Developments in Language Theory
9th International Conference, July 4–8, 2005, Palermo, Italy

LR parsing for Boolean grammars

Alexander Okhotin

Department of Mathematics, University of Turku
Turku, Finland

5 July, 2005.

Parsing beyond context-free grammars

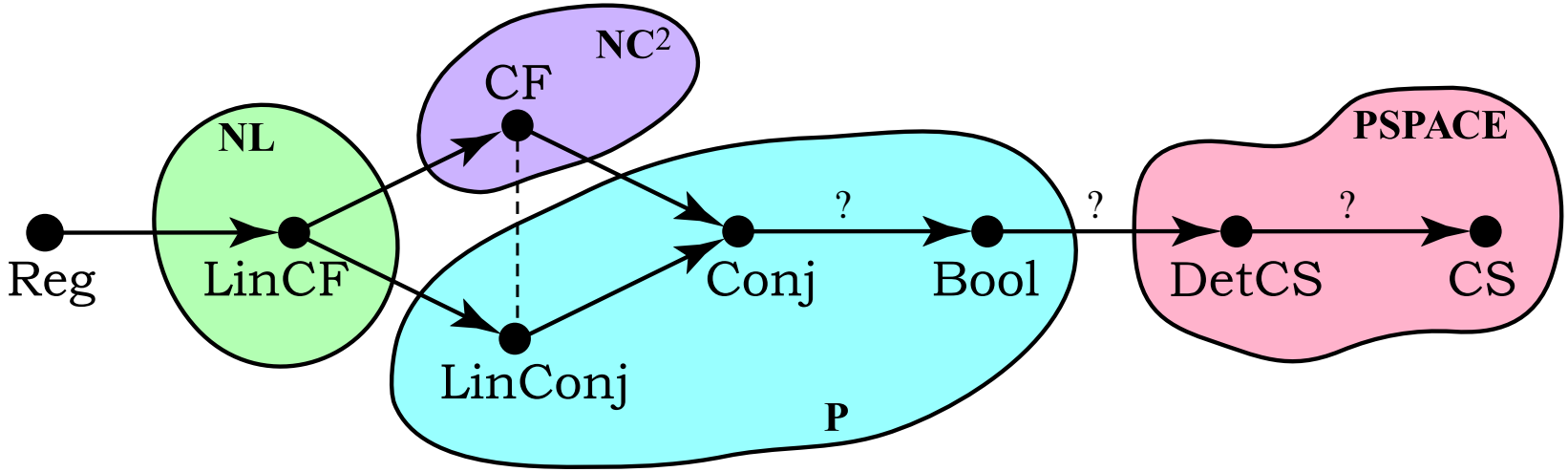
- Mathematical approach:
  1. Define a mathematical model more general than a CFG.
  2. Develop an algorithm to test the membership.
  3. Prove that the algorithm recognizes the right language.
  4. Prove its domain of applicability and complexity.
- Engineering approach:
  1. Take a context-free parsing algorithm,
     augment it with additional control structures.
  2. A CFG with directives to control them defines a parser.
  3. The computation of the parser defines a language.
  4. Applicability and complexity: programmer's concern.

Boolean grammars (Okhotin, DLT 2003)

- Context-free grammars with Boolean operations. The rules are

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n$$

- Further extension of *conjunctive grammars* (Okhotin, 2000).
- Can generate $\{a^n b^n c^n\}$, $\{wcw \mid w \in \{a,b\}^*\}$, $\{ww\}$, $\{a^{2^n}\}$, a **P**-complete language, a simple programming language.
- The generated languages are in $\mathrm{DTIME}(n^3) \cap \mathrm{DSPACE}(n)$.



Formal semantics: language equations

- A grammar as a system in variables $N = \{A_1, \ldots, A_n\}$.

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \cap \bigcap_{j=1}^{n} \overline{\beta_j} \right] \quad \text{(for all } A \in N\text{)}$$

- Unique solutions of such systems: any recursive sets (Okhotin, ICALP 2003).
- Strongly unique solution: unique modulo any finite $M \subseteq \Sigma^*$.
- Naturally reachable solution: models intuitive semantics.
  - A derivation on vectors of languages modulo a finite M.
  - $\{S \to \neg A,\ A \to A\}$: $L(A) = \varnothing$, $L(S) = \Sigma^*$.
  - If negation is not used, equals the least solution.
- Strongly unique and naturally reachable solutions: equal power.

