

Taxonomies and toolkits of regular tree language algorithms

Loek Cleophas

loek@loekcleophas.com

<http://www.win.tue.nl/~lcleopha>

Software Engineering & Technology Group
Department of Mathematics & Computer Science



2IS95 SET seminar
September 18th 2008

TU / **e**

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

- **Context**

Regular string and tree language theory, domain deficiencies, role of taxonomies & toolkits

- **Domain**

Tree algorithms, trees, tree pattern matching, tree grammars, match sets & tree automata, applications, algorithms

- **Taxonomies**

Algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Toolkits**

Motivation, toolkit vs. taxonomy, tree toolkit content, algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Concluding Remarks & Research Problems**

- **Context**

Regular string and tree language theory, domain deficiencies, role of taxonomies & toolkits

- **Domain**

Tree algorithms, trees, tree pattern matching, tree grammars, match sets & tree automata, applications, algorithms

- **Taxonomies**

Algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Toolkits**

Motivation, toolkit vs. taxonomy, tree toolkit content, algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Concluding Remarks & Research Problems**

- SET research themes:
 - Model-driven software engineering
 - Verified software engineering
 - Generic language technology
 - Theory, tools and applications
 - Taxonomies and toolkits of formal language algorithms
 - Taxonomies and toolkits of *regular tree language* algorithms

- *Regular Grammar (and Regular Expression)*
 - Different types, transformations between them
- *Finite Automaton*
 - Nondeterministic with/without ε -transitions, deterministic
- **Theoretical Results**
 - Equivalence of *NFA* and *DFA* (subset construction)
 - Equivalence of *RG*, *RE*, and *FA*
- **Problems**
 - *Membership/Acceptance* $s \in L$
 - *Keyword Pattern Matching (KPM)*
 - ...

- Theoreticians (1950s):
 - Solve by constructing and using *FA* based on *RG/RE*
 - Done, move on!
- In practice (1960s - now):
 - Many applications (text search, *DNA* processing)
 - Many *FA* constructions, acceptance/*KPM* algorithms
 - More efficient; for specific situations
 - Difficult to find, understand, compare
 - Separation between theory and practice
 - Hard to compare and choose implementations
- Taxonomies and toolkits (*Watson, 1995; Watson & Zwaan, 1996; Cleophas, Watson & Zwaan, 2004*)

- *Regular Tree Grammar (and Regular Tree Expression)*
 - Different types, transformations between them
- *Finite Tree Automaton (TA)*
 - Nondeterministic with/without ε -transitions, deterministic
 - Undirected, root-to-frontier (*RF*), frontier-to-root (*FR*)
- **Theoretical Results**
 - Equivalence of *TAs* (except *DRFTA*) (subset construction)
 - Equivalence of *RTG*, *RTE*, and *TA* (except *DRFTA*)
- **Problems**
 - *Membership/Tree (Grammar) Acceptance (TGA)*:
Given a regular tree grammar and an input tree, determine whether the input tree is in the language generated by the grammar.
 - *Tree Pattern Matching (TPM)*
 - ...

- Theoreticians (1960s):
 - Solve by constructing and using *TA* based on *RTG/RTE*
 - Done, move on!
- In practice (1975 - now):
 - Applications in code generation, term rewriting
 - Many *TA* constructions, *TGA/TPM* algorithms
 - More efficient; for specific situations
 - Difficult to find, understand, compare
 - Separation between theory and practice
 - Hard to compare and choose implementations
- Taxonomies and toolkit (*Cleophas, Hemerik & Zwaan, 2005/2006; Strolenberg, 2007; Cleophas, 2008*)

- Inaccessibility of theory and algorithms
- Difficulty of comparing algorithms
 - Difference in style
 - Difference in formality
- Distance between theory & practice
- Lack of large collection of implementations
- Difficulty of choosing between algorithms

Taxonomy shows commonality & variation in algorithm structure & data representation, discusses theoretical complexity, not actual performance

- Inaccessibility of theory and algorithms
- Difficulty of comparing algorithms
 - Difference in style
 - Difference in formality
- Distance between theory & practice
- Lack of large collection of implementations
- Difficulty of choosing between algorithms

Toolkit & GUI give insight into algorithm properties, performance; starting point in solving last two deficiencies

- **Context**

Regular string and tree language theory, domain deficiencies, role of taxonomies & toolkits

- **Domain**

Tree algorithms, trees, tree pattern matching, tree grammars, match sets & tree automata, applications, algorithms

- **Taxonomies**

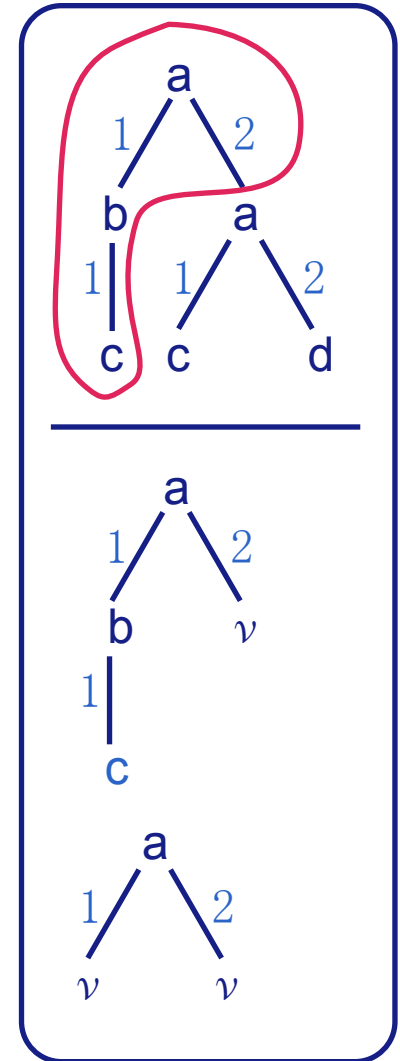
Algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Toolkits**

