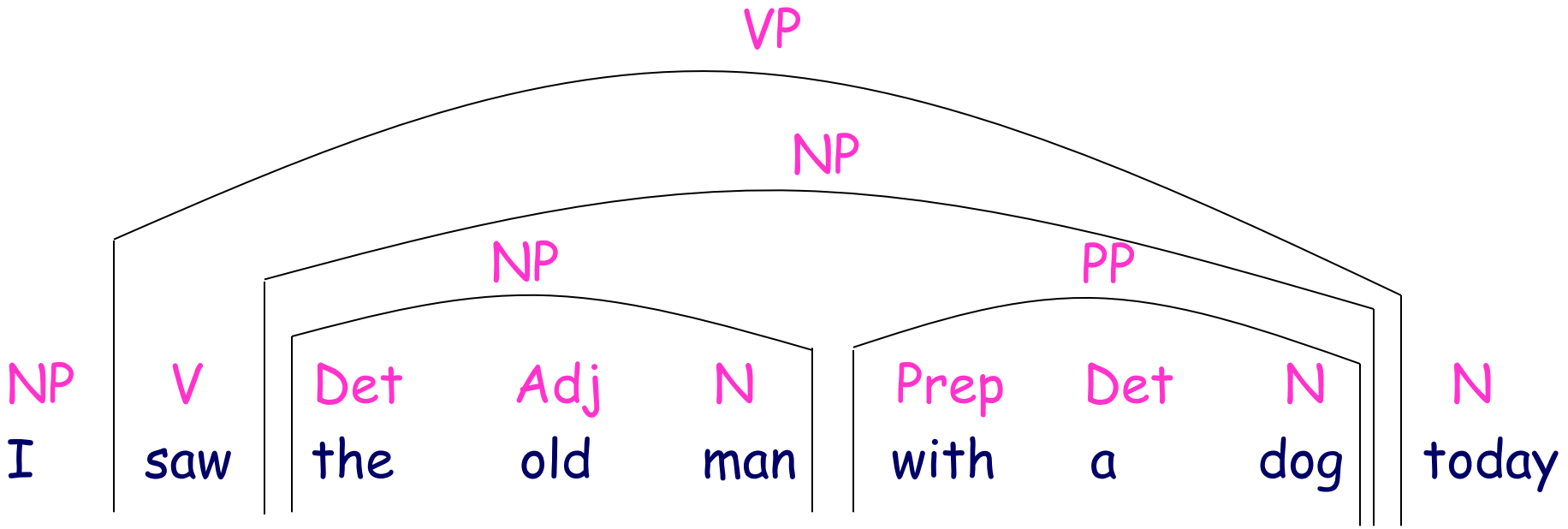
Positions classified as:

- Call positions: both linear and hierarchical outgoing edges
- Return positions: both linear and hierarchical incoming edges
- Internal positions: otherwise

Nested word:

Nested shape + Positions labeled with symbols in Σ

Linguistic Annotated Data



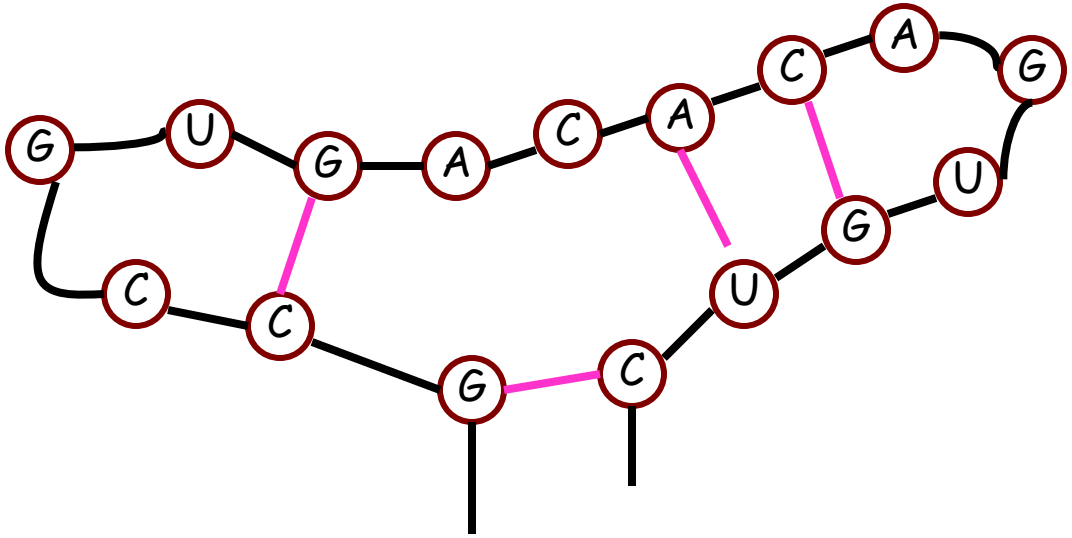
Linguistic data stored as annotated sentences (eg. Penn Treebank)

