Graham Cooper

June 29th, 2015

<u>**Choice:**</u>

1. Shift symbol from input to stack
2. reduce top of stack by grammar rule

Theorem (Donald Knuth, 1965)

- create tex, became latex - currently writing the art of computer programming
<u>**Theorem:**</u>
The set {wa | ∃x, S' $\implies$ *wax }
w is the stabck
a is the next characer
is a regular language.

If it is a regular language, then it can be described by a DFA.
- use the DFA to make shift/reduce decisions

LR Parsing

- left-to-right scan
- rightmost derivations

<u>**Example:**</u> S' $\rightarrow \vdash$ E $\dashv$
E $\rightarrow$ E + T
E $\rightarrow$ T
T $\rightarrow$ id
See pretty picture 1 in notes

LR(0) machine (simplest)

<u>**Definition:**</u> An item is a production with a dot ($\cdot$) somewhere on the right hand side.
(indiates partially completed rule)

- begin with a state with the starting state production rule and the dot at the beginning of input
- Label an arc with the symbol that follows the dot ; advance the dot in the next state
- If the dot precedes a non-terminal A, add al productions with A on the LHS to the state, dot in the leftmost position.
- Repeat until we get all of the transitions we can

Using the machine

- Start in the start state with empty stack
- Shifting
  - shift char from input to stack
  - follow transiition for that char to next state
  - if no transition, error, or reduce
- Reducing
  - "Reduce states" have only one item and the dot is rightmost
  - called a complete item
  - reduce by the rule in the state
  - reduce: pop RHS off the stack, backgrack size(RHS) states in the DFA, push LHS, follow shift transition for the LHS
- Backtracking the DFA
  - must remember the DFA states
  - Push the DFA states onto the stack as well.

Stack	Read	unread	Action
1	ϵ	$\vdash$id+id+id$\dashv$	S2 (shift and go to 2)
1 $\vdash$ 2	$\vdash$	id+id+id$\dashv$	S6 (Shift and go to 6)
1 $\vdash$ 2 id 6	$\vdash$id	+id+id$\dashv$	RT $\rightarrow$ id (Pop 1 symbol and 1 state) Now in state 2, Push T g
1 $\vdash$ 2 T 5	$\vdash$id	+id+id$\dashv$	R, E $\rightarrow$ T pop 1 sym, pop 1 state, push E goto 3
1 $\vdash$ 2 E 3	$\vdash$id	+id+id$\dashv$	S7
1 $\vdash$ 2 E 3 + 7	$\vdash$ id +	id+id$\dashv$	S6
1 $\vdash$ 2 E 3 + 7 + 6	$\vdash$ id + id	+id$\dashv$	R, T $\rightarrow$ id, goto 8
1 $\vdash$ 2 E 3 + 7 T 8	$\vdash$ id+id	+id$\dashv$	R, E $\rightarrow$ E + T (Pop 3 sym, 3 states, goto 2)
1 $\vdash$ 2 E 3	$\vdash$id + id	+id $\dashv$	S7
1 $\vdash$ 2 E 3 + 7	$\vdash$id + id +	id $\dashv$	S6
1 $\vdash$ 2 E 3 + 7 id 6	$\vdash$id+id+id	$\dashv$	R, T $\rightarrow$ id, goto 8
1 $\vdash$ 2 E 3 + 7 T 8	$\vdash$id+id+id	$\dashv$	R, E $\rightarrow$ E + T goto3
1 $\vdash$ 2 E 3	$\vdash$id+id+id	$\dashv$	S4
1 $\vdash$ 2 E 3 $\dashv$ 4	$\vdash$id+id+id$\dashv$	ϵ	Accept

What can go wrong?
What if the state looks like this:
A $\rightarrow \alpha\cdot$ c β
B $\rightarrow \gamma\ \cdot$
Shift c or reduce B $\rightarrow \gamma$
This is a shift-reduce conflict!

A $\rightarrow \alpha\ \cdot$
B $\rightarrow \beta\ \cdot$
reduce-reduce conflict

Whenever a complete item A $\rightarrow \alpha\cdot$ is not alone in a state there is a conflict and the grammar is not LR(0)

Example: Right-associative
S' $\rightarrow \vdash$ E $\dashv$
E $\rightarrow$ T + E
E $\rightarrow$ T

$T \rightarrow id$

See pretty picture 2 in notes

Example:
$\epsilon 1$ shift $->$ $\vdash 2$ shift $->$ $\vdash$id 6 reduce $->$ $\vdash$ T 5

Should we reduce E$\rightarrow$ T?
Depends: If input is $\vdash$ id $\dashv$ then YES, otherwise, no!

Add a lookahead to fix the conflict

For each $A \rightarrow \alpha \cdot$, attach Follow(A)
Follow(E) = $\{\dashv\}$
Follow(A) = $\{+, \dashv\}$

Interpretation:
A reduce action
$A \rightarrow \alpha\cdot$ X (X = follow(A))
only applies if the next char is in X.

So $E \rightarrow T \cdot$ applies when the next char is $\dashv$
$E \rightarrow T \cdot + E$ applies when next char is +

Conflict resolved!
Result is called an SLR(1) parser. = Simple LR with 1 char lookahead

SLR(1) resolves many, but not all conflicts

LR(1) - more powerful than SLR(1)
- PRoduces many more states

Building a Parse Tree

Top-Down

$\dashv$ S	$S \rightarrow A$ y B
$\dashv$ B y A	keep S make the new nodes its children

Bottom-up

$\dashv$ab	Reduce $A \rightarrow$ a b
$\dashv$A	Use A as parent, make a b as children