Inventive Algorithmics

Bruce W. Watson
Derrick Kourie
Ina Schaefer (TU Braunschweig)
Loek Cleophas (TU Eindhoven)

fastar
RESEARCH GROUP

Introduction & Motivation

- Inventing new algorithms is tough
  - Depends largely on inate talent, or luck
- There are many still to be invented
- Small fraction of SW is *correctness critical*
  But then it *really matters*
- Standards for automotive, aviation, medical, …

Introduction & Motivation (cont)

- Start with pre- and postcondition
- Co-develop program and annotations
- *Lightweight* correctness-by-construction
  Historically, the “other” camp
  Alternatives?
  - Testing
  - Verification
  - Posthoc proof

Random Quotes

Bjarne Stroustrup

“infrastructure software” has stronger quality and elegance requirements

C.A.R. (Tony) Hoare

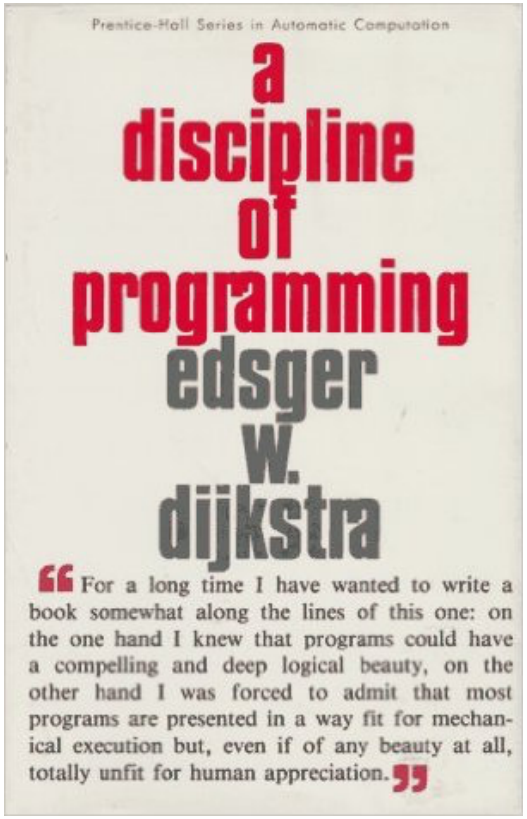
“…taxonomies are to the field of algorithmics what the Standard Model is to Particle Physics…”

fastar
RESEARCH GROUP

CbC in other Engineering Disciplines

- Common in electronic, mechanical, civil, …
- For example, CAD tools:
  - Component-based engineering from components with known properties
  - Standard libraries of building blocks used by drag-and-drop
  - Tools respect component properties and restrictions on composition

Correctness-by-Construction (CbC)



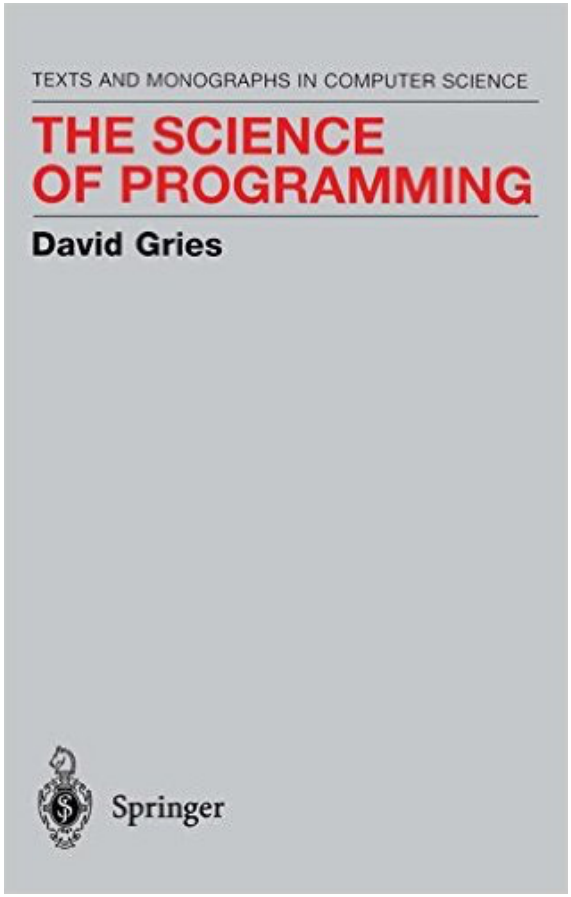
Worthless to the Working Programmer - Great for Computer Scientists

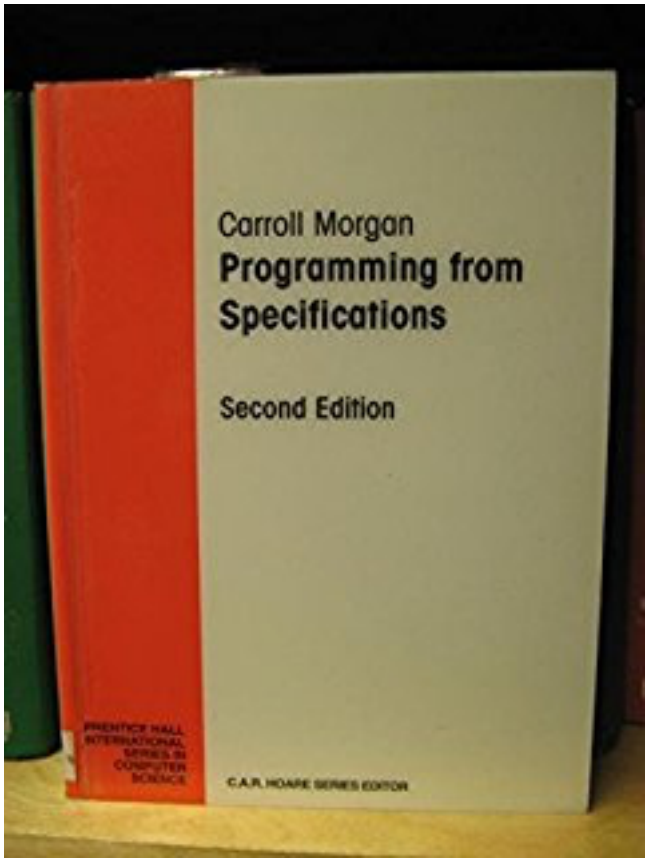
It's like someone writing a book entitled "A Discipline of Calculus" and then claiming that every engineer should use it to "properly" develop their projects, allowing the formalism to do their thinking for them.

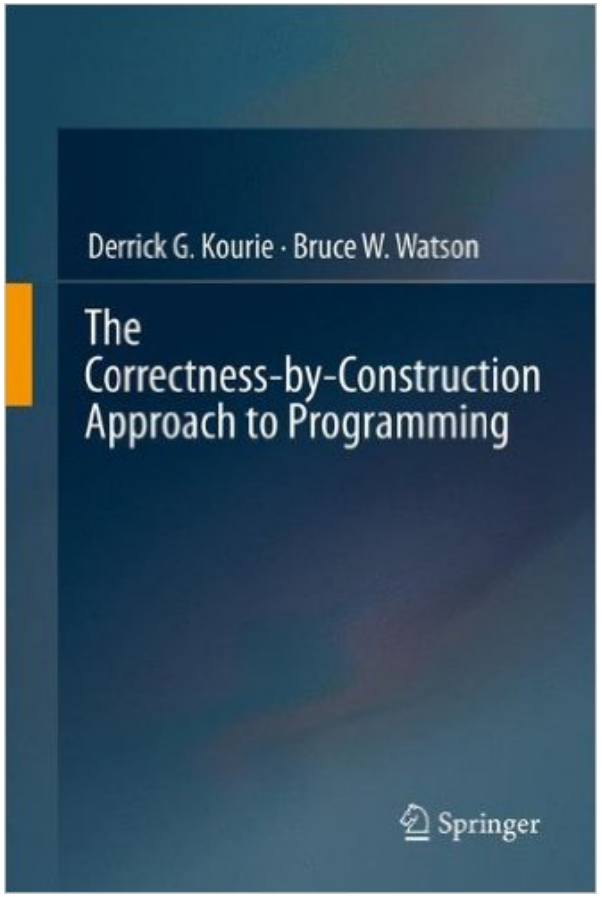
James R. Pannozzion November 12, 2011

fastar RESEARCH GROUP

CbC Round 2+







fastar
RESEARCH GROUP

What ***is*** CbC?

