LR Parsing

Three Different Parsers

Three Different Finite Automata for VP_G

SLR The simplest to construct:

- Compact FSA Tables
- Small set of **C**ontext **F**ree, **L**inear **T**ime Analyzable, Languages

LR Quite simple to construct:

- Large FSA Tables
- Largest set of **CF**, **LT** Analyzable, Languages

LALR A good compromise between the two, above:

- Same size of the SLR Tables
- Significantly large, set of **CF**, **LT** Aanalyzable, languages
- A bit intricate to construct

SLR Parsing

VP_G - Items of the LR(0) Collection

In order to detect **Viable Prefixes**, the grammar (canonical) productions can be examined. The use of **Items** is fundamental in such an exam.

Let **$A ::= \alpha\beta$** be a canonical production. Then:

$A ::= \alpha.\beta$ is a **LR(0) item.**

Moreover, let: $S \Rightarrow^* \gamma A \delta \Rightarrow \gamma\ \alpha\beta\ \delta$. Then:

- α is a trait of a viable prefix $\gamma\alpha$
- $A ::= \alpha.\beta$ is a **valid item** for $\gamma\alpha$

- **Valid Items** are collected into sets according to the prefixes that they can recognize;
- Each of such **sets is a state** of a finite state automaton of **VP_G**

SLR Parsing

The LR(0) Collection: State Closure

Two forms of closure (as for Dotted Automata):

- State Closure

- State Transition Closure

State Closure. Let I be a set of Valid Items. Then I is in the collection only if I=Closure(I

$$\textbf{Closure(I)} =_{\min} \textbf{I} \cup \textbf{Closure}\{B::=.\gamma \mid A::=\alpha.B\beta \in I\}$$

Why? What is the rational of the requirement, above?

If $A::=\alpha.B\beta$ is in I, then, for some ρ, it is considered valid for $\rho\alpha$. But $B::=.\gamma$ too, is valid for $\rho\alpha$. As a matter of this fact:

if $S \Rightarrow^* \rho A\delta \Rightarrow \rho\, \alpha B\beta\, \delta$ *then* $S \Rightarrow^* \rho\, \alpha B\beta\, \delta \Rightarrow \rho\alpha\, \gamma\, \beta\delta$

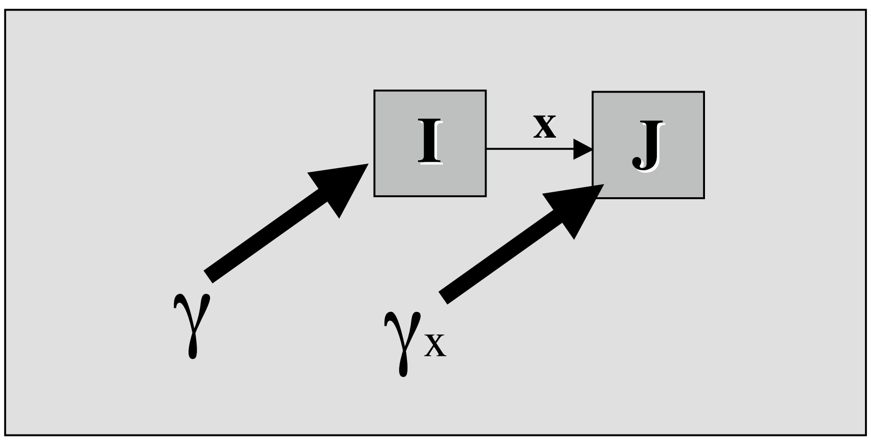
SLR Parsing

The LR(0) Collection: State Transition Closure

State Transition Closure. Let I be in the collection LR(0). Then all the outcomings of I are incomings of states of LR(0)

$$J = \mathrm{Goto}(I,x) = \mathrm{Closure}\{A::=\alpha x.\beta \mid A::=\alpha .x\beta \in I\}$$

Why? What is the rational of the requirement, abobe? - A graphical answer



SLR Parsing

The LR(0) Collection of G: The Initial State

The LR(0) Collection defines States and Transitions of the Viable Prefixes to SLR(1) parsing Handles.