Semantics of naturally reachable solution.

A solution $L = (L_1, \ldots, L_n)$ of $X = \varphi(X)$ is called a n.r.s. if for every M and for every string $u \notin M$ (such that all proper substrings of u are in M), every sequence of vectors of the form

$$L^{(0)}, L^{(1)}, \ldots, L^{(i)}, \ldots \quad \text{where}$$

1. $L^{(0)} = (L_1 \cap M, \ldots, L_n \cap M)$,

2. every next vector $L^{(i+1)} \neq L^{(i)}$ is obtained from the previous vector $L^{(i)}$ by substituting some j-th component with $\varphi_j(L^{(i)}) \cap (M \cup \{u\})$

converges to $(L_1 \cap (M \cup \{u\}), \ldots, L_n \cap (M \cup \{u\}))$ in finitely many steps regardless of the choice of components.

Is uniquely defined (if exists). Can be computed modulo every finite language just by following the definition. Checking the validity is co-RE-complete.

Parsing for Boolean grammars

- A $C \cdot n^3$ generalization of Cocke–Kasami–Younger algorithm.
- Recursive descent parsing for a subfamily of LL(k) Boolean gr.
  - A prodedure $s()$ for every $s \in \Sigma \cup N$.
  - A parsing table $T : N \times \Sigma^{\leqslant k} \to P$.
  - Conjunction implemented by scanning input multiple times.
  - Negation implemented via exception handling.
  - LL(k) Boolean grammars exceed Boolean closure of CFLs.

✓ The polar opposite approach to parsing: LR(k).

LR parsing (Knuth, 1965)

- Applicable to every deterministic CFL.
- Two operations on stack: Shift and Reduce.

$$\alpha\beta \xrightarrow{\text{Shift a}} \alpha\beta a \xrightarrow{\text{Reduce } B\to Ba} \alpha B$$

- Controlled by a finite automaton that processes the stack.

$$q_0 \overbrace{s_1 q_1 s_2 \ldots q_{m-1} s_m}^{\alpha} q_m \overbrace{\ldots}^{\beta} q'$$

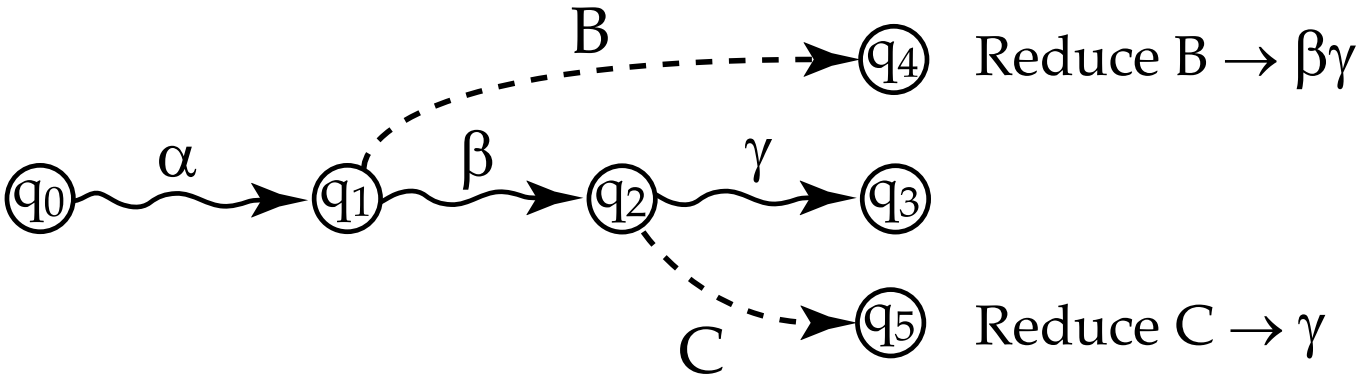
$$q_0 \overbrace{s_1 q_1 s_2 \ldots q_{m-1} s_m}^{\alpha} q_m \overbrace{\ldots}^{\beta} q'\, a\, q'' \qquad \text{Action}[q', a] = \text{Shift a}$$

$$q_0 \overbrace{s_1 q_1 s_2 \ldots q_{m-1} s_m}^{\alpha} q_m\, B\, q''' \qquad \text{Action}[q'', b] = \text{Reduce } B\to Ba$$

- The choice is nondeterministic for unsuitable grammars.

