6.035

Fall 2000

Lecture 5: Shift-Reduce Parsing

Implementing a Parser

- Different techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

Implementing a Parser

- Different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

Implementing a Parser

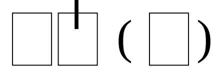
- Different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

  - **L -** parse from left to right
  - **R -** parse from right to left

Implementing a Parser

- Different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

  - **L -** leftmost derivation
  - **R -** rightmost derivation

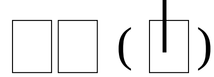


Martin Rinard 5 **6.035** ©MIT Fall 2000

Implementing a Parser

- Different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

  - Number of lookahead characters



Martin Rinard 6 **6.035** ©MIT Fall 2000

Implementing a Parser

- Different techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

  - Examples: LL(0), LR(1)
  - Classical Recursive Descent Parser is LL(1)

$\boxed{}\boxed{}\ (\boxed{})$

Implementing a Parser

- Different techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

  - Examples: LL(0), LR(1)
  - This lecture: LR(k) parsers

$\boxed{\mathbf{L}}\boxed{\mathbf{R}}\ (\boxed{\mathbf{k}})$

Shift-Reduce Parser Example

$$Expr \rightarrow Expr\ Op\ \ Expr$$
$$Expr \rightarrow (Expr)$$
$$Expr \rightarrow \text{-}\ Expr$$
$$Expr \rightarrow \text{num}$$
$$Op \rightarrow +$$
$$Op \rightarrow \text{-}$$
$$Op \rightarrow *$$

Stack

Input String

num	*	(	num	+	num	)

Shift-Reduce Parser Example

$$Expr \rightarrow Expr\ Op\ \ Expr$$
$$Expr \rightarrow (Expr)$$
$$Expr \rightarrow \text{-}\ Expr$$
$$Expr \rightarrow \text{num}$$
$$Op \rightarrow +$$
$$Op \rightarrow \text{-}$$
$$Op \rightarrow *$$

num	*	(	num	+	num	)

Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ \ Expr$

$Expr \rightarrow (Expr)$

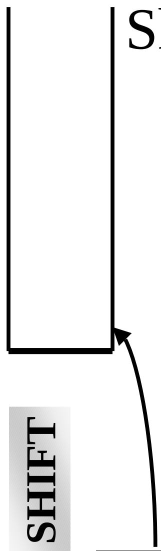
$Expr \rightarrow \text{-}\ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$



num	*	(	num	+	num	)

Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ \ Expr$

$Expr \rightarrow (Expr)$

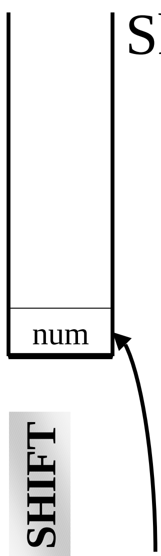
$Expr \rightarrow \text{-}\ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$



*	(	num	+	num	)

Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow -\ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

num

REDUCE

*	(	num	+	num	)

Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow -\ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Expr

REDUCE

num

*	(	num	+	num	)

Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow (Expr)$

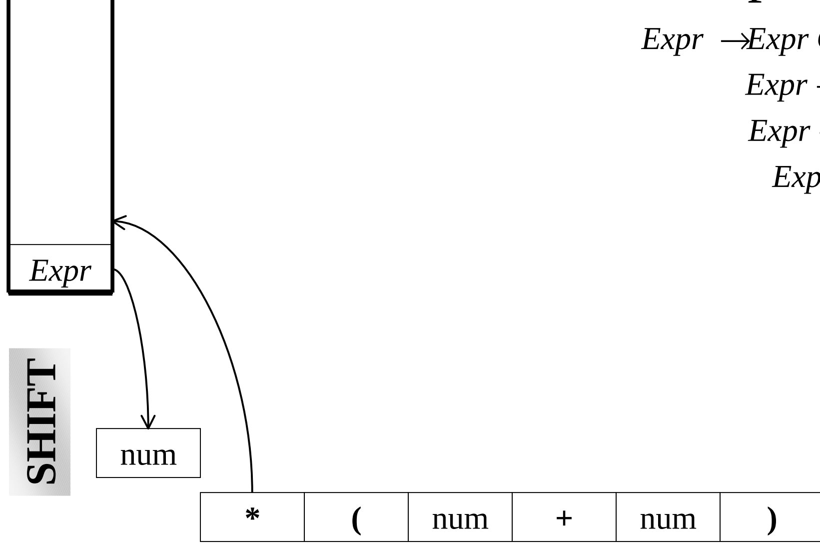
$Expr \rightarrow -\ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow (Expr)$

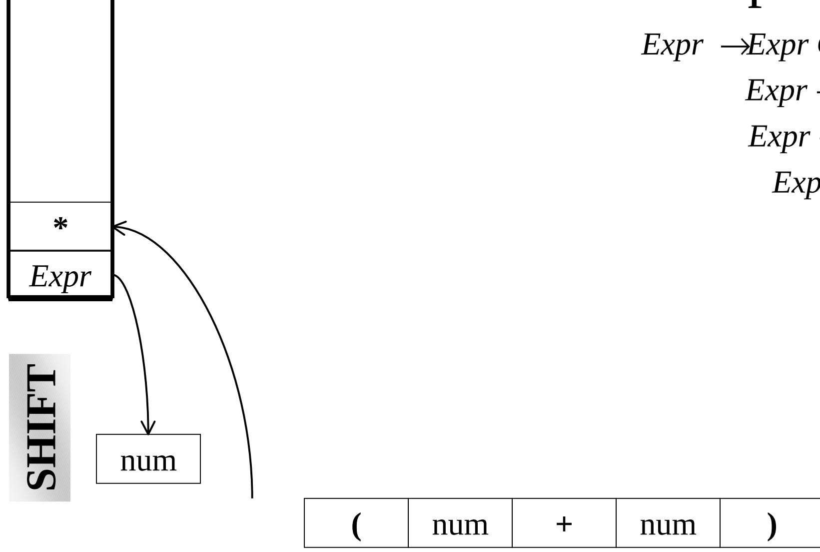
$Expr \rightarrow -\ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$



Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ \ Expr$

$Expr \rightarrow (Expr)$

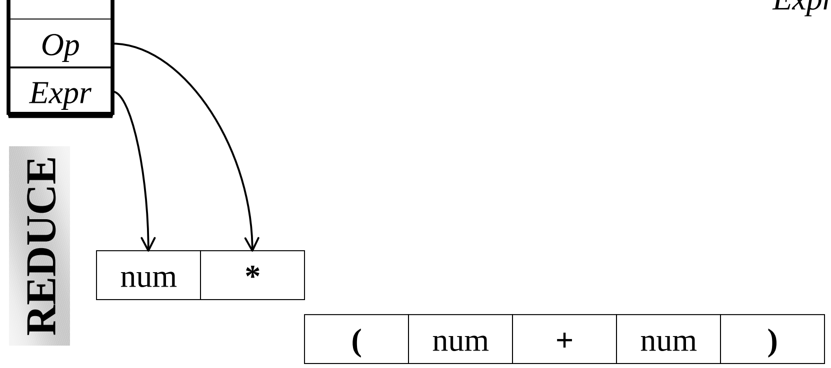
$Expr \rightarrow \text{-}\ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$



Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ \ Expr$

$Expr \rightarrow (Expr)$

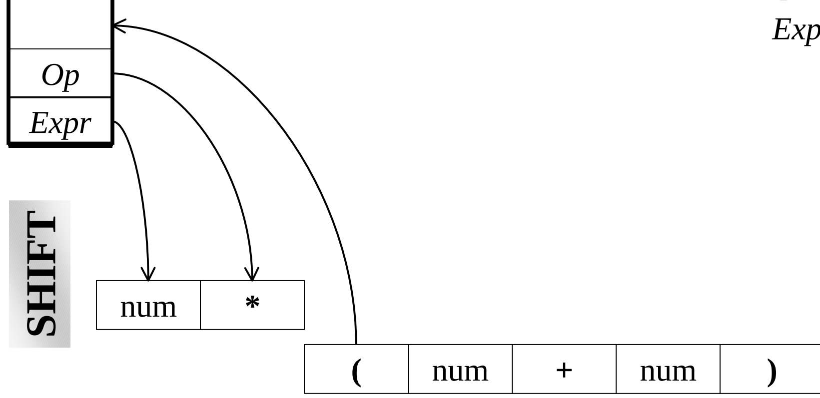
$Expr \rightarrow \text{-}\ Expr$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

