RUTGERS
THE STATE UNIVERSITY OF NEW JERSEY

CS415 Compilers
Syntax Analysis
Bottom-up Parsing

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University

Review: LR(k) Shift-reduce Parsing

Shift reduce parsers are easily built and easily understood

A shift-reduce parser has just four actions

- *Shift* — next word is shifted onto the stack
- *Reduce* — right end of handle is at top of stack
  - Locate left end of handle within the stack
  - Pop handle off stack & push appropriate *lhs*
- *Accept* — stop parsing & report success
- *Error* — call an error reporting/recovery routine

Accept & Error are simple

Shift is just a push and a call to the scanner

Reduce takes |*rhs*| pops (or 2*|rhs| pops) & 1 push

If handle-finding requires state, put it in the stack ⇒ 2x work

Review - LR(k) items

The LR(1) table construction algorithm uses LR(1) items to represent valid configurations of an LR(1) parser

An LR(k) item is a pair $[P, \delta]$, where

- P is a production $A \rightarrow \beta$ with a • at some position in the *rhs*
- δ is a lookahead string of length ≤ k (words or EOF)

The • in an item indicates the position of the top of the stack

LR(1):

$[A \rightarrow \bullet\beta\gamma, \underline{a}]$ means that the input seen so far is consistent with the use of $A \rightarrow \beta\gamma$ immediately after the symbol on top of the stack

$[A \rightarrow \beta\bullet\gamma, \underline{a}]$ means that the input seen so far is consistent with the use of $A \rightarrow \beta\gamma$ at this point in the parse, *and* that the parser has already recognized β.

$[A \rightarrow \beta\gamma\bullet, \underline{a}]$ means that the parser has seen $\beta\gamma$, *and* that a lookahead symbol of $\underline{a}$ is consistent with reducing to A.

Review – LR(1) Table Construction

High-level overview

1 Build the canonical collection of sets of LR(k) Items, *I*

a Begin in an appropriate state, s_0

- Assume: $S' \rightarrow S$, and S' is unique start symbol that does not occur on any RHS of a production (extended CFG - ECFG)
- [$S' \rightarrow \cdot S$,EOF], along with any equivalent items
- Derive equivalent items as *closure*(s_0)

b Repeatedly compute, for each s_k, and each X, *goto*(s_k,X)

- If the set is not already in the collection, add it
- Record all the transitions created by *goto*( )

This eventually reaches a fixed point

2 Fill in the table from the collection of sets of LR(1) items

The canonical collection completely encodes the transition diagram for the handle-finding DFA

Review - Computing Closures

Closure(s) adds all the items implied by items already in *s*

- Any item $[A \rightarrow \beta \bullet \underline{B}\delta, a]$ implies $[B \rightarrow \bullet \tau, x]$ for each production with *B* on the *lhs*, and each $x \in \text{FIRST}(\delta \underline{a})$ – for LR(1) item

The algorithm

Closure(s)
  while (s is still changing)
    $\forall$ *items* $[A \rightarrow \beta \cdot B\delta, \underline{a}] \in s$
      $\forall$ *productions* $B \rightarrow \tau \in P$
        $\forall\ \underline{b} \in \text{FIRST}(\delta \underline{a})$ // δ might be ε
          if $[B \rightarrow \cdot\ \tau, \underline{b}] \notin s$
            then add $[B \rightarrow \cdot\ \tau, \underline{b}]$ *to s*

- Classic fixed-point method
- Halts because $s \subset$ ITEMS
- *Closure "fills out" a state*

Review - Computing Gotos

Goto(s,x) computes the state that the parser would reach if it recognized an *x* while in state *s*

- *Goto(* { [$A \to \beta \bullet X \delta, \underline{a}$] }, *X)* produces [$A \to \beta X \bullet \delta, \underline{a}$] *(easy part)*
- Should also includes *closure(* [$A \to \beta X \bullet \delta, \underline{a}$] *)* *(fill out the state)*

The algorithm

Goto(s, X)
  new $\leftarrow \emptyset$
  $\forall$ *items* [$A \to \beta \cdot X \delta, \underline{a}$] $\in s$
    new $\leftarrow$ *new* $\cup$ [$A \to \beta X \cdot \delta, \underline{a}$]
  return closure(new)

- Not a fixed-point method!
- Straightforward computation
- Uses *closure*( )

Goto() moves forward

Review - Building the Canonical Collection

Start from s_0 = *closure*([*S'* →*S*,<u>EOF</u>])