CbC ==

Construct a program/algorithm

from a specification

using refinement/C-preserving transforms

In our case

Imperative programs (GCL)

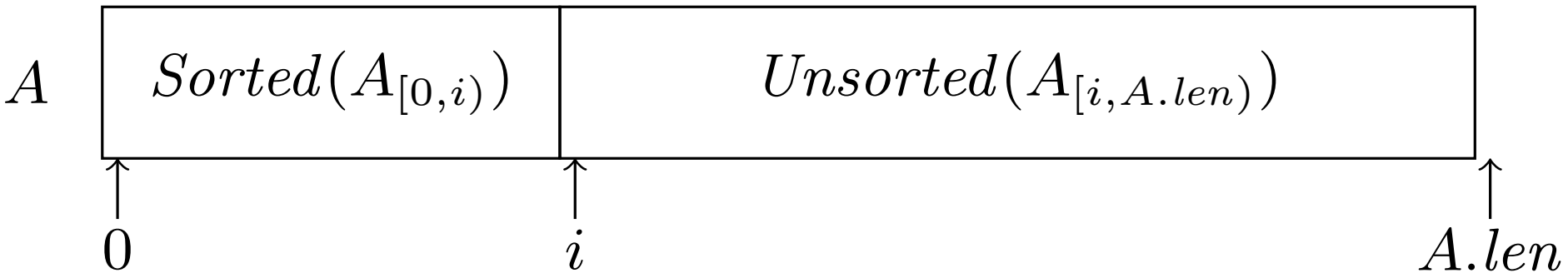
Requires FOPL

Ex: A Simple *sorting* Algorithm

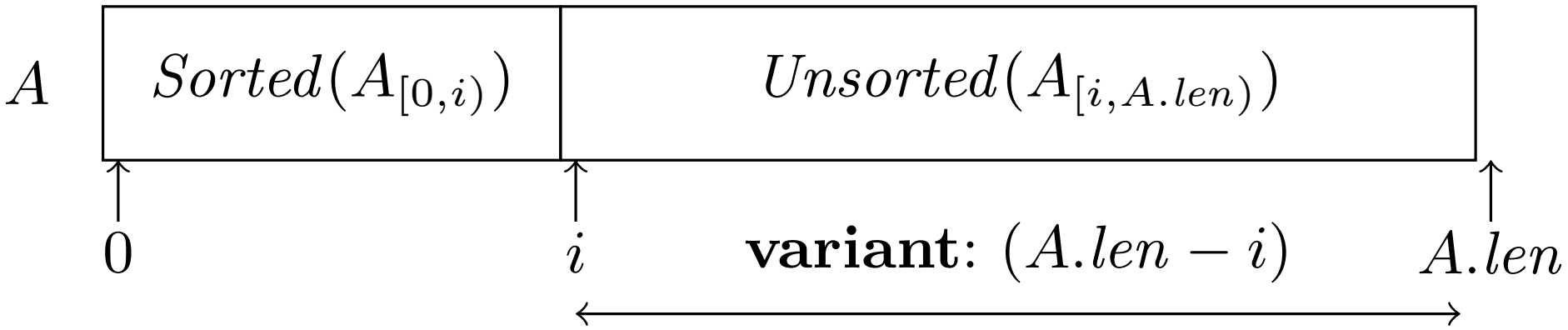
$$\{P\}\ S\ \{Q\}$$

$$\{A.len > 0\}\ S\ \{Sorted(A)\}$$

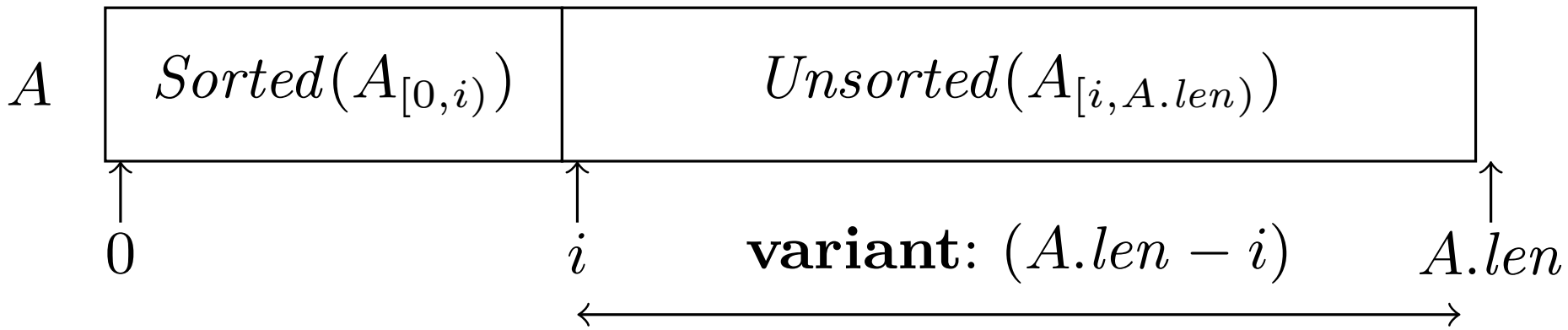
Sorting: introducing a loop



Sorting: introducing a loop



Invariant in FOPL



$$I : Sorted(A_{[0,i)}) \wedge (i \leq A.len)$$

$$I[i := A.len] \equiv Sorted(A_{[0,A.len)}) \wedge (A.len \leq A.len) \implies Sorted(A)$$

fastar
RESEARCH GROUP

First Refinements

$$\{A.len > 0\}\ S_1\ \{I\};\ S_2\ \{Sorted(A)\}$$

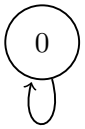
$$I[i := 0] \equiv Sorted(A_{[0,0)}) \wedge (0 \leq A.len) \equiv \mathbf{true}$$

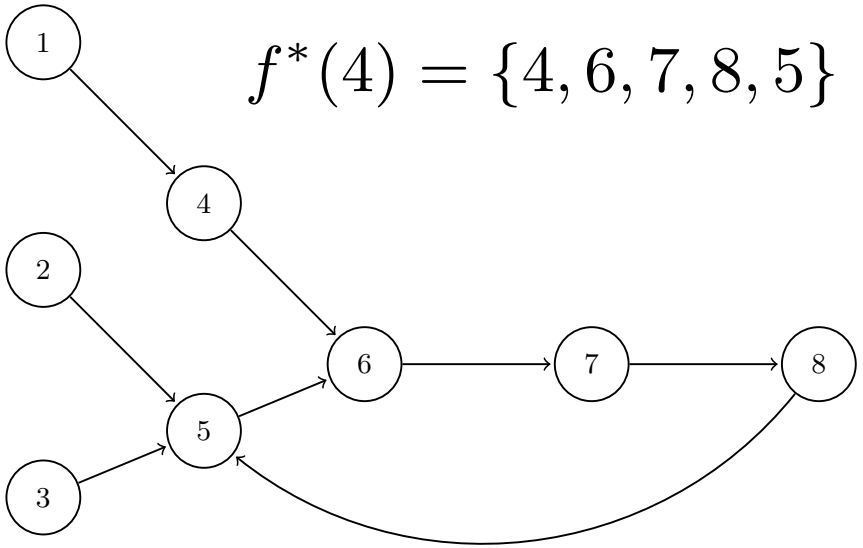
First Refinements (cont)

$\{\ A.len > 0\ \}$
$i := 0;$
$\{$ **invariant** I and **variant** $A.len - i\ \}$
do $\underbrace{\neg(i = A.len)}_{i \neq A.len} \rightarrow$
$\{\ I \wedge \underbrace{i \neq A.len}_{\text{loop guard}}\ \}$
$S_3;$
$i := i + 1$
$\{\ I \wedge$ **variant** $A.len - i$ has decreased and is non-negative $\}$
od
$\{\ \underbrace{I \wedge \underbrace{\neg\neg(i = A.len)}_{i = A.len}}_{Sorted(A)}\ \}$

fastar RESEARCH GROUP

Ex: A Simple *closure* Algorithm





$$f^*(4) = \{4, 6, 7, 8, 5\}$$

Given a finite set N, a total function

$$f : N \longrightarrow N$$

and an element $n_0 \in N$, compute the set

$$f^*(n_0) = \{f^k(n_0) : 0 \leq k\}$$

where

$$f^0(n_0) = n_0$$

and

$$f^k(n_0) = f(f^{k-1}(n_0))$$

for all $k > 0$.

fastar
RESEARCH GROUP

Closure Specification

$$\{N \text{ is finite} \wedge f : N \longrightarrow N \wedge n_0 \in N\}\ S\ \{D = f^*(n_0)\}$$

$$J : D = \{f^k(n_0) : k < i\} \wedge T = \{f^i(n_0)\}$$

fastar
RESEARCH GROUP

First Algorithm

$\{\ N \text{ is finite} \wedge f : N \longrightarrow N \wedge n_0 \in N\ \}$
$D, T, i := \emptyset, \{n_0\}, 0;$
$\{\ \textbf{invariant}\ J\ \}$
do $T \neq \emptyset \rightarrow$
$\quad \{\ J \wedge (T \neq \emptyset)\ \}\ S_0\ \{\ J\ \}$
od
$\{\ J \wedge (T = \emptyset)\ \}$
$\{\ D = f^*(n_0)\ \}$

$|N| - |D|$ $\qquad\qquad$ $|f^*(n_0)| - |D|$

fastar
RESEARCH GROUP

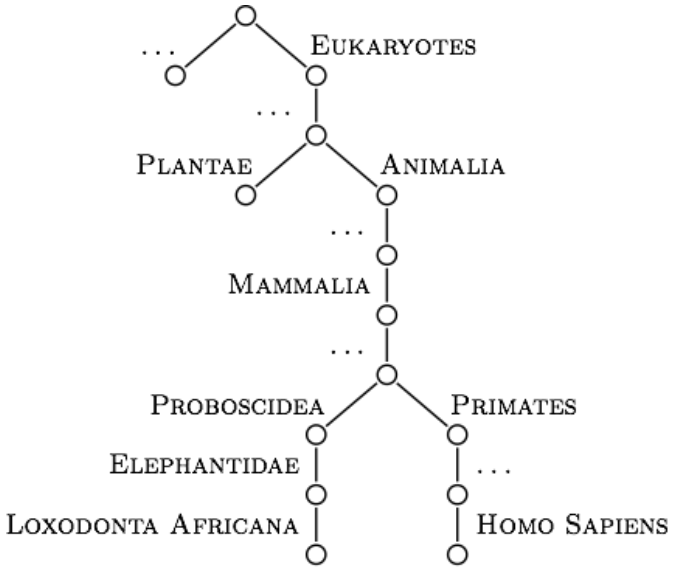
Final Algorithm

$\{\ N \text{ is finite} \wedge f : N \longrightarrow N \wedge n_0 \in N\ \}$
$D, T, i := \emptyset, \{n_0\}, 0;$
$\{$ **invariant** J and **variant** $|f^*(n_0)| - |D|\ \}$
do $T \neq \emptyset \rightarrow$
    $\{\ J \wedge (T \neq \emptyset)\ \}$
    let n such that $n \in T$;
    $D, T, i := D \cup \{n\}, T - \{n\}, i + 1;$
    $\{\ D = \{f^k(n_0) : k < i\}\ \}$
    if $f(n) \notin D \rightarrow T := T \cup \{f(n)\}$
    $[\!]$ $f(n) \in D \rightarrow$ **skip**
    fi
    $\{\ T = \{f^i(n_0)\}\ \}$
    $\{\ J \wedge$ **variant** $|f^*(n_0)| - |D|$ has decreased and is non-negative $\}$
od
$\{\ J \wedge (T = \emptyset)\ \}$
$\{\ D = f^*(n_0)\ \}$