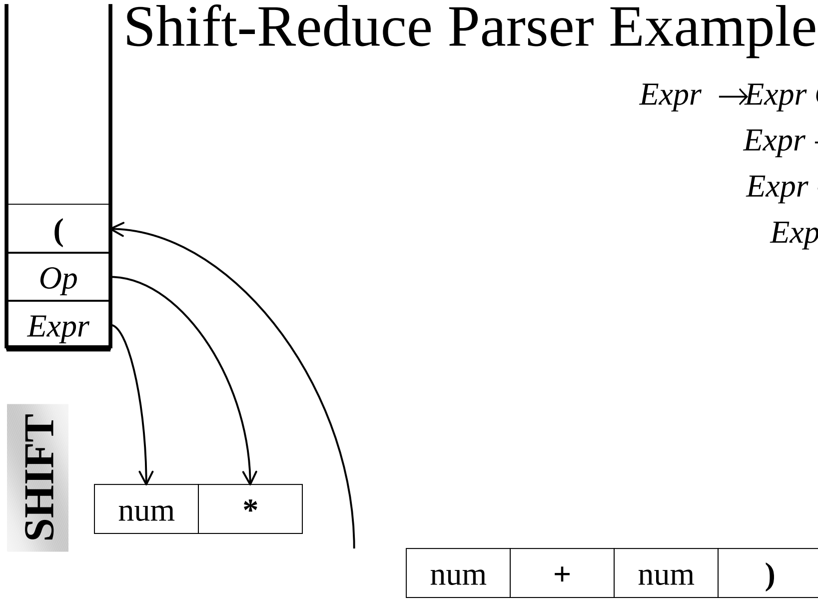
$Op \rightarrow \text{-}$

$Op \rightarrow *$



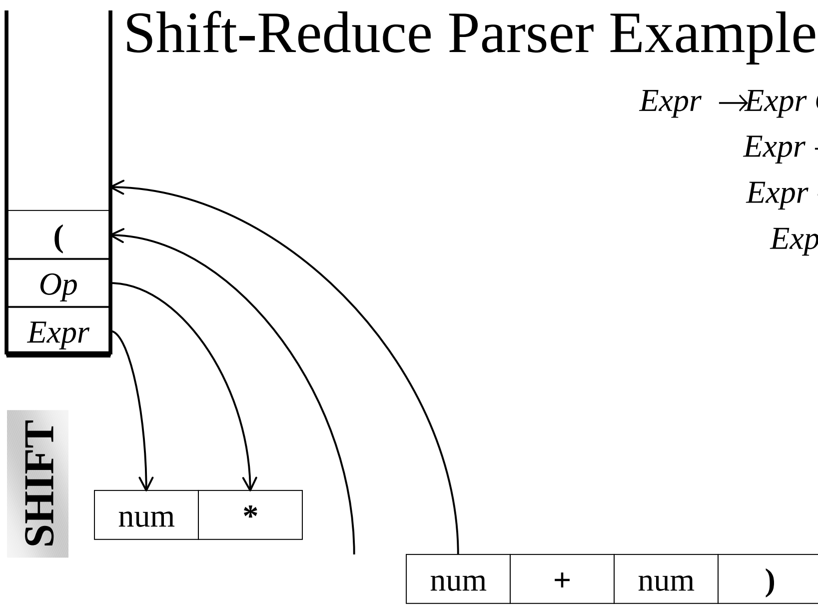
Shift-Reduce Parser Example

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *



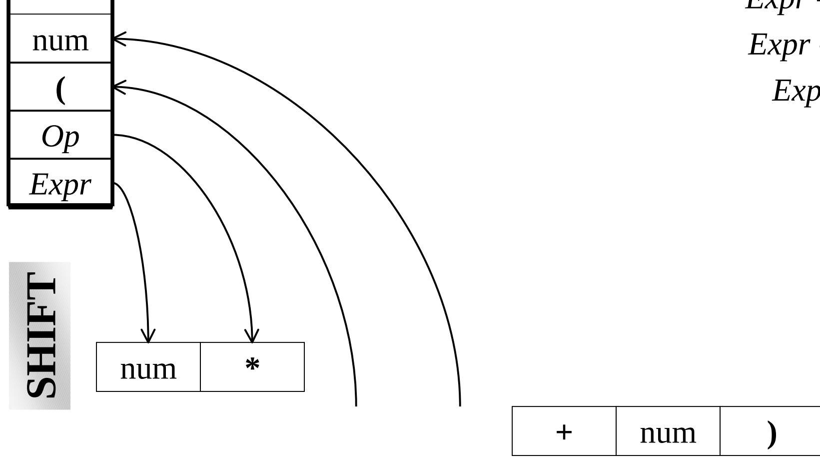
Shift-Reduce Parser Example

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *



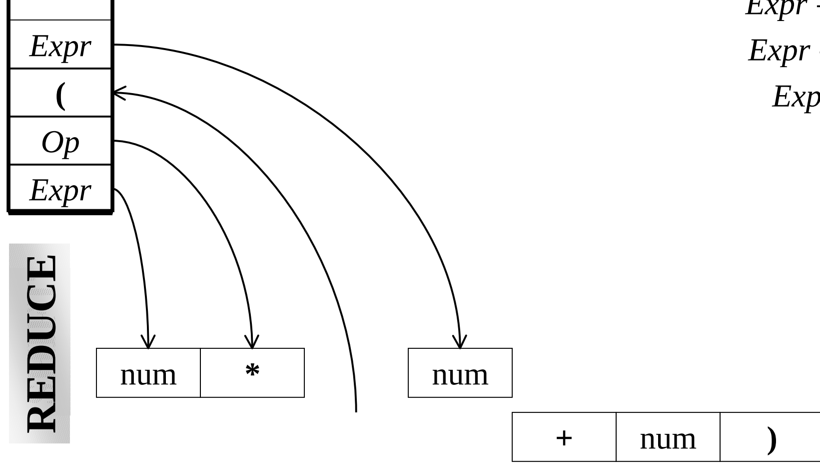
Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$
$Expr \rightarrow (Expr)$
$Expr \rightarrow \text{-}\ Expr$
$Expr \rightarrow \text{num}$
$Op \rightarrow +$
$Op \rightarrow \text{-}$
$Op \rightarrow *$



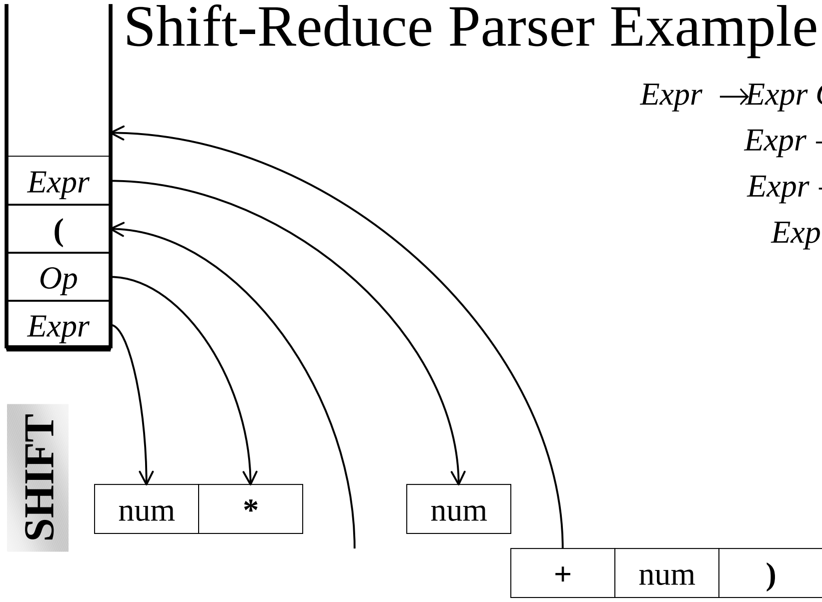
Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$
$Expr \rightarrow (Expr)$
$Expr \rightarrow \text{-}\ Expr$
$Expr \rightarrow \text{num}$
$Op \rightarrow +$
$Op \rightarrow \text{-}$
$Op \rightarrow *$