Nondeterministic LR parsing (Tomita, 1986)

- When two actions are possible, do both.
- Graph-structured stack represents the contents of linear stack for all branches of computation.
- $|Q| \cdot |w|$ nodes (each state appears at most once per symbol).
- Example of doing two reductions:

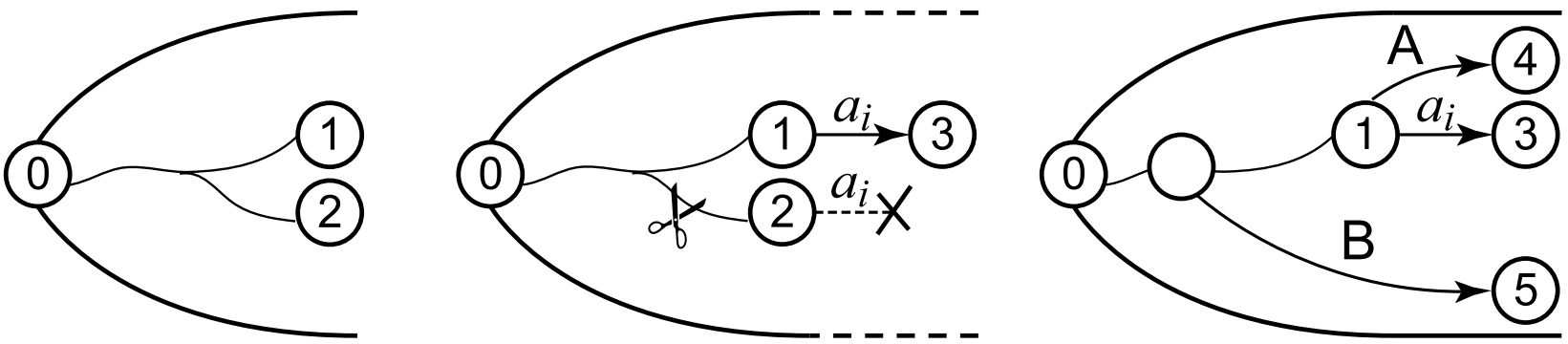


- An extension for conjunctive grammars (Okhotin, 2002).

✓ A much harder generalization for Boolean grammars.

The algorithm

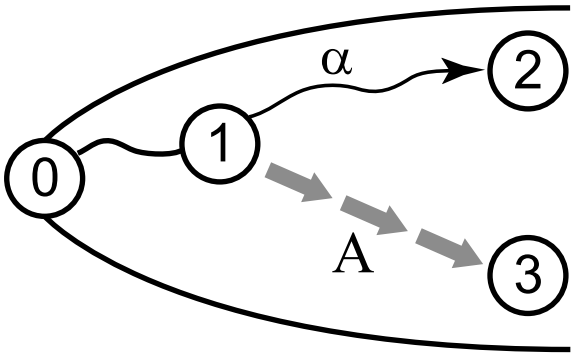
Alternation of *shift* and *reduction* phases.

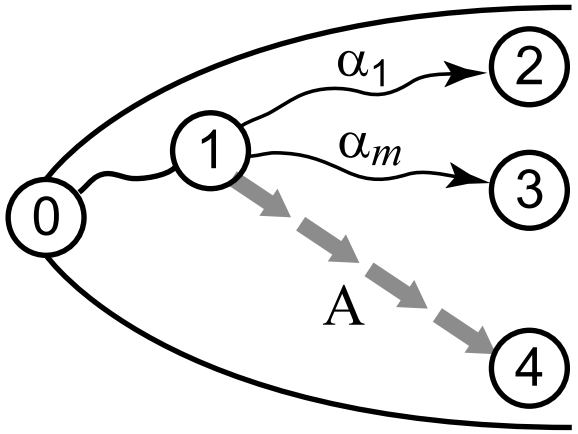


- *Shift phase*: read a new input symbol and apply it to all top vertices.
- *Reduction phase*: set nonterminal arcs according to the grammar rules.