Repeatedly construct new states, until all are found

The algorithm

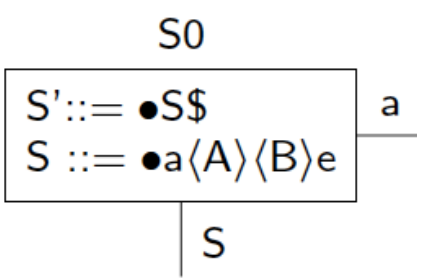
cc$_0$ ← *closure*([*S'* → •*S*, <u>EOF</u>])
CC ← { *cc*$_0$ }
while (*new sets are still being added to CC*)
  for each unmarked set cc$_j$ ∈ *CC*
    mark cc$_j$ *as processed*
    for each x following a • in an item in cc$_j$
      temp ← *goto*(*cc*$_j$, *x*)
      if temp ∉ *CC*
        then CC ← *CC* ∪ { *temp* }
      record transitions from cc$_j$ *to temp on x*

- Fixed-point computation (worklist version)
- Loop adds to *CC*
- $CC \subseteq 2^{ITEMS}$, so *CC* is finite

Construct LR(0) States

1

1	⟨S⟩::=a ⟨A⟩ ⟨B⟩ e
2	⟨A⟩::=⟨A⟩ b c
3	⟨A⟩::=b
4	⟨B⟩::=d

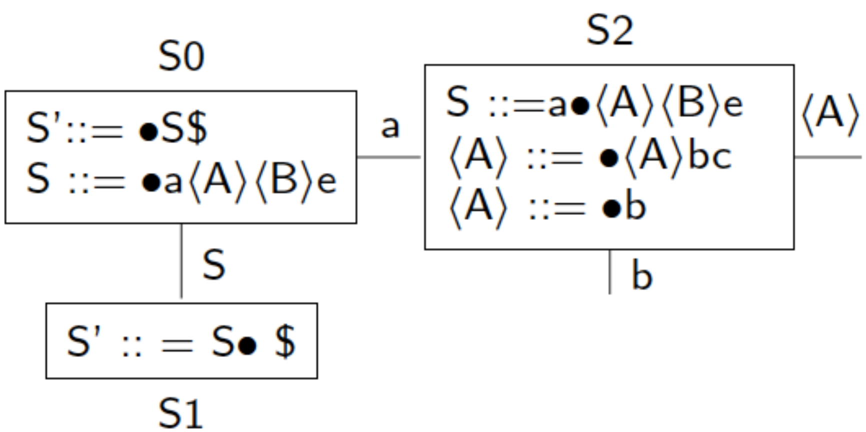


One Example

Construct LR(0) States

1

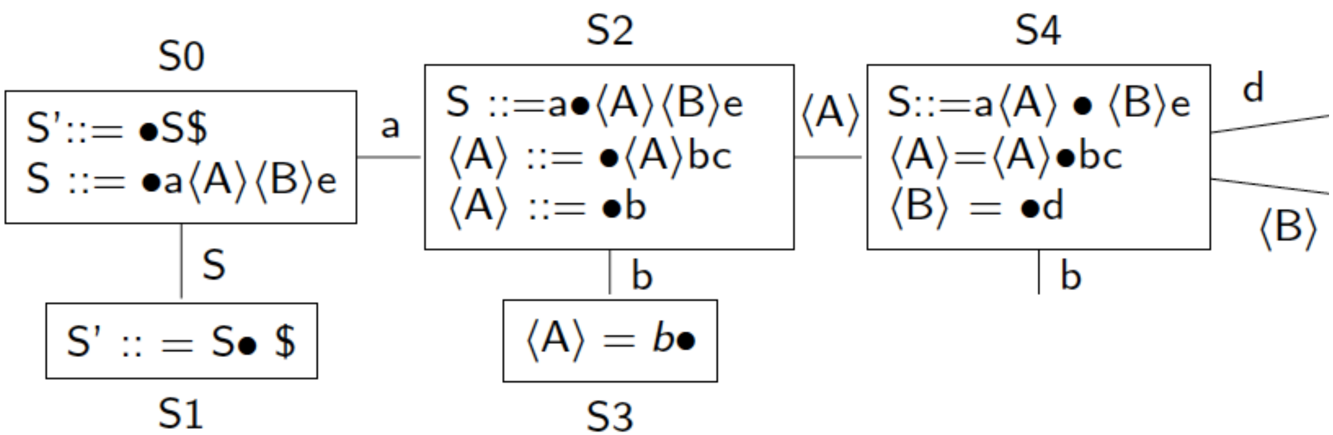
1	⟨S⟩::=a ⟨A⟩ ⟨B⟩ e
2	⟨A⟩::=⟨A⟩ b c
3	⟨A⟩::=b
4	⟨B⟩::=d