fastar
RESEARCH GROUP

Classifications
Biological Taxonomies

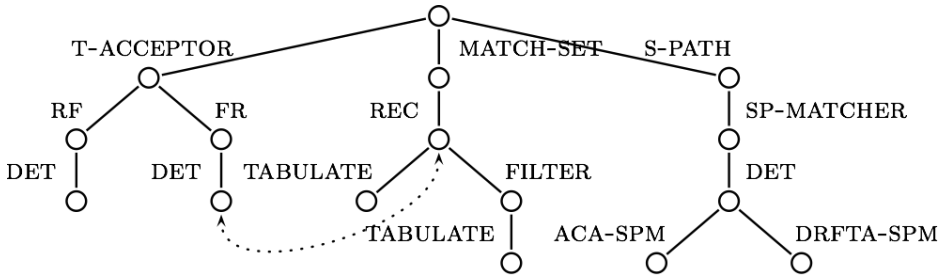
- Classify *organisms*
- From abstract, general to concrete, specific
- Properties (details) explicit
- Allow comparison



fastar
RESEARCH GROUP

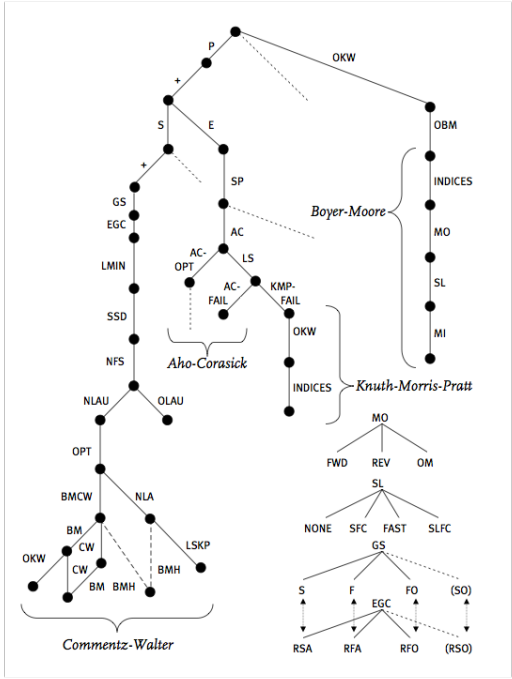
Classifications: Algorithm Taxonomies

- Similar to biological taxonomies
- Algorithm taxonomies classify *algorithms* based on *essential details*
- Depicted as tree/DAG
  Nodes refer to algorithms, branches to details
- Algorithms solving one algorithmic problem
  - From abstract, general to concrete, specific
  - Root represents high-level algorithm



Taxonomies

Presentation & Correctness—Top-down

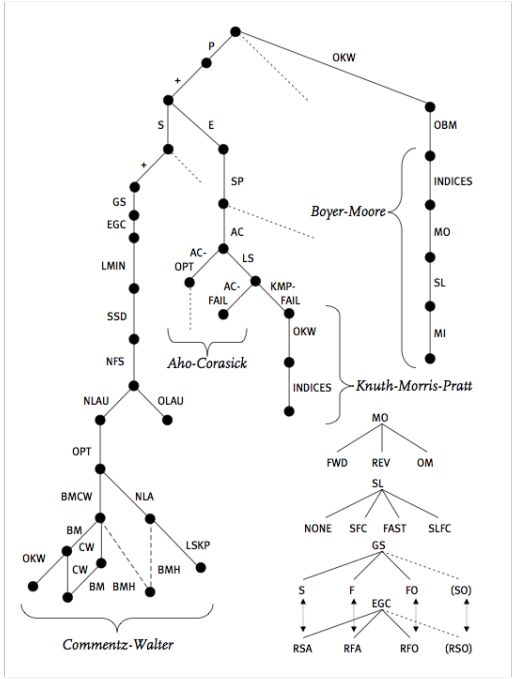


- Root represents high-level algorithm
  - With pre-/postcondition, invariants, ...
  - Correctness easily shown
- Adding *detail*
  - Obtains refinement/variation (from literature or new)
  - Branch connecting algorithm node to child node
  - Associated correctness arguments—*correctness-preserving*
- Correctness of root and of details on rootpath imply correctness of node—*correctness-by-construction approach* (Dijkstra et al., Eindhoven; Kourie & Watson, 2012)

fastar
RESEARCH GROUP

Taxonomies

Presentation & Correctness— Top-down



- Allow comparison
  - Commonalities lead to common path from root*
- Multiple paths to same solution possible
- Main goal: improve understanding of algorithms and their relations, i.e. commonalities and variabilities
- Secondary goal: highlight opportunities for new algorithms

fastar
RESEARCH GROUP

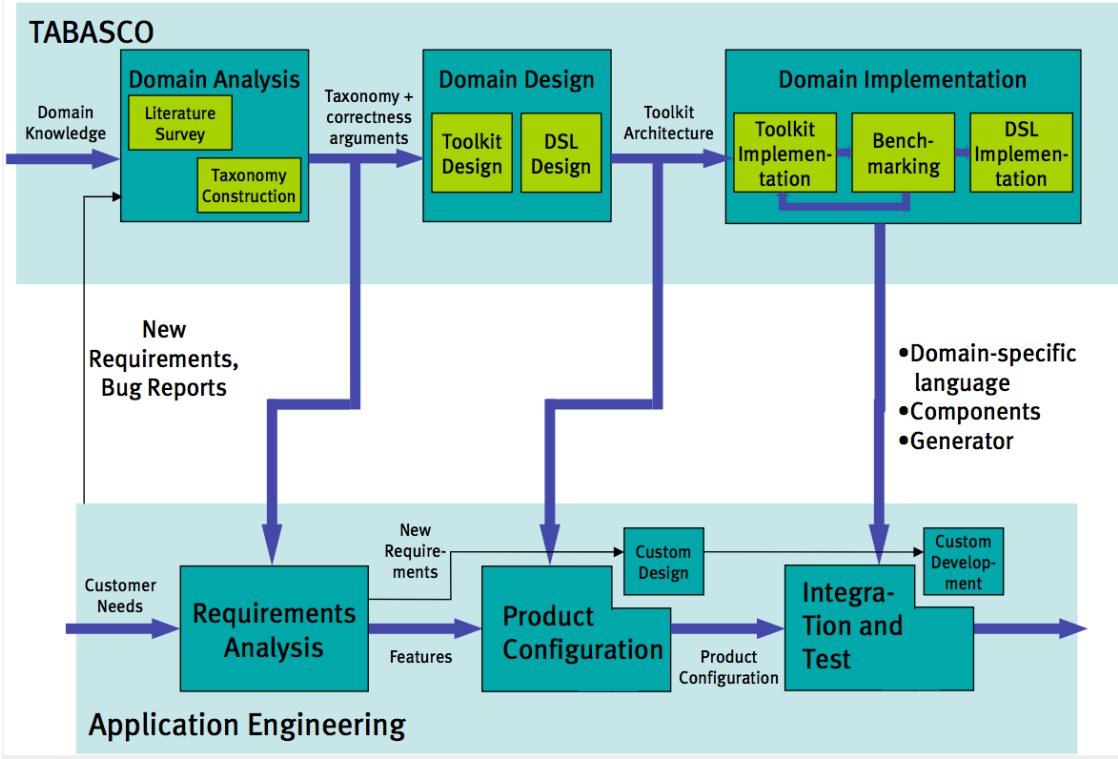
Taxonomies
Advantages and Disadvantages

+ Algorithm comparison easier
+ Clear and correct algorithm presentation
+ Leads naturally to inventive algorithmics
+ Orders field, usable as teaching aid
+ Formal specifications
+ Aids in construction of toolkit
- Takes much time and effort (*abstraction (bottom-up!), sequential addition of details*)
- Overkill for some domains?

