Outline

• Review of bottom-up parsing

• Computing the parsing DFA
  - Closures, LR(1) Items, States
  - Transitions

• Using parser generators
  - Handling Conflicts
In One Slide

- An **LR(1) parsing table** can be constructed automatically from a CFG. An **LR(1) item** is a pair made up of a **production** and a **lookahead** token; it represents a possible parser context. After we extend LR(1) items by closing them they become LR(1) **DFA states**. Grammars can have **shift/reduce** or **reduce/reduce** conflicts. You can fix most conflicts with **precedence** and **associativity** declarations. **LALR(1) tables** are formed from LR(1) tables by merging states with similar cores.
Bottom-up Parsing (Review)

- A bottom-up parser rewrites the input string to the start symbol
- The state of the parser is described as
  \[ \alpha \xrightarrow{\cdot} \gamma \]
  - \(\alpha\) is a stack of terminals and non-terminals
  - \(\gamma\) is the string of terminals not yet examined
- Initially: \(\xrightarrow{\cdot} x_1 x_2 \ldots x_n\)
Shift and Reduce Actions (Review)

- Recall the CFG: \( E \rightarrow \text{int} \mid E + (E) \)
- A bottom-up parser uses two kinds of actions:
  - **Shift** pushes a terminal from input on the stack
    
    \[
    E + (\uparrow \text{int} ) \Rightarrow E + (\text{int} \uparrow)
    \]
  - **Reduce** pops 0 or more symbols off of the stack (production RHS) and pushes a non-terminal on the stack (production LHS)
    
    \[
    E + (E + (E) \uparrow ) \Rightarrow E + (E \uparrow)
    \]
Key Issue: When to Shift or Reduce?

• Idea: use a finite automaton (DFA) to decide when to shift or reduce
  - The input is the stack
  - The language consists of terminals and non-terminals

• We run the DFA on the stack and we examine the resulting state $X$ and the token $tok$ after $\rightarrow$
  - If $X$ has a transition labeled $tok$ then shift
  - If $X$ is labeled with “$A \rightarrow \beta$ on $tok$” then reduce
LR(1) Parsing. An Example

E → int

E → int + (int) + (int)$ shift
int ▶ + (int) + (int)$ E → int
E ▶ + (int) + (int)$ shift(x3)
E + (int ▶ ) + (int)$ E → int
E + (E ▶ ) + (int)$ shift
E + (E) ▶ + (int)$ E → E+(E)
E ▶ + (int)$ shift (x3)
E + (int ▶ )$ E → int
E + (E ▶ )$ shift
E + (E) ▶$ E → E+(E)
E ▶$ accept
E → E + (E)
E → E + (E)
E → E + (E)
E → E + (E)
E → E + (E)
E → E + (E)
E → E + (E)
E → E + (E)
End of review
Key Issue: How is the DFA Constructed?

- The **stack** describes the **context** of the parse
  - What non-terminal we are looking for
  - What production rhs we are looking for
  - What we have seen so far from the rhs
Three hours later, you can finally parse $E \rightarrow E + E \mid \text{int}$.
Parsing Contexts

- Consider the state:

\[ E \]

\[ \text{int } + ( \text{ int } ) + ( \text{ int } ) \]

- The stack is

\[ E + ( \text{ int } ) + ( \text{ int } ) \]

- Context:
  - We are looking for an \( E \rightarrow E + ( \bullet E ) \):
    - Have have seen \( E + ( \text{ from the right-hand side} \)
  - We are also looking for \( E \rightarrow \bullet \text{ int or } E \rightarrow \bullet E + ( E ) \):
    - Have seen nothing from the right-hand side

- One DFA state describes several contexts
LR(1) Items

- An **LR(1) item** is a pair:
  \[ X \rightarrow \alpha \cdot \beta , a \]
  - \( X \rightarrow \alpha \beta \) is a production
  - \( a \) is a terminal (the **lookahead** terminal)
  - LR(1) means 1 lookahead terminal

- \([X \rightarrow \alpha \cdot \beta , a]\) describes a **context** of the parser
  - We are trying to find an \( X \) followed by an \( a \), and
  - We have \( \alpha \) already on top of the stack
  - Thus we need to see next a prefix derived from \( \beta a \)
Note