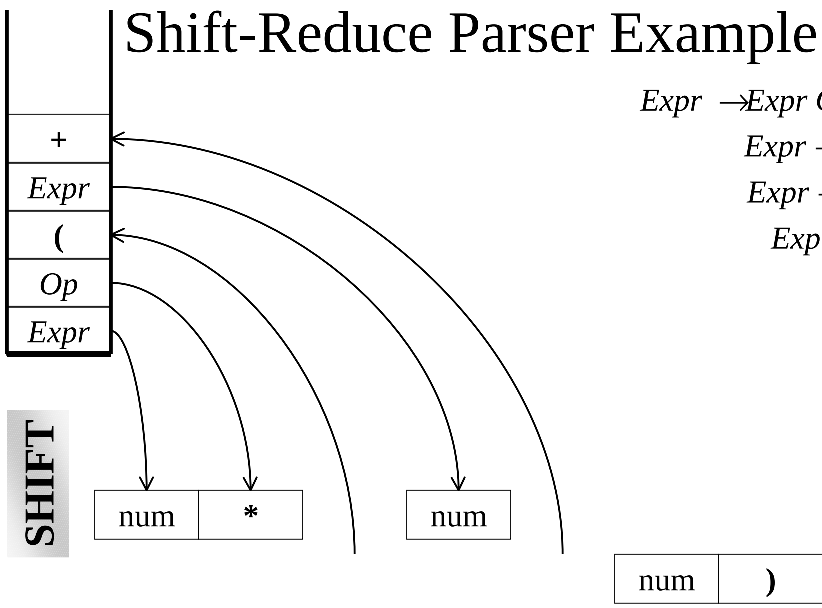
Shift-Reduce Parser Example

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *



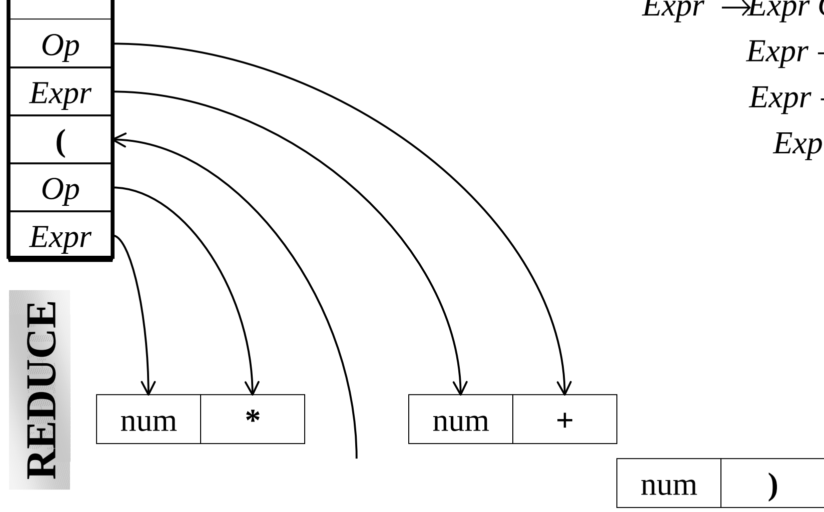
Shift-Reduce Parser Example

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *



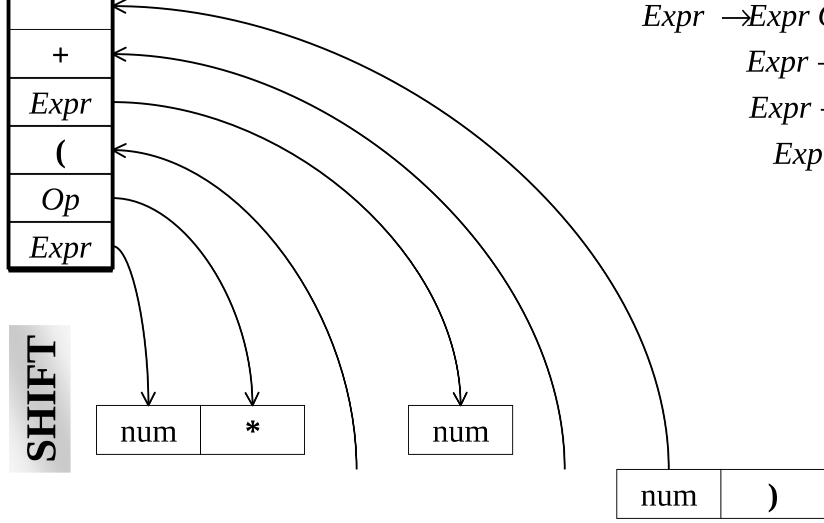
Shift-Reduce Parser Example

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *

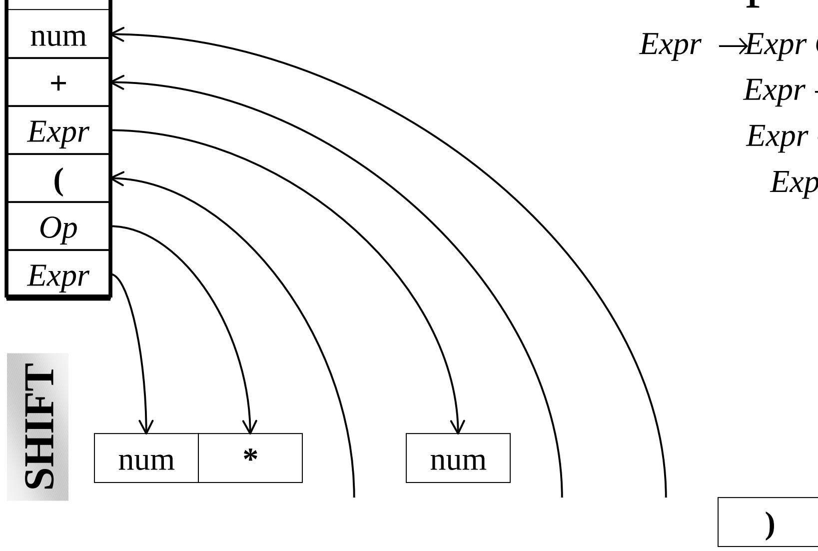


Shift-Reduce Parser Example

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *

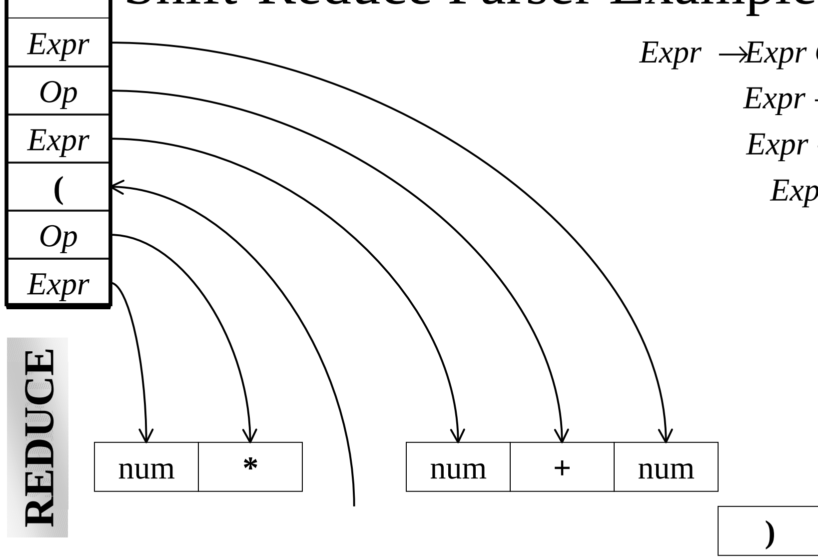


Shift-Reduce Parser Example



Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *

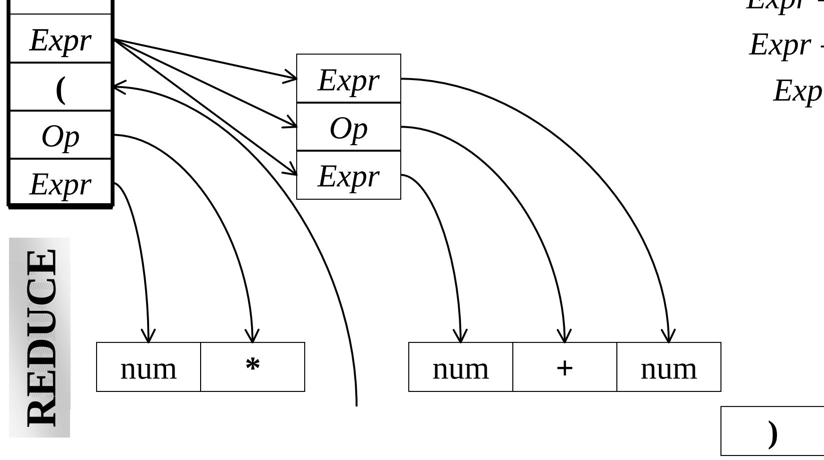
Shift-Reduce Parser Example



Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *

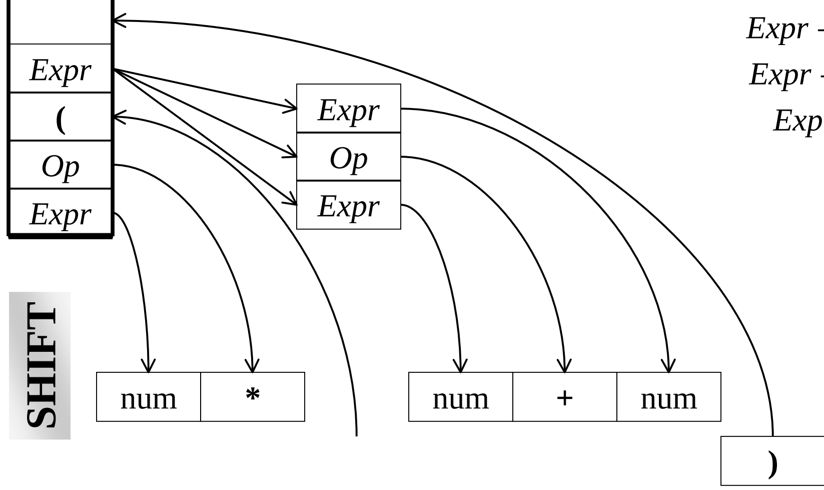
Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$
$Expr \rightarrow (Expr)$
$Expr \rightarrow -\ Expr$
$Expr \rightarrow \text{num}$
$Op \rightarrow +$
$Op \rightarrow -$
$Op \rightarrow *$