Sample query: Find nouns that follow a verb which is a child of a verb phrase

RNA as a Nested Word

Primary structure: Linear sequence of nucleotides (A, C, G, U)

Secondary structure: Hydrogen bonds between complementary nucleotides (A-U, G-C, G-U)

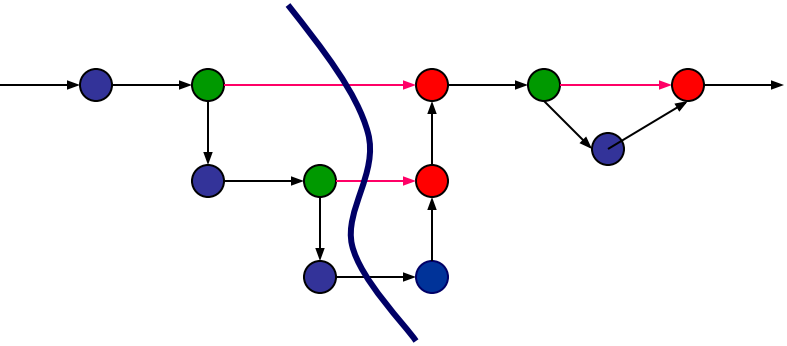


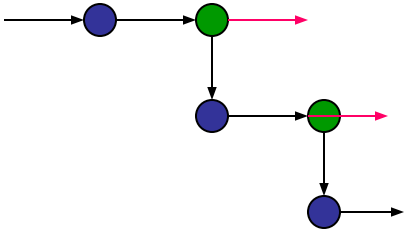
In literature, this is modeled as trees.

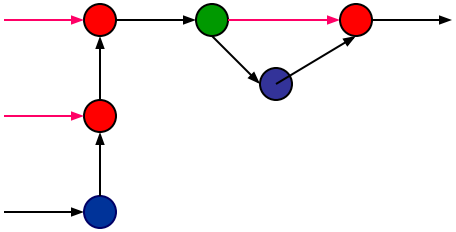
Algorithmic question: Find similarity between RNAs using edit distances

Word operations:

Prefixes, suffixes, concatenation, reverse

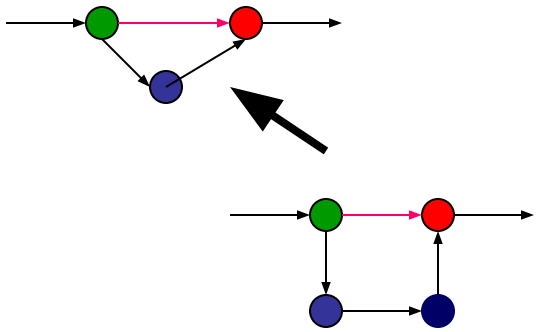


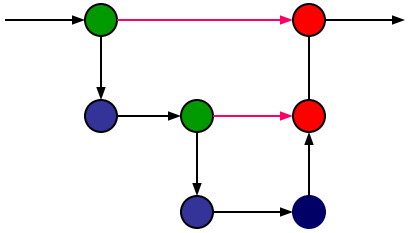




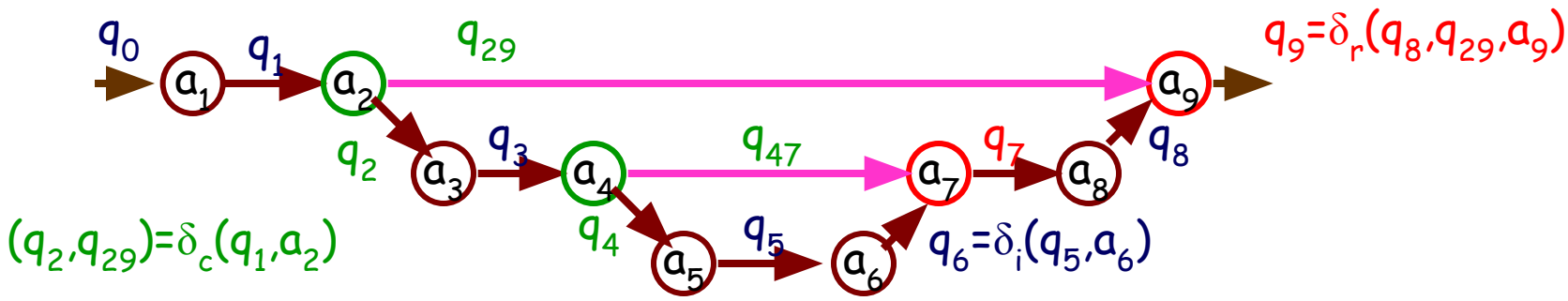
Tree operations:

- Inserting/deleting well-matched words
- Well-matched: no pending calls/returns





Nested Word Automata (NWA)



- States Q, initial state q_0, final states F
- Reads the word from left to right labeling edges with states
- Transition function:
  - $\delta_c : Q \times \Sigma \rightarrow Q \times Q$ (for call positions)
  - $\delta_i : Q \times \Sigma \rightarrow Q$ (for internal positions)
  - $\delta_r : Q \times Q \times \Sigma \rightarrow Q$ (for return positions)
- Nested word is accepted if the run ends in a final state

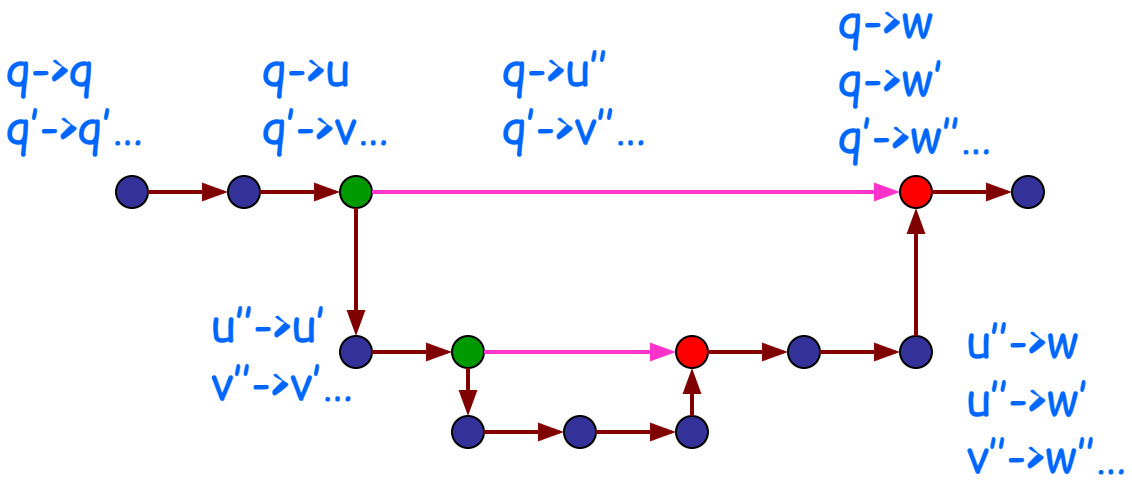
Regular Languages of Nested Words

- A set of nested words is regular if there is a finite-state NWA that accepts it

- Nondeterministic automata over nested words
  - Transition function: δ_c: $Q \times \Sigma \to 2^{Q \times Q}$, δ_i :$Q \times \Sigma \to 2^Q$, δ_r:$Q \times Q \times \Sigma \to 2^Q$
  - Can be determinized: blow-up 2^{n^2}

- Appealing theoretical properties
  - Effectively closed under various operations (union, intersection, complement, concatenation, prefix-closure, projection, Kleene-* …)
  - Decidable decision problems: membership, language inclusion, language equivalence …
  - Alternate characterization: MSO, syntactic congruences

Determinization



Goal: Given a nondeterministic automaton A with states Q, construct an equivalent deterministic automaton B

- Intuition: Maintain a set of "summaries" (pairs of states)
- State-space of B: $2^{Q \times Q}$
- Initially, and after every call, state contains q->q, for each q
- At any step q->q' is in B's state if A can be in state q' when started in state q at the most recent unmatched call position
- Acceptance: must contain q->q', where q is initial and q' is final

Closure Properties

The class of regular languages of nested words is effectively closed under many operations

- Intersection: Take product of automata (key: nesting given by input)
- Union: Use nondeterminism
- Closure under prefixes and suffixes
- Complementation: Complement final states of deterministic NWA
- Concatenation/Kleene*: Guess the split (as in case of word automata)
- Reverse (reversal of a nested word reverses nested edges also)

Decision Problems

- Membership: Is a given nested word w accepted by NWA A?
  - Solvable in polynomial time
  - If A is fixed, then in time O(|w|) and space O(nesting depth of w)

- Emptiness: Given NWA A, is its language empty?

