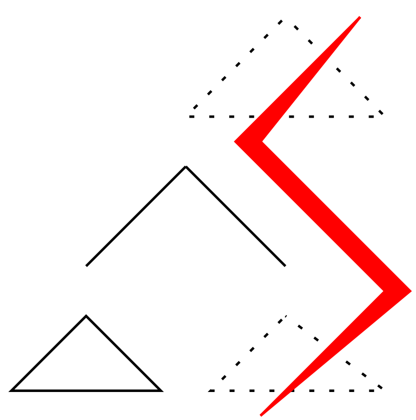
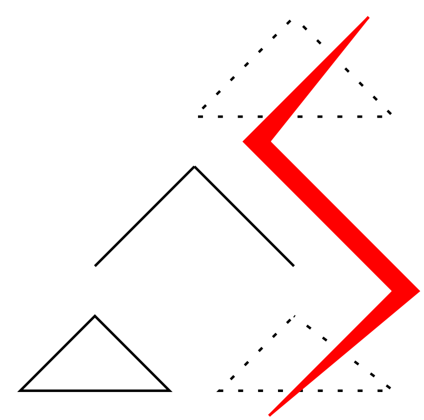
Finite-state Approximation of Constraint-based Grammars using Left-corner Grammar Transforms

Mark Johnson

Brown University



ACL-COLING 1998



Summary

- Approximating a Unification Grammar (UG) with a FSM
- FS approximations of top-down parsers
- Grammar transformation
  - Left-Corner (LC)
  - Composition/ϵ-removal
  - Partial evaluation

Why approximate UGs with FSMs?

- FSM processing is faster
  - linear time recognition
  - can be used as oracle to guide UG parser
- LC parsing has some psycholinguistic validity
- UG languages can be manipulated via FS calculus

Why use LC approximation?

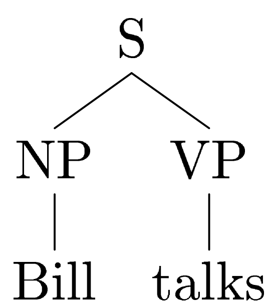
- LC parsing applies directly to UGs
- LC parsers require only finite stack-depth to parse left linear or right linear grammars

Non-deterministic top-down parsing

- Parser states are stacks of nonterminals and terminals $(N \cup T)^*$
- State transition function δ:

$$\gamma \in \delta(a\gamma, a) : a \in T, \gamma \in (N \cup T)^*.$$

$$\beta\gamma \in \delta(A\gamma, \epsilon) : A \in N, \gamma \in (N \cup T)^*, A \to \beta \in P.$$



State	**Remaining input**
S	*NP VP*
NP VP	*NP VP*
VP	*VP*
ϵ	ϵ

FS approximations to TD states

- Unbounded state stack size
  - ignore state stacks larger than some fixed bound

    $\Rightarrow$ approximation accepts a *subset* of UG language
  - collapse all states sharing a common prefix

    $\Rightarrow$ approximation accepts a *superset* of UG language
- Unbounded UG categories
  - Restriction (a.k.a. abstraction) (Shieber 1985)

    $\Rightarrow$ approximation accepts a *superset* of UG language
  - Ignore categories whose complexity exceeds some bound

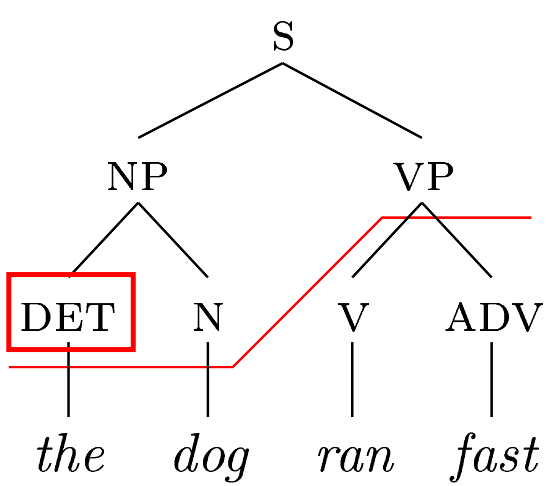
    $\Rightarrow$ approximation accepts a *subset* of UG language
    - In many UGs, the *syntactically potent features* range over finite values

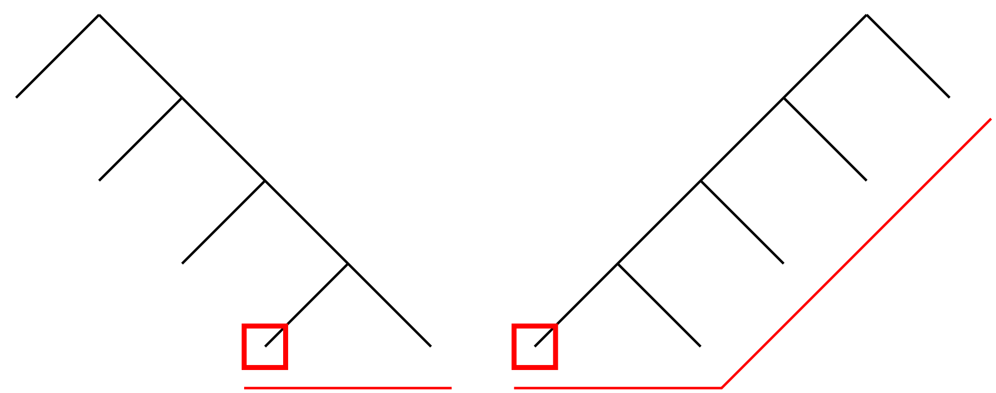
States of a TD parser

- Just before X is expanded, the TD parser's state consists of X followed by *the right siblings of it and all its ancestors.*