The reduction phase

- Context-free case: one operation, *reduction*.
  A rule $A \to \alpha$, a path $\alpha \Rightarrow$ add an arc A.
- Conjunctive case: *reduction* with multiple premises.
  A rule $A \to \alpha_1 \& \ldots \& \alpha_m$, paths $\alpha_1, \ldots, \alpha_m \Rightarrow$ add an arc A.



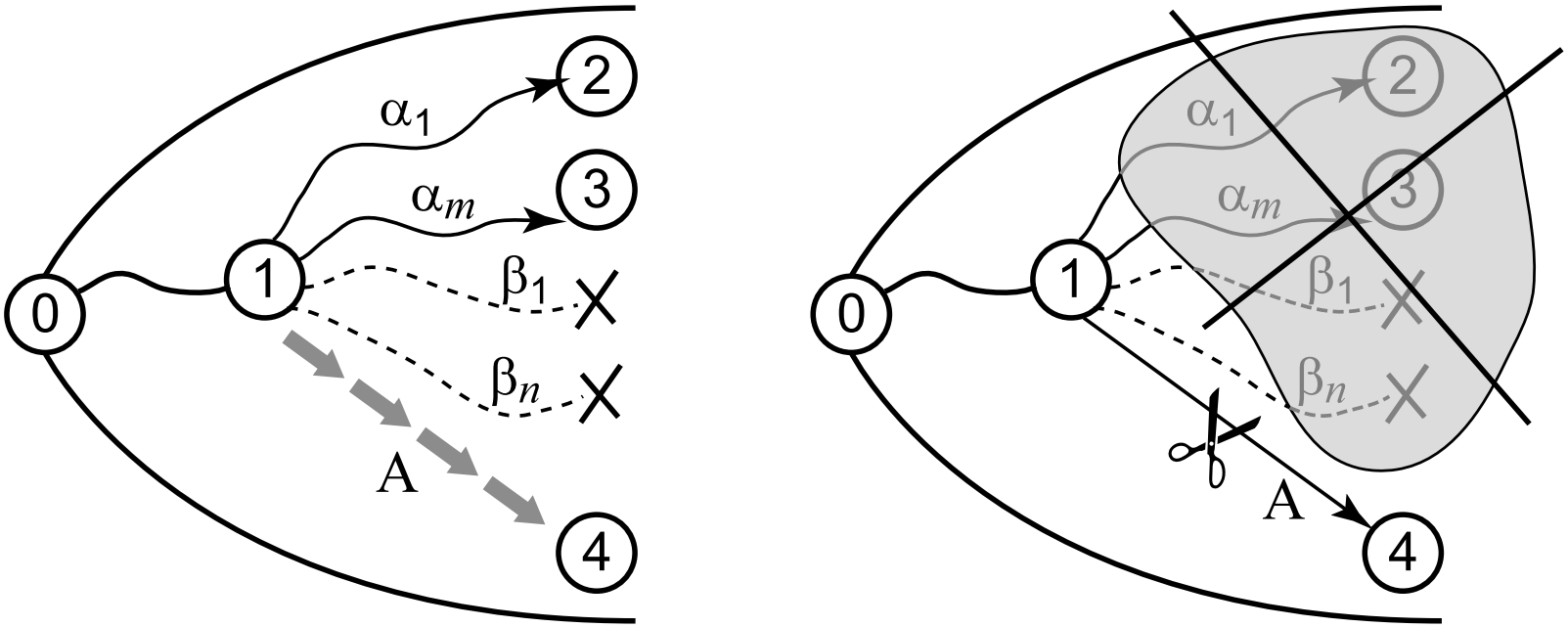



```
while the graph can be modified
        do a reduction
end while
```

The reduction phase for Boolean grammars

1. *Reduction*, with positive and negative premises.
   A rule $A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n$, each path α_i exists, each path β_j does not exist $\Rightarrow$ add an arc A.

2. *Invalidation*: if there is an arc A, but no rule that warrants its addition, then remove it.