The LR(0) Collection is a set of states of LR(0) Valid Items:

* Each state is closed w.r.t. a closure operation called **Closure**
* The transition set is closed w.r.t. a closure operation called **Goto**

The initial state of the LR(0) collection is **I0** and is defined below.
Let G=<S,V,P,s> be the grammar. Then, the augmented grammar of G, is G'=<S∪{s'},V,P ∪{s'::=s},s'>.

I0=Closure({s'::=.s})

Example

Apply The construction of LR(0) collection to the grammar G

Grammar G

S::= aABe
A::= Abc | b
B::= d

→

Augmented Grammar G'

S'::=S
S::= aABe
A::= Abc | b
B::= d

Initial State I0

I0:Closure({S'::=.S})={S'::=.S
S::= .aABe}

Computation of the remaining states

Example

Computation of the remaining states

S’::=S
S::= aABe
A::= Abc | b
B::= d

- **Handle Items**. They have the dot just on the right end (here, are red marked)
- **Shift Items**. They have the dot just on the left of a terminal symbol.
- Handles always have an Handle Item at the end of the prefix.

I0:Closure({S’::=.S}) ={S’::=.S
S::= .aABe}
I1:Goto(I0,S) ={S’::=S.}
I2:Goto(I0,a) ={S::= a.ABe
A::=.Abc
A::=.b}
I3:Goto(I2,A) ={S::= aA.Be
A::=A.bc
B::= .d}
I4:Goto(I2,b) ={A::=b.}
I5:Goto(I3,B) ={S::= aAB.e}
I6:Goto(I3,b) ={A::=Ab.c}
I7:Goto(I3,d) ={B::= d.}
I8:Goto(I5,e) ={S::= aABe.}
I9:Goto(I6,c) ={A::=Abc.}

Example

The automaton of VP_G: Final States and Handles

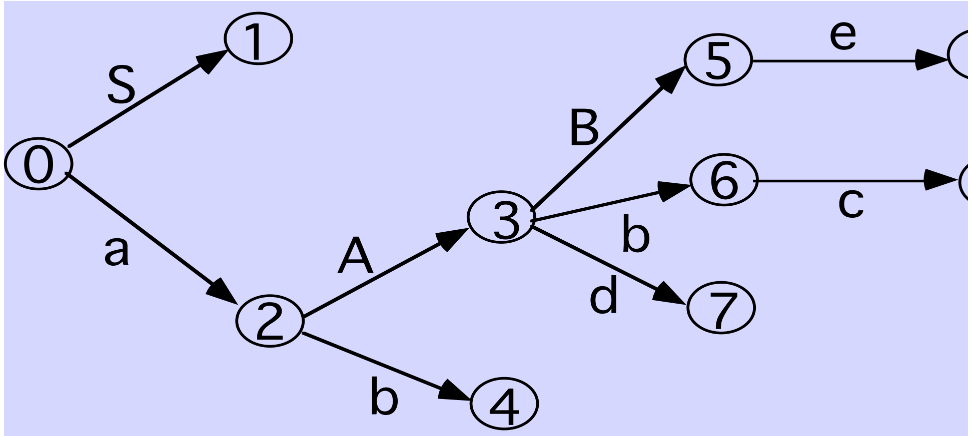
LR(0) Collection

I0:Closure({S'::=.S}) ={S'::=.S, S::= .aABe}
I1:Goto(I0,S) ={S'::=S.}
I2:Goto(I0,a) ={S::= a.Abe, A::=.Abc, A::=.b}
I3:Goto(I2,A) ={S::= aA.Be, A::=A.bc, B::= .d}
I4:Goto(I2,b) ={A::=b.}
I5:Goto(I3,B) ={S::= aAB.e}
I6:Goto(I3,b) ={A::=Ab.c}
I7:Goto(I3,d) ={B::= d.}
I8:Goto(I5,e) ={S::= aABe.}
I9:Goto(I6,c) ={A::=Abc.}

Augmented G'