$\Rightarrow$ Right-linear grammars ($A \to w\,B$) require *finite* state size

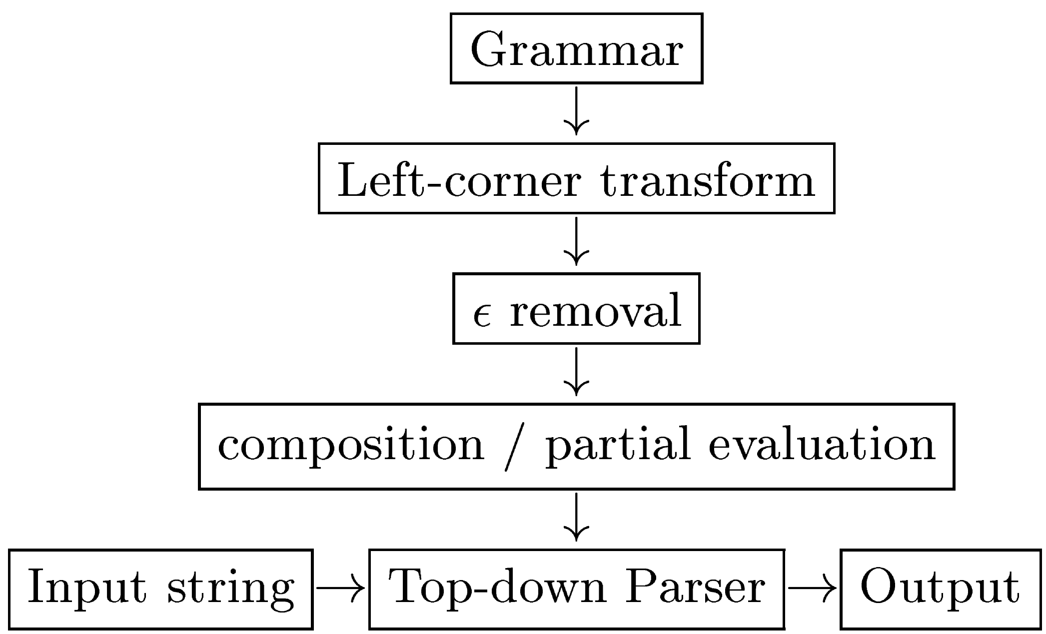
$\Rightarrow$ Left-linear grammars ($A \to B\,w$) require *unbounded* state size





Left-corner grammar transforms

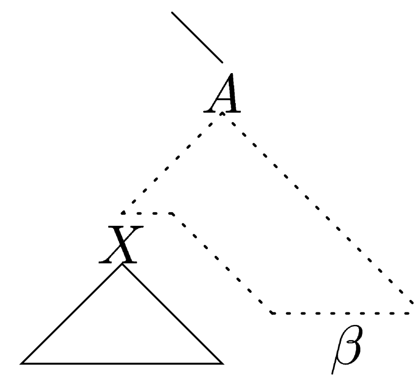
- A Left-Corner (LC) parser exhibits finite state size on both left-linear and right-linear CFGs (*)
- A LC parser for grammar G acts isomorphically to a top-down parser using $\mathcal{LC}(G)$.



Left-corner grammar transform

- Left-corner of each production is recognized bottom-up, everything else is predicted top-down
- Nonterminals of $\mathcal{LC}(G) = N \cup N \times (N \cup T)$

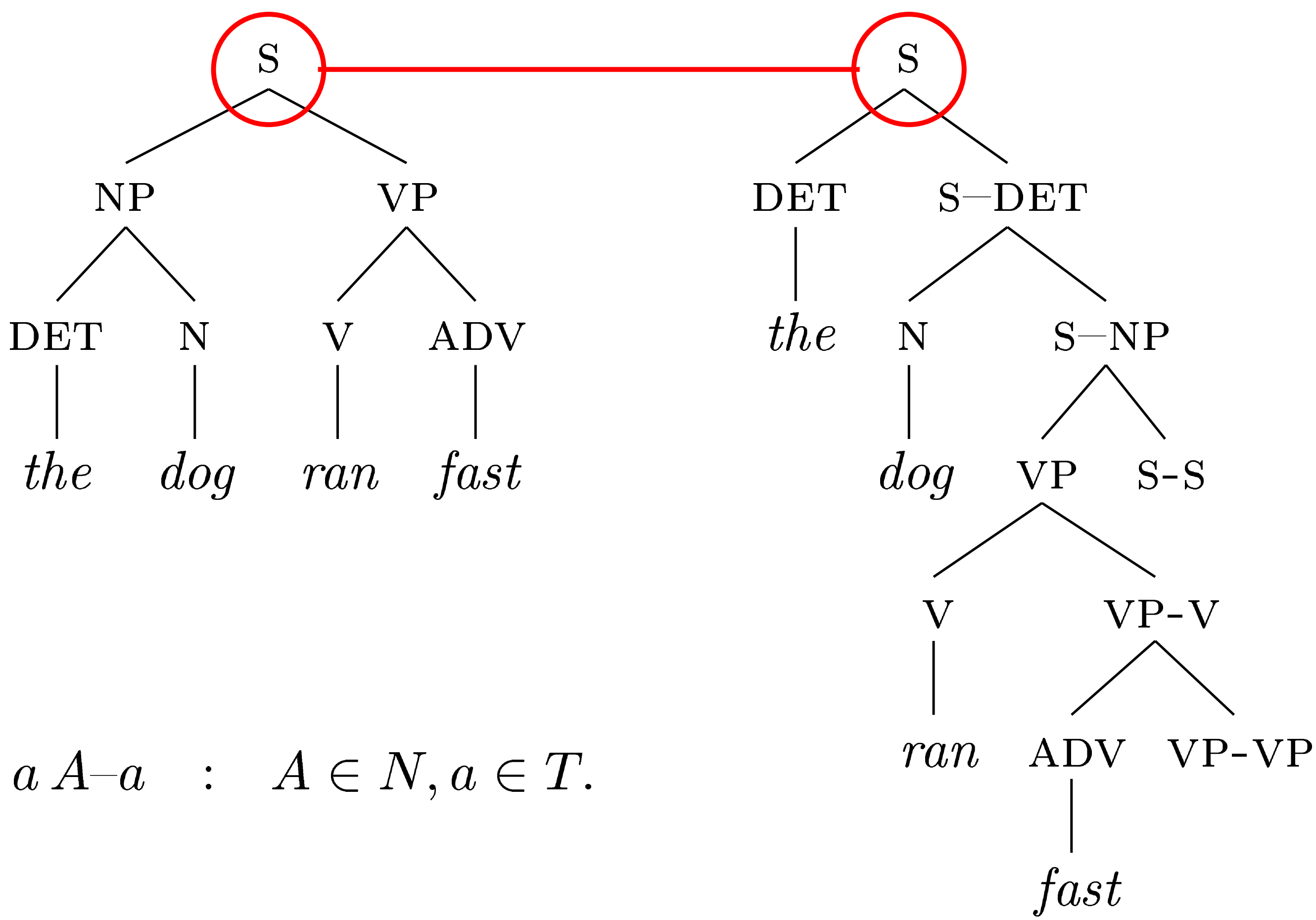
$$A \Rightarrow_G^* X\beta \text{ iff } A\text{–}X \Rightarrow_{\mathcal{LC}(G)}^* \beta$$