Example

The following grammar generates $(aa)^*$ in a very peculiar way:

$$S \to X\&\neg aS$$
$$X \to Xa \mid \varepsilon$$

	δ			R	
	a	S	X	ε	a
0	1	5	2	$X \to \varepsilon$	$X \to \varepsilon$
1	1	3	2	$X \to \varepsilon$	$X \to \varepsilon$
2	4			$S \to X$	
3				$S \to \neg aS$	
4				$X \to Xa$	$X \to Xa$
5					

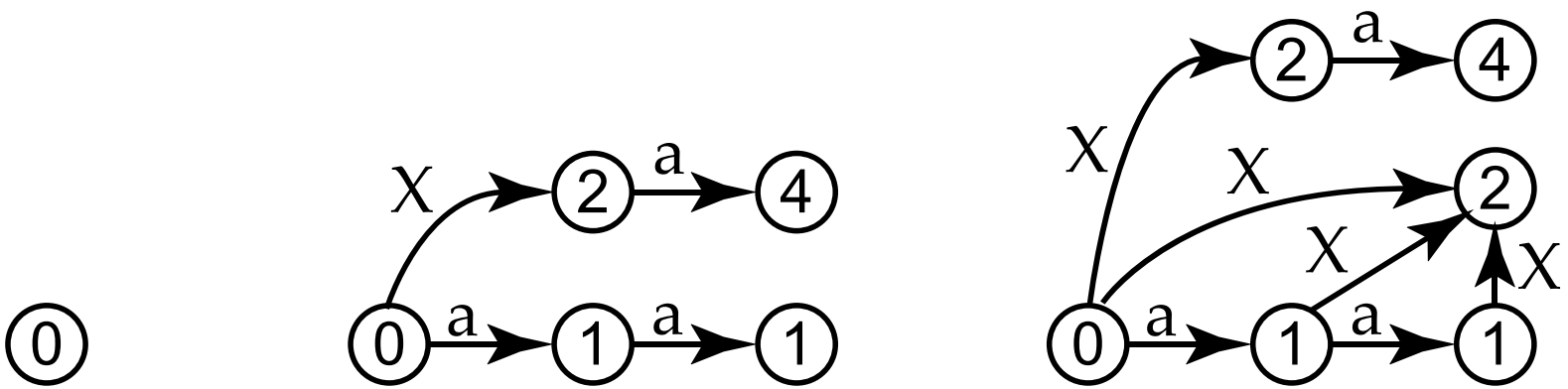
Let us trace the LR parser upon the input aa.

Example (continued)

$S \to X\&\neg aS$

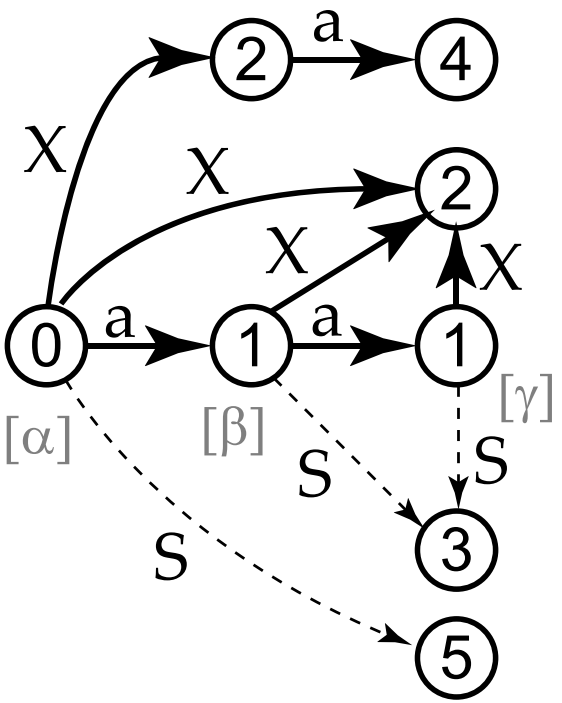
$X \to Xa \mid \varepsilon$

Until the end of the input, only X is used.



