Pushdown Automata

- The table-driven predictive parser is a specific example of a class of automata called pushdown automata (PDA) that recognize exactly the class of context-free languages.
  - The generalizations come in several areas.
    - The set of PDA (Q) can differ from the non-terminals of the grammar.
    - The stack symbols can be drawn from another alphabet Γ.
    - The actions taken for (X, a) combinations are generalized to be mappings from $Q \times F \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.

Pushdown Automata

- The table-driven predictive parser is a specific example of a class of automata called pushdown automata (PDA) that recognize exactly the class of context-free languages.
  - The generalizations come in several areas.
    - The set of PDA (Q) can differ from the non-terminals of the grammar.
    - The stack symbols can be drawn from another alphabet Γ.
    - The actions taken for (X, a) combinations are generalized to be mappings from $Q \times F \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.
- As in the case of finite automata, PDAs can come in deterministic or non-deterministic flavors.
  - Non-deterministic PDAs are more powerful than deterministic PDAs.

Towards Grammar Flow Graphs

- Is there a graphical interpretation equivalent to PDAs, just as there was for finite automata?
  - Yes, it's called a Grammar Flow Graph (GFG).
  - It's a little more complicated than the graph of a FA, because we need to split the *finite* structure of the graph (the states) from the *unbounded* amount of memory of the input prefix that it needs to retain (the stack).

Towards Grammar Flow Graphs

- Is there a graphical interpretation equivalent to PDAs, just as there was for finite automata?
  - Yes, it's called a Grammar Flow Graph (GFG).
  - It's a little more complicated than the graph of a FA, because we need to split the *finite* structure of the graph (the states) from the *unbounded* amount of memory of the input prefix that it needs to retain (the stack).
- Game plan
  - Introduce the structure of GFGs as a graphical representation of a CFG.
  - Show why an NFA simulation of the GFG (i.e., treating acceptance as a simple path problem) is insufficient and inaccurate.
  - Introduce the the NGA and its inherently non-deterministic semantics.

Towards Grammar Flow Graphs

- Is there a graphical interpretation equivalent to PDAs, just as there was for finite automata?
  - Yes, it's called a Grammar Flow Graph (GFG).
  - It's a little more complicated than the graph of a FA, because we need to split the *finite* structure of the graph (the states) from the *unbounded* amount of memory of the input prefix that it needs to retain (the stack).
- Game plan
  - Introduce the structure of GFGs as a graphical representation of a CFG.
  - Show why an NFA simulation of the GFG (i.e., treating acceptance as a simple path problem) is insufficient and inaccurate.
  - Introduce the the NGA and its inherently non-deterministic semantics.
  - Show a correct way to formulate acceptance as a path problem using tags (Earley's algorithm).
  - Cast other parsing algorithms in the GFG framework.
  - Show how the GFG devolves to a FA for right-linear grammars.

An Example GFG (Nodes Only)