- Productions of $\mathcal{LC}(G) =$

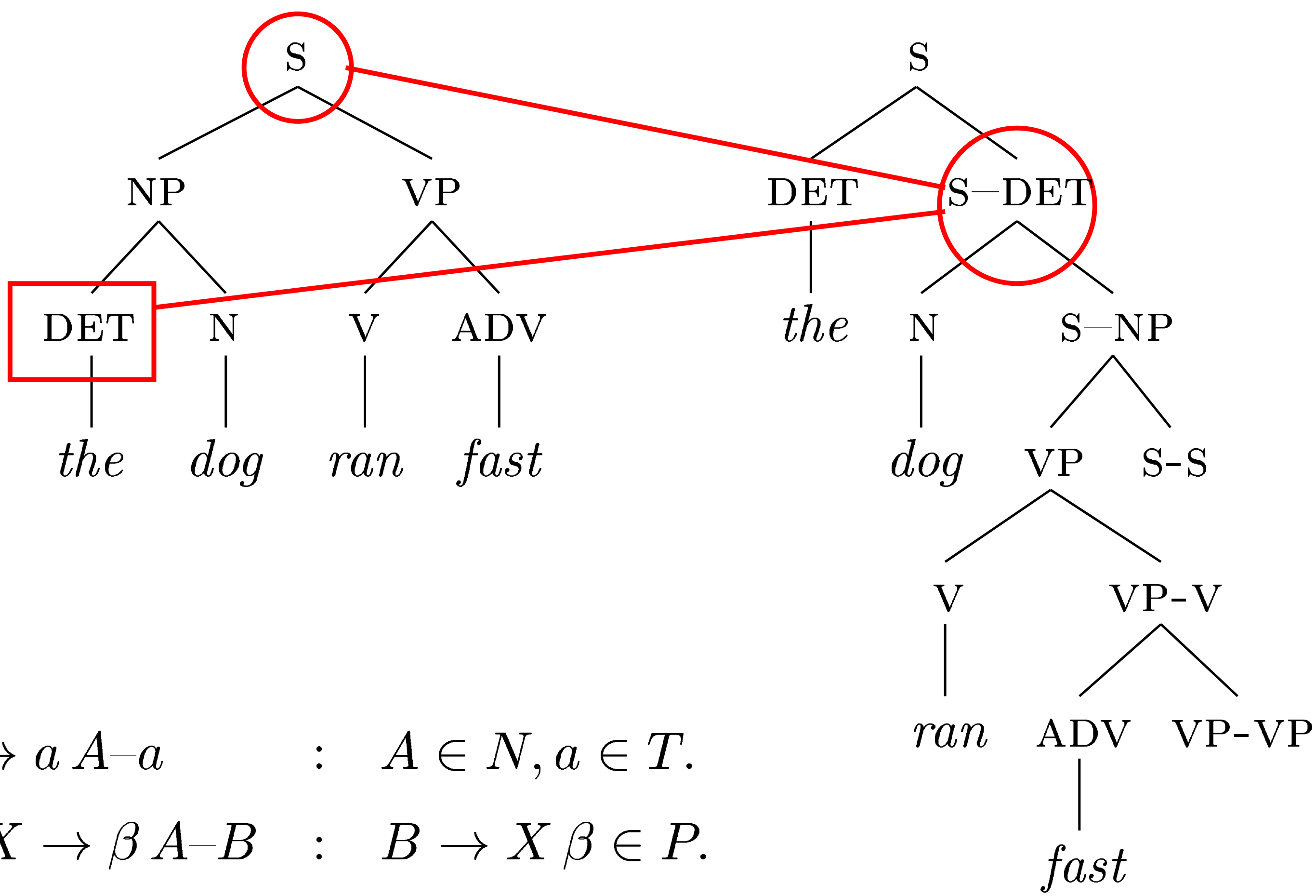
$$\begin{array}{lcl} A \to a\, A\text{–}a & : & A \in N, a \in T. \\ A\text{–}X \to \beta\, A\text{–}B & : & A \in N, B \to X\,\beta \in P. \\ A\text{–}A \to \epsilon & : & A \in N. \end{array}$$