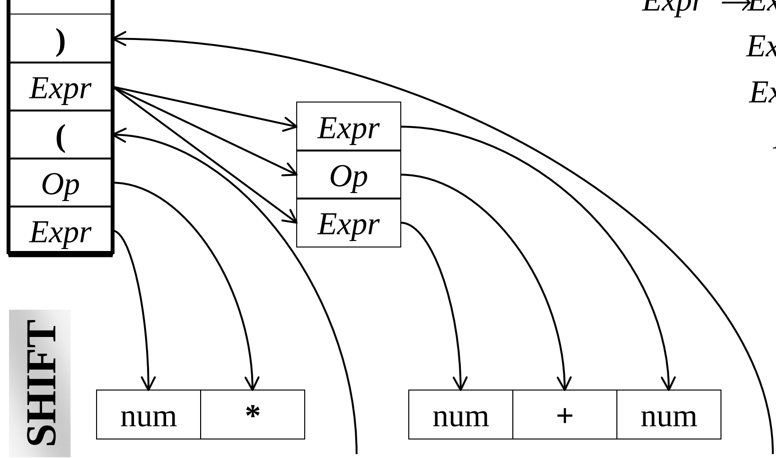
Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$
$Expr \rightarrow (Expr)$
$Expr \rightarrow -\ Expr$
$Expr \rightarrow \text{num}$
$Op \rightarrow +$
$Op \rightarrow -$
$Op \rightarrow *$



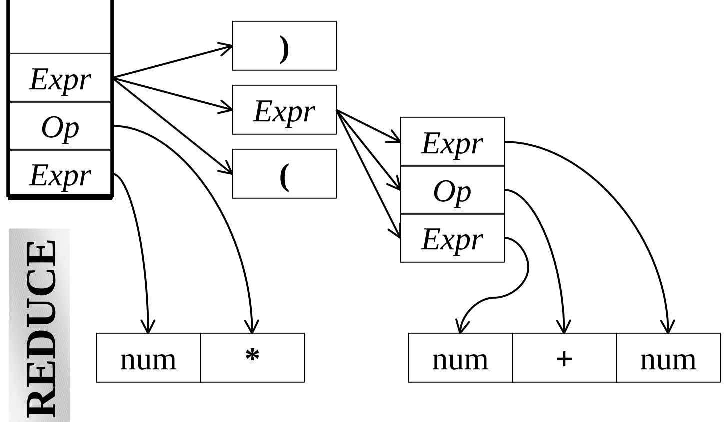
Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$
$Expr \rightarrow (Expr)$
$Expr \rightarrow -\ Expr$
$Expr \rightarrow$ num
$Op \rightarrow +$
$Op \rightarrow -$
$Op \rightarrow *$



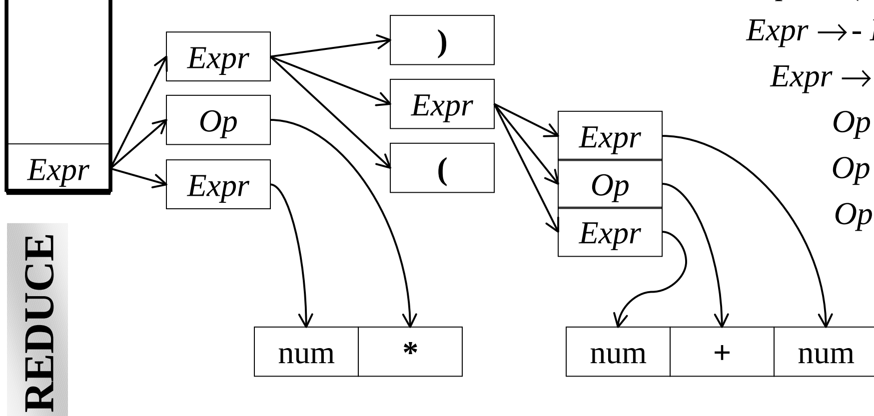
Shift-Reduce Parser Example

$Expr \rightarrow Expr\ Op\ Expr$
$Expr \rightarrow (Expr)$
$Expr \rightarrow -\ Expr$
$Expr \rightarrow$ num
$Op \rightarrow +$
$Op \rightarrow -$
$Op \rightarrow *$



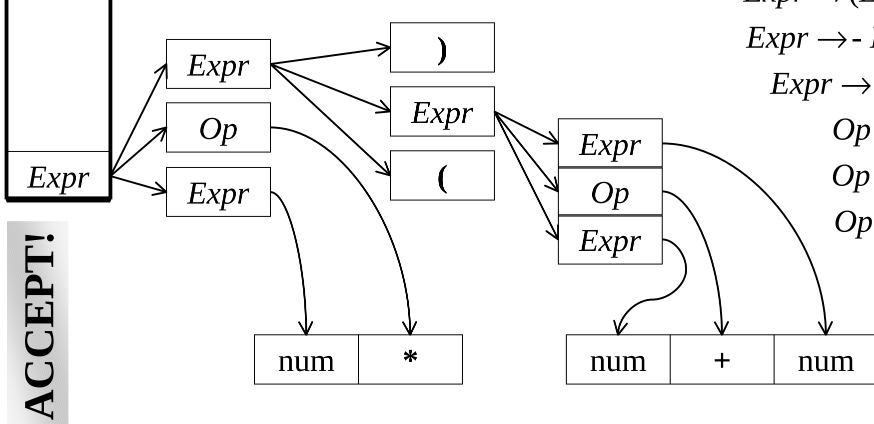
Shift-Reduce Parser Example

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *



Shift-Reduce Parser Example

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *



Parser Actions

- Shift:
  - Shift the next token onto the stack
- Reduce:
  - If the top symbols of the stack match an RHS of a production , can do the reduction
    - Pop the RHS from the top of the stack
    - push the LHS nonterminal onto the stack
- If the input is empty
  - accept if only the start symbol is on the stack
  - reject otherwise

Potential Conflicts

- Reduce/Reduce Conflict
  - Top of the stack may match RHS of multiple productions
  - Which to use?
- Shift/Reduce Conflict
  - Stack may match RHS of production
  - But that may not be the right match
  - May need to shift an input and later find a different reduction

Conflicts

- Original Grammar

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow -\ Expr$

$Expr \rightarrow \mathrm{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Conflicts

- New Grammar

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr - Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ -$

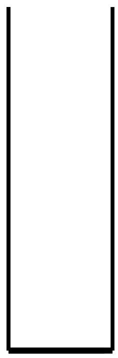
$Expr \rightarrow \mathrm{num}$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

Conflicts



$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ \text{-}\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ \text{-}$

$Expr \rightarrow \text{num}$

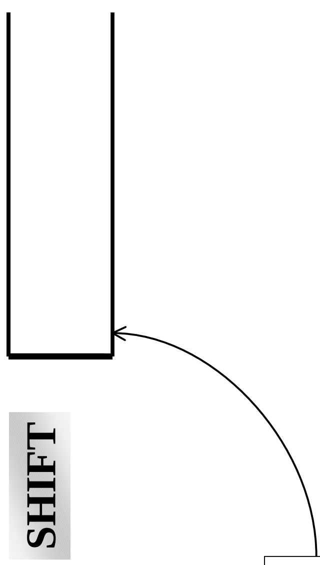
$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$

num	-	num

Conflicts



$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ \text{-}\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ \text{-}$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$

num	-	num

Conflicts

$Expr \rightarrow Expr\ Op\ Expr$
$Expr \rightarrow Expr\ \text{-}\ Expr$
$Expr \rightarrow (Expr)$
$Expr \rightarrow Expr\ \text{-}$
$Expr \rightarrow \text{num}$
$Op \rightarrow +$
$Op \rightarrow \text{-}$
$Op \rightarrow *$

num

SHIFT

-	num

Conflicts

$Expr \rightarrow Expr\ Op\ Expr$
$Expr \rightarrow Expr\ \text{-}\ Expr$
$Expr \rightarrow (Expr)$
$Expr \rightarrow Expr\ \text{-}$
$Expr \rightarrow \text{num}$
$Op \rightarrow +$
$Op \rightarrow \text{-}$
$Op \rightarrow *$

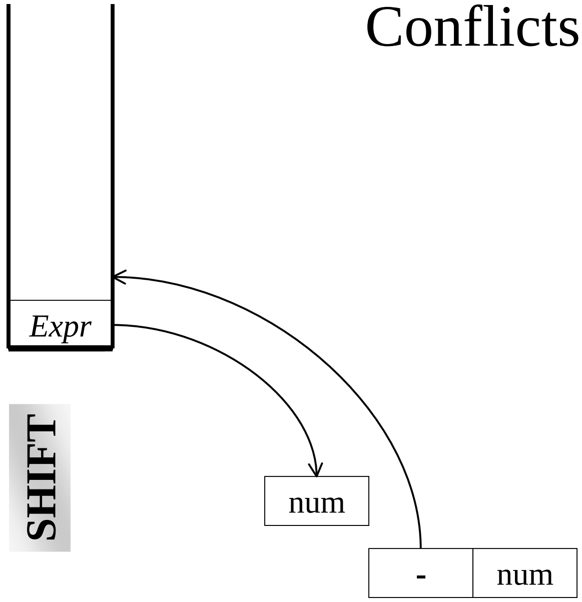
Expr

REDUCE

num

-	num

Conflicts



$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ \text{-}\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ \text{-}$