Construct LR(0) States

1

1	⟨S⟩::=a ⟨A⟩ ⟨B⟩ e
2	⟨A⟩::=⟨A⟩ b c
3	⟨A⟩::=b
4	⟨B⟩::=d

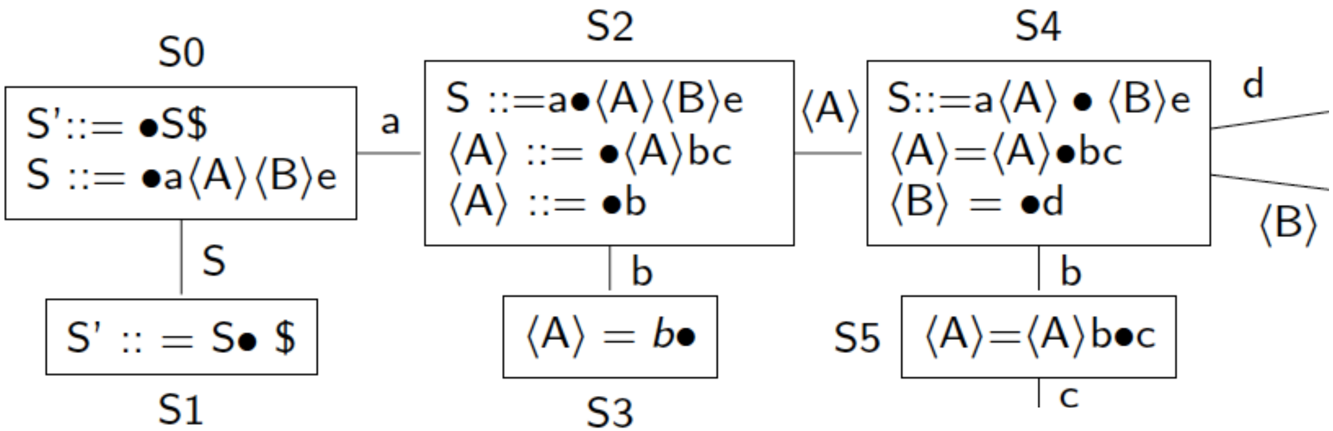


One Example

Construct LR(0) States

1

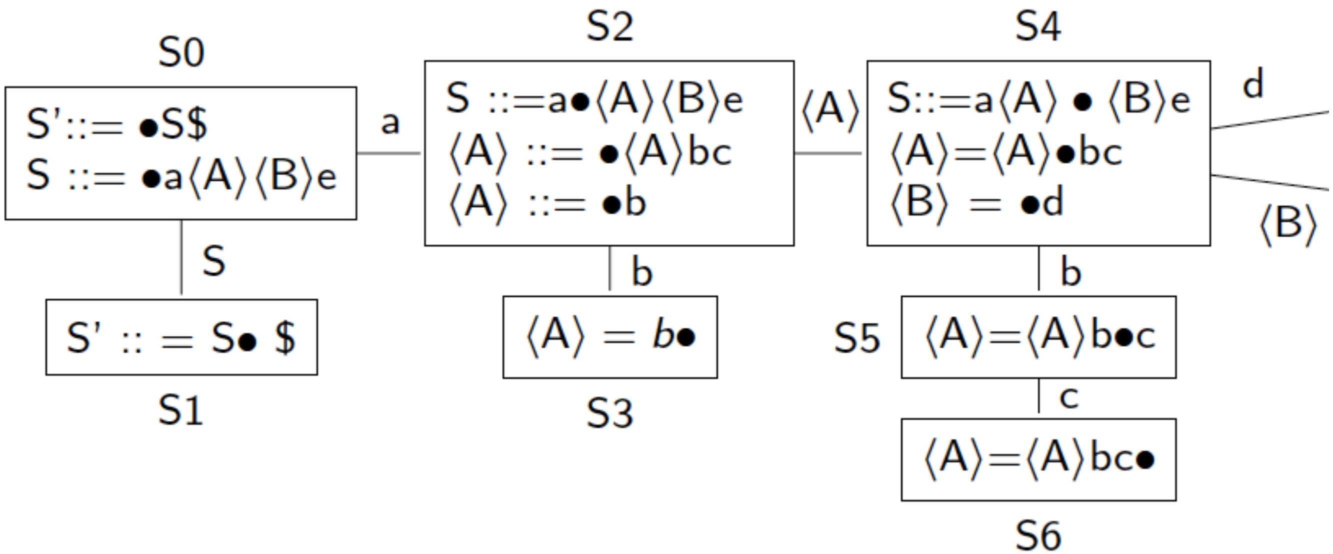
1	⟨S⟩::=a ⟨A⟩ ⟨B⟩ e
2	⟨A⟩::=⟨A⟩ b c
3	⟨A⟩::=b
4	⟨B⟩::=d