Parsing with $\mathcal{LC}(G)$: start



$A \rightarrow a\, A{-}a \quad : \quad A \in N, a \in T.$

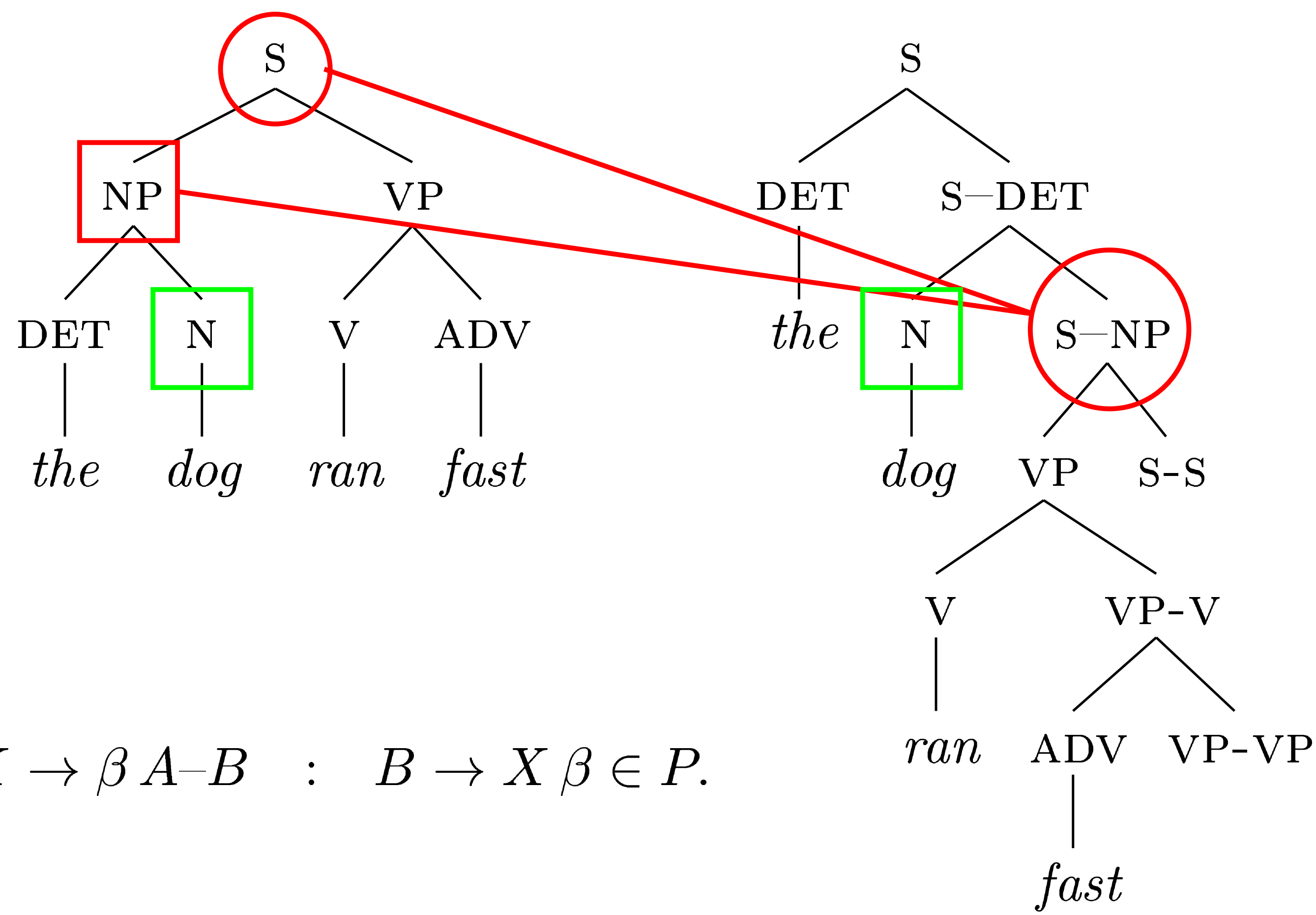
Parsing with $\mathcal{LC}(G)$: shift DET