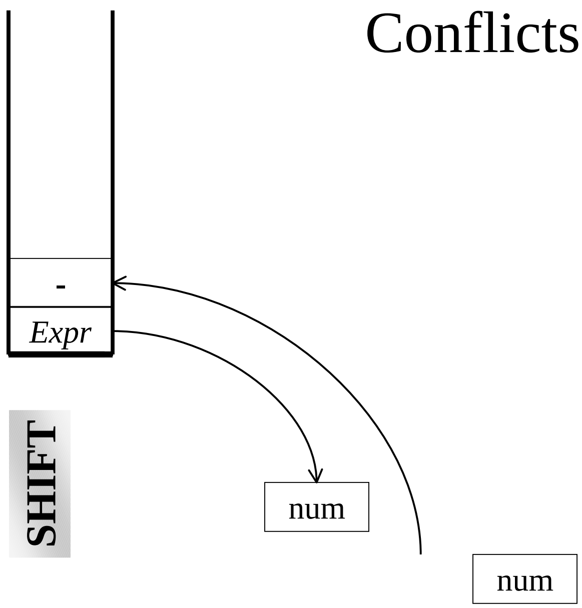
$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$

Conflicts



$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ \text{-}\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ \text{-}$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$

Shift/Reduce/Reduce Conflict

-

Expr

Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *

Options:

Reduce
Reduce
Shift

num

num

Shift/Reduce/Reduce Conflict

-

Expr

REDUCE

Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *

What Happens
if Choose

Reduce

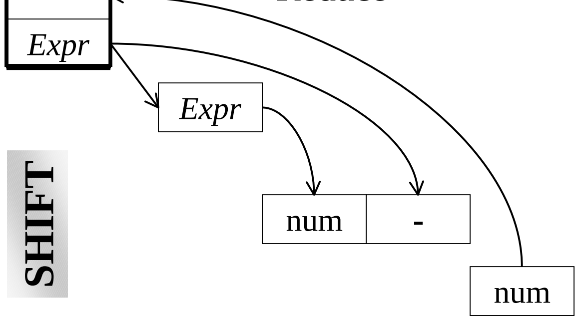
num

num

Shift/Reduce/Reduce Conflict

What Happens
if Choose
Reduce

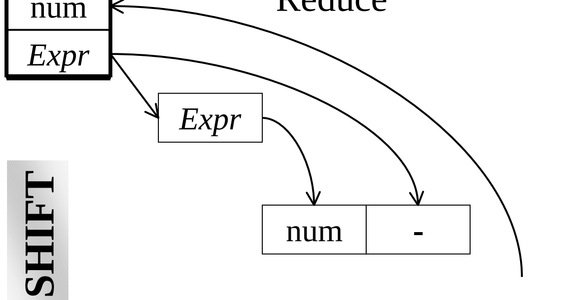
Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *



Shift/Reduce/Reduce Conflict

What Happens
if Choose
Reduce

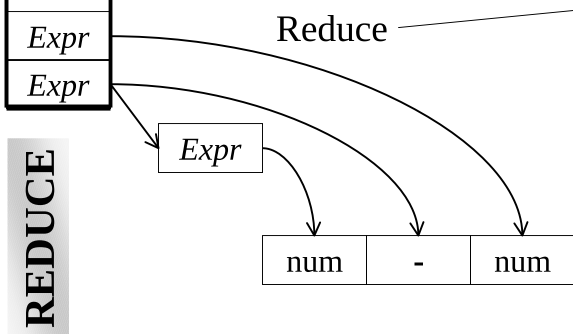
Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *



Shift/Reduce/Reduce Conflict

What Happens
if Choose
Reduce

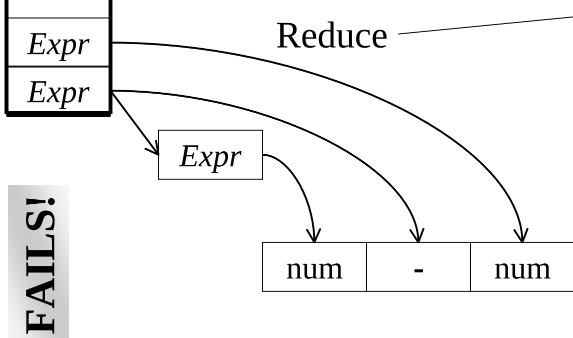
Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *



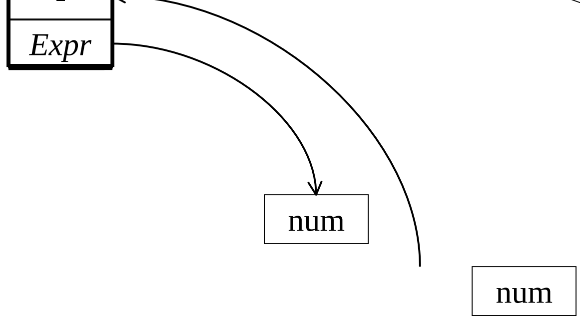
Shift/Reduce/Reduce Conflict

What Happens
if Choose
Reduce

Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *



Shift/Reduce/Reduce Conflict



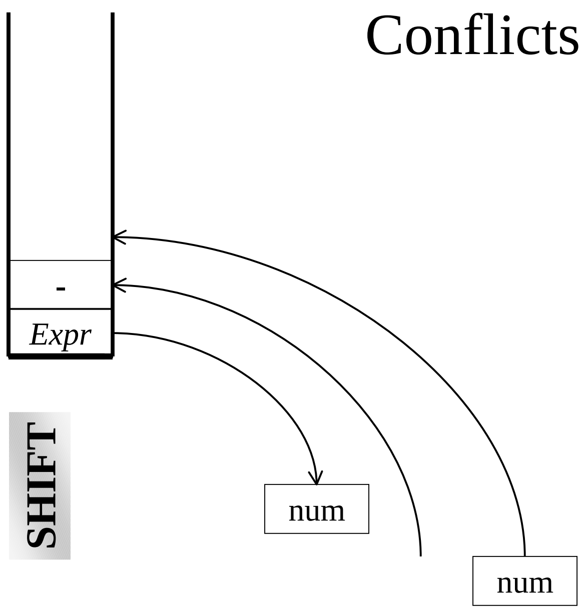
OnlyTwo Work

Reduce

Shift

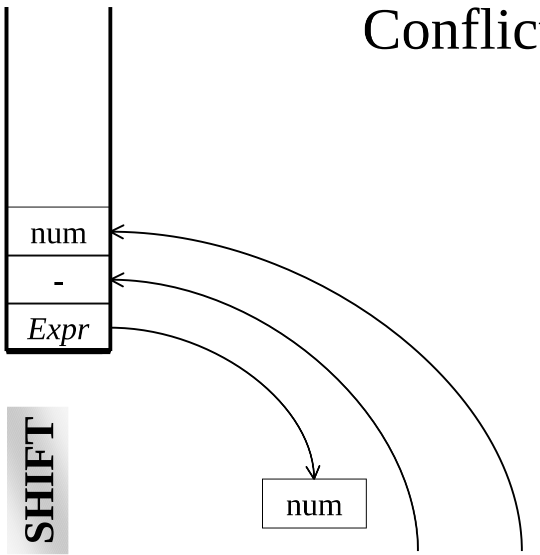
Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *

Conflicts



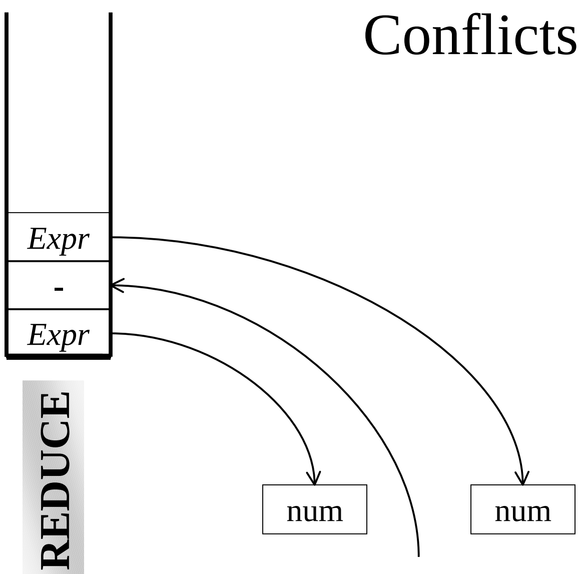
Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *

Conflicts



Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *

Conflicts



Expr → *Expr Op Expr*
Expr → *Expr* - *Expr*
Expr → (*Expr*)
Expr → *Expr* -
Expr → num
Op → +
Op → -
Op → *