Construct LR(0) States

1

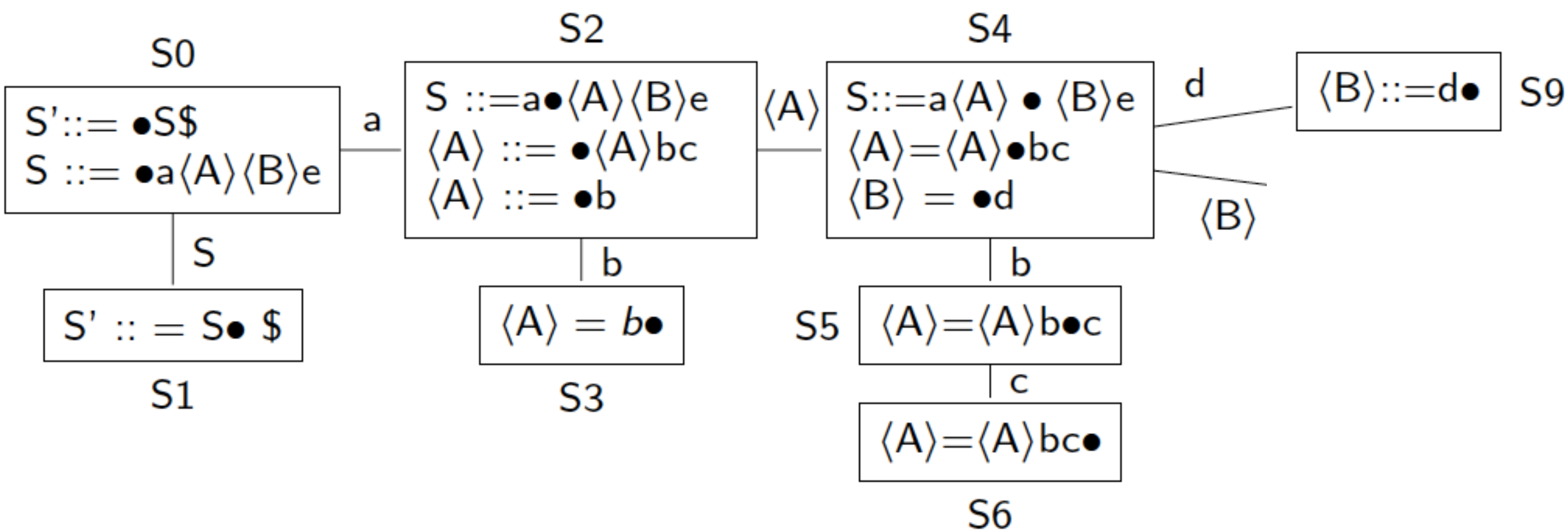
1	⟨S⟩::=a ⟨A⟩ ⟨B⟩ e
2	⟨A⟩::=⟨A⟩ b c
3	⟨A⟩::=b
4	⟨B⟩::=d



Construct LR(0) States

1

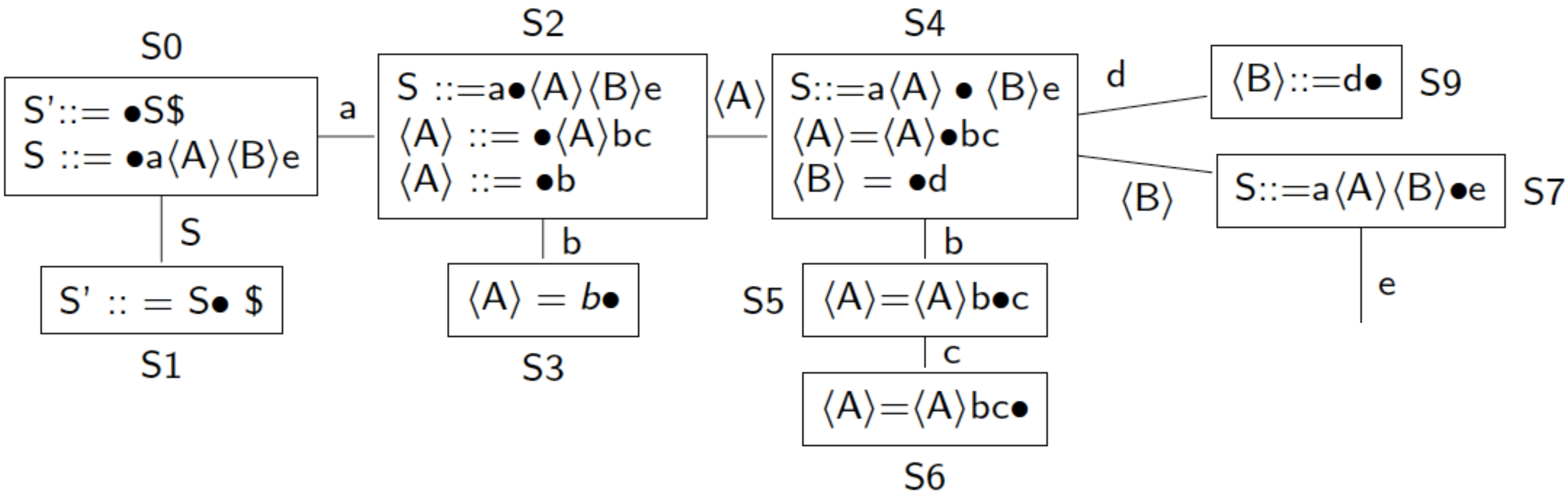
1	⟨S⟩::=a ⟨A⟩ ⟨B⟩ e
2	⟨A⟩::=⟨A⟩ b c
3	⟨A⟩::=b
4	⟨B⟩::=d