$S \rightarrow Ab \mid bA$
$A \rightarrow bb$

$.S$

$S \rightarrow .Ab$

$S \rightarrow A.b$

$S \rightarrow Ab.$

$.A$

$A \rightarrow .bb$

$A \rightarrow b.b$

$A \rightarrow bb.$

$A.$

$S \rightarrow .bA$

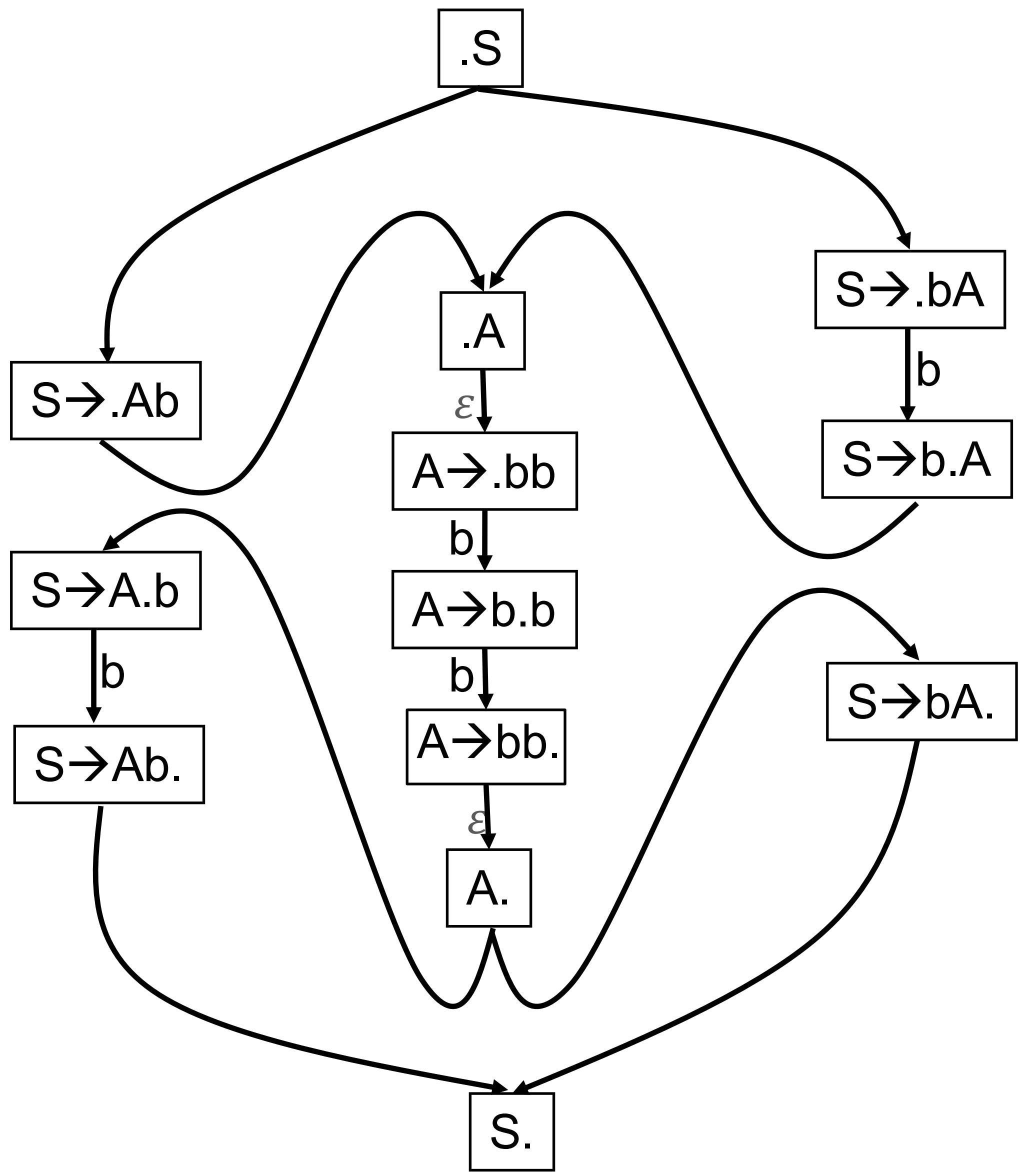
$S \rightarrow b.A$

$S \rightarrow bA.$

$S.$

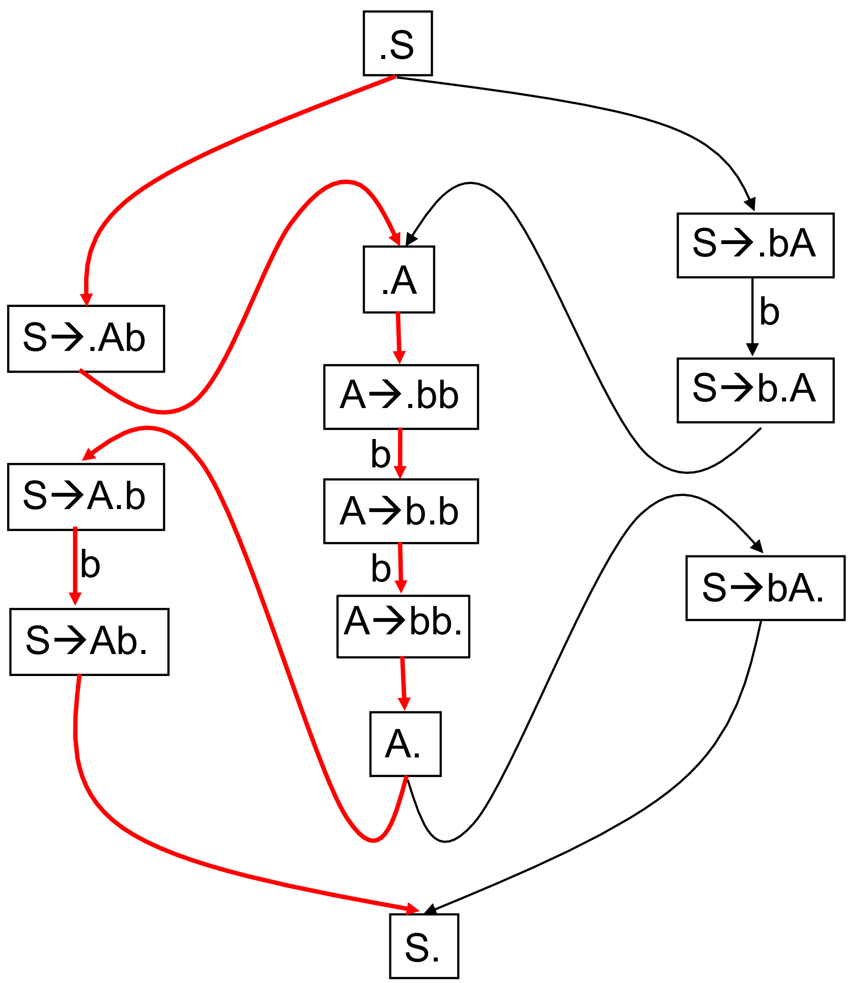
An Example GFG (Nodes + Edges)