  Solvable in time $O(|A|^3)$: view A as a pushdown automaton

- Universality, Language inclusion, Language equivalence:
  - Solvable in polynomial-time for deterministic automata
  - For nondeterministic automata, use determinization and complementation; causes exponential blow-up, Exptime-complete problems

MSO-based Characterization

- Monadic Second Order Logic of Nested Words
  - First order variables: x,y,z; Set variables: X,Y,Z…
  - Atomic formulas: a(x), X(x), x=y, x < y, x -> y
  - Logical connectives and quantifiers

- Sample formula:

  For all x,y. ((a(x) and x -> y) implies b(y))

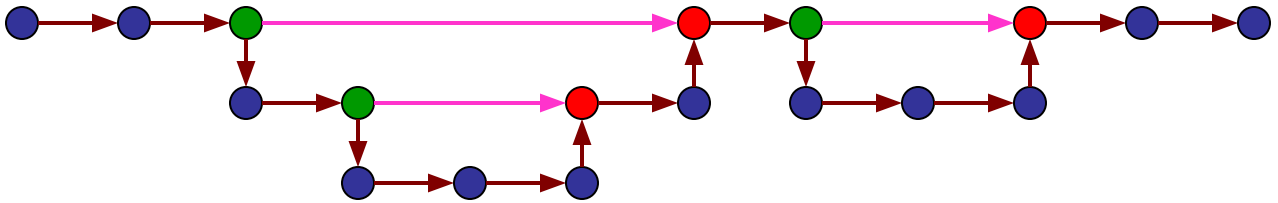
  Every call labeled *a* is matched by a return labeled *b*

- Thm: *A language L of nested words is regular iff it is definable by an MSO sentence*
  - Robust characterization of regularity as in case of languages of words and languages of trees

Application: Software Analysis

- A program *P* with stack-based control is modeled by a set *L* of nested words it generates
  - If *P* has finite data (e.g. pushdown automata, Boolean programs, recursive state machines) then *L* is regular
- Specification *S* given as a regular language of nested words
  - Allows many properties not specifiable in classical temporal logics
  - PAL: instrumentation language of *C* programs (SPIN 2007)
- Verification: Does every behavior in *L* satisfy *S* ?
  - Take product of *P* and complement of *S* and analyze
  - Runtime monitoring: Check if current execution is accepted by S (compiled as a deterministic automaton)
  - Model checking: Check if *L* is contained in *S*, decidable when *P* has finite data (no extra cost, as analysis still requires context-free reachability)

Writing Program Specifications



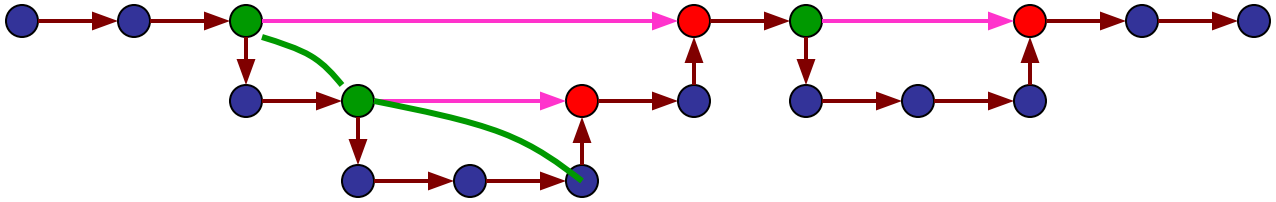
Intuition: Keeping track of context is easy; just skip using a summary edge

- Finite-state properties of paths, where a path can be a local path, a global path, or a mixture

Sample regular properties:

- If *p* holds at a call, *q* should hold at matching return
- If *x* is being written, procedure *P* must be in call stack
- Within a procedure, an *unlock* must follow a *lock*
- All properties specifiable in standard temporal logics (LTL)

Temporal Logic of Nested Time: CaRet



Global paths, Local paths, Caller paths

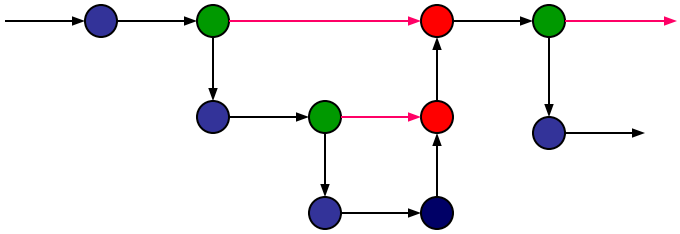
- Three versions of every temporal modality

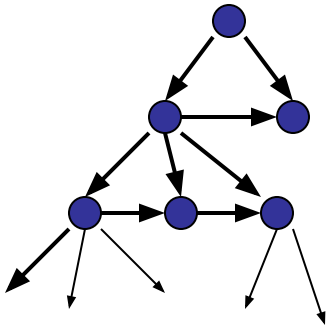
Sample CaRet formulas:

- (if p then local-next q) global-unless r
- if p then caller-eventually q
- Global-always (if p then local-eventually q)