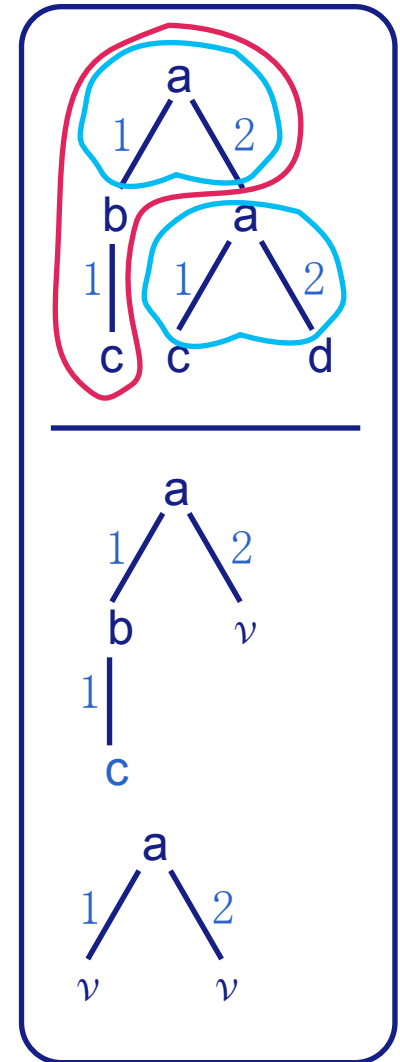
Motivation, toolkit vs. taxonomy, tree toolkit content, algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Concluding Remarks & Research Problems**

- Algorithms ...
 - for acceptance, pattern matching, parsing
 - on node-labeled, ordered, ranked trees
- Usable in ...
 - compilers, for instruction selection
 - term rewriting



- Algorithms ...
 - for acceptance, pattern matching, parsing
 - on node-labeled, ordered, ranked trees
- Usable in ...
 - compilers, for instruction selection
 - term rewriting



Domain

Regular Tree Grammars

$$(1) S \rightarrow \begin{array}{c} a \\ / \quad \backslash \\ B \quad d \end{array},$$

$$(2) S \rightarrow \begin{array}{c} a \\ / \quad \backslash \\ b \quad B \\ | \\ c \end{array},$$

$$(3) S \rightarrow c,$$

$$(4) B \rightarrow \begin{array}{c} b \\ | \\ B \end{array},$$

$$(5) B \rightarrow S,$$

$$(6) B \rightarrow d$$

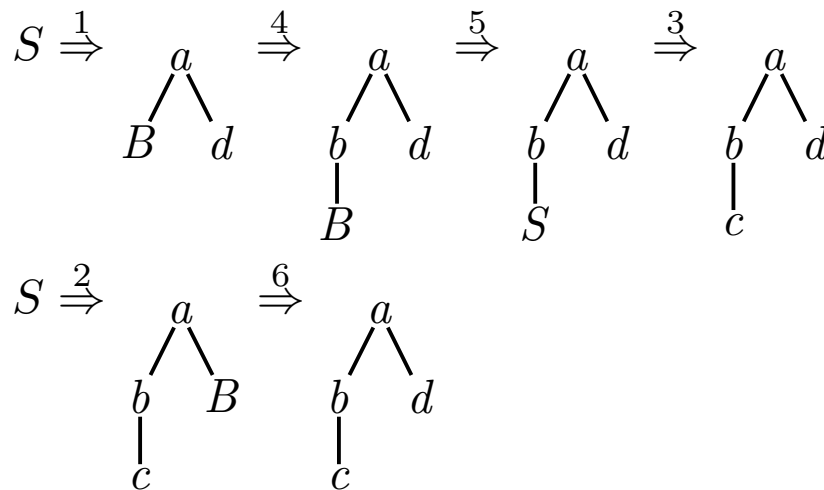
- Generalization of regular string grammar