$$A \to a\ A\text{–}a \quad : \quad A \in N, a \in T.$$

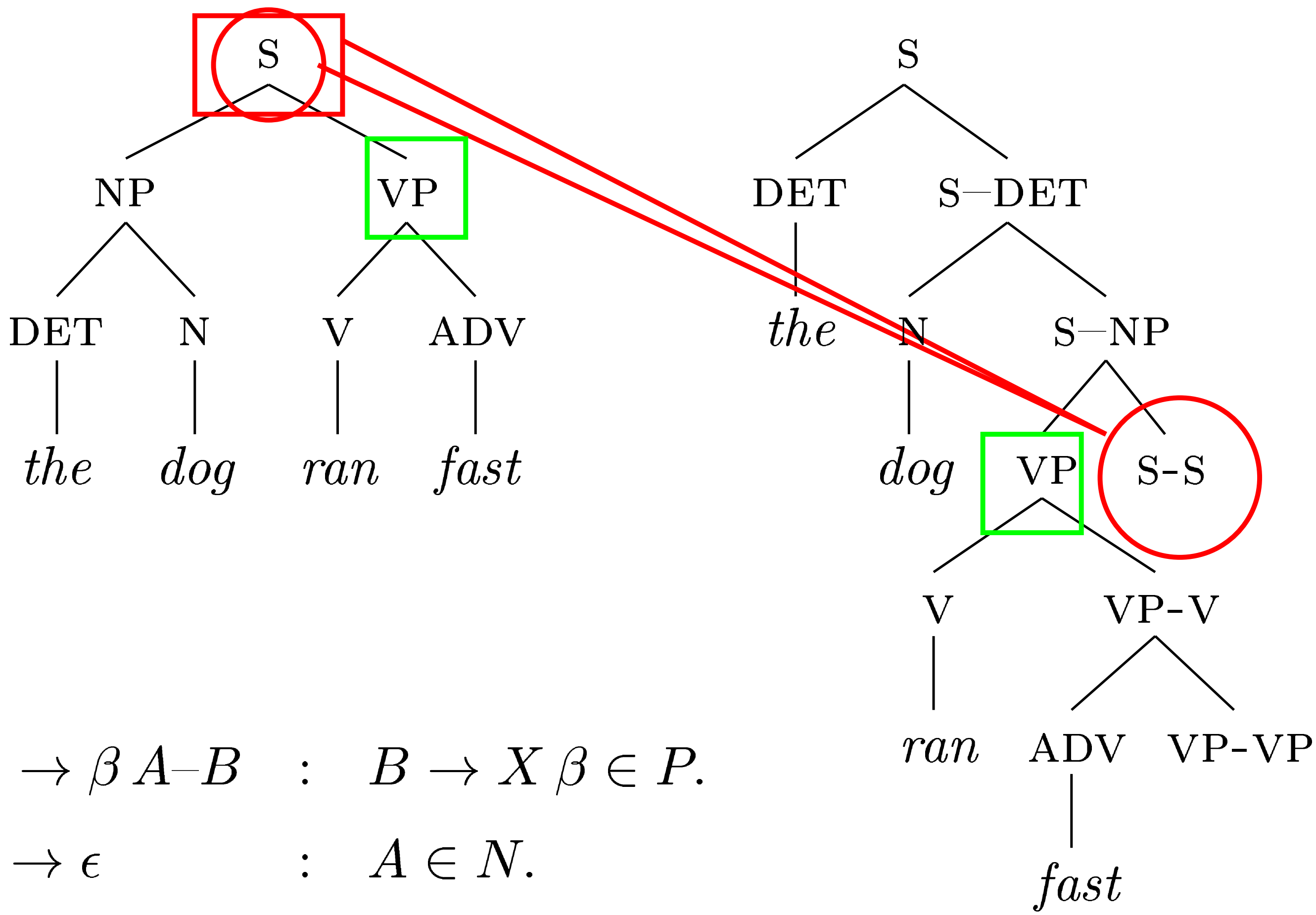
$$A\text{–}X \to \beta\ A\text{–}B \quad : \quad B \to X\ \beta \in P.$$