TABASCO—Steps

Process consists of multiple steps:

1. Selection of domain
2. Literature survey
3. Classification construction
4. Toolkit design
5. Toolkit implementation
6. Benchmarking
7. DSL/GUI design
8. DSL/GUI implementation



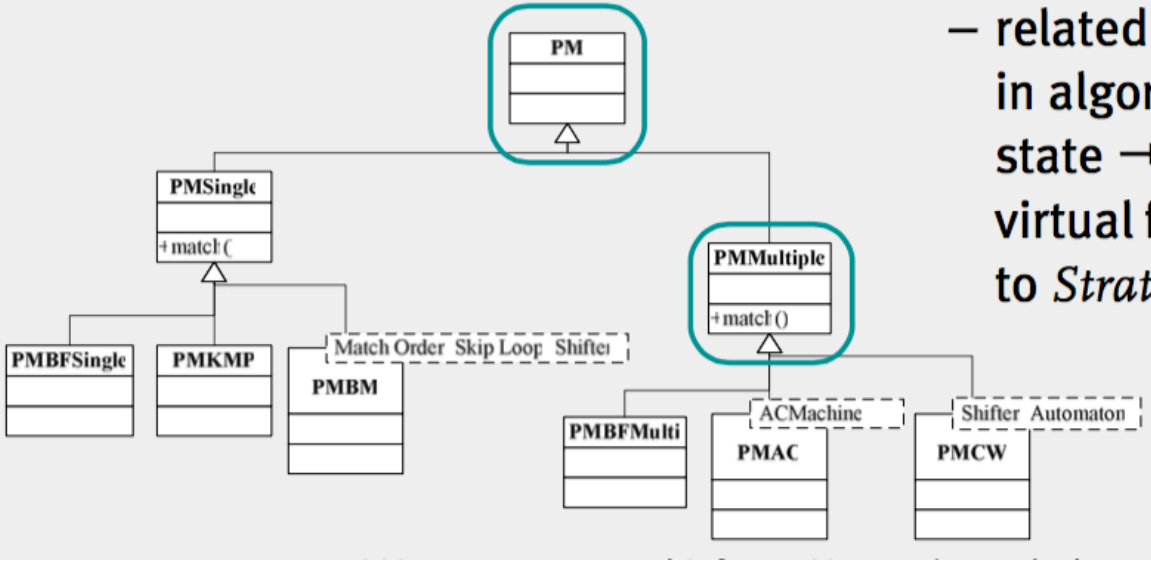
Toolkit Design

- Domain design not straightforward, requires experience and creativity
- TABASCO makes domain design more straightforward:
  - domains consist of closely related algorithms
  - taxonomy relates algorithms → makes commonalities and variabilities explicit
- Get to high-level design using *multi-paradigm design* [Coplien 1998]
  - maps commonality/variability types (*name, behavior, data structure, fine or gross algorithm, defaults, state values*) to (*C++*) language constructs (*inheritance, overloading, virtual functions, containment, structs, enumerations*) or to design patterns (*Strategy, Template Method, Singleton, Bridge, Adapter*) [Gamma et al. 1995]

The Design of SPARE Time — I

Example: toolkit based on *kpm* taxonomy

- Algorithms for similar problems, multiple and single *kpm*
- Slightly different interfaces → classes *PMSingle* and *PMMultiple* deriving from abstract class *PM*
- Approaches for multiple *kpm* (brute-force, Aho-Corasick, Commentz-Walter)

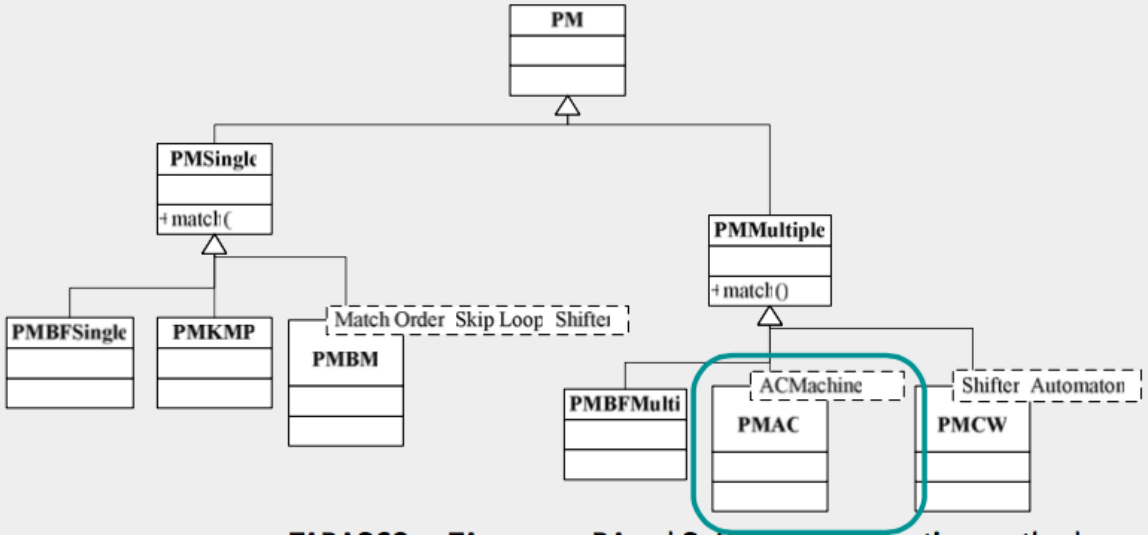


– related operations, difference in algorithm, data structure, state → inheritance with virtual functions; corresponds to *Strategy* design pattern

fastar RESEARCH GROUP

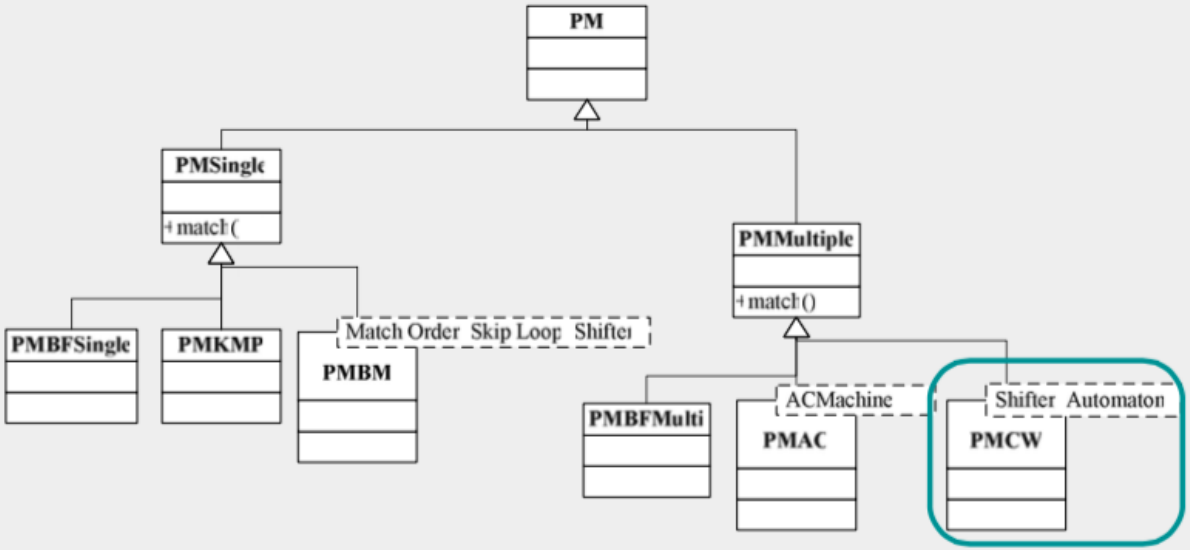
The Design of SPARE TIME — II

- Three variants of Aho-Corasick (failure function, goto function, multiple-keyword Knuth-Morris-Pratt)
  - share algorithm, differ in state update/automaton → template class with automaton parameter (*ACMachine*); similar to *Template Method* design pattern
- Implementation straightforward, given taxonomy-based toolkit design and algorithm derivations in common, abstract format



Sublinear algorithms in SPARE TIME

- As we saw, many variants of sublinear algorithms, based on different shift functions and automata
  - share algorithm, differ in automata and shift function → template class with shift function and automata parameters (*ACMachine*); similar to *Template Method* or *Strategy* design pattern



fastar RESEARCH GROUP

Advantages of Taxonomy-based Toolkit Design

- Taxonomy availability simplifies design task
  - leads to homogeneity of design, interface → easier to understand
  - leads to easier to understand/debug code
- Uniformity of style gives confidence in accuracy of implementations' relative performance
- Taxonomy and toolkit can serve as example of toolkit design & implementation techniques
- Correctness arguments in taxonomy give confidence in toolkit correctness and safety
- Taxonomy gives formal specification of requirements satisfied; helps in understanding components, creating mapping from user requirements to components

Conclusions

- CbC always constructs correct algorithms
- Correctness proof is integrated in derivation
- CbC *lite* should be widely used
- Multi-algorithm CbC == taxonomy
- Taxonomy-gap exploration == new algorithms
- CbC should be taught more widely.