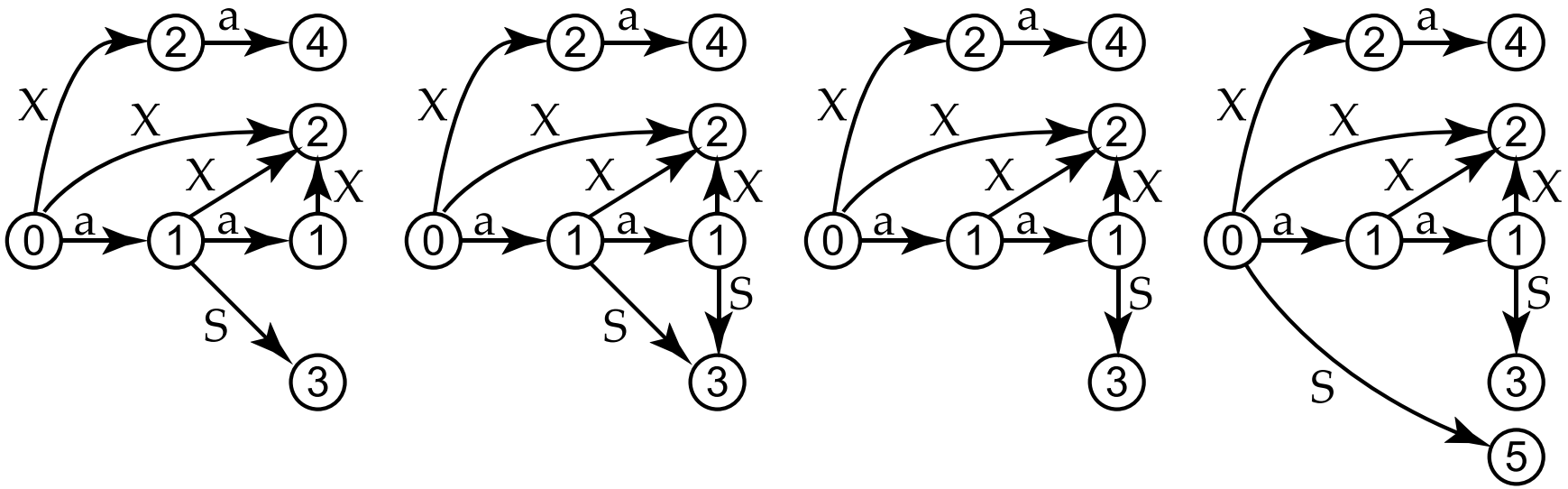
Example (continued)

$S \to X\&\neg aS$
$X \to Xa \mid \varepsilon$



Three possible reductions: α, β, γ.

- Conditions of existence:
  $\alpha = \neg\beta$, $\beta = \neg\gamma$, $\gamma = true$
- If we reduce by γ and α, we are done.
- Suppose we reduce by β. Then:
  1. Reduce by γ.
  2. Invalidate β.
  3. Reduce by α.


0
1
2
3
4
5
a
X
S

Applicability and correctness

As per the engineering approach to parsing, the algorithm is here.