Conflicts

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ \text{-}\ Expr$

$Expr \rightarrow (Expr)$

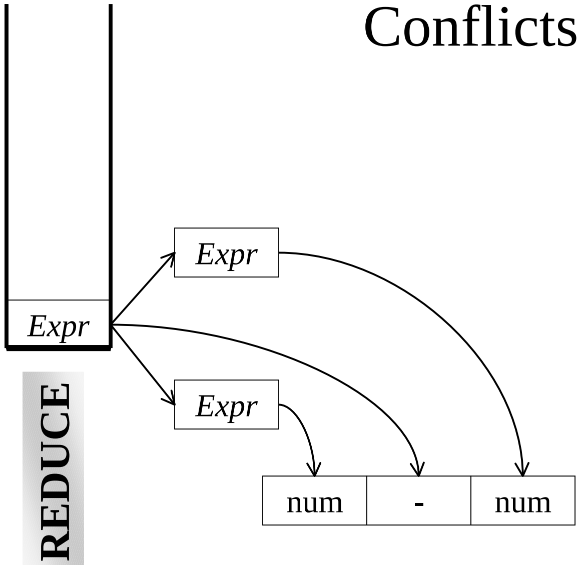
$Expr \rightarrow Expr\ \text{-}$

$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$



Conflicts

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ \text{-}\ Expr$

$Expr \rightarrow (Expr)$

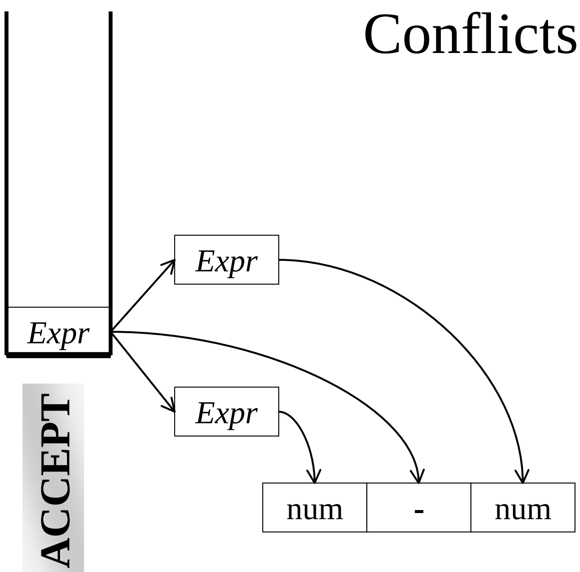
$Expr \rightarrow Expr\ \text{-}$

$Expr \rightarrow \text{num}$

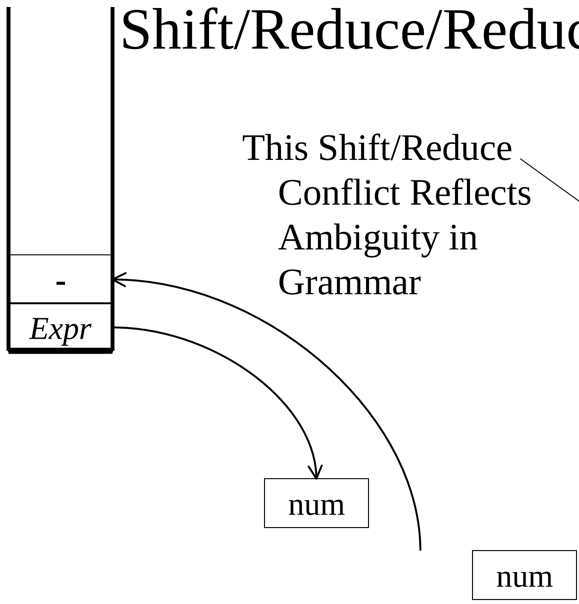
$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$



Shift/Reduce/Reduce Conflict



This Shift/Reduce Conflict Reflects Ambiguity in Grammar

$Expr \rightarrow Expr\ Op\ Expr$

$Expr \rightarrow Expr\ \text{-}\ Expr$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ \text{-}$

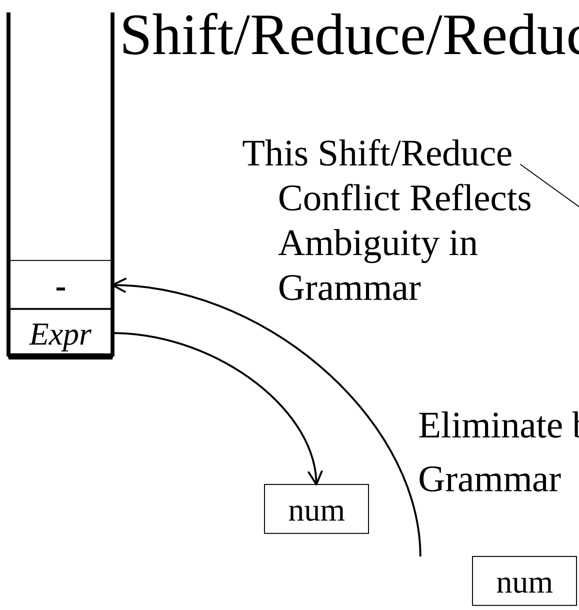
$Expr \rightarrow \text{num}$

$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$

Shift/Reduce/Reduce Conflict



This Shift/Reduce Conflict Reflects Ambiguity in Grammar

$Expr \rightarrow Expr\ Op\ Expr$

$\cancel{Expr \rightarrow Expr\ \text{-}\ Expr}$

$Expr \rightarrow (Expr)$

$Expr \rightarrow Expr\ \text{-}$

$Expr \rightarrow \text{num}$

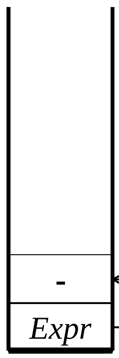
$Op \rightarrow +$

$Op \rightarrow \text{-}$

$Op \rightarrow *$

Eliminate by Hacking Grammar

Shift/Reduce/Reduce Conflict



$Expr \to Expr\ Op\ Expr$

$Expr \to Expr\ \text{-}\ Expr$

$Expr \to (Expr)$

$Expr \to Expr\ \text{-}$

$Expr \to \text{num}$

$Op \to +$

$Op \to \text{-}$

$Op \to *$

This Shift/Reduce Conflict Can Be Eliminated By Lookahead of One Symbol

Parser Generator Should Handle It

num

num

Constructing a LR(k) Parser

- We will construct LR(0), SLR(0), LR(1) parsers
- Key Decisions
  - Shift or Reduce
  - Which Production to Reduce
- Basic Idea
  - Build a DFA to Control Shift and Reduce Actions
  - In Effect, Convert Grammar to PDA
  - Encode PDA in Parse Table

Parser State

- Input Token Sequence
  – Special “Token” $ for end of input
- Current State from Finite State Machine
- Two Stacks
  – State Stack (helps implement parser control)
  – Symbol Stack (terminals from input and nonterminals from reductions)
- Actions
  – Push Symbols and States Onto Stack
  – Reduce According to a Given Production

Parse Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- At Each Step
- Look up
  – Table[top of state stack] [input symbol]
- Carry out the action

Parse Table Example

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack | Symbol Stack | Input | Grammar

(())

$S \to X\,\$$ (1)

$X \to (X)$ (2)

$X \to (\ )$ (3)

s0

X

Parser Tables

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Shift to s*n*
  - Push input token into the symbol stack
  - Push s*n* into state stack
  - Advance to next input symbol

Parser Tables

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Reduce (*n*)
  - Pop both stacks as many times as the number of symbols on the RHS of rule *n*
  - Push LHS of rule *n* into symbol stack

Parser Tables

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Reduce (*n*) (continued)
  - Look up
    - Table[top of the state stack][top of symbol stack]
  - Push that state (in goto k) into state stack

Parser Tables

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Accept
  - Stop parsing and report success

Parse Table In Action

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
s0		(())$	$S \to X\,\$$ (1) $X \to (X)$ (2) $X \to (\ )$ (3)

Parse Table In Action

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
s0		(())$	$S \to X\,\$$ (1) $X \to (X)$ (2) $X \to (\ )$ (3)

Parse Table In Action

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
s2 s0	(	())$	$S \to X\,\$$ (1) $X \to (X)$ (2) $X \to (\ )$ (3)

Parse Table In Action

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	**shift to s2**	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

s2
s0

Symbol Stack

(

Input

())$

Grammar

$S \rightarrow X\ \$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Parse Table In Action

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

s2
s2
s0

Symbol Stack

(
(

Input

))$

Grammar

$S \rightarrow X\ \$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Parse Table In Action

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s2, s2, s0

Symbol Stack: (, (

Input:))$

Grammar

$S \to X\,\$$ (1)

$X \to (X)$ (2)

$X \to (\ )$ (3)

Parse Table In Action

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s5, s2, s2, s0

Symbol Stack:), (, (

Input:)$

Grammar

$S \to X\,\$$ (1)

$X \to (X)$ (2)

$X \to (\ )$ (3)

Parse Table In Action

State	ACTION			Goto
	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
s5		)$	$S \rightarrow X\,\$$ (1)
s2	)		$X \rightarrow (X)$ (2)
s2	(		$\boxed{X \rightarrow (\ )\quad (3)}$
s0	(		

Step One: Pop Stacks

State	ACTION			Goto
	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
s5		)$	$S \rightarrow X\,\$$ (1)
s2	)		$X \rightarrow (X)$ (2)
s2	(		$\boxed{X \rightarrow (\ )\quad (3)}$
s0	(		

Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack | Symbol Stack | Input | Grammar

Input:)$

State Stack: s2, s0

Symbol Stack: (

$S \rightarrow X\,\$$ (1)

$X \rightarrow (X)$ (2)

$\boxed{X \rightarrow (\ ) \quad (3)}$

Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack | Symbol Stack | Input | Grammar

Input:)$

State Stack: s2, s0

Symbol Stack: (

$S \rightarrow X\,\$$ (1)

$X \rightarrow (X)$ (2)

$\boxed{X \rightarrow (\ ) \quad (3)}$

Step Two: Push Nonterminal

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

s2
s0

Symbol Stack

X
(

Input

)$

Grammar

$S \rightarrow X\,\$$ (1)

$X \rightarrow (X)$ (2)

$\boxed{X \rightarrow (\ )\quad (3)}$

Step Three: Use Goto, Push New State

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

s2
s0

Symbol Stack

X
(

Input

)$

Grammar

$S \rightarrow X\,\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input
s3		)$
s2	X	
s0	(	

Grammar

$S \to X\ \$$ (1)