Future Work

- CbC approaches for programming models and languages other than sequential-imperative programs, e.g., parallelism, cloud-based programs or DSLs, such as Matlab/Simulink, GP, etc.
- CbC tools in the form of structured editors that directly support the CbC style of code derivation

References

- D.G. Kourie & B.W. Watson
  The Correctness-by-Construction Approach to Programming
  Springer, 2012.
- B.W. Watson, D.G. Kourie & L. Cleophas
  Experience with Correctness-by-Construction.
  Science of Computer Programming, special issue on New Ideas and Emerging Results in Understanding Software, 2013.
- L. Cleophas & B.W. Watson
  Taxonomy-based software construction of SPARE Time: a case study.
  In IEE Proceedings – Software, 152(1), February 2005.
- L. Cleophas, B.W. Watson, D.G. Kourie, A. Boake & S. Obiedkov
  TABASCO: Using Concept-Based Taxonomies in Domain Engineering.
  SACJ, 37:30–40, December 2006.

fastar
RESEARCH GROUP

Case Study: Generalised Stringology

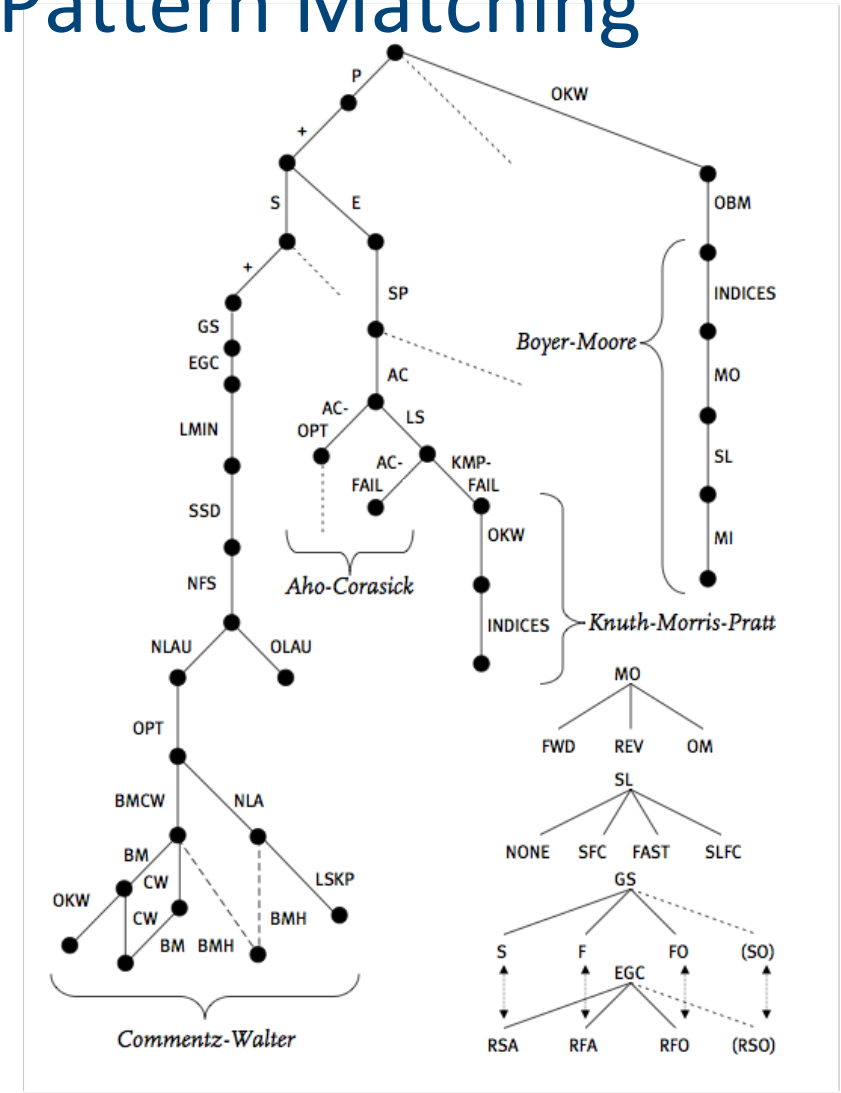
- *Regular Grammar* and *Regular Expression*
  - Different types, transformations between them
- Problems
  - *Membership/Acceptance*
  - *Keyword Pattern Matching* (*KPM*)
- *Finite Automaton*
  - Nondeterministic with/without *epsilon*-transitions, deterministic
- Theoretical Results (1950s)
  - Equivalence of *NFA* and *DFA* (subset construction)
  - Equivalence of *RG*, *RE*, and *FA*
  - Solve by constructing and using *FA* based on *RG*/*RE*

Case Study: Generalised Stringology (cont.)

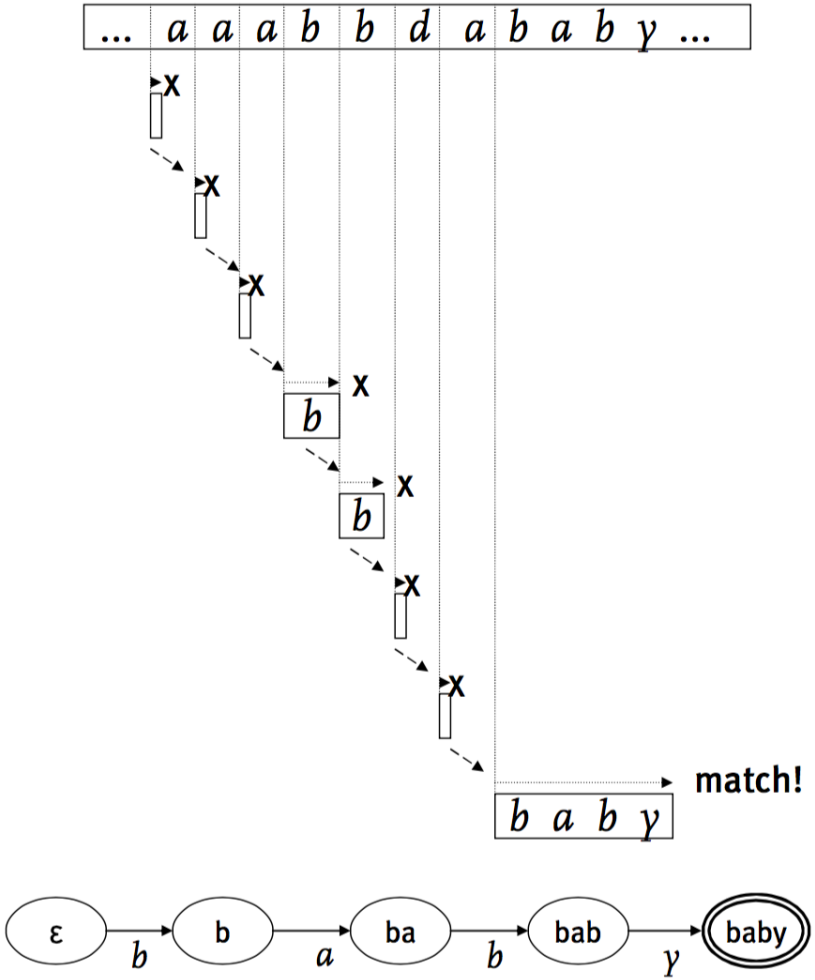
- In practice (1960s - now):
  - Many applications
    - Natural language text search
    - DNA processing
    - Network intrusion and virus detection
  - Many *FA* constructions, acceptance/*KPM* algorithms—$O(10^2)$
    - More efficient; for specific situations
  - Difficult to find, understand, compare
  - Separation between theory and practice
  - Hard to compare and choose implementations

Taxonomies
Example: Keyword Pattern Matching

- Detail choice and order depend on personal preference & domain understanding
- Inclusion of different orders for single algorithm leads to directed acyclic graph
- Initial version by Watson & Zwaan (1992-1996)
- Revised & extended
  - Cleophas (2003)
  - Cleophas, Watson & Zwaan (2004; 2010)