Parsing with $\mathcal{LC}(G)$: NP



$$A\text{–}X \to \beta\, A\text{–}B \quad : \quad B \to X\, \beta \in P.$$

Parsing with $\mathcal{LC}(G)$: S

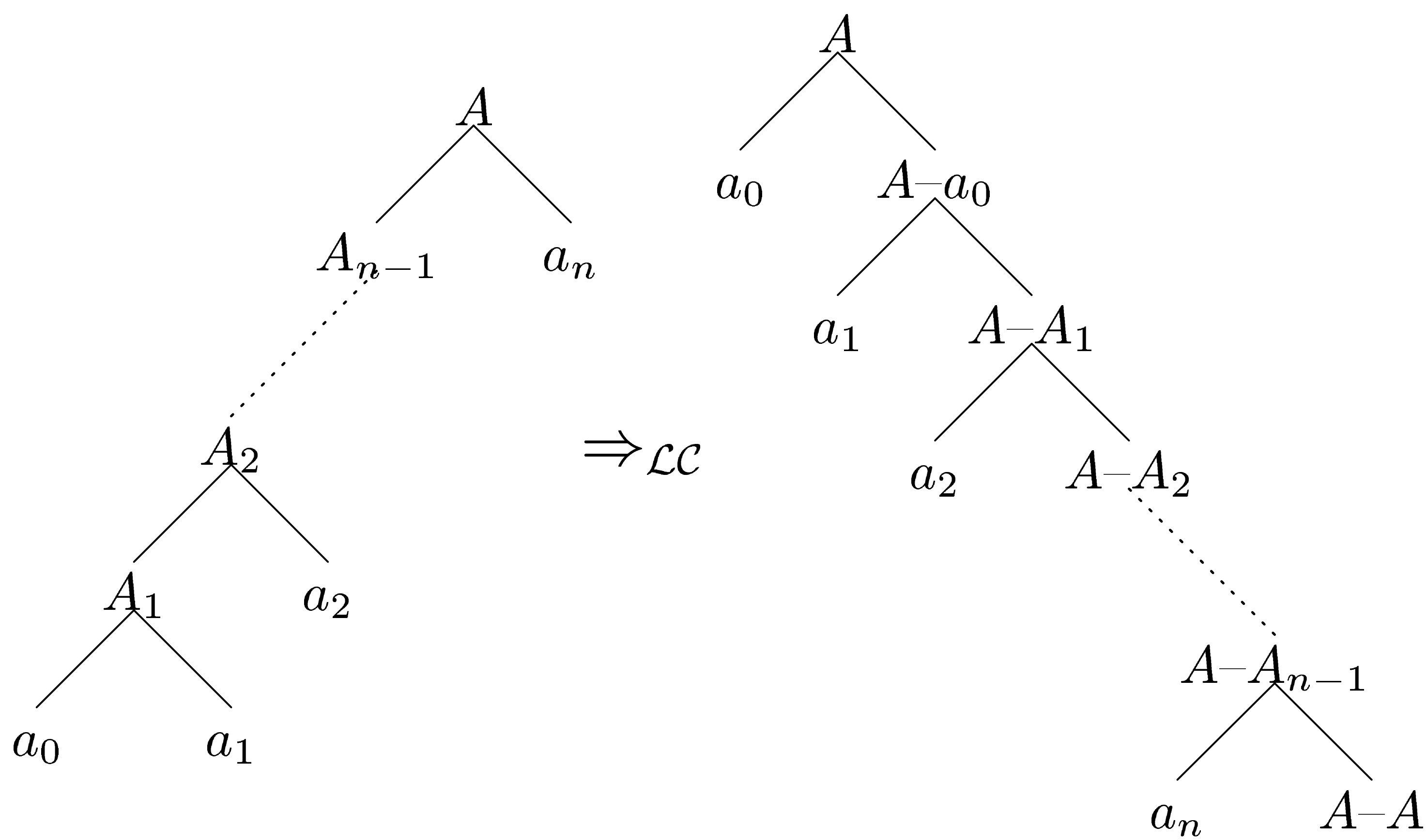


$$A\text{–}X \rightarrow \beta\, A\text{–}B \quad : \quad B \rightarrow X\,\beta \in P.$$

$$A\text{–}A \rightarrow \epsilon \quad : \quad A \in N.$$

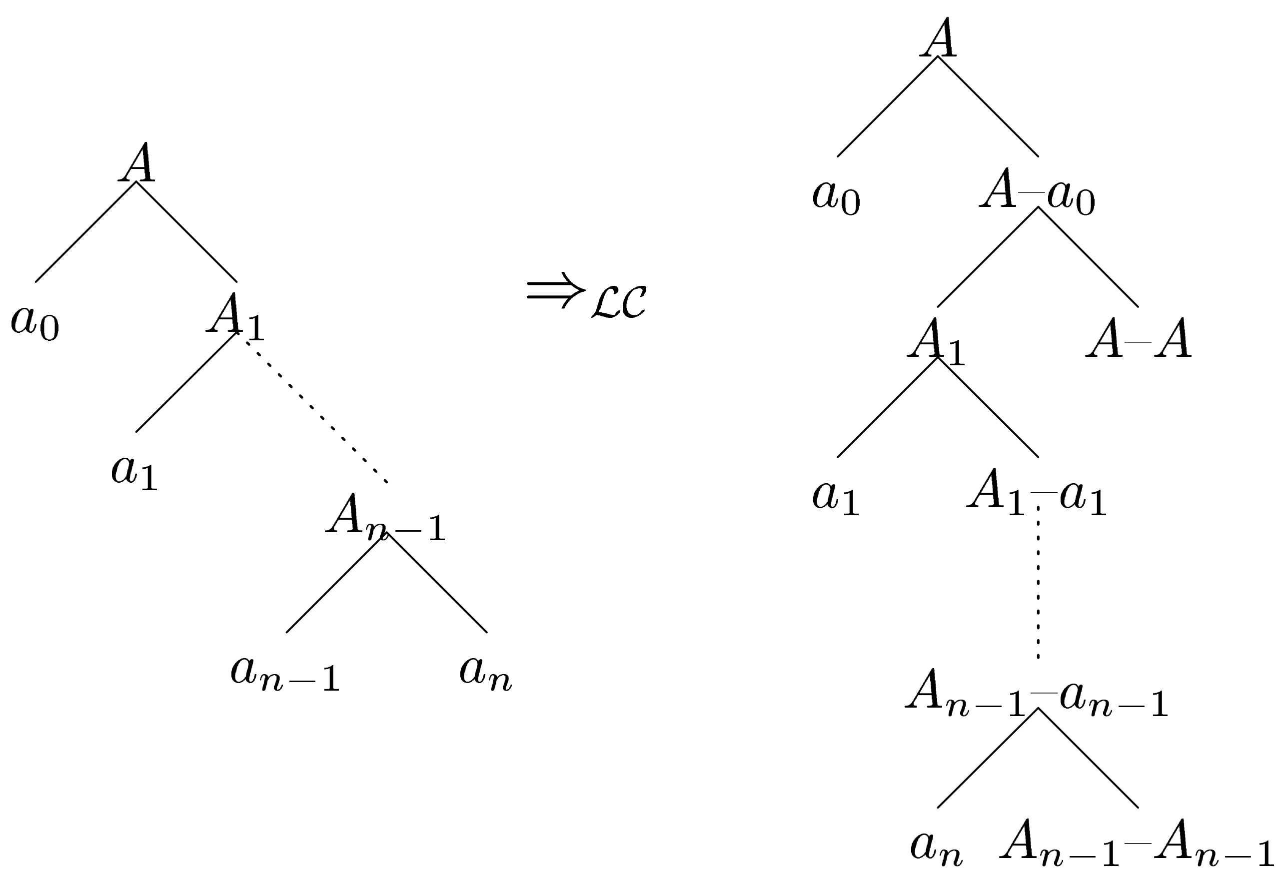
States of an LC parser

- Left-linear $G \Rightarrow$ right-linear $\mathcal{LC}(G) \Rightarrow$ finite states