- Recall right regular string grammar production forms

$$A \rightarrow wB, A \rightarrow w \quad (w \in \Sigma^*)$$

- Regular tree grammar

- Form $A \rightarrow t$ with t a tree, nonterminals at leaves



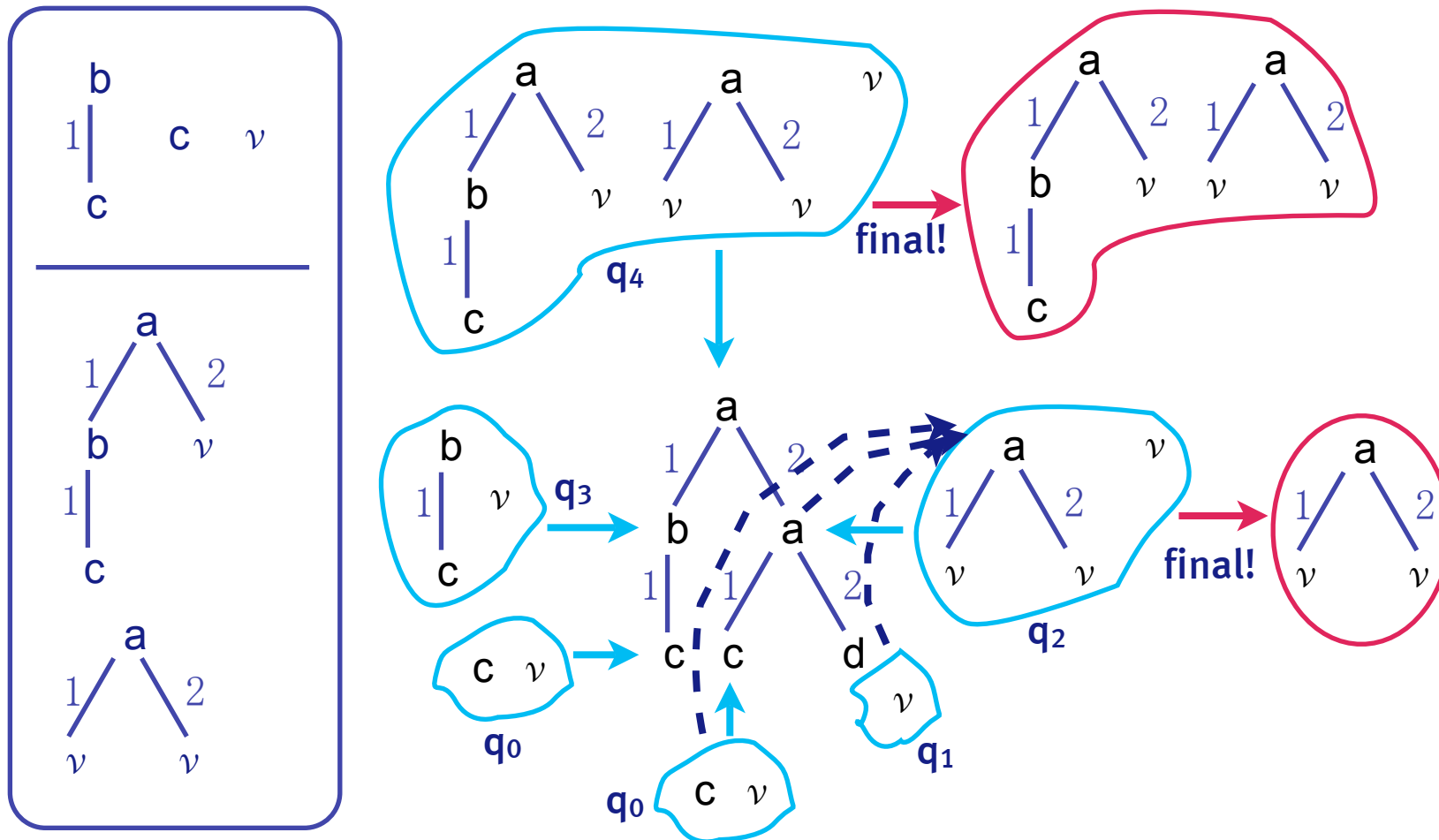
Domain

Computing match sets; tree automata

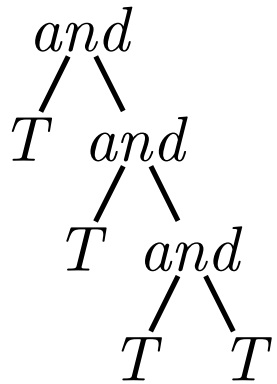
Idea: Compute matching *sub*patterns -> includes patterns

Idea: Do so recursively

Idea: Precompute, tabulate -> Det. FR tree automaton



- Need to find instances of rewrite rules' left hand sides in subject
- Tree Pattern Matching!



$$(1) \begin{array}{c} \text{and} \\ / \quad \backslash \\ T \quad T \end{array} \rightarrow T \qquad (2) \begin{array}{c} \text{and} \\ / \quad \backslash \\ \nu \quad F \end{array} \rightarrow F$$

$$(3) \begin{array}{c} \text{and} \\ / \quad \backslash \\ F \quad \nu \end{array} \rightarrow F \qquad (4) \begin{array}{c} \text{and} \\ / \quad \backslash \\ \nu_1 \quad \text{and} \\ \quad / \quad \backslash \\ \quad \nu_2 \quad \nu_3 \end{array} \rightarrow \begin{array}{c} \text{and} \\ / \quad \backslash \\ \text{and} \quad \nu_3 \\ / \quad \backslash \\ \nu_1 \quad \nu_2 \end{array}$$

Domain

Application: Instruction Selection

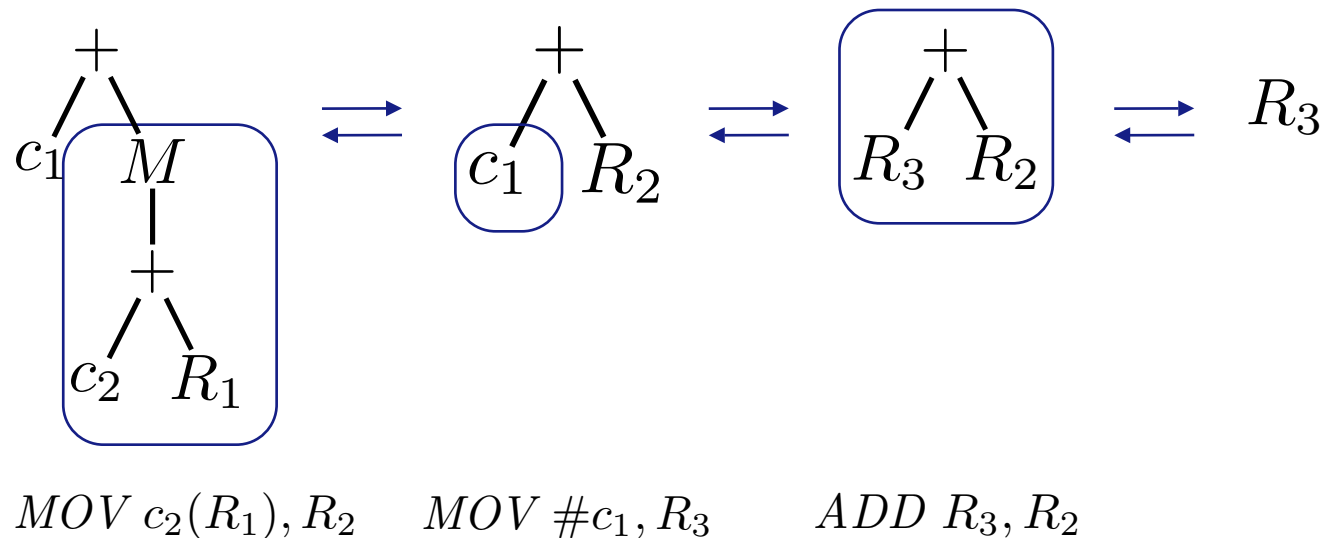
$$(1) R_i \rightarrow c \\ \text{MOV } \#c, R_i$$

$$(2) R_i \rightarrow M \\ \quad \quad \quad | \\ \quad \quad \quad R_j \\ \text{MOV } *R_j, R_i$$

$$(3) R_i \rightarrow \begin{array}{c} M \\ | \\ + \\ / \quad \backslash \\ c \quad R_j \end{array} \\ \text{MOV } c(R_j), R_i$$

$$(4) R_i \rightarrow \begin{array}{c} + \\ / \quad \backslash \\ R_i \quad R_j \end{array} \\ \text{ADD } R_i, R_j$$

- Given intermediate representation tree, determine covering/instructions
- Use RTG; each production has associated instruction; determine derivation to obtain covering/instructions
- Tree Parsing, extending Tree Acceptance!