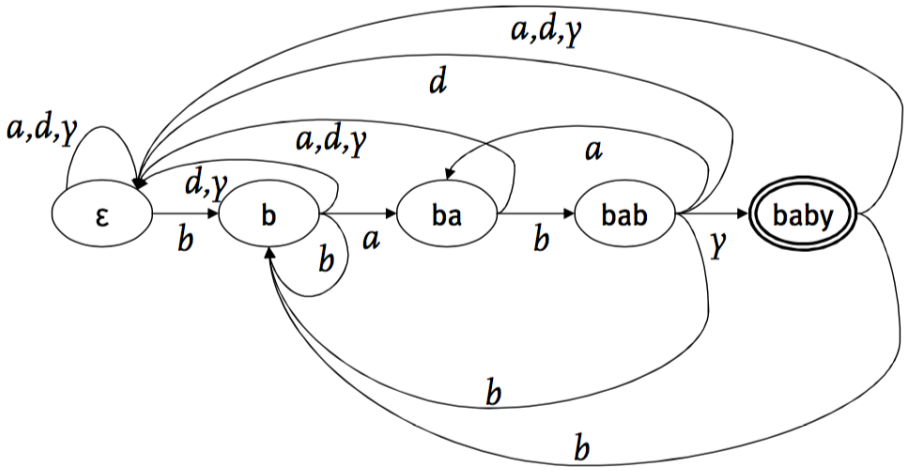
Example – Naive



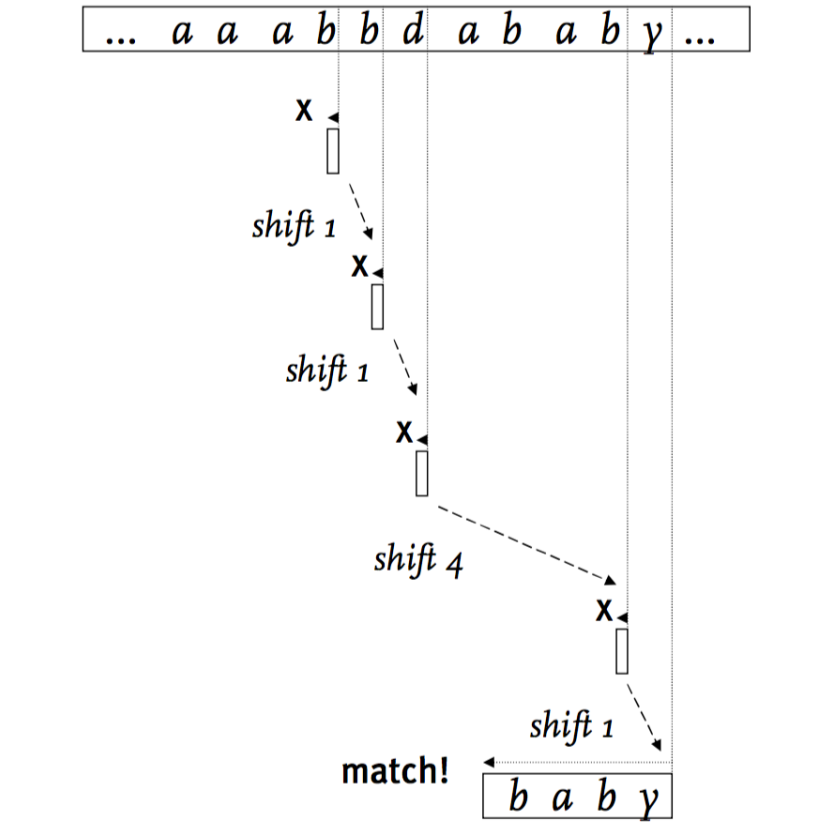
Example – Prefix-based

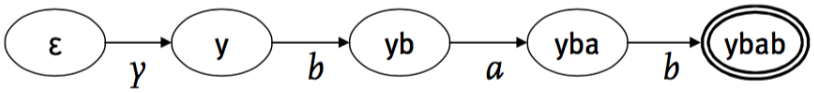
…	a	a	a	b	b	d	a	b	a	b	y	…

read character and take automaton transition

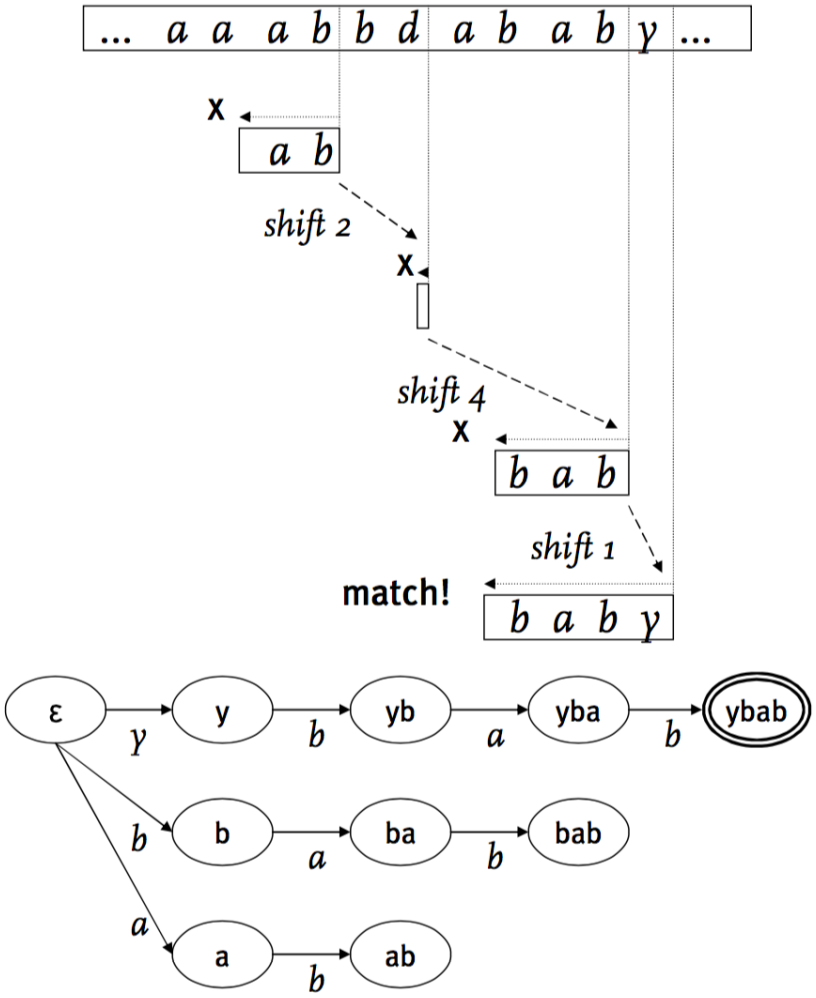


Example – Suffix-based

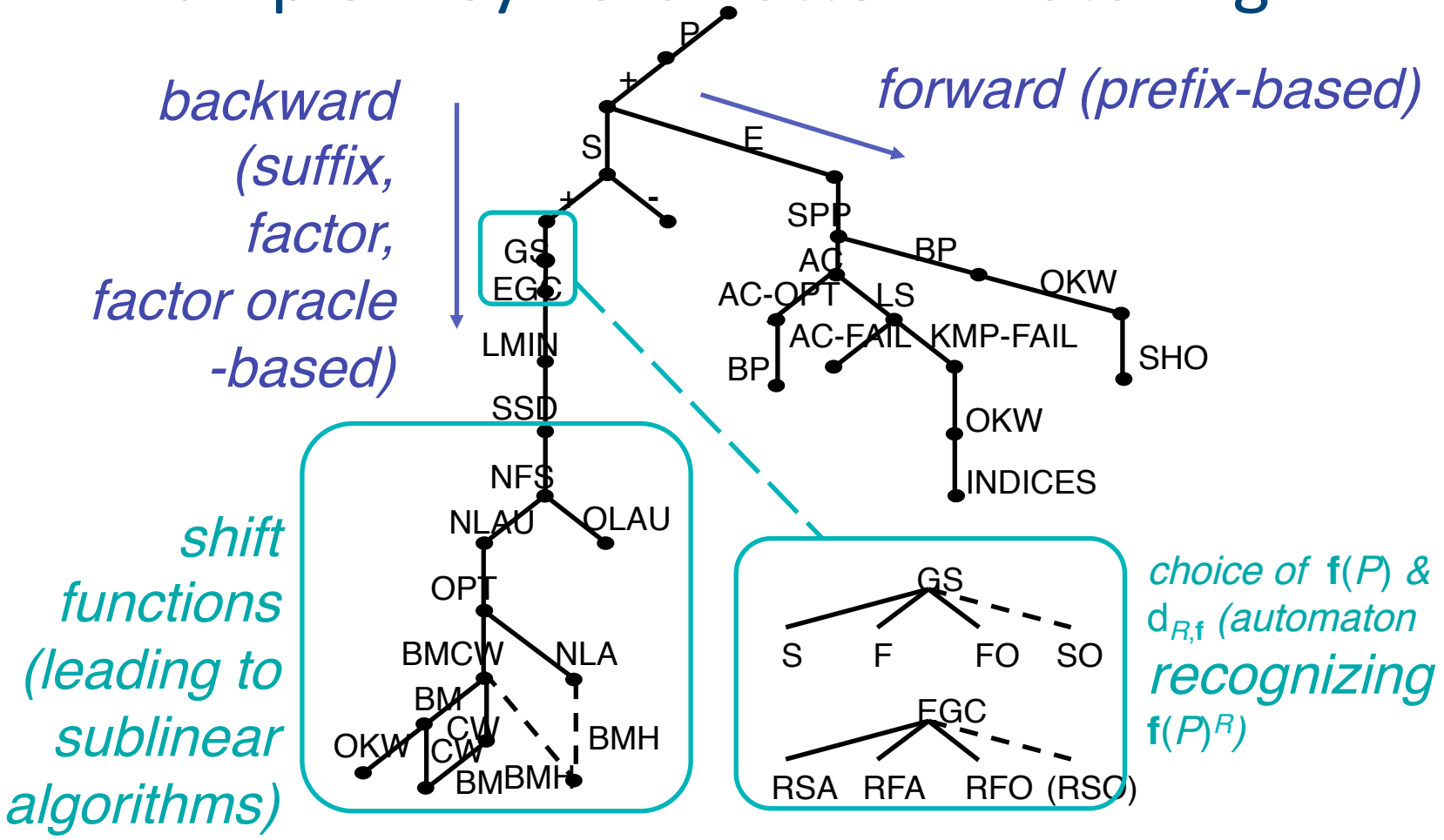




Example – Factor-based



Taxonomies
Example: Keyword Pattern Matching

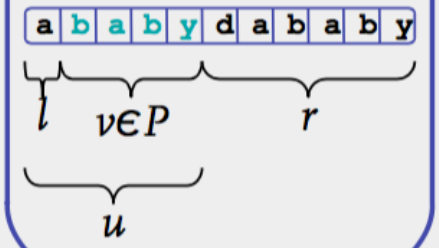


A particular taxonomy path — I

Example:

$O =$
$\{$
$(a, baby, dababy),$
$(ababyda, baby, \epsilon)$
$\}$

(u,r) pairs nondeterministically chosen



Formally, the KPM problem is to establish

$$O = (\cup\ l, v, r : lvr = S \wedge v \in P : \{(l, v, r)\})$$

for text S and keyword set P

$O := (\cup\ l, v, r : lvr = S \wedge v \in P : \{(l, v, r)\})$ **Algorithm ()**

$\{\ R : O = (\cup\ l, v, r : lvr = S \wedge v \in P : \{(l, v, r)\})\ \}$

Can be rewritten to

$$O = (\cup u, r : ur = S : (\cup\ l, v : lv = u \wedge v \in P : \{(l, v, r)\}))$$

$O := \varnothing;$

for $(u, r) : ur = S \rightarrow$

$O := O\ (\ (\cup l, v : lv = u \wedge v \in P : \{(l, v, r)\})$

rof

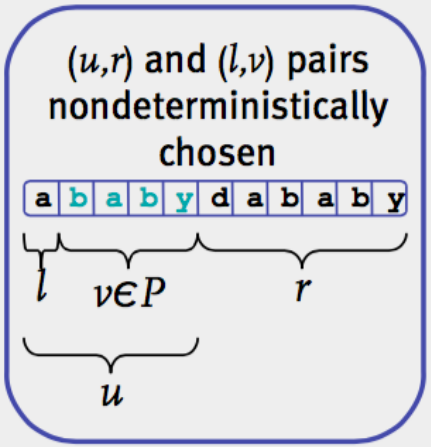
$\{\ R\ \}$

Algorithm (P)

fastar RESEARCH GROUP

A particular taxonomy path — II

Can also consider suffixes of such prefixes in some order



Algorithm (P, S)