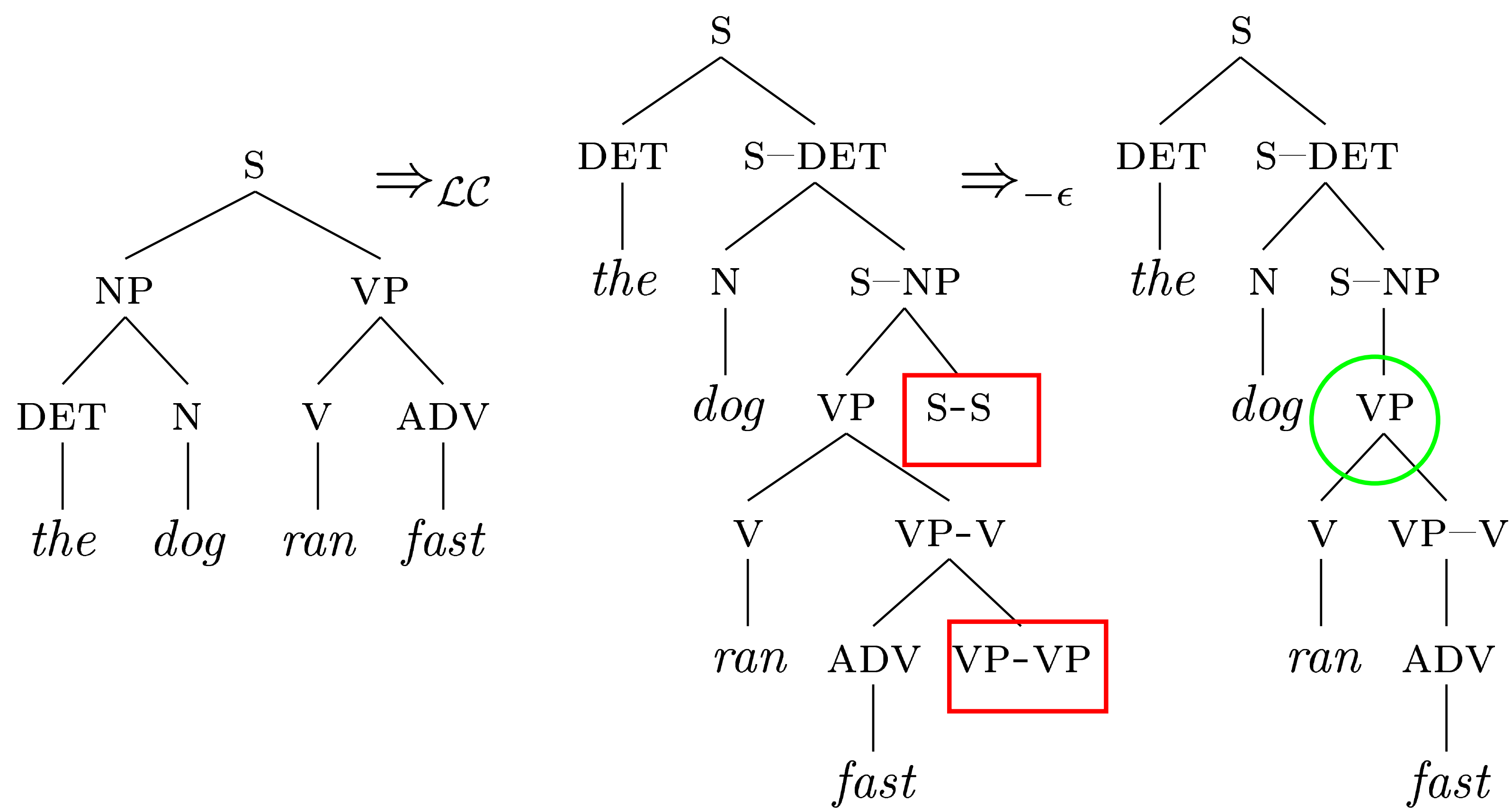
States of an LC parser (cont.)