Domain

Appearance of algorithms

- *Brainerd, 1967 & 1969*
- *Kron, 1975*
- *Hoffmann & O'Donnell, 1980 & 1982*
- *Hatcher, 1985; Hatcher & Christopher, 1986*
- *Turner, 1986*
- *van Dinther, 1987*
- *Chase, 1987*
- *Aho, Ganapathi & Tjang, 1985, 1988*
- *van de Meerakker, 1988*
- *Weisgerber & Wilhelm, 1989*
- *Hemerik & Katoen, 1989*
- *Balachandran, Dhamdhere & Biswas, 1990*
- *Ferdinand, Seidl & Wilhelm, 1994*
- *Wilhelm & Mauer, 1995*
- *Comon et al., 2003*
- *Cleophas, Hemerik & Zwaan, 2005 & 2006*
- *Cleophas, 2008*

- **Context**

Regular string and tree language theory, domain deficiencies, role of taxonomies & toolkits

- **Domain**

Tree algorithms, trees, tree pattern matching, tree grammars, match sets & tree automata, applications, algorithms

- **Taxonomies**

Algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Toolkits**

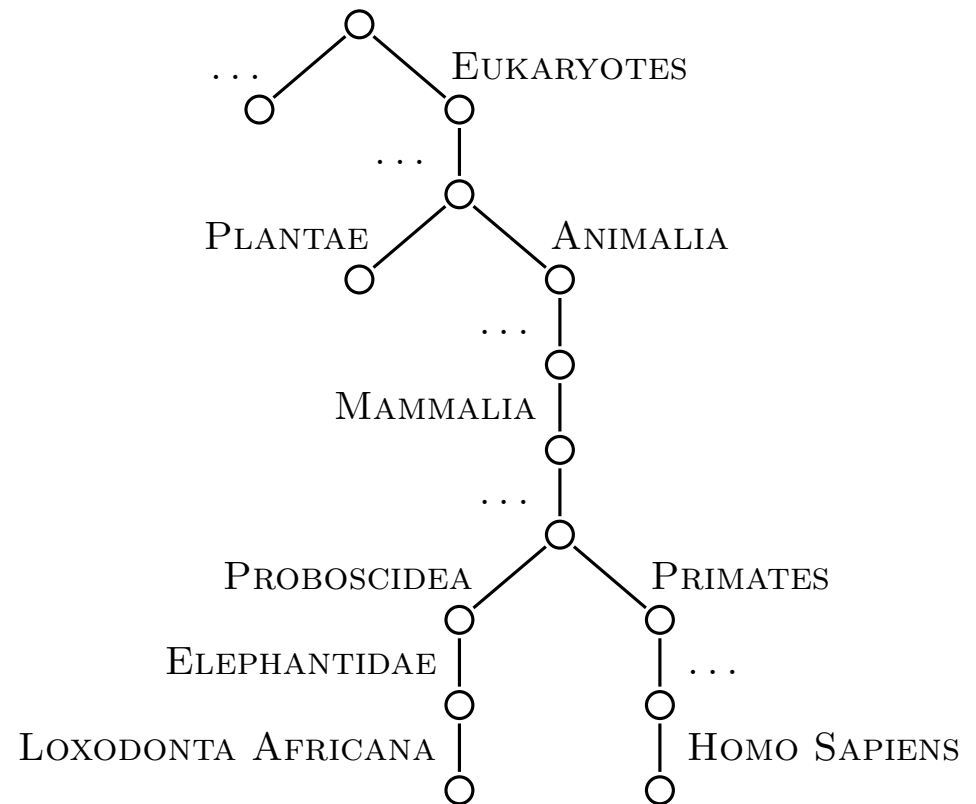
Motivation, toolkit vs. taxonomy, tree toolkit content, algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Concluding Remarks & Research Problems**

Taxonomies

Biological Taxonomies

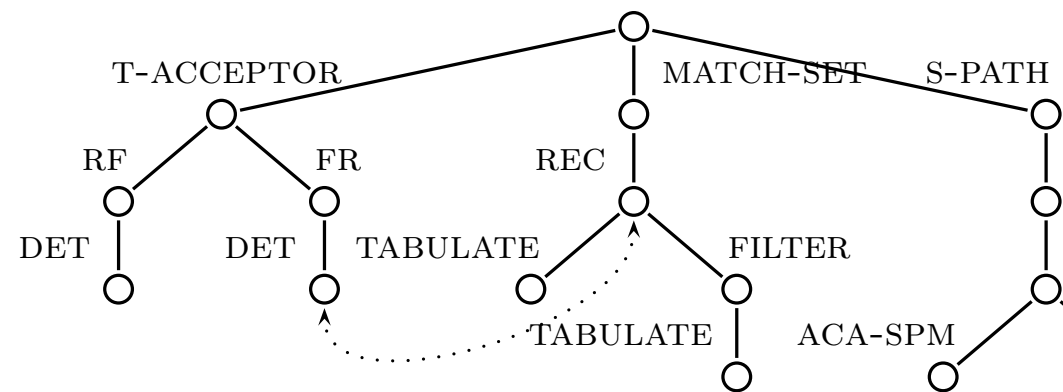
- Classify *organisms*
- From abstract, general to concrete, specific
- Properties (details) explicit
- Allow comparison



Taxonomies

Algorithm Taxonomies

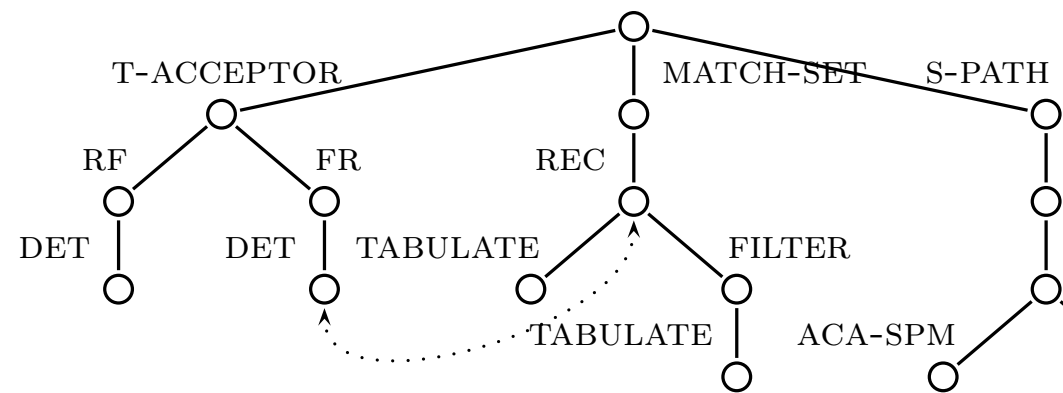
- *Algorithm taxonomies classify algorithms*
- From abstract, general to concrete, specific
- Properties (details) explicit
- Allows comparison