$X \to (X)$ (2)

$X \to (\ )$ (3)

Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input
s3		)$
s2	X	
s0	(	

Grammar

$S \to X\ \$$ (1)

$X \to (X)$ (2)

$X \to (\ )$ (3)

Parse Table In Action

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s4, s3, s2, s0

Symbol Stack:), X, (

Input: $

Grammar

$S \rightarrow X\,\$ \quad (1)$

$X \rightarrow (X) \quad (2)$

$X \rightarrow (\;) \quad (3)$

Parse Table In Action

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s4, s3, s2, s0

Symbol Stack:), X, (

Input: $

Grammar

$S \rightarrow X\,\$ \quad (1)$

$\boxed{X \rightarrow (X) \quad (2)}$

$X \rightarrow (\;) \quad (3)$

Step One: Pop Stacks

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s4, s3, s2, s0

Symbol Stack:), X, (

Input: $

Grammar

$S \rightarrow X\ \$$ (1)

$\boxed{X \rightarrow (X) \quad (2)}$

$X \rightarrow (\ )$ (3)

Step One: Pop Stacks

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s0

Symbol Stack:

Input: $

Grammar

$S \rightarrow X\ \$$ (1)

$\boxed{X \rightarrow (X) \quad (2)}$

$X \rightarrow (\ )$ (3)

Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar

$

$S \to X\ \$ \quad (1)$

$\boxed{X \to (X) \quad (2)}$

$X \to (\ ) \quad (3)$

s0

Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar

$

$S \to X\ \$ \quad (1)$

$\boxed{X \to (X) \quad (2)}$

$X \to (\ ) \quad (3)$

s0 X

Step Three: Use Goto, Push New State

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
		$	$S \rightarrow X\,\$$ (1)
			$\boxed{X \rightarrow (X) \quad (2)}$
s0	X		$X \rightarrow (\ )$ (3)

Step Three: Use Goto, Push New State

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
		$	$S \rightarrow X\,\$$ (1)
s1			$\boxed{X \rightarrow (X) \quad (2)}$
s0	X		$X \rightarrow (\ )$ (3)

Accept the String!

State	ACTION (	ACTION)	ACTION $	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s1, s0

Symbol Stack: S

Input: $

Grammar

$S \to X\,\$$ (1)

$X \to (X)$ (2)

$X \to (\ )$ (3)

Constructing a LR(k) Parser

- Synthesize A DFA
  - Captures all the possible states that the parser can be in
  - State transitions for terminals and non-terminals
- Use DFA to create an parse table

LR Example

- The grammar

$S \to X \ \$$ (1)

$X \to (X)$ (2)

$X \to (\ )$ (3)

DFA States Based on LR(0) Items

- We need to capture how much of a given production we have scanned so far

$$\boldsymbol{X \to (\ X\ )}$$

Are we here? **Or here?** **Or here?** **Or here?**

LR(0) Items

- We need to capture how much of a given production we have scanned so far

$$X \rightarrow (\quad X \quad)$$

- Production Generates 4 items
  - $X \rightarrow \bullet\ (X)$
  - $X \rightarrow (\ \bullet X)$
  - $X \rightarrow (X \bullet)$
  - $X \rightarrow (X)\ \bullet$

Example of Items

- The grammar

  $S \rightarrow X\ \$$

  $X \rightarrow (X)$

  $X \rightarrow (\ )$

- Items

  $S \rightarrow \bullet\ X\ \$$

  $S \rightarrow X \bullet\ \$$

  $X \rightarrow \bullet\ (X)$

  $X \rightarrow (\bullet X)$

  $X \rightarrow (X \bullet)$

  $X \rightarrow (X)\ \bullet$

  $X \rightarrow \bullet\ (\ )$

  $X \rightarrow (\ \bullet\ )$

  $X \rightarrow (\ )\ \bullet$

Key idea behind items

- If the "current state" contains the item $A \rightarrow \alpha \bullet c\, \beta$ and the current symbol in the input buffer is c
  - the state prompts parser to perform a shift action
  - next state will contain $A \rightarrow \alpha\ c \bullet \beta$
- If the "state" contains the item $\mathbf{A} \rightarrow \alpha \bullet$
  - the state prompts parser to perform a reduce action
- If the "state" contains the item $S \rightarrow \alpha \bullet \$$ and the input buffer is empty
  - the state prompts parser to accept

Closure() of a set of items

- Closure finds all the items in the same "state"
- Fixed Point Algorithm for Closure**(I)**
  - Every item in **I** is also an item in Closure**(I)**
  - If $A \rightarrow \alpha \bullet B\, \beta$ is in Closure**(I)** and $B \rightarrow \bullet\, \gamma$ is an item, then add $B \rightarrow \bullet\, \gamma$ to Closure**(I)**
  - Repeat until no more new items can be added to Closure**(I)**

Example of Closure

- Closure({$X \to (\bullet X)$})

$$\left\{ \begin{array}{l} X \to (\bullet X) \\ X \to \bullet (X) \\ X \to \bullet (\ ) \end{array} \right\}$$

- Items

$S \to \bullet X \$$
$S \to X \bullet \$$
$X \to \bullet (X)$
$X \to (\bullet X)$
$X \to (X \bullet)$
$X \to (X) \bullet$
$X \to \bullet (\ )$
$X \to (\bullet)$
$X \to (\ ) \bullet$

Another Example

- closure({$S \to \bullet X \$$})

$$\left\{ \begin{array}{l} S \to \bullet X \$ \\ X \to \bullet (X) \\ X \to \bullet (\ ) \end{array} \right\}$$

- Items

$S \to \bullet X \$$
$S \to X \bullet \$$
$X \to \bullet (X)$
$X \to (\bullet X)$
$X \to (X \bullet)$
$X \to (X) \bullet$
$X \to \bullet (\ )$
$X \to (\bullet)$
$X \to (\ ) \bullet$

Goto() of a set of items

- Goto finds the new state after consuming a grammar symbol while at the current state
- Algorithm for *Goto*(I, X)
  where I is a set of items
  and X is a grammar symbol

Goto(I, X) = Closure($\{ A \to \alpha\ X \bullet \beta \mid A \to \alpha \bullet X \beta \text{ in I} \}$)

- goto is the new set obtained by "moving the dot" over X

Example of Goto

- Goto $(\{X \to (\bullet X)\}, X)$

$\{ X \to (X \bullet\ ) \}$

- Items

$S \to \bullet X \$$
$S \to X \bullet \$$
$X \to \bullet (X)$
$X \to (\bullet X)$
$X \to (X \bullet)$
$X \to (X) \bullet$
$X \to \bullet (\ )$
$X \to (\bullet)$
$X \to (\ ) \bullet$

Example of Goto

- Goto ({$X \to \bullet(X)$}, ()

$$\left\{\begin{array}{l} X \to (\ \bullet X) \\ X \to \bullet\ (X) \\ X \to \bullet\ (\ ) \end{array}\right\}$$

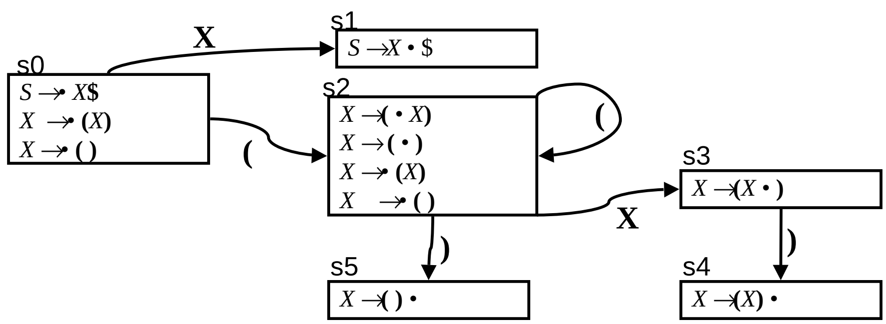
- Items

$S \to \bullet X\ \$$
$S \to X \bullet\ \$$
$X \to \bullet\ (X)$
$X \to (\bullet X)$
$X \to (X \bullet)$
$X \to (X)\ \bullet$
$X \to \bullet\ (\ )$
$X \to (\ \bullet\ )$
$X \to (\ )\ \bullet$

Building the DFA states

- Start with the item $S \to \bullet\ \beta\ \$$
- Create the first state to be Closure({ $Goal \to \bullet\ S\ \$$})
- Pick a state **I**
  - for each item $A \to \alpha\ \bullet X\ \beta$ in **I**
    - find Goto**(I, X)**
    - if Goto**(I, X)** is not already a state, make one
    - Add an edge X from state **I** to Goto**(I, X)** state
- Repeat until no more additions possible

DFA Example



Constructing a LR(0) Parse Engine

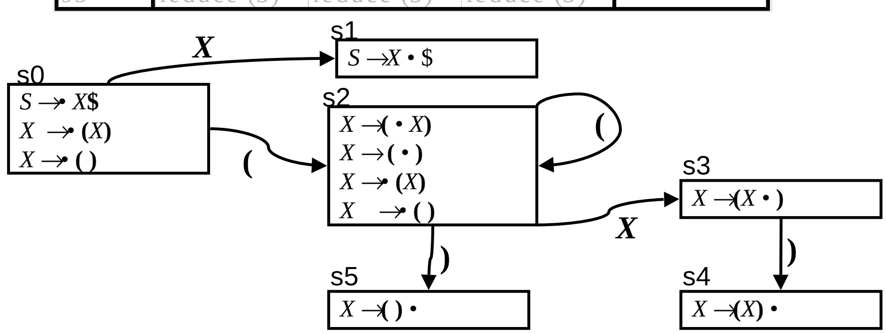
- Build a DFA
  - DONE
- Construct a parse table using the DFA