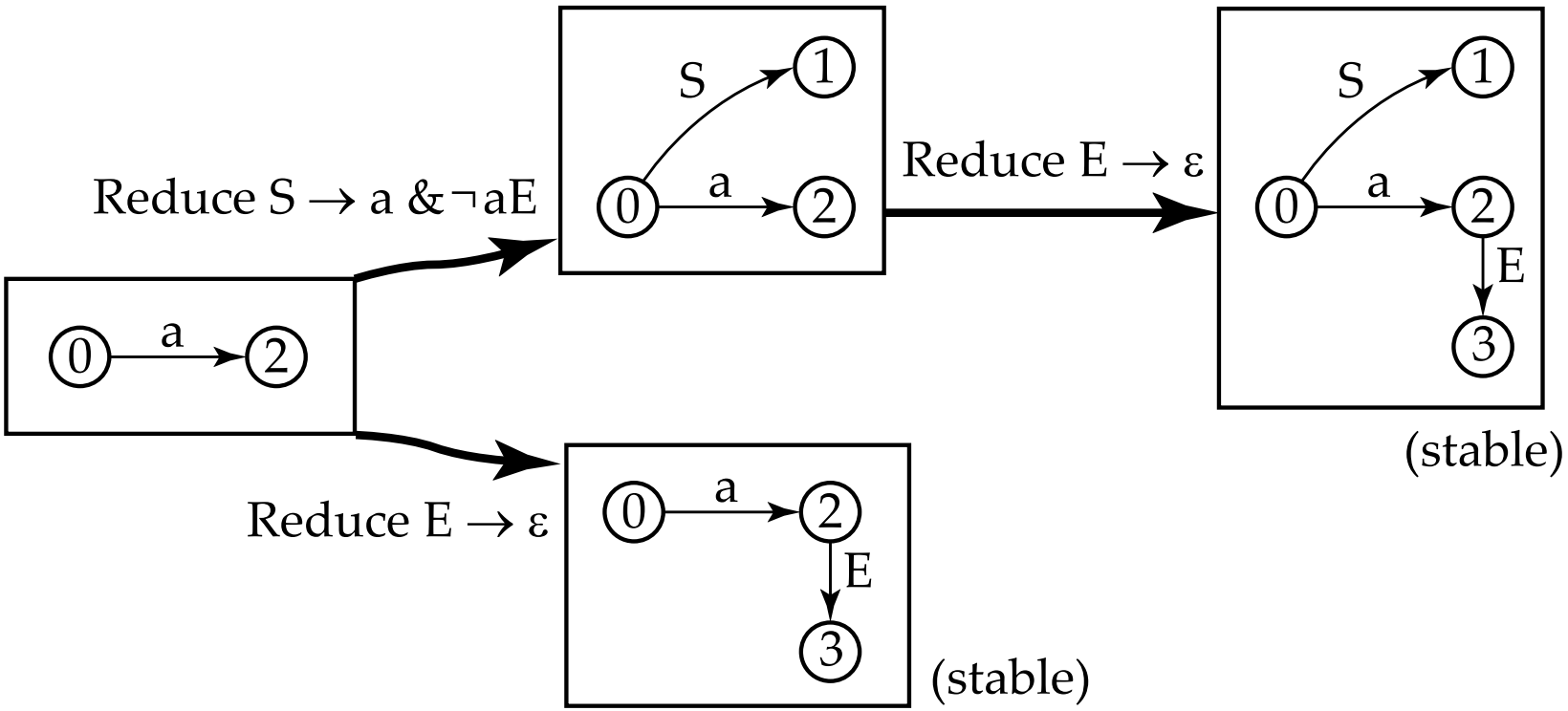
- Justified doubts about its soundness.
- Does it always terminate?
- Are the results consistent for different choice of reductions/invalidations.
- Do the results follow the grammar (a language equation)?

✓ Yes, except for a small class of "bad" grammars.