Taxonomies

Algorithm Taxonomy – I

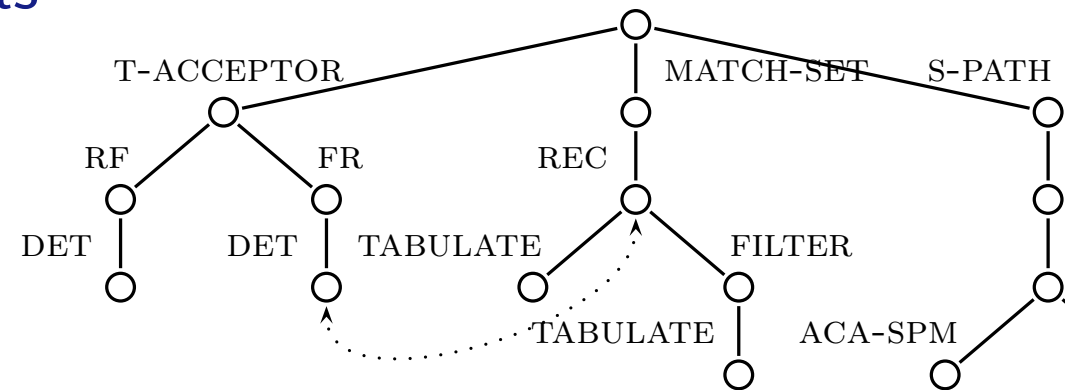
- Classification of *algorithms* based on their *essential details*
- Algorithms in it solve one algorithmic problem
- Depicted as tree/directed acyclic graph
 - Nodes refer to algorithms
 - Branches refer to details
- Root represents high-level algorithm
 - Correctness easily shown



Taxonomies

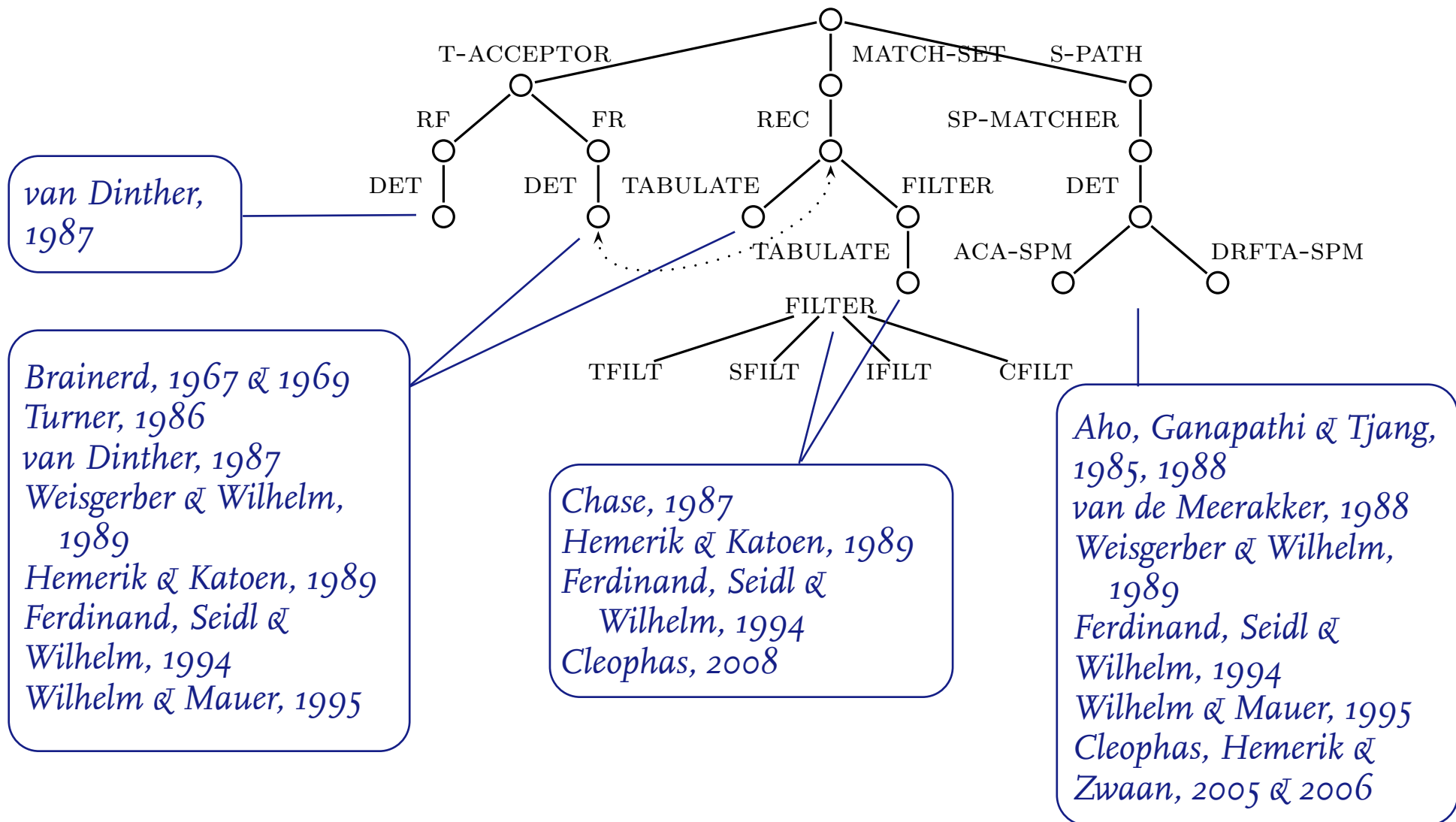
Algorithm Taxonomy – II

- Adding *detail* to algorithm
 - Depicted by branch connecting algorithm node to new child node
 - Obtains refinement or variation
 - Leads to algorithms from literature, new algorithms
 - Associated correctness arguments
- Algorithm correctness follows from correctness of root and of details added