• The symbol ▶ was used before to separate the stack from the rest of input
  - $\alpha ▶ \gamma$, where $\alpha$ is the stack and $\gamma$ is the remaining string of terminals

• In LR(1) items • is used to mark a prefix of a production rhs:
  $$X \rightarrow \alpha \bullet \beta, \ a$$
  - Here $\beta$ might contain non-terminals as well

• In both case the stack is on the left
Convention

• We add to our grammar a fresh new start symbol $S$ and a production $S \rightarrow E$
  - Where $E$ is the old start symbol
  - No need to do this if $E$ had only one production

• The **initial parsing context** contains:
  
  $$S \rightarrow \bullet E, \$$$

  - Trying to find an $S$ as a string derived from $E\$$
  - The stack is empty
LR(1) Items (Cont.)

• In context containing

\[ E \rightarrow E + \bullet ( E ), + \]

- If ( follows then we can perform a shift to context containing

\[ E \rightarrow E + ( \bullet E ), + \]

• In context containing

\[ E \rightarrow E + ( E ) \bullet, + \]

- We can perform a reduction with \( E \rightarrow E + ( E ) \)

- But only if a + follows
LR(1) Items (Cont.)

• Consider a context with the item

   \[ E \rightarrow E + ( \cdot E ) , + \]

• We expect next a string derived from \( E ) + \)

• There are two productions for \( E \)

   \[ E \rightarrow \text{int} \quad \text{and} \quad E \rightarrow E + ( E \) \)

• We describe this by extending the context with two more items:

   \[ E \rightarrow \cdot \text{int} , ) \]

   \[ E \rightarrow \cdot E + ( E ) , ) \]
The Closure Operation

• The operation of extending the context with items is called the closure operation

\[
\text{Closure(Items) = repeat}
\]

\[
\text{for each } [X \rightarrow \alpha \bullet Y\beta, a] \text{ in Items}
\]

\[
\text{for each production } Y \rightarrow \gamma
\]

\[
\text{for each } b \in \text{First(}\beta \text{a)}
\]

\[
\text{add } [Y \rightarrow \bullet \gamma, b] \text{ to Items}
\]

\[
\text{until } \text{Items is unchanged}
\]
Constructing the Parsing DFA (1)

• Construct the start context:

\[
\text{Closure} \left( \{ S \rightarrow \bullet E, \$ \} \right) =
\]

\[
S \rightarrow \bullet E, \$
E \rightarrow \bullet E+(E), \$
E \rightarrow \bullet \text{int}, \$
E \rightarrow \bullet E+(E), +
\]

• We abbreviate as:

\[
S \rightarrow \bullet E, \$
E \rightarrow \bullet E+(E), \$/+
E \rightarrow \bullet \text{int}, \$/+
\]
PLANNING

You... have a plan, right?
Constructing the Parsing DFA (2)

• An **LR(1) DFA state** is a **closed** set of **LR(1) items**
  - This means that we performed **Closure**

• The start state contains \([S \rightarrow \cdot E, \$]\)

• A state that contains \([X \rightarrow \alpha \cdot, b]\) is labeled with “**reduce with** \(X \rightarrow \alpha\) on \(b\)”

• And now the transitions ...
The DFA Transitions

- A state “State” that contains $[X \rightarrow \alpha \cdot y\beta , b]$ has a transition labeled $y$ to a state that contains the items “Transition(State, y)"
  - $y$ can be a terminal or a non-terminal

Transition(State, y) =

Items $\leftarrow \emptyset$

for each $[X \rightarrow \alpha \cdot y\beta , b] \in$ State

  add $[X \rightarrow \alpha y\cdot\beta , b]$ to Items

return Closure(Items)
LR(1) DFA Construction Example

\[ S \rightarrow \cdot E, \$
\]
\[ E \rightarrow \cdot E+(E), $/+ 
\]
\[ E \rightarrow \cdot \text{int}, $/+ 
\]
LR(1) DFA Construction Example

\[\begin{align*}
S & \rightarrow \cdot E, \$
E & \rightarrow \cdot E+(E), \$, +
E & \rightarrow \cdot \text{int}, \$, +
\end{align*}\]
LR(1) DFA Construction Example