So far: Nested words have appealing theoretical properties with possible applications

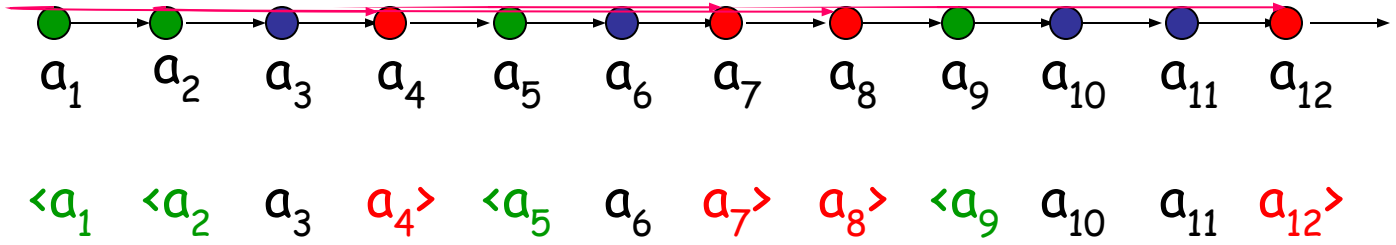
Coming up: How do finite nested words compare with ordered trees/hedges?





Common framework: linear encoding using brackets/tags

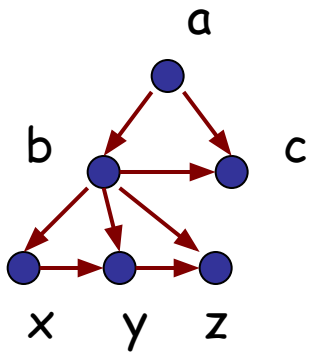
Linear Encoding of Nested Words



Nested word over Σ is encoded as a word over tagged alphabet $\langle\Sigma\rangle$

- For each symbol a, call $\langle a$, return $a\rangle$, internal a
- Two views are isomorphic: every word over $\langle\Sigma\rangle$ corresponds to a nested word over Σ
- Linear view useful for streaming, and word operations such as prefixes
- Number of nested words of length k: $(3\ |\Sigma|)^k$