One Example

Construct LR(0) States 1

1	⟨S⟩::=a ⟨A⟩ ⟨B⟩ e
2	⟨A⟩::=⟨A⟩ b c
3	⟨A⟩::=b
4	⟨B⟩::=d



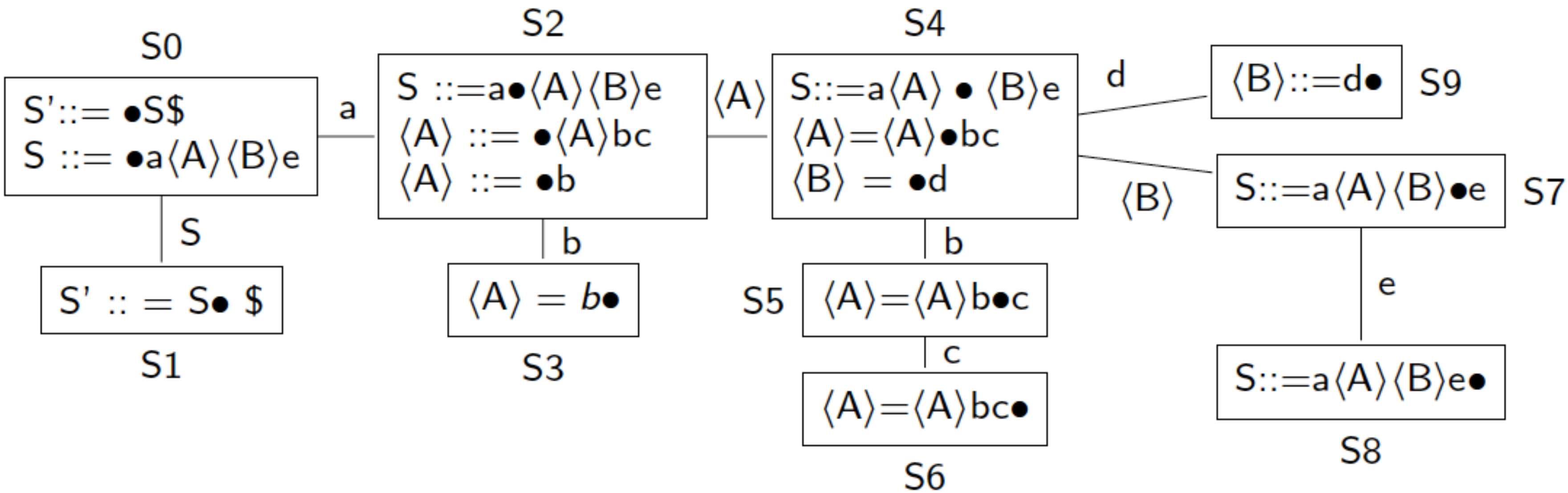
RUTGERS

One Example - LR(0) states

Construct LR(0) States

1

1	⟨S⟩::=a ⟨A⟩ ⟨B⟩ e
2	⟨A⟩::=⟨A⟩ b c
3	⟨A⟩::=b
4	⟨B⟩::=d



Another Example - LR(1) states

Simplified, right recursive expression grammar

1: *Goal* → *Expr*
2: *Expr* → *Term* - *Expr*
3: *Expr* → *Term*
4: *Term* → *Factor* * *Term*
5: *Term* → *Factor*
6: *Factor* → ident

Symbol	FIRST
Goal	{ ident }
Expr	{ ident }
Term	{ ident }
Factor	{ ident }
-	{ - }
*	{ * }
ident	{ ident }