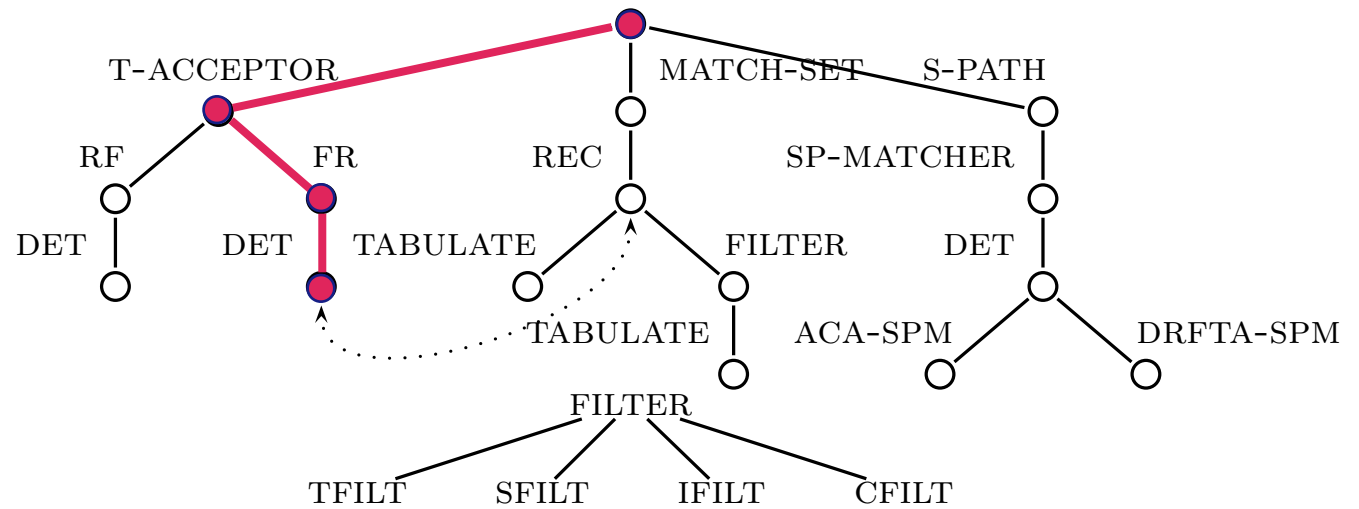
Taxonomies

Tree Acceptance Taxonomy



Taxonomies

One Algorithm Path – I



Taxonomies

One Algorithm Path — II

()

```
[[ const  $G = \dots;$ 
     $t : \dots;$ 
    var  $b : \mathbb{B}$ 
    |  $b := t \in \mathcal{L}(G)$ 
  ]]
```

(T-ACCEPTOR)

```
[[ const  $G = \dots;$ 
     $t : \dots;$ 
    var  $b : \mathbb{B}$ 
    | let  $M = \dots$  be a TA such that  $\mathcal{L}(M) = \mathcal{L}(G);$ 
       $b := t \in \mathcal{L}(M)$ 
  ]]
```

(T-ACCEPTOR, FR)

```
[[ const  $G = \dots$ ;  
     $t : \dots$ ;  
    var  $b : \mathbb{B}$   
| let  $M = \dots$  be an  $\varepsilon$ NFRTA such that  $\mathcal{L}(M) = \mathcal{L}(G)$ ;  
  
     $b := \text{Traverse}(\varepsilon) \cap Q_{ra} \neq \emptyset$   
  
    func  $\text{Traverse}(\downarrow n : D) : \mathcal{P}(Q) =$   
    [[ var  $s_1, \dots, s_n : \mathcal{P}(Q)$   
    | let  $a = t(\mathbf{n})$ ;  
    | if  $n > 0 \rightarrow$   
    |   for  $i : 1 \leq i \leq n \rightarrow$   
    |      $s_i := \text{Traverse}(\mathbf{n} \cdot i)$   
    |   rof;  
    |    $\text{Traverse} := \emptyset$ ;  
    |   for  $(q_1, \dots, q_n) : q_1 \in s_1, \dots, q_n \in s_n \rightarrow$   
    |      $\text{Traverse} := \text{Traverse} \cup R_\varepsilon^*(R_a(q_1, \dots, q_n))$   
    |   rof  
    |  $n = 0 \rightarrow$   
    |    $\text{Traverse} := R_\varepsilon^*(R_a())$   
    | fi  
    ]]  
  ]]
```

