The Parse Machine

Chris Healy
Department of Computer Science
Furman University

Rationale

- All CS students use a compiler
- Shouldn’t have to wait until 400/500 level course to see how a compiler works
  - “It’s that thing that gives me error msg about my code”
- Concept of “state” already useful, e.g. by the time students reach CS 2
- My university does not have a compiler course, and only rarely offers programming languages.
- Classic technique of using parse tables is less intuitive, and there is no standard notation for how to go backwards.

What is a parse machine?

- Similar to Deterministic PDA, drawn like an FA
- Parse stack: maintain a history of visited states
- States: determined from all possible positions the cursor could be in while reading input w.r.t. grammar
  - Special states called "Reduce Blocks" tell you to go backwards.
- Transitions: Go to the next state based on current terminal or nonterminal in the input.
  - Crash/reject if unspecified → "syntax error"
  - When you "reduce", you insert a nonterminal into the input.

“Reduce”

- One important concept is reducing a nonterminal.
- Once we have read some input, we may have just finished an important part of the input.
  - This happens when the cursor is at the <u>end</u> of the RHS of a production.
- Example: S → AB   A → aaa   B → bb
- When we read the string “aaabb”…
  - We arrive at aaa • bb. We can reduce the “aaa” to A to obtain A • bb.
  - When we arrive at Abb •, we can reduce the “bb” to B to obtain AB •.
  - Knowing that we just read AB, we can reduce it to S •.

Example

- First example machine that I show my class uses this grammar.