S'::=S
S::= aABe
A::= Abc | b
B::= d

VP_G Automaton, A_{VP}



- Q: How a A_{VP} recognizes viable prefixes?
  - A: The set VP_G of the Viable Prefixes of the handle of G is just the language of A_{VP}, i.e. $L(A_{VP}) = VP_G$
- Q: Which are the final states of A_{VP}?
  - A: Each state is a final state. In particular, I0 is final for the prefix λ.
- Q: Which of the following is not a prefix: a) λ, b) a, c) ab, d) abb?
  - A: abb is not.
- Q: When we cannot continue to traversing the automaton:
  - -(1) we are on the right of the handle?
  - -(2) we have just passed the end of a prefix whose terminal trait is or is not the string handle depending from the symbol following the prefix
  - A: Answer can be obtained considering what happens in the following cases: a) ac, b) abd, c) abb.

Exercise: Use all the tools in this slide, and the answers to the questions above, to show the rightmost derivation of the string: abbcde

ab bcde$ <= aAbc de$ <= aAd e$ <= aABe $ <= S $ <= S'$

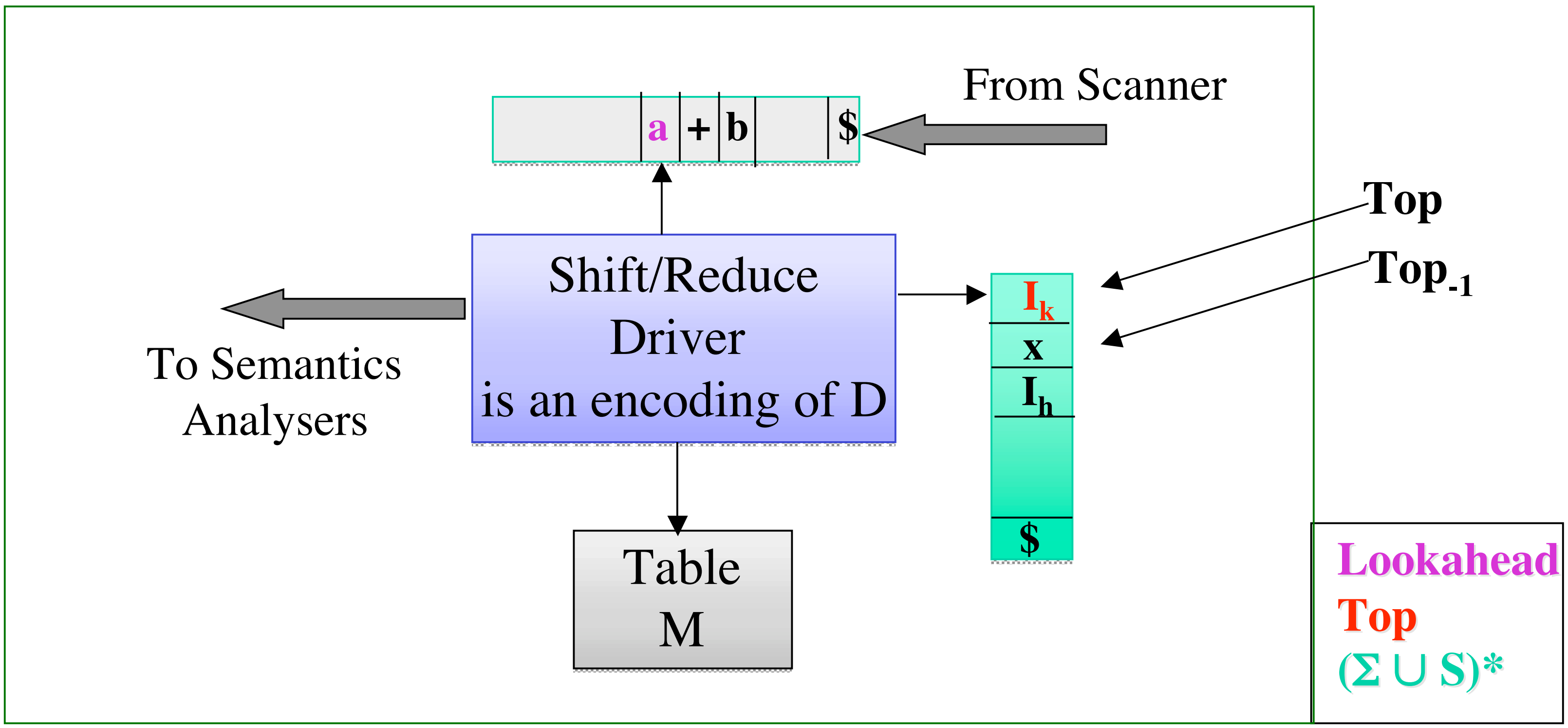
Push-Down Automata

are the perfect supports for SLR parsers

A **Pushdown Automaton** extends FSA and can be defined by the 6-tuple below:

$$<S,\ \Sigma,\ M: S \times \Sigma \dashrightarrow S,\ D: M \times (\Sigma \cup S)^* \dashrightarrow (\Sigma \cup S)^*,\ s_0 \in S,\ F \subseteq S>$$

where **S, Σ, M, s_0, F** are the same of FSA, while **(Σ ∪ S)*** is a stack.



Push-Down Automata

The definition of D for SLR(1) grammars

The function D for SLR(1)

- **Shift(k) =** push(lookahead); push(I_k)
- **Reduce(A::= α) =** popn(2*|α|);
  push(A);
  push(Goto(Top_{-1},A));

where **M≡<Action,Goto>**

SLR Parsing
The Table M (Action-Goto) for SLR