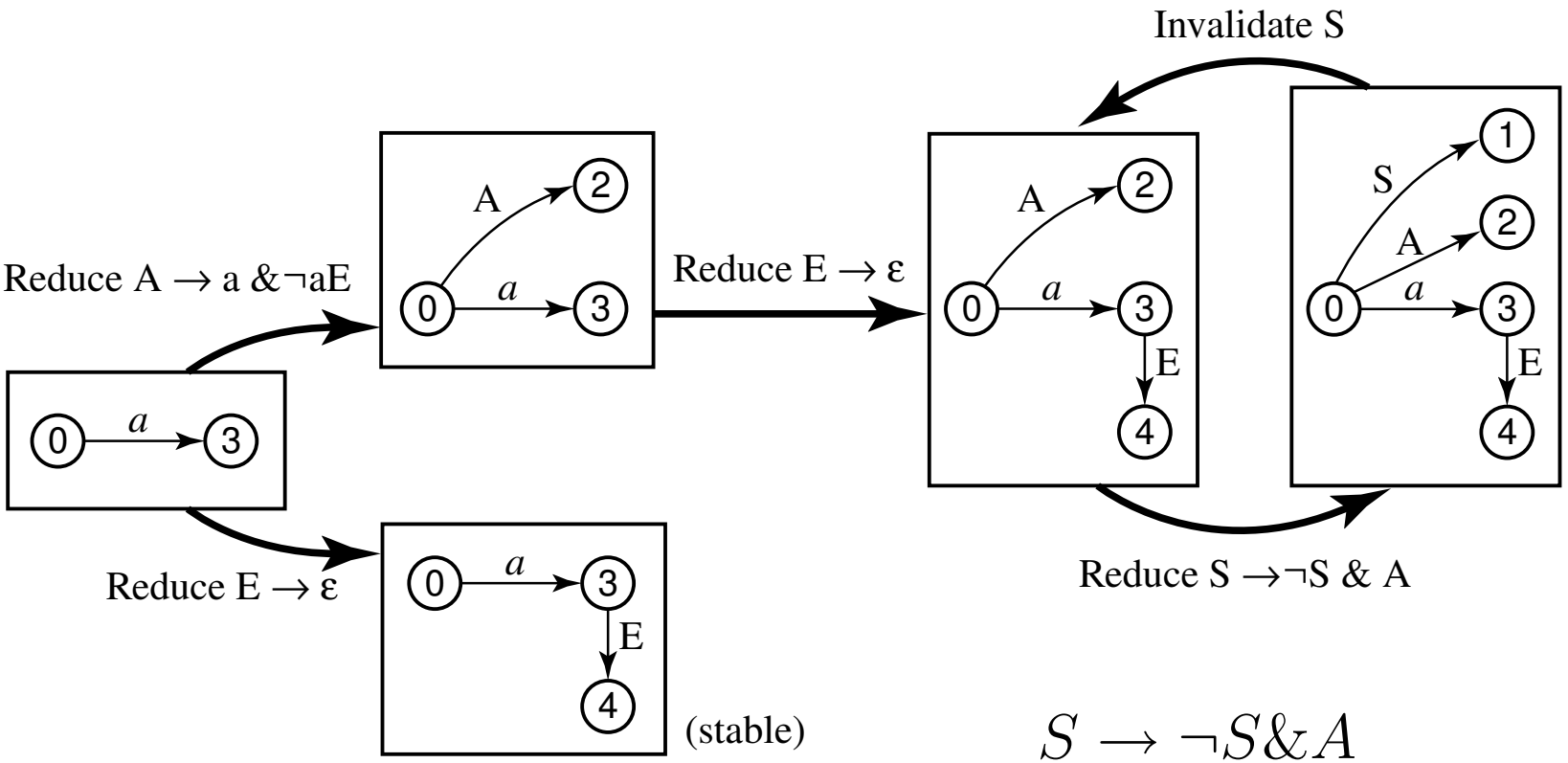
Counterexample One: inconsistent results

$S \to S \mid a\&\neg aE$

$E \to \varepsilon$



Counterexample Two: non-termination



$$S \to \neg S \& A$$
$$A \to A \mid a \& \neg a E$$
$$E \to \varepsilon$$

Resolving the doubts

A very weak syntactical condition upon the grammar is introduced.

- **Convergence lemma:** the reduction phase terminates for any choice of actions and produces consistent results.
- **Model computation:** there exists a sequence of reductions that sets the right arcs one by one.
- Why invalidation is not redundant?

Ensuring polynomial time complexity

How to implement the reduction phase?

- For an arbitrary implementation, time is $O(2^n)$.
- Do all possible reductions and invalidations at once.
- $O(n^3)$ or $O(n^4)$ upper bound (depends upon search methods).

An extensive example

- A Boolean grammar for the set of well-formed programs in a simple programming language.

```
average(x, y) { return (x+y)/2; }
factorial(n)
{
   var i, prod; i=1; prod=1;
   while(i<=n) {
      prod=prod*i;
      i=i+1;
   }
   return prod;
}
```

```
factorial2(n) {
   if(n>=2)
      return n*factorial2(n-1);
   else
      return 1;
}
main(arg) {
   return average(factorial(arg),
       factorial2(arg));
}
```

- 54 terminals ($\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots\}$), 123 nonterminals, 368 rules.
- The parser: 743 LR states, works in time $C \cdot n^2$.
- 256-symbol example: 3695 shifts, 7832 local errors, 24342 reductions, 57 invalidations; about 1 s on Pentium III.

Conclusion

- Boolean grammars as better context-free grammars.
- Many theoretical issues left to explore.
- Does $\mathcal{L}(Conj.) \subset \mathcal{L}(Bool.) \subset \mathcal{L}(DetCS)$ hold?
- Automaton representation? Known APDAs do not fit.
- Deterministic LR parsing?
- $o(n^2)$ parsing for a subclass?

Systematizing parsing for Boolean grammars

- *Parsing as deduction* vs. **parsing as search for a solution**.
- Parsing schemata (Sikkel, 1997) — a nonmonotone extension?