Another Example (*building the collection*)

1: *Goal* → *Expr*
2: *Expr* → *Term* – *Expr*
3: *Expr* → *Term*
4: *Term* → *Factor* * *Term*
5: *Term* → *Factor*
6: *Factor* → ident

Symbol	FIRST
Goal	{ ident }
Expr	{ ident }
Term	{ ident }
Factor	{ ident }
–	{ – }
*	{ * }
ident	{ ident }

Initialization Step

$s_0 \leftarrow$ *closure*({ [*Goal* → •*Expr* , EOF] }) =

{ [Expr → • Term – Expr, EOF], [Expr → • Term, EOF],
[Term → • Factor * Term, -], [Term → • Factor, -], [Term → • Factor * Term, EOF], [Term → • Factor, EOF],
[Factor → •ident, *], [Factor → •ident, -], [Factor → •ident, EOF]}

S $\leftarrow \{ S_0 \}$

Example (building the collection)

$s_0 \leftarrow$ closure({ [Goal → •Expr , EOF] })

{ [Goal → • Expr , EOF], [Expr → • Term – Expr , EOF],
[Expr → • Term , EOF], [Term → • Factor * Term , EOF],
[Term → • Factor * Term , –], [Term → • Factor , EOF],
[Term → • Factor , –], [Factor → • ident , EOF],
[Factor → • ident , –], [Factor → • ident , *] }

Iteration 1

$s_1 \leftarrow goto(s_0 , Expr)$

$s_2 \leftarrow goto(s_0 , Term)$

$s_3 \leftarrow goto(s_0 , Factor)$

$s_4 \leftarrow goto(s_0 ,$ ident $)$

Example (*building the collection*)

$s_0 \leftarrow closure(\{ [Goal \rightarrow \cdot Expr , EOF] \})$

{ [*Goal* → • *Expr* , EOF], [*Expr* → • *Term* - *Expr* , EOF],
[*Expr* → • *Term* , EOF], [*Term* → • *Factor* * Term , EOF],
[*Term* → • *Factor* * *Term* , -], [*Term* → • *Factor* , EOF],
[*Term* → • *Factor* , -], [*Factor* → • ident , EOF],
[*Factor* → • ident , -], [*Factor* → • ident , *] }

Iteration 1

$s_1 \leftarrow goto(s_0 , Expr)$ = { [*Goal* → *Expr* •, EOF] }

$s_2 \leftarrow goto(s_0 , Term)$ = { [*Expr* → *Term* • - *Expr* , EOF], [*Expr* → *Term* •, EOF] }

$s_3 \leftarrow goto(s_0 , Factor)$ = { [*Term* → *Factor* • * *Term* , EOF],[*Term* → *Factor* • * *Term* , -], [*Term* → *Factor* •, EOF], [*Term* → *Factor* •, -] }

$s_4 \leftarrow goto(s_0 , ident)$ = { [*Factor* → ident •, EOF],[*Factor* → ident •, -], [*Factor* → ident •, *] }

Example (*building the collection*)

Iteration 1

$s_1 \leftarrow goto(s_0, Expr)$ = { [*Goal* → *Expr* •, EOF] }

$s_2 \leftarrow goto(s_0, Term)$ = { [*Expr* → *Term* • - *Expr* , EOF], [*Expr* → *Term* •, EOF] }

$s_3 \leftarrow goto(s_0, Factor)$ = { [*Term* → *Factor* • * *Term* , EOF],[*Term* → *Factor* • * *Term* , -], [*Term* → *Factor* •, EOF], [*Term* → *Factor* •, -] }

$s_4 \leftarrow goto(s_0,$ <u>ident</u> $)$ = { [*Factor* → <u>ident</u> •, EOF],[*Factor* → <u>ident</u> •, -], [*Factor* → <u>ident</u> •, *] }

Iteration 2

$s_5 \leftarrow goto(s_2, -)$

$s_6 \leftarrow goto(s_3, *)$

Example (*building the collection*)

Iteration 1

$s_1 \leftarrow goto(s_0, Expr)$ = { [*Goal* → *Expr* •, EOF] }

$s_2 \leftarrow goto(s_0, Term)$ = { [*Expr* → *Term* • - *Expr* , EOF], [*Expr* → *Term* •, EOF] }

$s_3 \leftarrow goto(s_0, Factor)$ = { [*Term* → *Factor* • * *Term* , EOF],[*Term* → *Factor* • * *Term* , -], [*Term* → *Factor* •, EOF], [*Term* → *Factor* •, -] }