S → •E, $
E → •E+(E), $/+ 
E → •int, $/+ 

0
1
int

E → int•, $/+
LR(1) DFA Construction Example

S → •E, $  
E → •E+(E), $/+  
E → •int, $/+  

E → int•, $/+  
E → int on $, +
LR(1) DFA Construction Example

\[
S \rightarrow \cdot E, \$
E \rightarrow \cdot E+(E), \$/+
E \rightarrow \cdot \text{int}, \$/+
\]

0  1

\[
E \rightarrow \text{int}, \$/+
E \rightarrow \text{int} \quad \text{on } \$, +
\]
LR(1) DFA Construction Example

S → •E, $
E → •E+(E), $/+
E → •int, $/+  

S → E•, $
E → E•+(E), $/+  

E → int•, $/+  

E → int  
on $, +

accept  
on $
LR(1) DFA Construction Example

S → \bullet E, $ 
E → \bullet E+(E), $/+ 
E → \bullet int, $/+ 

E → int\bullet, $/+ 
E → E+(E), $/+ 

S → E\bullet, $ 
E → E+(E), $/+ 

accept on $
LR(1) DFA Construction Example

S → E, $
E → E+(E), $/+ 
E → int, $/+ 

S → E, $
E → E+(E), $/+ 

E → int, $/+ 

E → E+(E), $/+ 

E → int 
on $, + 

accept 
on $
LR(1) DFA Construction Example

S → •E, $
E → •E+(E), $/+ 
E → •int, $/+ 

S → E•, $
E → E•+(E), $/+ 

E → int•, $/+ 
E → E+(E), $/+ 
E → int, $/+ 

E → int on $, + 

E → E+(E), $/+ 
E → int, $/+ 

accept on $
LR(1) DFA Construction Example

S → •E, $  
E → •E+(E), $/+  
E → •int, $/+  

0  
E → int•, $/+  

1  
E → int  
on $, +  

2  
S → E•, $  
E → E•+(E), $/+  

3  
E → E+(E), $/+  

4  
E → •E+(E), $/+  
E → •E+(E), )/+  
E → •int, )/+  

accept  
on $
LR(1) DFA Construction Example

S → •E, $  
E → •E+(E), $/+
E → •int, $/+  

0

E → int•, $/+  

1

E → E+•(E), $/+  

3

(E)

E → E+(•E), $/+  
E → •E+(E), )/+  
E → •int, )/+  

4

int

E

E → int•, )/+  

5
LR(1) DFA Construction Example

S → •E, $  
E → •E+(E), $/+  
E → •int, $/+  

E → int•, $/+  
E → E+•(E), $/+  

E → E+(E), $/+  
E → •E+(E), )/+  
E → •int, )/+  

E → int•, )/+  
E → int on ), +  

S → E•, $  
E → E•+(E), $/+  

accept on $
LR(1) DFA Construction Example

S → •E, $
E → •E+(E), $/+ 
E → •int, $/+ 

E → int•, $/+ 
E → int, $/+ 
E → int, $/+ 

E → E+(E•), $/+ 
E → E•+(E), $/+ 

E → E+(E•), $/+ 
E → E•+(E), )/+ 
E → •int, )/+ 

E → E+(•E), $/+ 
E → •E+(E), )/+ 

E → int•, )/+ 
E → int, )/+ 

E → int on $, +
E → int on ), +

and so on...
• Identify the United Kingdom poet associated with one of these verses:

He was my North, my South, my East and West,
My working week and my Sunday rest
My noon, my midnight, my talk, my song;
I thought that love would last forever, I was wrong.