- Right-linear $G \Rightarrow$ unbounded TD states in $\mathcal{LC}(G)$



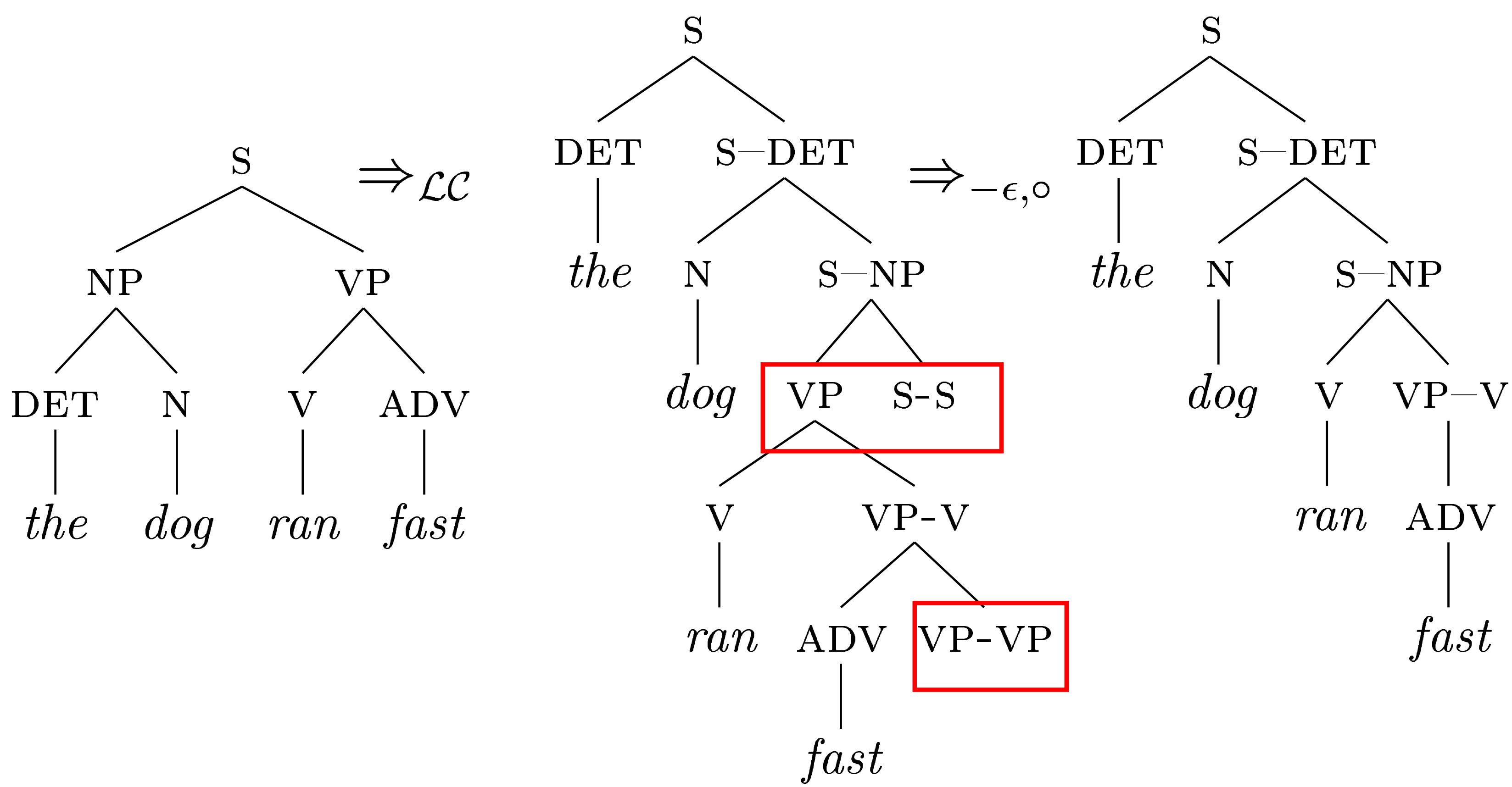
Epsilon-removal after $\mathcal{LC}$ transform

- Linear $G \Rightarrow$ right-linear $\mathcal{LC}'(G) \Rightarrow$ finite TD states



Partial evaluation/composition

- Converts binary branches into (almost) binary branches



Special case of binary productions

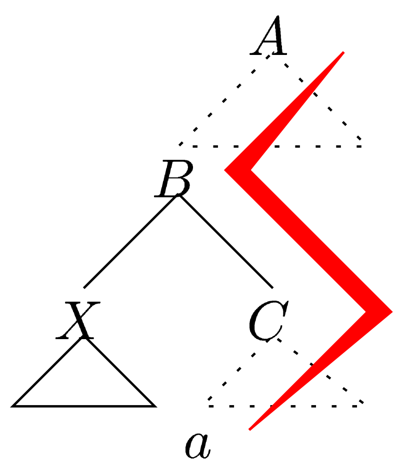
$$\begin{array}{lcl}
S \to a\, S\text{–}a & : & a \in T. \\
A\text{–}X \to a\ A\text{–}B & : & A \in N, B \to X\, a \in P. \\
A\text{–}X \to a & : & A \to X\, a \in P. \\
A\text{–}X \to a\ C\text{–}a & : & A \to X\, C \in P. \\
A\text{–}X \to a\ C\text{–}a\ A\text{–}B & : & A \in N, B \to X\, C \in P.
\end{array}$$

- All but one schema are right-linear
- Exactly one transformed rule per input item
- Such productions can be implemented as FSM arcs, e.g.:

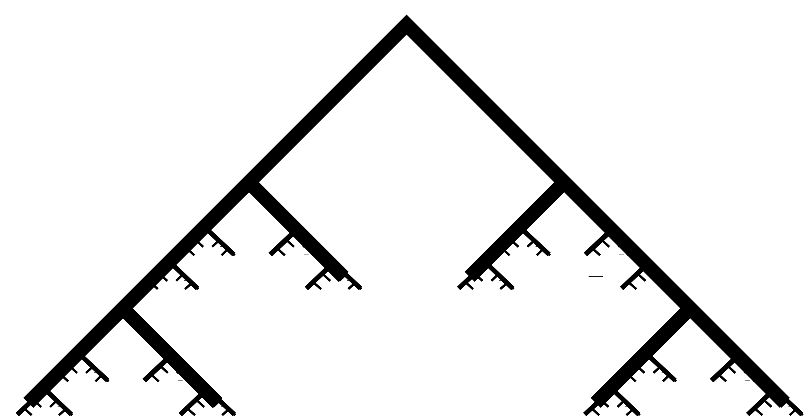
$$A\text{–}B\beta \ \in\ \delta(A\text{–}X\beta, a)\ :\ B \to X\, a \in P.$$

Geometry of LC state complexity

- Because only one production schema increases the stack state size, LC state complexity is associated with a specific tree geometry
- Helps characterize the errors in a FS approximation



$A\text{–}X \rightarrow a\ C\text{–}a\ A\text{–}B$



Odds and ends

- Classifying unification grammar categories
- Identifying useless productions in $\mathcal{LC}(G)$ (link table)
- Obtaining parse trees from FSM transitions
  - FS transducer emits rule schema used at each transition,
  - which guides LC parser for G

Conclusion

- Left-corner grammar transforms convert left recursion into right recursion
- A finite-state approximation can be directly constructed from transformed unification grammars
- The approximation is exact for left linear and right linear CFGs
- A characterization of LC state complexity identifies constructions for which the approximation is inexact