$s_4 \leftarrow goto(s_0,$ <u>ident</u> $)$ = { [*Factor* → <u>ident</u> •, EOF],[*Factor* → <u>ident</u> •, -], [*Factor* → <u>ident</u> •, *] }

Iteration 2

$s_5 \leftarrow goto(s_2, -)$ = { [*Expr* → *Term* - • *Expr* , EOF], [*Expr* → • *Term* - *Expr* , EOF], [*Expr* → • *Term* , EOF], [*Term* → • *Factor* * *Term* , -], [*Term* → • *Factor* , -], [*Term* → • *Factor* * *Term* , EOF], [*Term* → • *Factor* , EOF], [*Factor* → • <u>ident</u> , *], [*Factor* → • <u>ident</u> , -], [*Factor* → • <u>ident</u> , EOF] }

$s_6 \leftarrow goto(s_3, *)$ = … *see next page*

Example (*building the collection*)

Iteration 2

$s_5 \leftarrow goto(s_2, -) = \{ [Expr \rightarrow Term - \cdot Expr, \text{EOF}], [Expr \rightarrow \cdot Term - Expr, \text{EOF}], [Expr \rightarrow \cdot Term, \text{EOF}], [Term \rightarrow \cdot Factor * Term, -], [Term \rightarrow \cdot Factor * Term, \text{EOF}], [Term \rightarrow \cdot Factor, -], [Term \rightarrow \cdot Factor, \text{EOF}], [Factor \rightarrow \cdot \underline{ident}, *], [Factor \rightarrow \cdot \underline{ident}, -], [Factor \rightarrow \cdot \underline{ident}, \text{EOF}] \}$

$s_6 \leftarrow goto(s_3, *) = \{ [Term \rightarrow Factor * \cdot Term, \text{EOF}], [Term \rightarrow Factor * \cdot Term, -], [Term \rightarrow \cdot Factor * Term, \text{EOF}], [Term \rightarrow \cdot Factor * Term, -], [Term \rightarrow \cdot Factor, \text{EOF}], [Term \rightarrow \cdot Factor, -], [Factor \rightarrow \cdot \underline{ident}, \text{EOF}], [Factor \rightarrow \cdot \underline{ident}, -], [Factor \rightarrow \cdot \underline{ident}, *] \}$

Iteration 3

$s_7 \leftarrow goto(s_5, Expr) = \{ [Expr \rightarrow Term - Expr \cdot, \text{EOF}] \}$

$s_8 \leftarrow goto(s_6, Term) = \{ [Term \rightarrow Factor * Term \cdot, \text{EOF}], [Term \rightarrow Factor * Term \cdot, -] \}$

$goto(s_5, Term) = S_2$, $goto(s_5, factor) = s_3$, $goto(S_5, ident) = s_4$

$goto(s_6, Factor) = s_3$, $goto(S_6, ident) = s_4$

Example (*Summary*)

S_0 : { [*Goal* → • *Expr* , EOF], [*Expr* → • *Term* - *Expr* , EOF],
[*Expr* → • *Term* , EOF], [*Term* → • *Factor* * *Term* , EOF],
[*Term* → • *Factor* * *Term* , -], [*Term* → • *Factor* , EOF],
[*Term* → • *Factor* , -], [*Factor* → • <u>ident</u> , EOF],
[*Factor* → • <u>ident</u> , -], [*Factor*→ • <u>ident</u>, *] }

S_1 : { [*Goal* → *Expr* •, EOF] }

S_2 : { [*Expr* → *Term* • - *Expr* , EOF], [*Expr* → *Term* •, EOF] }

S_3 : { [*Term* → *Factor* • * *Term* , EOF],[*Term* → *Factor* • * *Term* , -],
[*Term* → *Factor* •, EOF], [*Term* → *Factor* •, -] }

S_4 : { [*Factor* → <u>ident</u> •, EOF],[*Factor* → <u>ident</u> •, -], [*Factor* → <u>ident</u> •, *] }

S_5 : { [*Expr* → *Term* - • *Expr* , EOF], [*Expr* → • *Term* - *Expr* , EOF],
[*Expr* → • *Term* , EOF], [*Term* → • *Factor* * *Term* , -],
[*Term* → • *Factor* , -], [*Term* → • *Factor* * *Term* , EOF],
[*Term* → • *Factor* , EOF], [*Factor* → • <u>ident</u> , *],
[*Factor* → • <u>ident</u> , -], [*Factor* → • <u>ident</u> , EOF] }