Let G=<S,V,Π,s> and I be the set of LR(0) Collection of G.

Table M (also, called, Action-Goto) consists of two Tables:

Action: is **|I|-rows** by **|V∪{$}|-columns**

Goto: is **|I|-rows** by **|S|-columns**

For each state, of **I**:

Action: The operation to apply (shift or reduce)

Goto: The state to go after reduction

The definition of M requires the computation of:

- LR(0) Collection
- Follow(s), for each s∈S

SLR Parsing
The Table Action for SLR(1)

Component, Action, of table M for SLR(1)

ACTION(i,a)=s/j

if goto(Ii,a)=Ij and a∈Σ

ACTION(i,a)= r/p

if A::=α.∈Ii and p≡ A::= α and a ∈follow(A)

ACTION(i,$)= <accept,->

if S'::=S.∈ Ii

Where: s/j = shortening for Shift(Ij);
r/p = shortening for Reduce(p)

SLR Parsing
The Table Goto for SLR(1)

Let G=<S,V,Π,s>

GOTO(i,A)=j
if goto(Ii,A)=Ij and A ∈S

Example
Table Action-Goto SLR(1)

	a	b	c	d	e	$
0	S/2					
1						ACC
2		S/4				
3		S/6		S/7		
4		R/2		R/2		
5					S/8	
6			S/9			
7					R/3	
8						R/0
9		R/1		R/1		

	S	A	B
0	1		
1			
2		3	
3			5
4			
5			
6			
7			
8			
9			

LR(0) Collection

- **I0:Closure({S'::=.S}) ={S'::=.S, S::= .aABe}**
- **I1:Goto(I0,S) ={S'::=S.}**
- **I2:Goto(I0,a) ={S::= a.Abe, A::=.Abc, A::=.b}**
- **I3:Goto(I2,A) ={S::= aA.Be, A::=A.bc, B::= .d}**
- **I4:Goto(I2,b) ={A::=b.}**
- **I5:Goto(I3,B) ={S::= aAB.e}**
- **I6:Goto(I3,b) ={A::=Ab.c}**
- **I7:Goto(I3,d) ={B::= d.}**
- **I8:Goto(I5,e) ={S::= aABe.}**
- **I9:Goto(I6,c) ={A::=Abc.}**

Source Grammar

	S'::=S
0	S::= aABe
1|2	A::= Abc | b
3	B::= d

Augmented G'

	S'::=S
0	S::= aABe
1|2	A::= Abc | b
3	B::= d

Table of Follow

follow(S)={$}
follow(A)={b,d}
follow(B)={e}

DRIVER: shift-reduce