Turning and turning in the widening gyre
The falcon cannot hear the falconer;
things fall apart; the centre cannot hold;
Mere anarchy is loosed upon the world,
The blood-dimmed tide is loosed, and everywhere
The ceremony of innocence is drowned;
The best lack all conviction, while the worst
Are full of passionate intensity.
In a 1995 Disney movie that has been uncharitably referred to as "Hokey-Hontas", the Stephen Schwartz lyrics "what I love most about rivers is: / you can't step in the same river twice" refer to the ideas of which Greek philosopher?
Leading Pop Culture Question

• Name either one of:
  - The *Star Trek: The Next Generation* two-part cliffhanger episode in which the Borg attack Earth and Picard becomes Locutus.
  - The bubblegum rock *Hannah Montana* main theme, performed by Miley Cyrus, that showcases the protagonist's double life.
Philosophy

- In his 1710 French work *Essais de Théodicée sur la bonté de Dieu, la liberté de l'homme et l'origine du mal*, Gottfried Leibniz (yes, the Calculus one) coined this term to describe his solution to the problem of evil in theodicy:

  - If God is omnibenevolent, omnipotent and omniscient, how do we account for the suffering and injustice that exist?
Natural Language, Poetry

• This metrical line is commonly used in verse. The rhythm involves a fixed number of double-syllable “feet”, as in this example by Keats:

<table>
<thead>
<tr>
<th></th>
<th>/</th>
<th></th>
<th>/</th>
<th></th>
<th>/</th>
<th></th>
<th>/</th>
<th></th>
<th>/</th>
</tr>
</thead>
</table>
| To| swell| the| gourd,| and| plump| the| ha-zel| shells
LR Parsing Tables. Notes

• Parsing tables (= the DFA) can be constructed automatically for a CFG
  - “The tables which cannot be constructed are constructed automatically in response to a CFG input. You asked for a miracle, Theo. I give you the L-R-1.” - Hans Gruber, Die Hard

• But we still need to understand the construction to work with parser generators
  - e.g., they report errors in terms of sets of items

• What kind of errors can we expect?
PARTY CONFLICT

Sometimes, you should back down.
Shift/Reduce Conflicts

• If a DFA state contains both 
  \[ X \rightarrow \alpha \cdot a\beta, b \] and \[ Y \rightarrow \gamma\cdot, a \] 

• Then on input “a” we could either 
  - Shift into state \[ X \rightarrow \alpha a\cdot\beta, b \], or 
  - Reduce with \[ Y \rightarrow \gamma \] 

• This is called a shift-reduce conflict
Shift/Reduce Conflicts

• Typically due to *ambiguities in the grammar*

• Classic example: the dangling else

  \[ S \rightarrow \text{if } E \text{ then } S \mid \text{if } E \text{ then } S \text{ else } S \mid \text{OTHER} \]

• Will have DFA state containing

  \[ [S \rightarrow \text{if } E \text{ then } S, \text{else}] \]
  \[ [S \rightarrow \text{if } E \text{ then } S \text{ else } S, \text{x}] \]

• If *else* follows then we can shift or reduce

• Default (bison, CUP, etc.) is to shift
  - Default behavior is as needed in this case
More Shift/Reduce Conflicts

• Consider the ambiguous grammar
  \[ E \rightarrow E + E \mid E \ast E \mid \text{int} \]

• We will have the states containing
  \[
  [E \rightarrow E \ast \bullet E, +] \quad [E \rightarrow E \ast E \bullet, +] \\
  [E \rightarrow \bullet E + E, +] \Rightarrow^E [E \rightarrow E \bullet + E, +] \\
  \ldots \quad \ldots
  \]

• Again we have a shift/reduce on input +
  - We need to reduce (* binds more tightly than +)
  - Solution: declare the precedence of * and +
More Shift/Reduce Conflicts

- In bison declare precedence and associativity:
  
  ```
  %left +
  %left *
  // high precedence
  ```

- Precedence of a rule = that of its last terminal
  - See bison manual for ways to override this default

- Resolve shift/reduce conflict with a shift if:
  - no precedence declared for either rule or terminal
  - input terminal has higher precedence than the rule
  - the precedences are the same and right associative
Using Precedence to Solve S/R Conflicts

• Back to our example:

\[
\begin{align*}
[E \rightarrow E \ast \bullet E, \ +] & \quad [E \rightarrow E \ast E\bullet, \ +] \\
[E \rightarrow \bullet E + E, \ +] & \Rightarrow^E [E \rightarrow E \bullet + E, \ +] \\
\ldots & \quad \ldots
\end{align*}
\]

