Lancaster University

Unit 7: Bottom-Up Parsing General Introduction

SCC 312 Compilation

Aims

- Before we start our examination of LR parsing, we wish to introduce the general concept of bottom-up parsing
- The concept of
  - reduction
  - use of the parse stack

Bottom-Up Parsing

- Bottom-Up parsers work by reducing a sentence of the language to the sentence symbol by successively applying productions of the grammar.
  - **(1) S → real IDLIST**
  - **(2) IDLIST → IDLIST, ID**
  - **(3) IDLIST → ID**
  - **(4) ID → A | B | C | D**

- for simplicity, (4) is thought of as a single production

Parse Stack

- The technique uses a “parse stack”. Each symbol read by the parser is immediately placed on top of the parse stack.
- The parse stack is operated on by grammar reductions.
- The sentence “real A, B, C” belongs to the language and might be parsed in the following bottom-up manner.

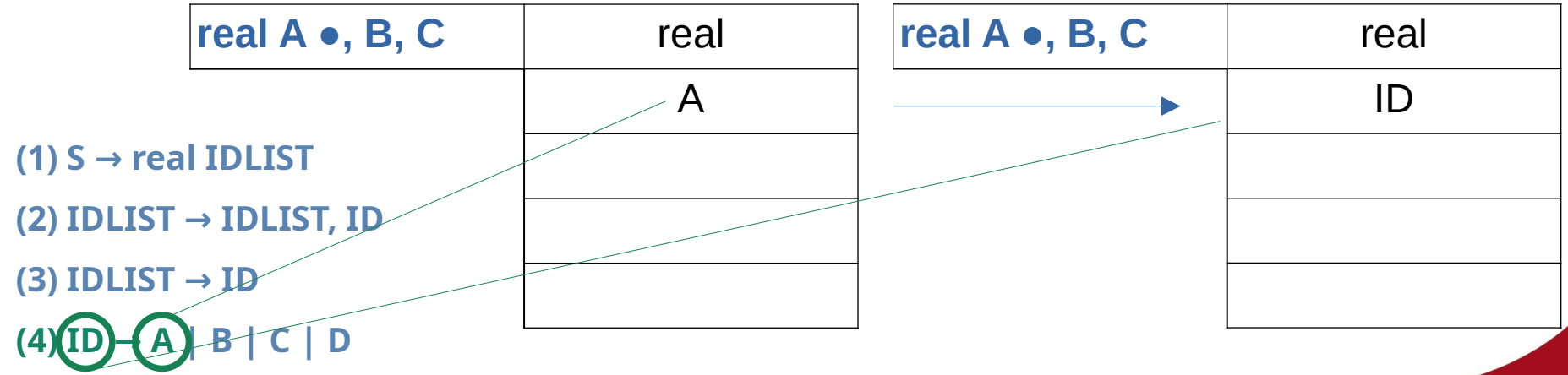
Example

- “real” is read and put on the stack.
- This move can be represented diagrammatically. The dot is placed immediately after the last symbol read, and the stack shows it holds one entry, “real”.

real ● A, B, C	real

Example

- We read **A** and put it on the stack.
- The next move of the parser is to replace “A” (on the stack) with “ID” using production (4).
- This is known as **reduction**.



Example

- In bottom-up parsing, productions are applied the other way round from top-down parsing i.e. right-hand sides of productions are replaced by their corresponding left-hand side rather than vice-versa

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- Production (3) is then used to perform another reduction.

real A ●, B, C	real
	ID

real A ●, B, C	real
	IDLIST

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- Now the parser reads another symbol

real A, ● B, C
real
IDLIST
comma

- ...and another

real A, B ●, C
real
IDLIST
comma
B

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- Now we can perform a reduction using production (4)

real A, B ●, C	real
	IDLIST
	comma
	B

real A, B ●, C	real
	IDLIST
	comma
	ID

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- And a further reduction using production (2)

real A, B ●, C	real
	IDLIST
	comma
	ID

real A, B ●, C	real
	IDLIST

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- Now two more symbols are read and stacked

real A, B, ● C	real
	IDLIST
	comma

real A, B, C ●	real
	IDLIST
	comma
	C

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- Now we can perform a reduction using production (4)

real A, B, C ●	real
	IDLIST
	comma
	C

real A, B, C ●	real
	IDLIST
	comma
	ID

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- Now we can perform a reduction using production (2)

real A, B, C ●	real
	IDLIST
	comma
	ID

real A, B, C ●	real
	IDLIST

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- Finally, we're at the end of the sentence with no more input to read; here we look for any further reductions and find we can use production (1)

real A, B, C ●	real
	IDLIST

real A, B, C ●	S

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Example

- The parse is complete when the stack contains only the distinguished symbol (S) and the complete input sentence has been read; if these conditions are not mutually true something has gone wrong

real A, B, C ●	real
	IDLIST

real A, B, C ●	S

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Two types of “move”

- shift: a symbol is read and added to the stack
- reduce: the top “n” elements of the stack are replaced by some non-terminal of the grammar using a production (where “n” is the amount of elements on the right hand side of the production)
- We seek a deterministic parser, but problems can arise :
  - in a particular situation, do we “shift” or “reduce”? Or
  - might we have more than one possible “reduce” move available?

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Ambiguity

- Here is a situation from earlier in our example
- We chose to leave things as they are and read the next token.

- ...but the contents of the stack match production (1), so why not reduce by that rule?

real A ●, B, C	real
	IDLIST

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Ambiguity

- We also had this situation earlier; at this stage we can reduce by either production (2) or production (3)
- We chose production (2); using production (3) would have resulted in a remaining stack which cannot be parsed by our grammar

real A, B ●, C	real
	IDLIST
	comma
	ID

(1) S → real IDLIST

(2) IDLIST → IDLIST, ID

(3) IDLIST → ID

(4) ID → A | B | C | D

Learning Outcomes

- You should now understand the use of the parse stack and the reduction technique as used in bottom-up parsing
- However, you should have (at least!) a couple of unresolved questions!
  - how do we decide which rule to use in a reduction?
  - how do we decide when to read the next token?
- You will see when we look at a particular bottom-up parsing technique, LR(0), next.