Creating the parse tables

- For each state
  - Transition to another state using a terminal symbol is a shift to that state (*shift to sn*)
  - Transition to another state using a non-terminal is a goto that state (*goto sn*)
  - If there is an item **A** $\rightarrow \alpha$ • in the state do a reduction with that production for all terminals (*reduce k*)

Building Parse Table Example

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	



Problem With LR(0) Parsing

- No lookahead
- Vulnerable to unnecessary conflicts
  – Shift/Reduce Conflicts (may reduce too soon in some cases)
  – Reduce/Reduce Conflicts
- Solutions:
  – SLR(0) parsing - reduce only when next symbol can occur after nonterminal from production
  – LR(1) parsing - systematic lookahead

SLR(0) Parsing

- If a state contains $A \to \beta \bullet$
- Reduce by $A \to \beta$ only if next input symbol can follow A in some derivation
- Example Grammar

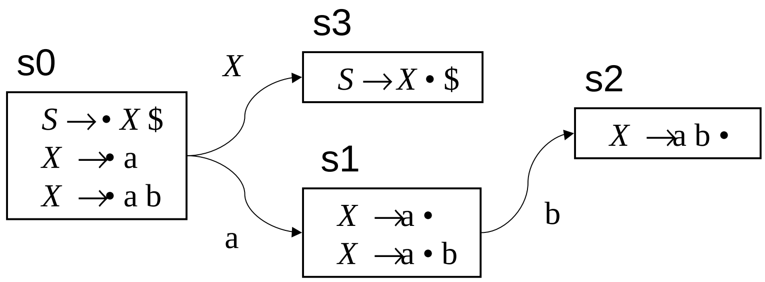
$S \to X$ \$

$X \to$ a

$X \to$ a b

LR(0) Parser

		ACTION		Goto
State	a	b	$	X
s0	shift to s1	error	error	goto s3
s1	reduce(2)	S/R Conflict	reduce(2)	
s2	reduce(3)	reduce(3)	reduce(3)	
s3	error	error	accept	



Creating SLR parse tables

- For each state
  - Transition to another state using a terminal symbol is a shift to that state (*shift to sn*) (same as LR(0))
  - Transition to another state using a non-terminal is a goto that state (*goto sn*) (same as LR(0))
  - If there is an item $A \to \alpha \bullet$ in the state do a reduction with that production for all terminals T that may follow A in some derivation (more precise than LR(0))

Follow() sets

For each non-terminal A, Follow(A) is the set of terminals that can come after A in some derivation

Constraints for Follow()

- $\$ \in$ Follow(S), where S is the start symbol
- If $A \to \alpha\ B\beta$ is a production
  then First(β) $\subseteq$ Follow(B)
- If $A \to \alpha\ B$ is a production then Follow(A) $\subseteq$ Follow(B)
- If $A \to \alpha\ B\beta$ is a production and β derives ε
  then Follow(A) $\subseteq$ Follow(B)

Algorithm for Follow

for all nonterminals NT

  Follow(NT) = {}

Follow(S) = {}

while Follow sets keep changing

  for all productions $A \to \alpha\ B\beta$

    Follow(B) = Follow(B) $\cup$ First(β)

    if (β derives ε) Follow(B) = Follow(B) $\cup$ Follow(A)

  for all productions $A \to \alpha\ B$

    Follow(B) = Follow(B) $\cup$ Follow(A)

Augmenting Example with Follow

- Example Grammar for Follow

  $S \to X$ \$

  $X \to$ a

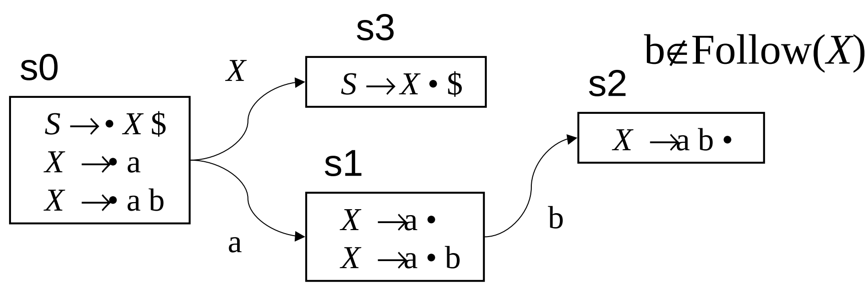
  $X \to$ a b

  Follow(S) = { \$ }

  Follow(X) = { \$ }

SLR Eliminates Shift/Reduce Conflict

		ACTION		Goto
State	a	b	$	X
s0	shift to s1	error	error	goto s3
s1	reduce(2)	shift to s2	reduce(2)	
s2	reduce(3)	reduce(3)	reduce(3)	
s3	error	error	accept	



Basic Idea Behind LR(1)

- Split states in LR(0) DFA based on lookahead
- Reduce based on item and lookahead

LR(1) Items

- Items will keep info on
  - production
  - right-hand-side position (the dot)
  - look ahead symbol
- LR(1) item is of the form [$A \to \alpha \bullet \beta$ T]
  - $A \to \alpha\ \beta$ is a production
  - The dot in $A \to \alpha \bullet \beta$ denotes the position
  - T is a terminal or the end marker ($)

Meaning of LR(1) Items

- Item [$A \to \alpha \bullet \beta$ T] means
  - The parser has parsed an α
  - If it parses a β and the next symbol is T
  - Then parser should reduce by $A \to \alpha\ \beta$

- The grammar

  $S \to X\,\$$

  $X \to (X)$

  $X \to \varepsilon$

- Terminal symbols
  - ‘(‘ ‘)’
- End of input symbol
  - ‘$’

LR(1) Items

$[S \to \bullet X\,\$ \quad)\,]$

$[S \to \bullet X\,\$ \quad (\,]$

$[S \to \bullet X\,\$ \quad \$\,]$

$[S \to X \bullet \$ \quad)\,]$

$[S \to X \bullet \$ \quad (\,]$

$[S \to X \bullet \$ \quad \$]$

$[X \to \bullet (X) \quad)\,]$

$[X \to \bullet (X) \quad (\,]$

$[X \to \bullet (X) \quad \$\,]$

$[X \to (\bullet X) \quad)\,]$

$[X \to (\bullet X) \quad (\,]$

$[X \to (\bullet X) \quad \$\,]$

$[X \to (X \bullet \quad)\,]$

$[X \to (X \bullet) \quad (\,]$

$[X \to (X \bullet) \quad \$\,]$

$[X \to (X) \bullet \quad)\,]$

$[X \to (X) \bullet \quad (\,]$

$[X \to (X) \bullet \quad \$\,]$

$[X \to \bullet \quad)\,]$

$[X \to \bullet \quad (\,]$

$[X \to \bullet \quad \$\,]$

Creating a LR(1) Parser Engine

- Need to define Closure() and Goto() functions for LR(1) items
- Need to provide an algorithm to create the DFA
- Need to provide an algorithm to create the parse table

Closure algorithm

Closure(I)

repeat

for all items [A $\rightarrow \alpha \bullet X \beta$ c] in I

for any production $X \rightarrow \gamma$

for any $d \in$ First(β c)

$I = I \cup \{ [X \rightarrow \bullet \gamma \quad d] \}$

until I does not change

Goto algorithm

Goto(I, X)

J = { }

for any item [A $\rightarrow \alpha \bullet X \beta$ c] in I

$J = J \cup \{[A \rightarrow \alpha X \bullet \beta \quad c]\}$

return Closure(J)

Building the LR(1) DFA

- Start with the item [<S'> → • <S> $?]
  - ? irrelevant because we will never shift $
- Find the closure of the item and make an state
- Pick a state I
  - for each item [A→α • X β c] in I
    - find Goto(I, X)
    - if Goto(I, X) is not already a state, make one
    - Add an edge X from state I to Goto(I, X) state
- Repeat until no more additions possible

Creating the parse tables

- For each LR(1) DFA state
  - Transition to another state using a terminal symbol is a shift to that state (*shift to sn*)
  - Transition to another state using a non-terminal is a goto that state (*goto sn*)
  - If there is an item [**A** →α • a] in the state, do a reduction for input symbol a with the production **A** →α (*reduce k*)

LALR(1) Parser

- Motivation
  - LR(1) parse engine has a large number of states
  - Simple method to eliminate states
- If two LR(1) states are identical except for the look ahead symbol of the items
  Then Merge the states
- Result is LALR(1) DFA
- Typically has many fewer states than LR(1)
- May also have more reduce/reduce conflicts

Summary

- Bottom Up Shift-Reduce Parsing
- Conflicts: Shift/Reduce, Reduce/Reduce
- Automatic Generation of Parser
  - Finite State Control Plus Push Down Stack
  - Table driven implementation
- Use of Lookahead to eliminate conflicts
  - SLR parsing
  - LR(1) parsing