S→Ab | bA
A→bb



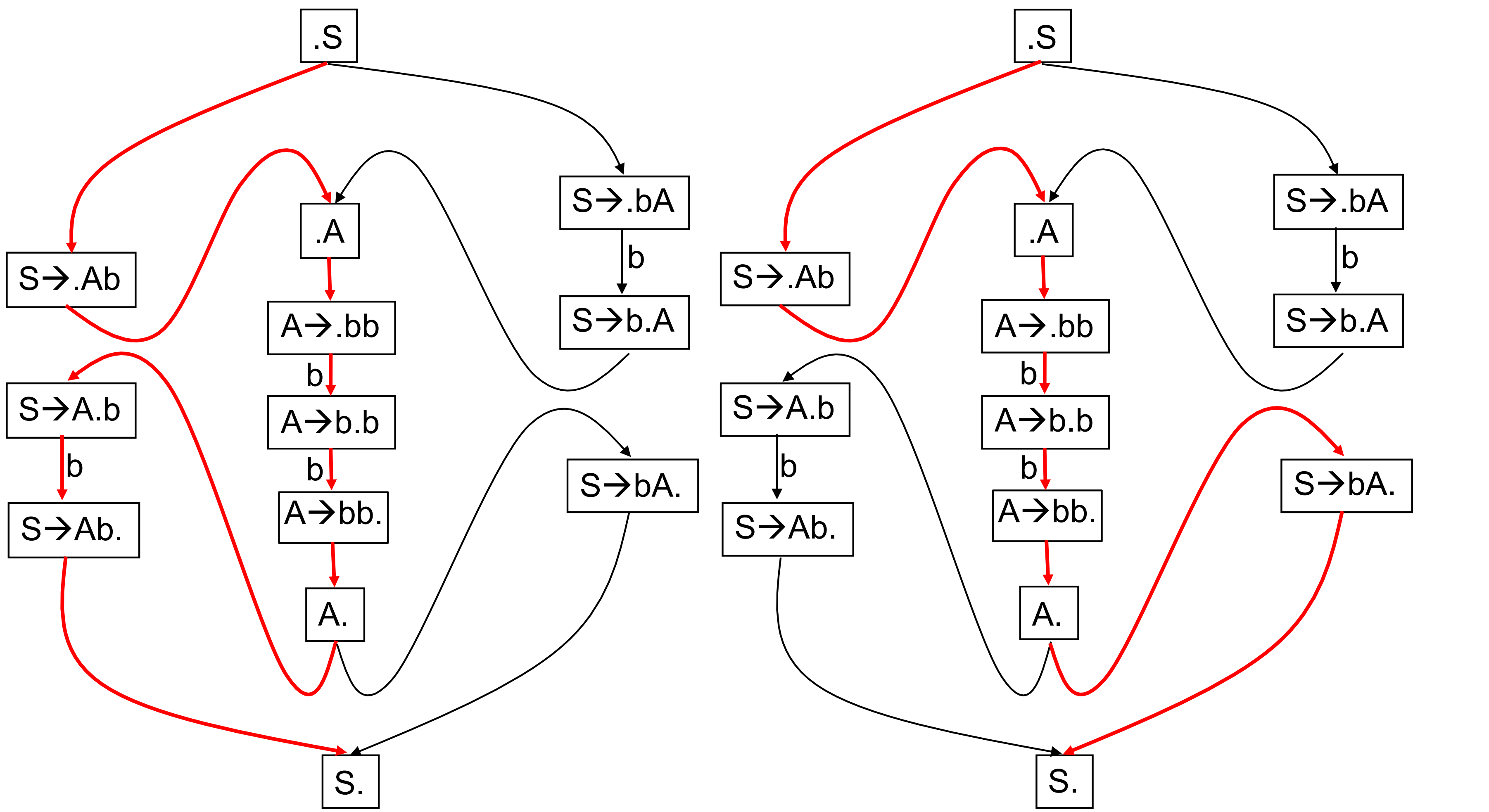
NFA Interpretation of GFG Is Incorrect

S→Ab | bA
A→bb

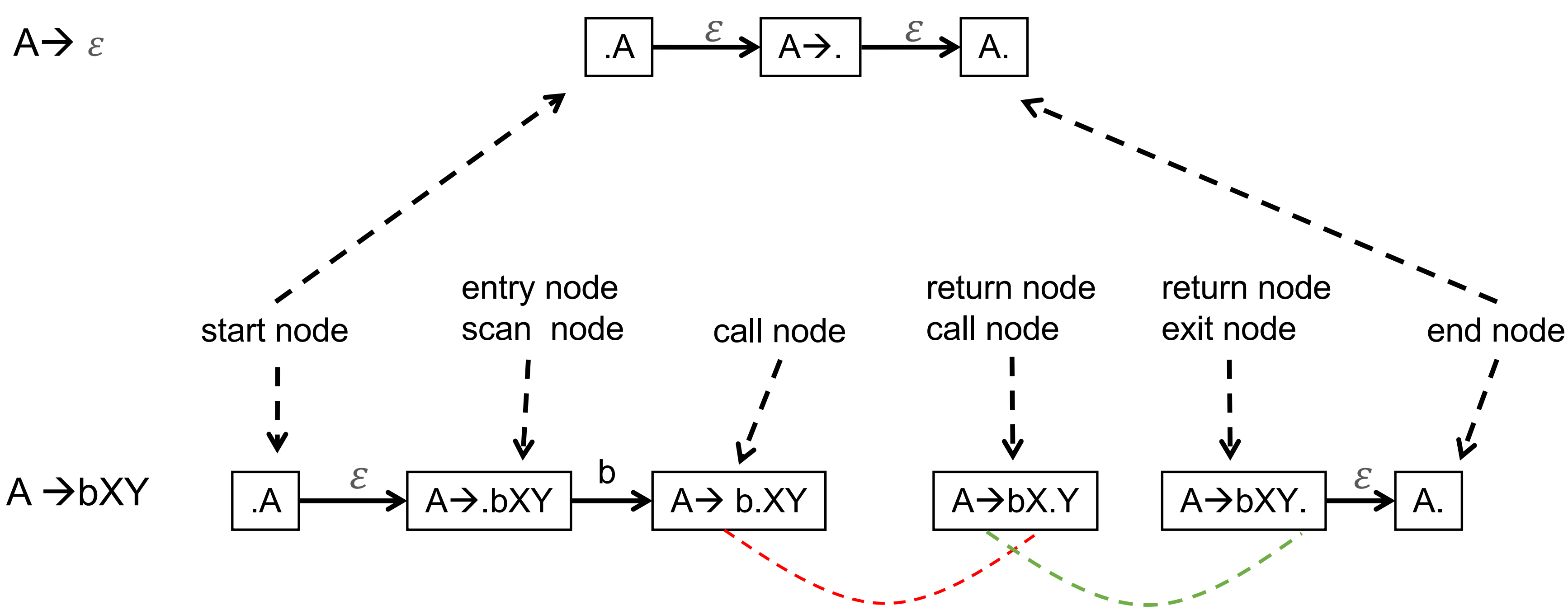


NFA Interpretation of GFG Is Incorrect

S→Ab | bA
A→bb



GFG Terminology



Key property: Call and return nodes come in matched pairs.

Nondeterministic GFG Automaton (NGA)

- Given: a CFG $G = (N, T, P, S)$ and its corresponding GFG $\Gamma(G) = \big(V(G), E(G)\big)$.
  - Let $R \subset V$ be the set of return nodes of Γ.
- A configuration is a triple $\langle v, k, w\rangle$ with $v \in V$, $k \in R^*$, $w \in T^* \times T^*$ ("node-stack-input").
- Initial configuration: $\langle \bullet\, S, [], \bullet\, s\rangle$, where s is the input string.
- Accepting configuration: $\langle S\, \bullet, [], s\, \bullet\rangle$.
- The transition function depends on the node type.
  - Start node: $\langle \bullet\, B, k, w\rangle \Longrightarrow \langle B \rightarrow \bullet\, \beta, k, w\rangle$ (non-deterministic choice).
  - Exit node: $\langle B \rightarrow \beta\, \bullet, k, w\rangle \Longrightarrow \langle B\, \bullet, k, w\rangle$.