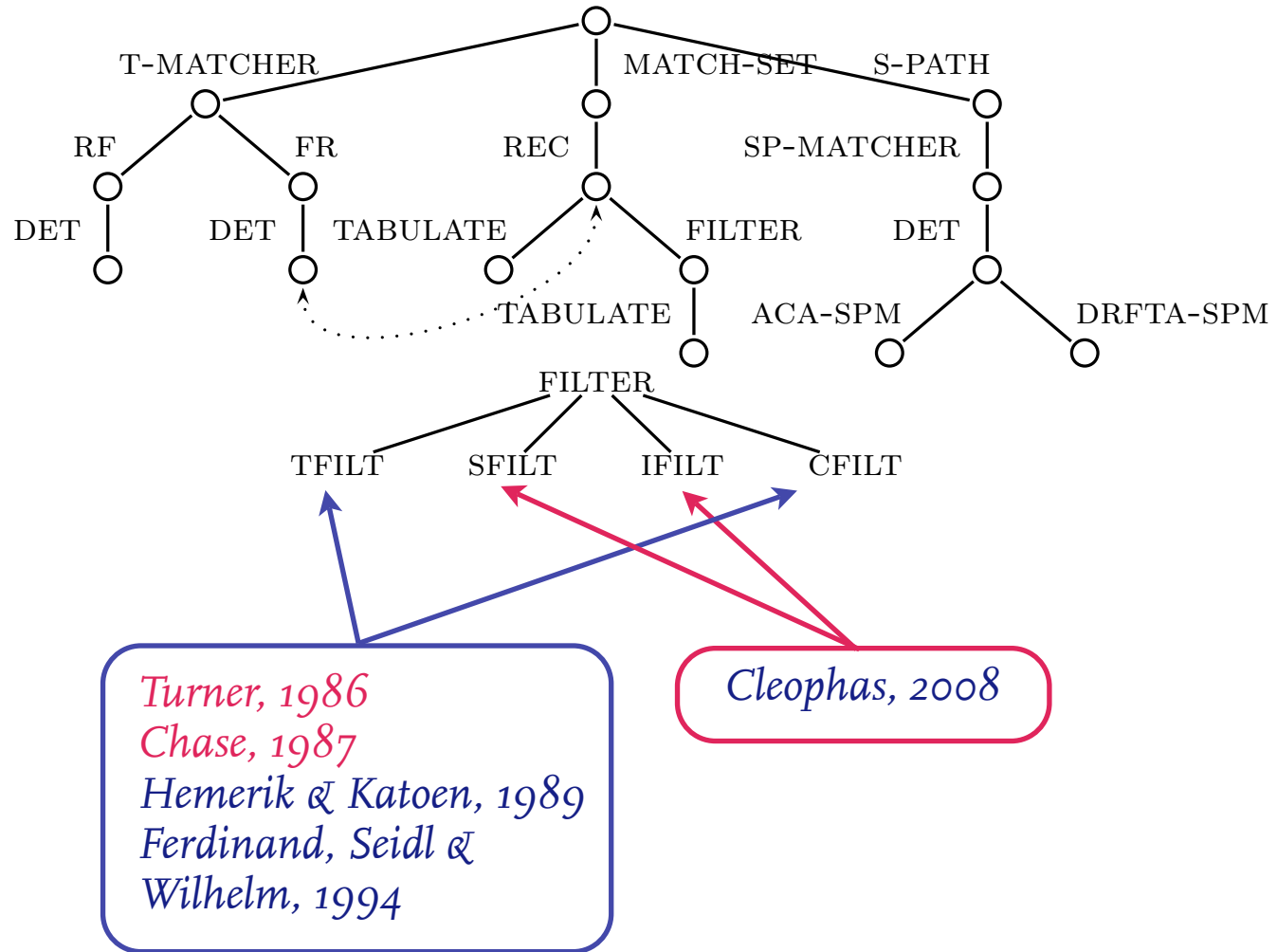
(T-ACCEPTOR, FR, DET)

```
[[ const  $G = \dots$ ;  
     $t : \dots$ ;  
    var  $b : \mathbb{B}$   
| let  $M = \dots$  be a DFRTA such that  $\mathcal{L}(M) = \mathcal{L}(G)$ ;  
  
     $b := \text{Traverse}(\varepsilon) \in Q_{ra}$   
  
    func  $\text{Traverse}(\downarrow \mathbf{n} : D) : Q =$   
    [[ var  $q_1, \dots, q_n : Q$   
    | let  $a = t(\mathbf{n})$ ;  
    | if  $n > 0 \rightarrow$   
         $\text{Traverse} := R_a(\text{Traverse}(\mathbf{n} \cdot 1), \dots, \text{Traverse}(\mathbf{n} \cdot n))$   
    |  $n = 0 \rightarrow$   
         $\text{Traverse} := R_a()$   
    | fi  
    ] ]
```

- 1 basic, undirected construction
- 3 different state sets; correspond to the set of all subtrees of *RTG* production rules, or reduction
- Add direction: undirected, *RF*, *FR*
- Remove ε -transitions
- Make deterministic by subset construction
- For *DFRTA*, additional *filtering* to reduce transition tables, based on *RTG*/patterns' structure
- Over 30 constructions in total
- Similar kind/amount for tree pattern matching taxonomy

Taxonomies

New filters → new algorithms



- **Context**

Regular string and tree language theory, domain deficiencies, role of taxonomies & toolkits

- **Domain**

Tree algorithms, trees, tree pattern matching, tree grammars, match sets & tree automata, applications, algorithms

- **Taxonomies**

Algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Toolkits**

Motivation, toolkit vs. taxonomy, tree toolkit content, algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Concluding Remarks & Research Problems**

Toolkit

Motivation

- Why?
 - Hard to find implementations, let alone collections of them
 - Compare performance, verify assumptions on algorithms made in literature
 - Use to experiment and gain insight
- Design based on taxonomies
 - Factoring, commonalities and variations suggest design
 - Implementation easy given abstract algorithms

```
func Traverse( $\downarrow n : D$ ) :  $Q =$   
[[ var  $q_1, \dots, q_n : Q$   
 | let  $a = t(n)$ ;  
 | if  $n > 0 \rightarrow$   
 |    $Traverse := R_a(Traverse(n \cdot 1), \dots, Traverse(n \cdot n))$   
 | |  $n = 0 \rightarrow$   
 | |    $Traverse := R_a()$   
 | fi  
 | ]]
```

```
private static AbstractAutomatonState Traverse(AbstractDFRTA M, Node n)  
{  
  AbstractTASState[] childStates = new AbstractTASState[n.children().size()];  
  for (int i=0; i < n.children().size(); i++)  
  { childStates[i] = Traverse(M, n.children().get(i)); }  
  
  if (n.children().size() > 0) {  
    state = M.nextState(childStates, (RankedSymbol)n.symbol());  
  }  
  else { state = M.nextState(childStates, (RankedSymbol)n.symbol()); }  
  
  return state;  
}
```