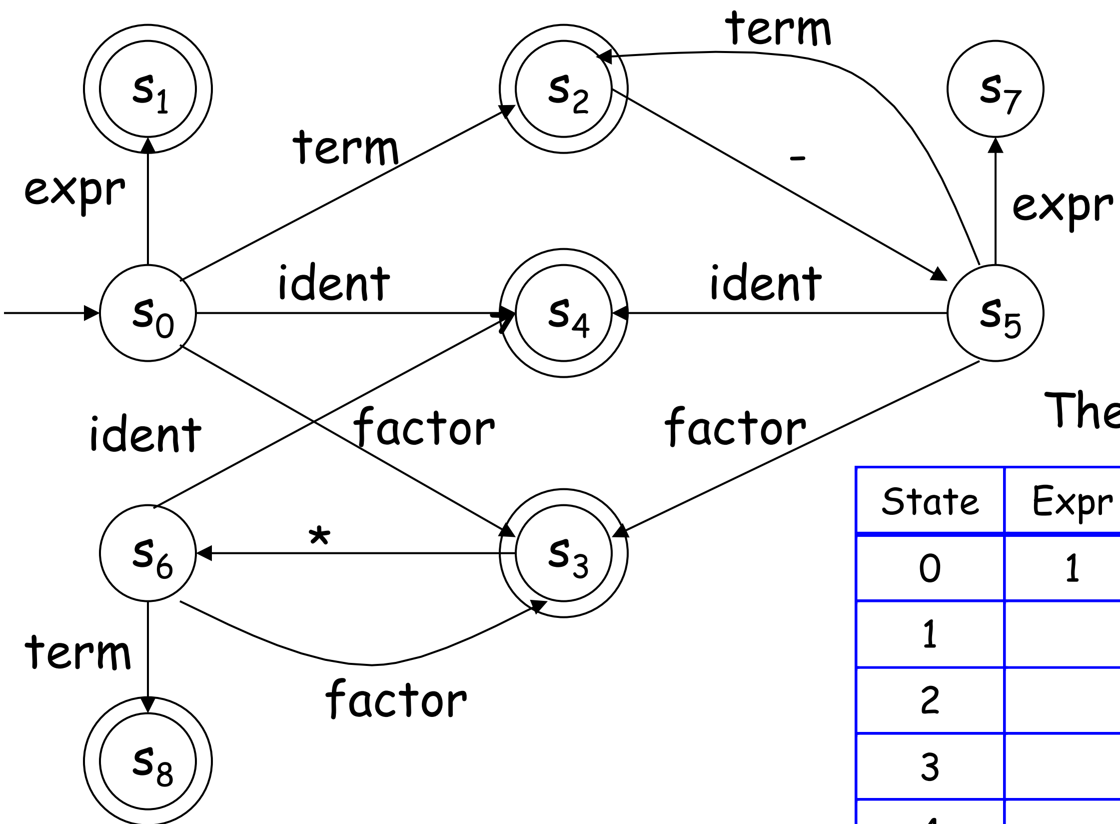
Example (*Summary*)

S_6 : { [*Term* → *Factor* * • *Term* , EOF], [*Term* → *Factor* * • *Term* , –],
[*Term* → • *Factor* * *Term* , EOF], [*Term* → • *Factor* * *Term* , –],
[*Term* → • *Factor* , EOF], [*Term* → • *Factor* , –],
[*Factor* → • ident , EOF], [*Factor* → • ident , –], [*Factor* → • ident , *] }

S_7: { [*Expr* → *Term* – *Expr* •, EOF] }

S_8 : { [*Term* → *Factor* * *Term* •, EOF], [*Term* → *Factor* * *Term* •, –] }

Example (DFA)



The State Transition Table

State	Expr	Term	Factor	-	*	Ident
0	1	2	3			4
1						
2				5		
3					6	
4						
5	7	2	3			4
6		8	3			4
7						
8						

Filling in the ACTION and GOTO Tables

The algorithm

$\forall$ *set* $s_x \in S$
  $\forall$ *item* $i \in s_x$
    if i *is* $[A \to \beta \bullet \underline{a}d, \underline{b}]$ *and* $goto(s_x, \underline{a}) = s_k$, $\underline{a} \in T$
      then ACTION$[x, \underline{a}] \leftarrow$ "*shift k*"
    else if i *is* $[S' \to S \bullet, \text{EOF}]$
      then ACTION$[x, \text{EOF}] \leftarrow$ "*accept*"
    else if i *is* $[A \to \beta \bullet, \underline{a}]$
      then ACTION$[x, \underline{a}] \leftarrow$ "*reduce* $A \to \beta$"
  $\forall$ $n \in NT$
    if $goto(s_x, n) = s_k$
      then GOTO$[x, n] \leftarrow k$

Many items generate no table entry

Next class

Wrap Up Syntax Analysis

Context-Sensitive Analysis

Read EaC: Chapters 3.4, 4.1 – 4.3