$O := \emptyset$;

for $(u, r) : ur = S \rightarrow$

  for $(l, v) : lv = u \rightarrow$

    as $v \in P \rightarrow O := O \ (\ \{(l, v, r)\}$ **sa**

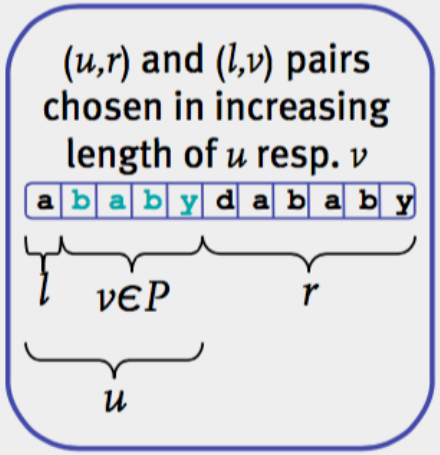
  rof

rof

$\{ R \}$

A particular taxonomy path — III

Can consider prefixes in increasing length order, and suffixes of such prefixes in increasing length order

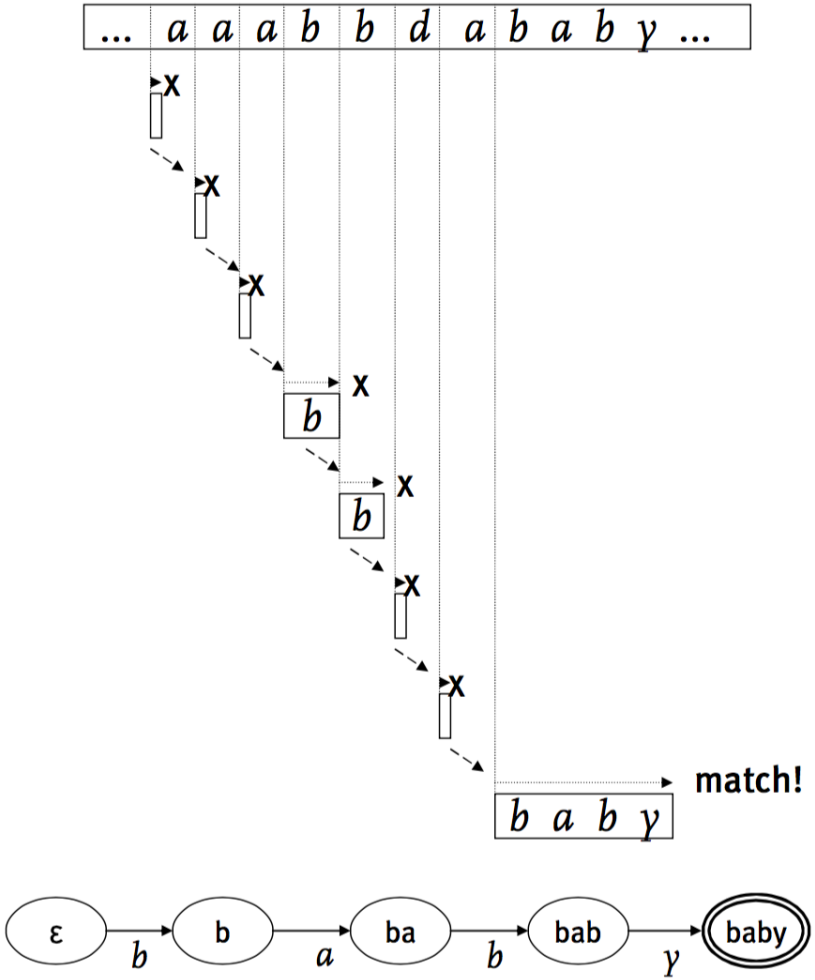


Algorithm (P+, S+)

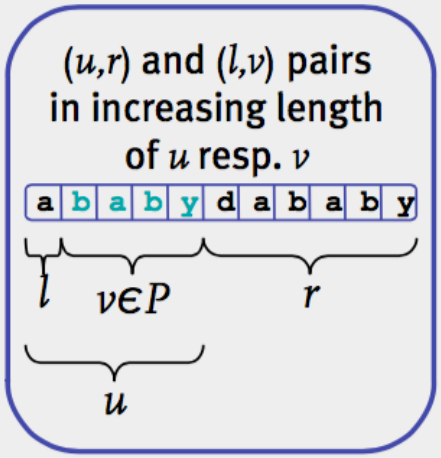
$u,r := \epsilon, S;$
$O := \emptyset;$

$l,v := \epsilon, \epsilon;$
do $r \neq \epsilon \rightarrow$
  $u,r := u(r\uparrow 1), r\downarrow 1;$
  $l,v := u, \epsilon;$
  do $l \neq \epsilon \rightarrow$
    $l,v := l\downarrow 1, (l\uparrow 1)v;$
    as $v \in P \rightarrow O := O \,(\, \{(l, v, r)\}$ **sa**
  od
od
$\{ R \}$

Example – Naive



A particular taxonomy path — V



f(P): $P \subseteq f(P) \wedge$

$suff(f(P)) \subseteq f(P)$

$d_{R,f}$: *automaton recognizing* $f(P)^R$

```
u,r := ϵ, S;
O := ∅;
l,v := ϵ, ϵ;
do r ≠ ϵ →
        u,r := u(r↑1), r↓1;
        l,v := u, ϵ;
        q := d_{R,f}(q_o, l↑I);
        { invariant: q = d_{R,f}(q_o, (l↑I)v)^R }
        do l ≠ ϵ cand q ≠ ⊥ →
                l,v := l↓I, (l↑I)v;
                q := d_{R,f}(q_o, l↑I);
        od      as v ∈ P → O := O ( {(l, v, r)} sa
od
{ R }
```

Algorithm
(P+, S+,
GS, EGC)

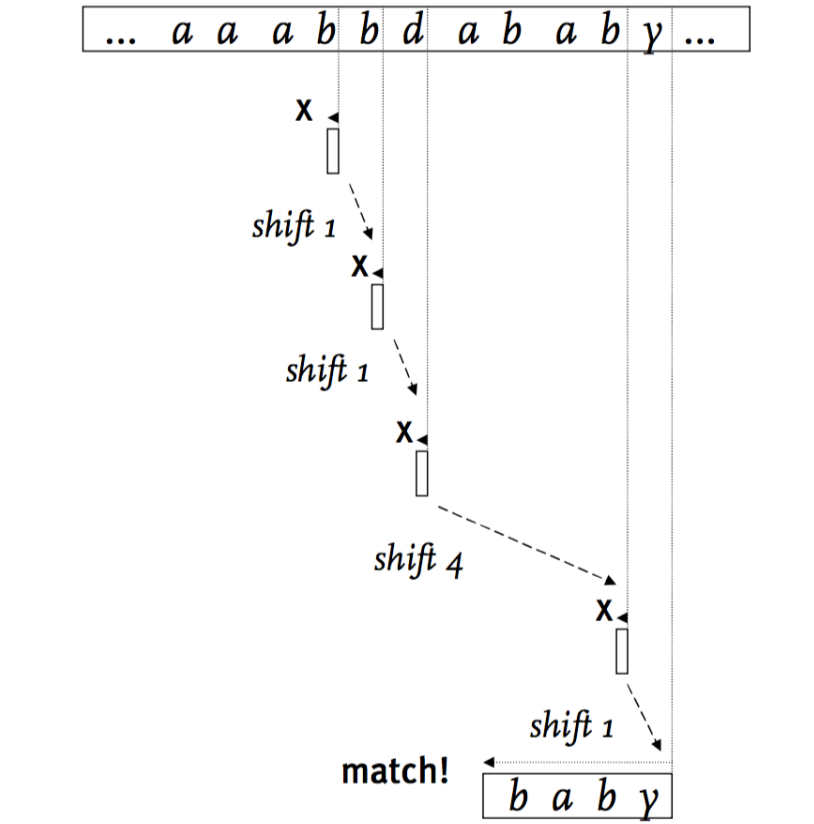
A particular taxonomy path — VI

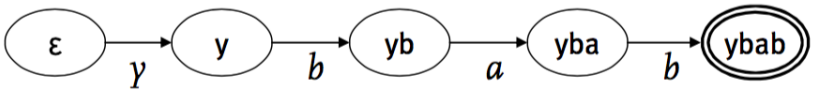
$k(l,v,r)$:
shift function
`at least 1 and
at most distance to
next match'

Algorithm
(P+, S+,
GS, EGC,
SSD)