Toolkit

Content

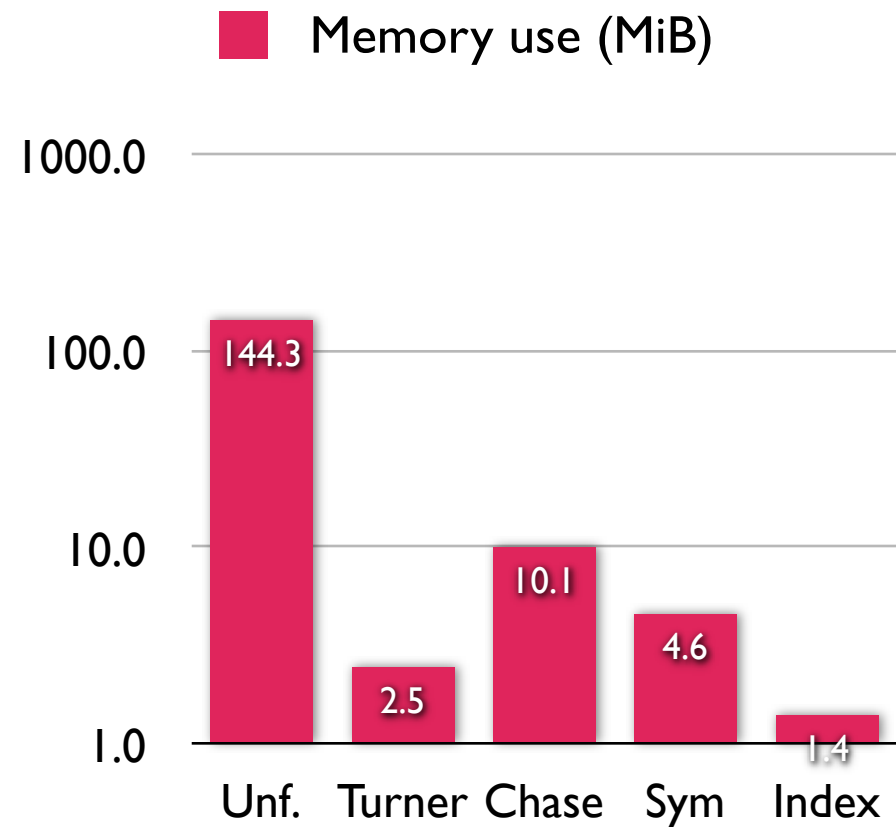
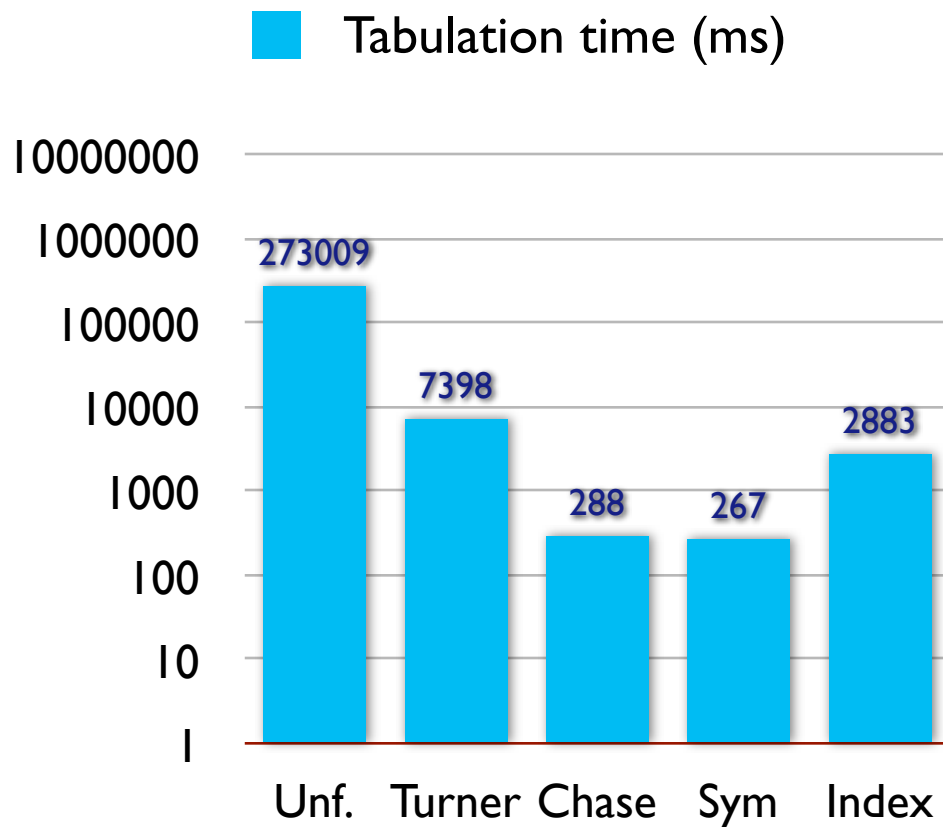
- Representations of ...
 - trees, tree patterns, regular tree grammars, tree automata
- Tree pattern matching, acceptance, parsing algorithms
 - most algorithms near taxonomy leafs
 - corresponding automata generators/tabulators
- Analyses, transformations (chain rule removal, epsilon transition removal)
- Implementation
 - *Forest FIRE* toolkit 82 interfaces/classes, ~8800 LOC
 - *FIRE Wood* GUI 56 interfaces/classes, ~7400 LOC
 - *Java, SWT*
 - *OS X, Windows XP, Linux*

Toolkit

Some Results

- Provides large, coherent set of implementations
 - Simplify choice, use
- Most work was in selecting efficient representations for the basic data structures
- Adding algorithms given basic data structures quite easy
 - Pseudo-code part of algorithms' presentation in taxonomies
 - Even further extensions: two tree parsing algorithms done in 2 hours
- New practical algorithms
 - Two new filters outperforming existing ones

- Intel X86 CPU



Demo?

Demo?

- **Context**

Regular string and tree language theory, domain deficiencies, role of taxonomies & toolkits

- **Domain**

Tree algorithms, trees, tree pattern matching, tree grammars, match sets & tree automata, applications, algorithms

- **Taxonomies**

Algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Toolkits**

Motivation, toolkit vs. taxonomy, tree toolkit content, algorithm taxonomies, taxonomies of tree acceptance and tree pattern matching algorithms

- **Concluding Remarks & Research Problems**

Concluding Remarks

- Classification & implementation of tree algorithms
 - Increase accessibility, link theory and practice
 - Two closely related taxonomies
 - Classify algorithms, show commonalities and differences
 - Give additional insight
 - Lead to new algorithms
 - Taxonomy-based toolkit of algorithms
 - Provides usable implementations
 - Allowed comparing algorithms in practice
 - Gave additional insight

- Tree parsing
- Tree automata minimization
- Tree automata construction based on tree regular expressions
- Combining different tree pattern matching algorithms and different term rewriting strategies
 - vs. just leftmost-innermost; simplifies specification?
- Tree transducer construction