Encoding Ordered Trees/Hedges

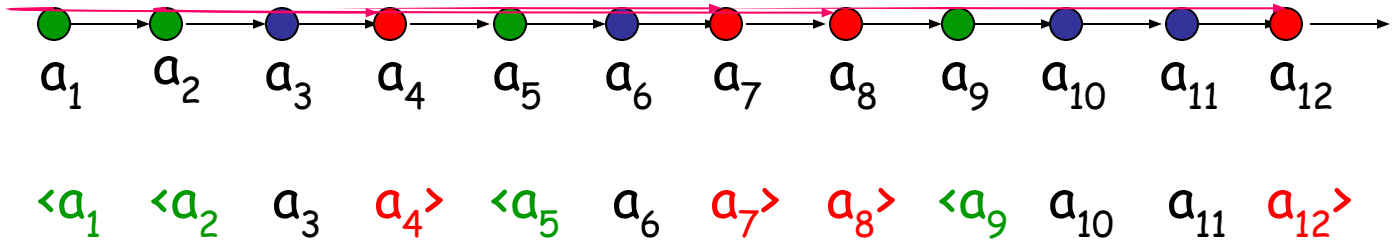


<a <b <x x> <y y> <z z> b> <c c> a>

An ordered tree/hedge over Σ is encoded as a word over $\langle\Sigma\rangle$

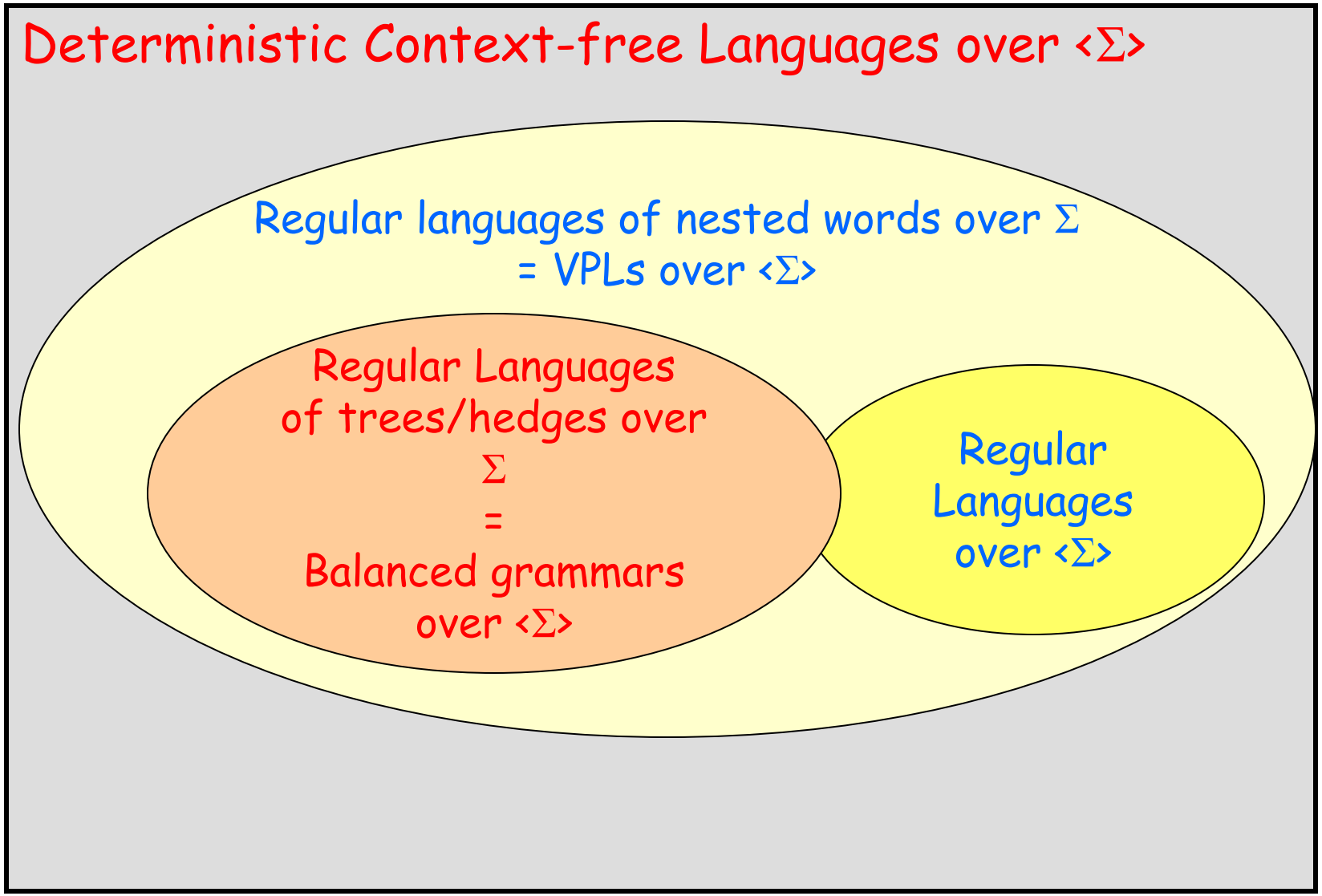
- For a node labeled a, print <a, process children in order, print a>
- Same as SAX representation of XML
- Hedge words: Words over $\langle\Sigma\rangle$ that correspond to ordered forests
  1. Well-matched (no pending calls/returns)
  2. No internals
  3. Matching calls and returns have same symbol
- Note: Tree traversals are not closed under prefixes/suffixes

Relating to Word languages



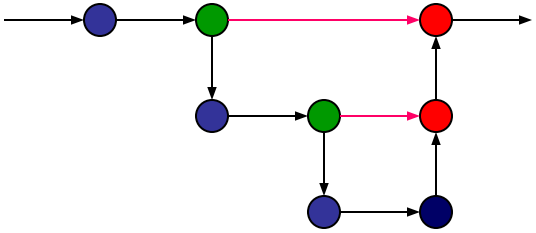
Visibly Pushdown Automata

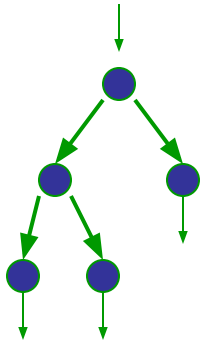
- Pushdown automaton that must push while reading a call, must pop while reading a return, and not update stack on internals
- Visibly pushdown language over $\langle\Sigma\rangle$ is word encoding of a regular language of nested words over Σ
- VPLs form a subclass of deterministic context-free languages

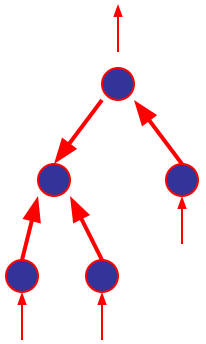

Deterministic Context-free Languages over <Σ>
Regular languages of nested words over Σ
= VPLs over <Σ>
Regular Languages
of trees/hedges over
Σ
=
Balanced grammars
over <Σ>
Regular
Languages
over <Σ>

Comparing NWAs with Tree Automata

- Over hedge words same expressiveness
- Same complexity of analysis problems (e.g. emptiness test: cubic)
- What about succinctness? Succinctness -> better query complexity