• Will choose \textbf{reduce} on input $+$ because precedence of rule $E \rightarrow E \ast E$ is higher than of terminal $+$
Using Precedence to Solve S/R Conflicts

- Same grammar as before
  \[ E \rightarrow E + E \mid E \ast E \mid \text{int} \]

- We will also have the states

\[ [E \rightarrow E + \bullet E, +] \quad [E \rightarrow E + E\bullet, +] \]
\[ [E \rightarrow \bullet E + E, +] \Rightarrow^E \quad [E \rightarrow E \bullet + E, +] \]
\[ \ldots \quad \ldots \]

- Now we also have a shift/reduce on input $+$
  - We choose **reduce** because $E \rightarrow E + E$ and $+$ have the same precedence and $+$ is left-associative
Using Precedence to Solve S/R Conflicts

• Back to our dangling else example
  
  \[
  [S \rightarrow \text{if } E \text{ then } S\text{•}, \quad \text{else}]
  
  [S \rightarrow \text{if } E \text{ then } S\text{•} \text{ else } S, \quad x]
  \]

• Can eliminate conflict by declaring \textbf{else} with higher precedence than \textbf{then}
  
  - Or just rely on the default shift action

• But this starts to look like “hacking the parser”

• Avoid overuse of precedence declarations or you’ll end with unexpected parse trees
  
  - The kiss of death ...
Reduce/Reduce Conflicts

- If a DFA state contains both
  \[X \rightarrow \alpha \bullet, a\] and \[Y \rightarrow \beta \bullet, a\]
  - Then on input “a” we don’t know which production to reduce

- This is called a reduce/reduce conflict
Reduce/Reduce Conflicts

• Usually due to **gross ambiguity** in the grammar

• Example: a sequence of identifiers

  \[
  S \rightarrow \varepsilon \mid \text{id} \mid \text{id}\ S
  \]

• There are **two parse trees** for the string **id**

  \[
  S \rightarrow \text{id}
  
  S \rightarrow \text{id} \ S \rightarrow \text{id}
  \]

• How does this confuse the parser?
More on Reduce/Reduce Conflicts

• Consider the states

  \[ [S' \rightarrow \bullet S, \ \$] \quad [S \rightarrow \bullet, \ \$] \quad [S \rightarrow \bullet \ id, \ \$] \quad [S \rightarrow \bullet \ id \ S, \ \$] \quad [S \rightarrow \bullet \ id \ S, \ \$] \]

  \[ [S' \rightarrow \bullet \ id \ S, \ \$] \quad [S \rightarrow \bullet \ id \ S, \ \$] \quad [S \rightarrow \bullet \ id \ S, \ \$] \quad [S \rightarrow \bullet \ id \ S, \ \$] \quad [S \rightarrow \bullet \ id \ S, \ \$] \]

• Reduce/reduce conflict on input \$ 

  \[ S' \rightarrow S \rightarrow id \]

  \[ S' \rightarrow S \rightarrow id \ S \rightarrow id \]

• Better rewrite the grammar:  \[ S \rightarrow \varepsilon \ | \ id \ S \]
Can’s someone learn this for me?

No, you can't have a neural network
Using Parser Generators

- **Parser generators** construct the parsing DFA given a CFG
  - Use precedence declarations and default conventions to **resolve conflicts**
  - The parser algorithm is the same for all grammars (and is provided as a library function)

- But most parser generators do not construct the DFA as described before
  - *Why might that be?*
Using Parser Generators

• **Parser generators** construct the parsing DFA given a CFG
  - Use precedence declarations and default conventions to **resolve conflicts**
  - The **parser algorithm is the same** for all grammars (and is provided as a library function)

• But most parser generators do not construct the DFA as described before
  - Because the LR(1) parsing DFA has 1000s of states even for a simple language
LR(1) Parsing Tables are Big

• But many states are similar, e.g.

\[
\begin{align*}
E &\rightarrow \text{int} \bullet, \, $/+ \\
E &\rightarrow \text{int} \\
&\quad \text{on } $, + \\
E &\rightarrow \text{int} \bullet, \, )/+ \\
&\quad \text{on } ), +
\end{align*}
\]

and

\[
\begin{align*}
E &\rightarrow \text{int} \\
&\quad \text{on } $, +, )
\end{align*}
\]