  - Scan node: $\langle A \rightarrow \alpha \bullet t\gamma, k, u \bullet tv\rangle \Longrightarrow \langle A \rightarrow \alpha t \bullet \gamma, k, ut \bullet v\rangle$.
  - Call node: $\langle A \rightarrow \alpha \bullet B\gamma, k, c\ \rangle \Longrightarrow \langle \bullet\, B, [A \rightarrow \alpha B \bullet \gamma, k], c\rangle$.
  - End node: $\langle B\, \bullet, [A \rightarrow \alpha B \bullet \gamma, k], w\rangle \Longrightarrow \langle A \rightarrow \alpha B \bullet \gamma, k, w\rangle$.
  - Return node: Nothing to do (why?).

Properties of the NGA

- The NGA is a special kind of PDA.
- Paths traversed in the GFG by the NGA are complete balanced paths.
  - Complete: Goes from$\bullet\, S$ to $S\, \bullet$.
  - Balanced: Call and return nodes are correctly matched.
- Theorems
  - Every complete balanced path corresponds to a parse tree, and every parse tree has a corresponding complete balanced path.
  - The label of every completely balanced path is a sentence generated by the grammar, and every sentence generated by the grammar corresponds to the label of a completed balanced path.
- An unambiguous grammar is one in which every string in the language of the grammar is generated by exactly one complete balanced path.