  $S \rightarrow 0A0$

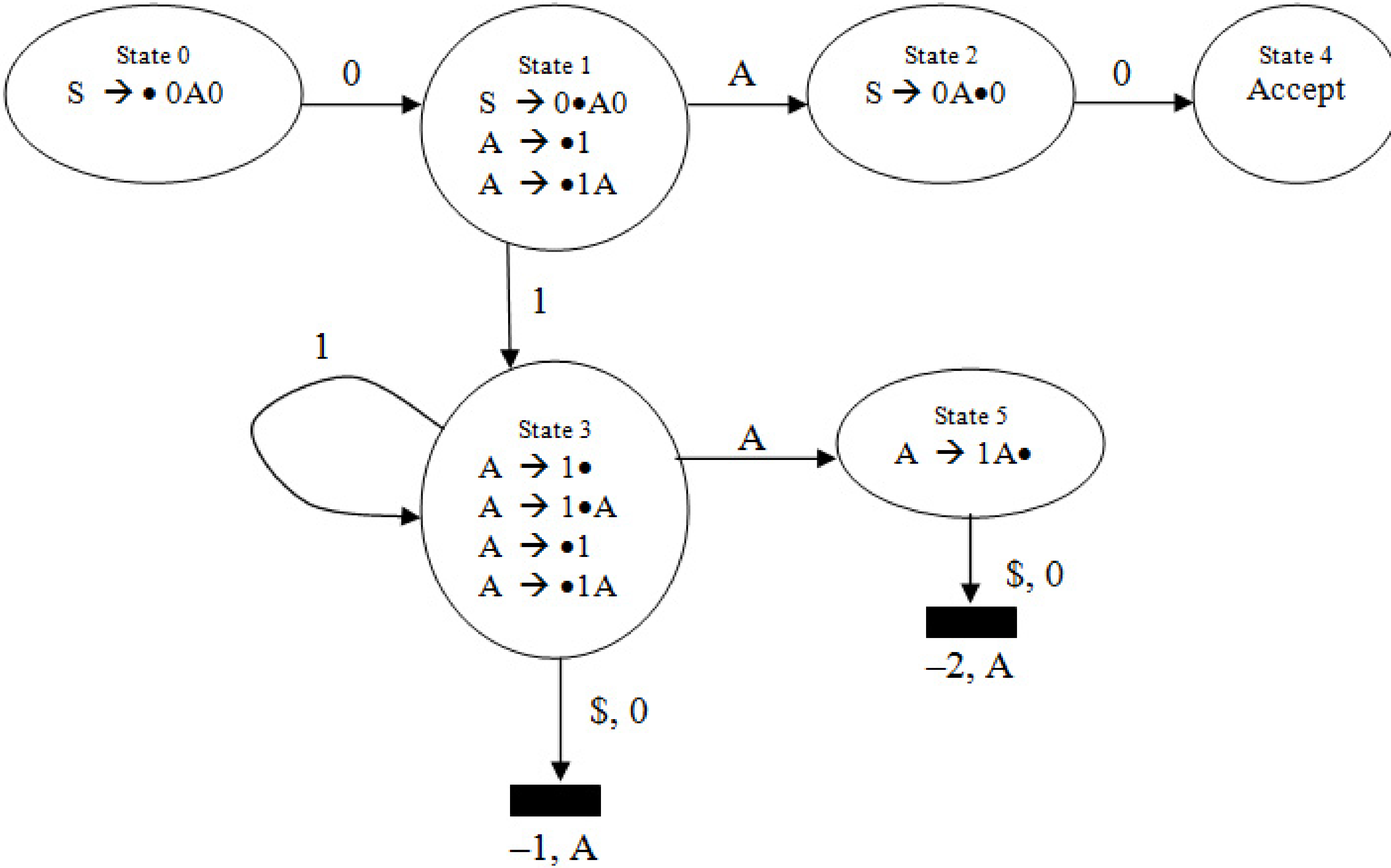
  $A \rightarrow 1 \mid 1A$

- There are a total of 6 states
  - The start state is $S \rightarrow \bullet 0A0$
  - One state is the accept state

Creating states

Creating states for the example parse machine.

State	Items	Go to state	On input(s)
0	S → • 0 A 0	1	0
1	S → 0 • A 0 A → • 1 A → • 1 A	2 3 3	A 1 1
2	S → 0 A • 0	4	0
3	A → 1 • A → 1 • A A → • 1 A → • 1 A	Reduce 5 3 3	$, 0 A 1 1
4	S → 0 A 0 •	Accept	$
5	A → 1 A •	Reduce	$, 0


State 0
S → • 0A0
0
State 1
S → 0•A0
A → •1
A → •1A
A
State 2
S → 0A•0
0
State 4
Accept
1
1
State 3
A → 1•
A → 1•A
A → •1
A → •1A
A
State 5
A → 1A•
$, 0
–2, A
$, 0
–1, A

Example with Input

- Suppose we want to parse 0110 with the grammar $S \rightarrow 0A0$ and $A \rightarrow 1 \mid 1A$.
- The steps in the trace are as follows.

State Stack	Input String	Next State
0	• 0 1 1 0	1
0 1	0 • 1 1 0	3
0 1 3	0 1 • 1 0	3
0 1 3 3	0 1 1 1 • 0	Need to backtrack: –1, A
0 1 3	0 1 1 • A 0	5
0 1 3 5	0 1 1 A • 0	Need to backtrack: –2, A
0 1	0 1 1 A • A 0	2
0 1 2	0 1 1 A A • 0	4 (accept state)

Fitting into a course

- Unit on bottom-up parsing takes 1 week
- Currently in our Computational Theory course
- Pre-requiste ideas
  - Helpful to know basic phases of compilation: scanning, parsing, code generation
  - Simple CFGs: can motivate with how we define mathematical expressions to enforce precedence & associativity
  - If desired: CYK algorithm to see if input string can be generated by grammar. Dynamic programming $O(n^3)$. Motivates need for efficient algorithm.

Outline of lessons

- Assumes 50-minute period
- Day 1
  - Running a parse machine. (~20 minutes)
  - Goto and reduce actions. The parse stack.
  - Introduction to <u>creating</u> parse machine: how to create the individual states. "Sets of items"
- Day 2
  - How to specify "reduce" actions.
  - Calculate the First and Follow of a nonterminal.
- Day 3: Extended example, which handles the special case where a nonterminal can generate ε.

Conclusion

- Can discuss with student what happens with real compiler
  - Hundreds of states…
  - Distinction between syntax and semantic errors
- After the unit, can talk about parse table as a convenient representation for implementation
  - Realization that many transitions are not specified!
- Goal is to gain a deeper appreciation of a programming language as defined by CFG
- Do it early in the curriculum, if you can spare a week in discrete math or CS 2.