Nested Word Automata:

Call: $\delta_c : Q \times \Sigma \rightarrow Q \times Q$

Return: $\delta_r : Q \times Q \times \Sigma \rightarrow Q$

Deterministic are sufficient

Bottom-up Tree Automata:

(binary trees) $\delta : Q \times Q \times \Sigma \rightarrow Q$

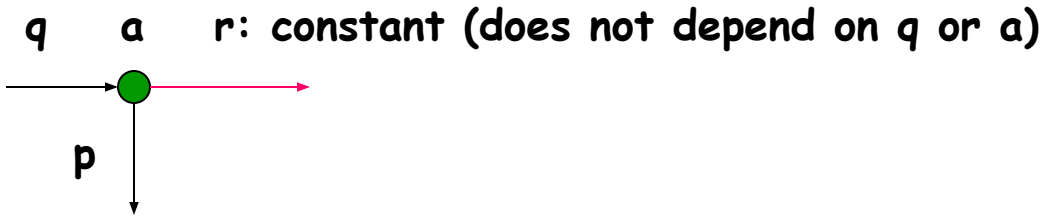
deterministic are sufficient

Top-down Tree Automata:

(binary trees) $\delta : Q \times \Sigma \rightarrow Q \times Q$

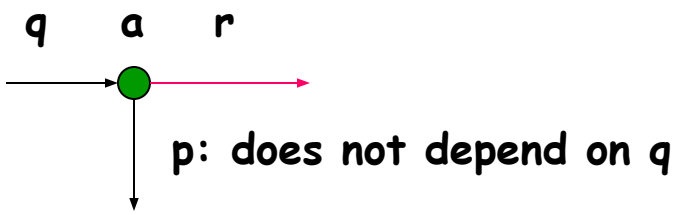
deterministic are not sufficient

Flat Automata



- Flat NWAs: no information flows across summary edges
- Syntactic special case: if $\delta_c(q,a)=(p,r)$ then $r=q_0$
- Flat NWAs are exactly like word automata: Every (non)deterministic word automaton can be interpreted as a flat (non)deterministic NWA with same number of states
- NWAs are more expressive than flat NWAs
- Exponential succinctness of NWAs: There exists a family L_s of regular word languages over $\langle\Sigma\rangle$ such that each L_s has NWA with $O(s)$ states, but every nondeterministic word automaton for L_s must have 2^s states

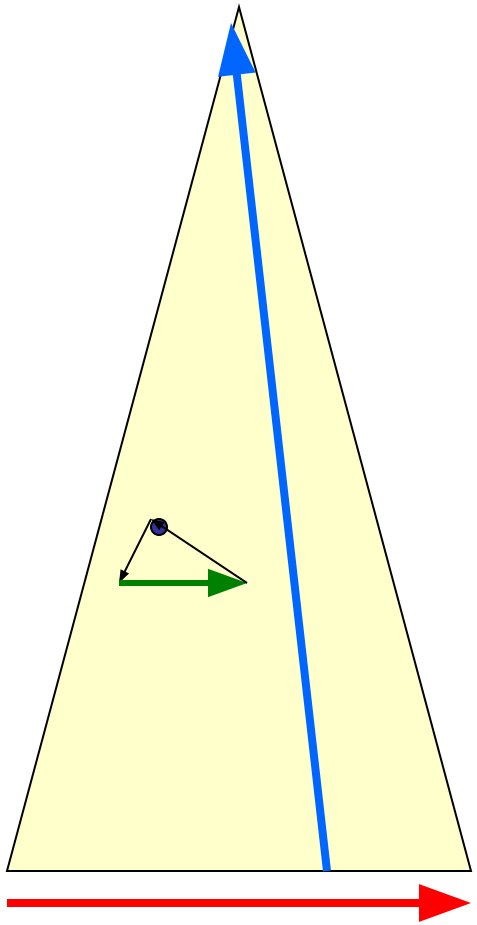
Bottom-up Automata



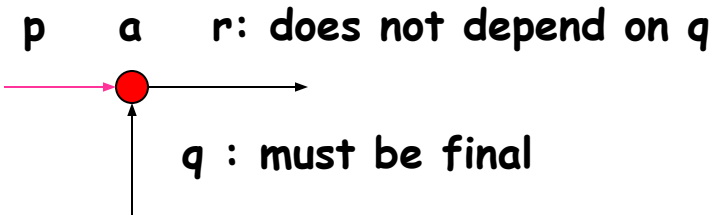
- Bottom-up NWAs: Processing of a nested subword does not depend on the current state
- Syntactic special case: if $\delta_c(q,a)=(p,r)$ and $\delta_c(q',a)=(p',r')$ then $p=p'$
- Step-wise bottom-up tree automata are a special case of bottom-up NWAs (i.e. no blow-up from bottom-up tree automata to NWAs)
- Over well-matched words, deterministic bottom-up NWAs can specify all regular languages of nested words
- Exponential succinctness of NWAs: There exists a family L_s of regular languages nested words such that each L_s has NWA with $O(s)$ states, but every bottom-up NWA for L_s must have 2^s states