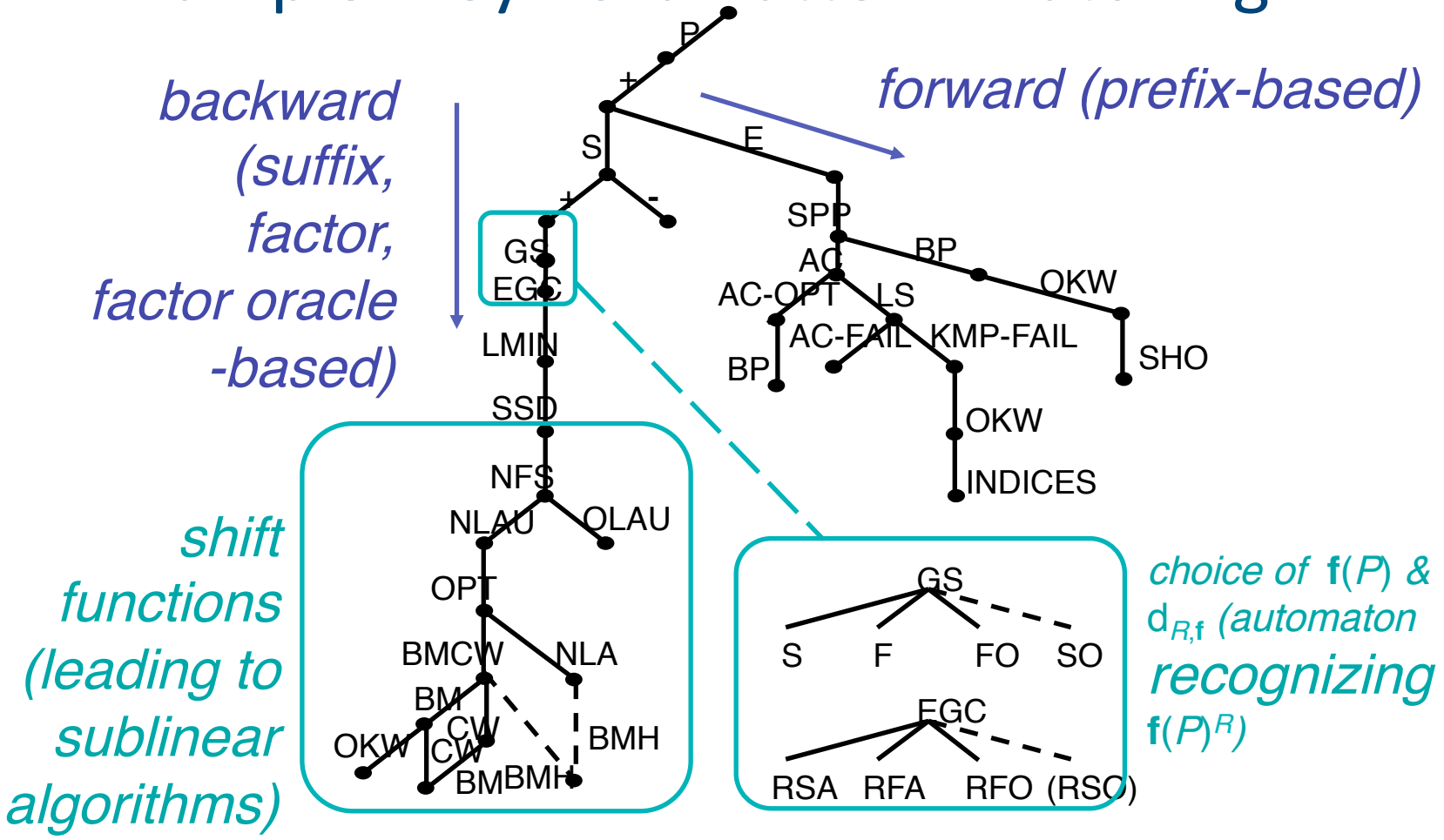
$u,r := \epsilon, S;$
$O := \emptyset;$
$l,v := \epsilon, \epsilon;$
do $r \neq \epsilon \rightarrow$
  $u,r := u(r\uparrow k(l,v,r)), r\downarrow k(l,v,r);$
  $l,v := u, \epsilon;$
  $q := \mathrm{d}_{R,\mathrm{f}}(q_o, l\restriction \mathrm{I});$
  { **invariant:** $q = \mathrm{d}_{R,\mathrm{f}}(q_o, (l\restriction \mathrm{I})v)^R$ }
  do $l \neq \epsilon$ **cand** $q \neq \perp \rightarrow$
    $l,v := l\downarrow \mathrm{I}, (l\restriction \mathrm{I})v;$
    $q := \mathrm{d}_{R,\mathrm{f}}(q_o, l\restriction \mathrm{I});$
    as $v \in P \rightarrow O := O$ ($\{(l, v, r)\}$ **sa**
  od
od
{ R }

Example – Suffix-based





Taxonomies
Example: Keyword Pattern Matching



Boyer-Moore algorithms

Matching “abracadabra” in “The quick brown fox...”

```
Attempting a match at 0
The quick brown fox jumped over the lazy dog
abracadabra
Match got as far as i = 0. Will now shift right by 2

Attempting a match at 2
The quick brown fox jumped over the lazy dog
  abracadabra
Match got as far as i = 0. Will now shift right by 11

Attempting a match at 13
The quick brown fox jumped over the lazy dog
             abracadabra
Match got as far as i = 0. Will now shift right by 11

Attempting a match at 24
The quick brown fox jumped over the lazy dog
                        abracadabra
Match got as far as i = 0. Will now shift right by 11
```

fastar RESEARCH GROUP

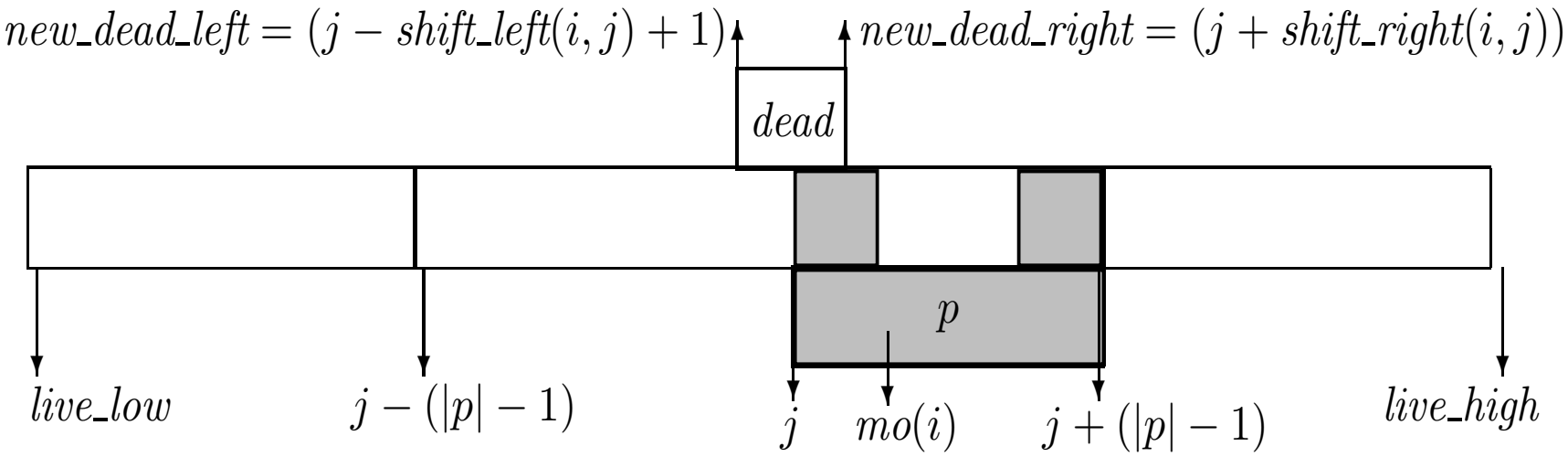
Single-keyword dead-zone

```
Invoked with a live-zone of [0,34). Attempting a match at 17
The quick brown fox jumped over the lazy dog
                  abracadabra
Match got as far as i = 0. Will now shift left/right by 11/11
New dead-zone is [7,28).
Left will be [0,7) and right will be [28,34)

Invoked with a live-zone of [0,7). Attempting a match at 3
The quick brown fox jumped over the lazy dog
   abracadabra
Match got as far as i = 0. Will now shift left/right by 11/11
New dead-zone is [-7,14).
Left will be [0,-7) and right will be [14,7)

Invoked with a live-zone of [28,34). Attempting a match at 31
The quick brown fox jumped over the lazy dog
                                  abracadabra
Match got as far as i = 0. Will now shift left/right by 11/4
```

A match attempt-and-shift



proc $dzmat(live\_low, live\_high) \rightarrow$

if $(live\_low \geq live\_high) \rightarrow$ **skip**

$[\!]$ $(live\_low < live\_high) \rightarrow$

$j := \lfloor (live\_low + live\_high)/2 \rfloor;$

$i := 0;$

do $((i < |p|)$ **cand** $(p_i = S_{j+i})) \rightarrow$

$i := i + 1$

od;

if $i = |p| \rightarrow print(\text{`Match at '}, j)$

$[\!]$ $i < |p| \rightarrow$ **skip**

fi;

$new\_dead\_left := j - shift\_left(i, j) + 1;$

$new\_dead\_right := j + shift\_right(i, j);$

$dzmat(live\_low, new\_dead\_left\ );$

$dzmat(new\_dead\_right + 1, live\_high)$

fi

corp

Dead-Zone example (best case)

```
Invoked with a live-zone of [0,27). Attempting a match at 13
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              01234
Match got as far as i = 0. Will now shift left/right by 5/5
New dead-zone is [9,18).
Left will be [0,9) and right will be [18,27)

Invoked with a live-zone of [0,9). Attempting a match at 4
aaaaaaaaa/////////aaaaaaaaaaaaa
    01234
Match got as far as i = 0. Will now shift left/right by 5/5
New dead-zone is [0,9).
Left will be [0,0) and right will be [9,9)

Invoked with a live-zone of [18,27). Attempting a match at 22
/////////////////aaaaaaaaaaaaa
                      01234
Match got as far as i = 0. Will now shift left/right by 5/5
New dead-zone is [18,27).
Left will be [18,18) and right will be [27,27)
```

fastar
RESEARCH GROUP