• Idea: **merge** the DFA states whose items differ only in the lookahead tokens
  - We say that such states have the same core

• We obtain
The Core of a Set of LR Items

• Definition: The core of a set of LR items is the set of first components
  - Without the lookahead terminals

• Example: the core of

\[
\{ [X \rightarrow \alpha \rightarrow \beta, b], [Y \rightarrow \gamma \rightarrow \delta, d]\}
\]

is

\[
\{X \rightarrow \alpha \rightarrow \beta, Y \rightarrow \gamma \rightarrow \delta\}
\]
LALR States

• Consider for example the LR(1) states
  \[\{[X \rightarrow \alpha \cdot, a], [Y \rightarrow \beta \cdot, c]\}\]
  \[\{[X \rightarrow \alpha \cdot, b], [Y \rightarrow \beta \cdot, d]\}\]

• They have the **same core** and can be merged

• And the merged state contains:
  \[\{[X \rightarrow \alpha \cdot, a/b], [Y \rightarrow \beta \cdot, c/d]\}\]

• These are called **LALR(1)** states
  - Stands for LookAhead LR
  - Typically 10x fewer LALR(1) states than LR(1)
LALR(1) DFA

• **Repeat** until all states have distinct core
  - Choose two distinct states with same core
  - *Merge* the states by creating a new one with the union of all the items
  - *Point* edges from predecessors to new state
  - *New state* points to all the previous successors

![Diagram of LALR(1) DFA](image)
Example LALR(1) to LR(1)
The LALR Transform Can Introduce Conflicts

- Consider for example the LR(1) states

  \{[X \rightarrow \alpha \bullet, a], [Y \rightarrow \beta \bullet, b]\}

  \{[X \rightarrow \alpha \bullet, b], [Y \rightarrow \beta \bullet, a]\}

- And the merged LALR(1) state

  \{[X \rightarrow \alpha \bullet, a/b], [Y \rightarrow \beta \bullet, a/b]\}

- Has a new reduce-reduce conflict

- In practice such cases are rare
LALR vs. LR Parsing

• LALR languages are not natural
  - They are an efficiency hack on LR languages

• Any “reasonable” programming language has a LALR(1) grammar

• LALR(1) has become a standard for programming languages and for parser generators
A Hierarchy of Grammar Classes

From Andrew Appel, “Modern Compiler Implementation in Java”
Notes on Parsing

- Parsing
  - A solid foundation: context-free grammars
  - A simple parser: LL(1)
  - A more powerful parser: LR(1)
  - An efficiency hack: LALR(1)
  - LALR(1) parser generators

- Now we move on to semantic analysis
Take a bow, you survived!
Supplement to LR Parsing

Strange Reduce/Reduce Conflicts Due to LALR Conversion (from the bison manual)
Strange Reduce/Reduce Conflicts

• Consider the grammar

\[ S \rightarrow P \ R \ , \quad NL \rightarrow N \ | \ N \ , \ NL \]
\[ P \rightarrow T \ | \ NL : T \quad R \rightarrow T \ | \ N : T \]
\[ N \rightarrow \text{id} \quad T \rightarrow \text{id} \]

• \( P \) - parameters specification
• \( R \) - result specification
• \( N \) - a parameter or result name
• \( T \) - a type name
• \( NL \) - a list of names
Strange Reduce/Reduce Conflicts

- In P an id is a
  - N when followed by , or :
  - T when followed by id
- In R an id is a
  - N when followed by :
  - T when followed by ,
- This is an LR(1) grammar.
- But it is not LALR(1). Why?
  - For obscure reasons
A Few LR(1) States

LALR reduce/reduce conflict on ","
What Happened?

- Two distinct states were confused because they have the same core
- Fix: add dummy productions to distinguish the two confused states
- E.g., add

  \[ R \rightarrow \text{id} \text{bogus} \]

  - \text{bogus} is a terminal not used by the lexer
  - This production will never be used during parsing
  - But it distinguishes \text{R} from \text{P}
A Few LR(1) States After Fix

Different cores ⇒ no LALR merging
Homework

• WA2 Due
• Midterm 1 in Class