Expressing Linear Queries with Tree Automata

- Tree Automata can naturally express constraints on sequence of labels along a tree path and also along a sibling path
- Linear order over all nodes (or all leaves) is only a derived relation, and query over this order is difficult to express
- For a regular word language L, consider the query: is the sequence of leaves (left-to-right) in L?
- For $L = \Sigma^* a_1 \Sigma^* a_2 \ldots \Sigma^* a_s \Sigma^*$, there a flat NWA of size $O(s)$, but every bottom-up automaton must have 2^s states
- Implication: Processing a document as a word (text-string) may be beneficial than processing it as a tree!

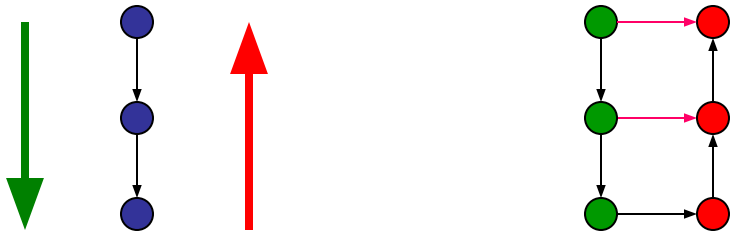


Top-down Automata



- Only information flowing across a return edge: whether inside subword is accepted or not
- Return transition relation specified $\delta_r^h : Q \times \Sigma \rightarrow 2^Q$ such that r in $\delta_r(q,p,a)$ iff q in F and q in $\delta_r^h(p,a)$
- Every (non)deterministic top-down tree automaton can be translated to an equivalent (non)deterministic top-down NWA with same number of states
- Over well-matched words, nondeterministic top-down NWAs are as expressive as NWAs (but deterministic top-down NWAs are less expressive)
- See Joinless NWAs in paper (both top-down & flat are special cases)

Processing Paths



- For a language L of words, let path(L) be language of unary trees such that the sequence of labels of nodes on the path is in L
- The minimal deterministic top-down tree automaton for path(L) is same as the minimal DFA for L
- The minimal deterministic bottom-up tree automaton for path(L) is same as the minimal DFA for Reverse(L)
- The minimal NWA for path(L) can be exponentially smaller than both these

Pushdown Automata over Nested Words

- Nondeterministic joinless transition relation
- Finite-state control augmented with stack
- Expressiveness: Contains both context-free word languages and context-free tree languages
- Example: Language of trees with same number of a-labeled nodes as b-labeled nodes

  Context-free tree languages do not include context-free word languages
- Membership: NP-complete (as for pushdown tree automata)
- Emptiness: EXPTIME-complete (as for pushdown tree automata)
- Inclusion/Equivalence: Undecidable (as for pushdown word automata)

Related Work

- Restricted context-free languages
  - Parantheses languages, Dyck languages
  - Input-driven languages
- Logical characterization of context-free languages (LST'94)
- Connection between pushdown automata and tree automata
  - Set of parse trees of a CFG is a regular tree language
  - Pushdown automata for query processing in XML
- Algorithms for pushdown automata compute summaries
  - Context-free reachability
  - Inter-procedural data-flow analysis
- Game semantics for programming languages (Abramsky et al)
- Model checking of pushdown automata
  - LTL, CTL, μ-calculus, pushdown games
  - LTL with regular valuations of stack contents
  - CaRet (LTL with calls and returns)

Conclusions

1. Nested words for modeling data with linear + hierarchical structure
   - Words are special cases; ordered trees/hedges can be encoded
   - Correct parsing is not a pre-requisite
   - Allow both word operations and tree operations

2. Regular languages of nested words have appealing properties
   - Closed under various operations
   - Multiple characterizations
   - Solvable decision problems (typically same complexity as tree automata)
   - Theory connects pushdown automata and tree automata

3. Nested word automata
   - Word automata, top-down tree automata, and bottom-up tree automata are all special cases
   - Traversal is natural for streaming applications
   - Exponential succinctness without any extra cost in analysis

Ongoing and Future Work

- Many follow-up papers/results already published
- Can the results be used to improve XML query processing ?
- Minimization
- Infinite nested words and temporal logics (see LICS'07)
- Two-way automata and transducers
- Nested trees (dually hierarchical structures)