CS406: Compilers

Spring 2021

Week 6: Parsers (LR(k)) and Semantic Processing

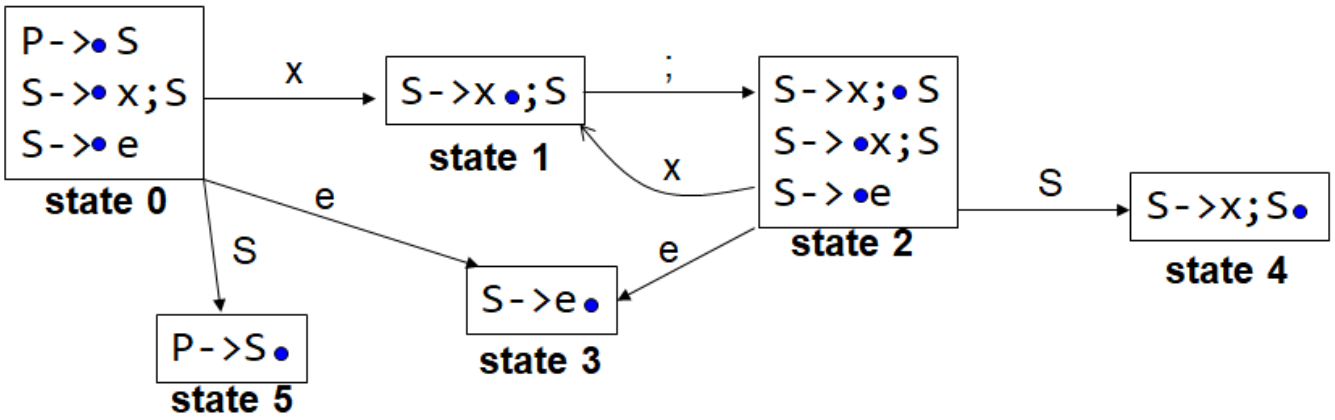
LR(0) Parsing

- Previous Example of LR Parsing was LR(0)
  - No (0) lookahead involved
  - Operate based on the parse stack state and with goto and action tables (How?)

LR(0) Parsing

- Assume: Parse stack contains α == saying that a e.g. prefix of x;x is seen in the input string

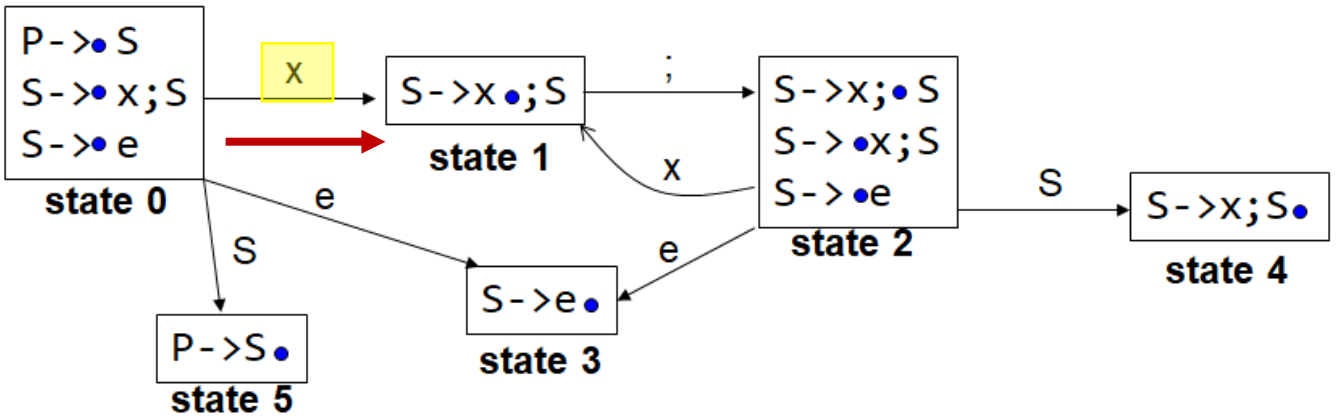
Parse Stack
0
0 1
0 1 2
0 1 2 1



LR(0) Parsing

- Assume: Parse stack contains α == saying that a prefix of x;x is seen in the input string

Parse Stack
0
0 1
0 1 2
0 1 2 1

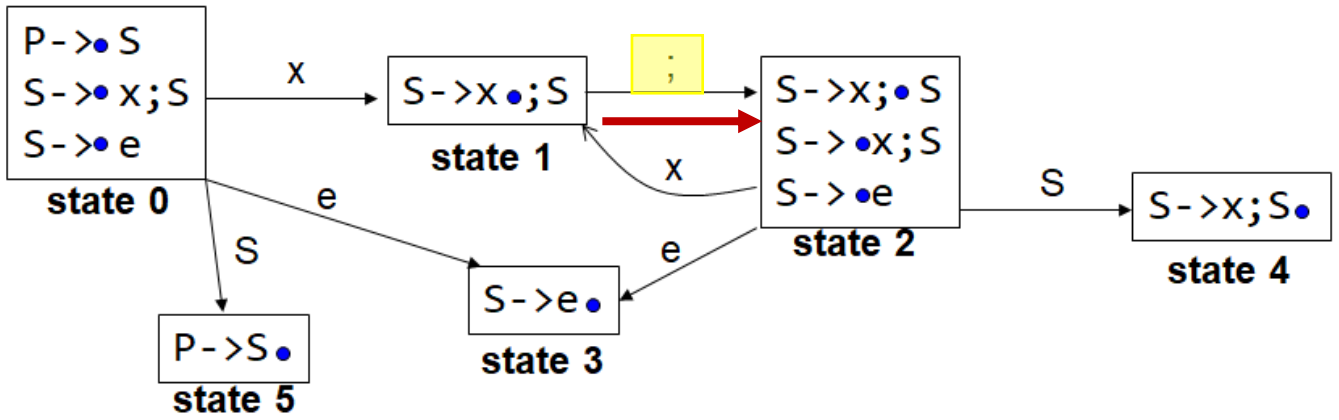


Go from state 0 to state 1 consuming x

LR(0) Parsing

- Assume: Parse stack contains α == saying that a prefix of x;x is seen in the input string

Parse Stack
0
0 1
0 1 2
0 1 2 1

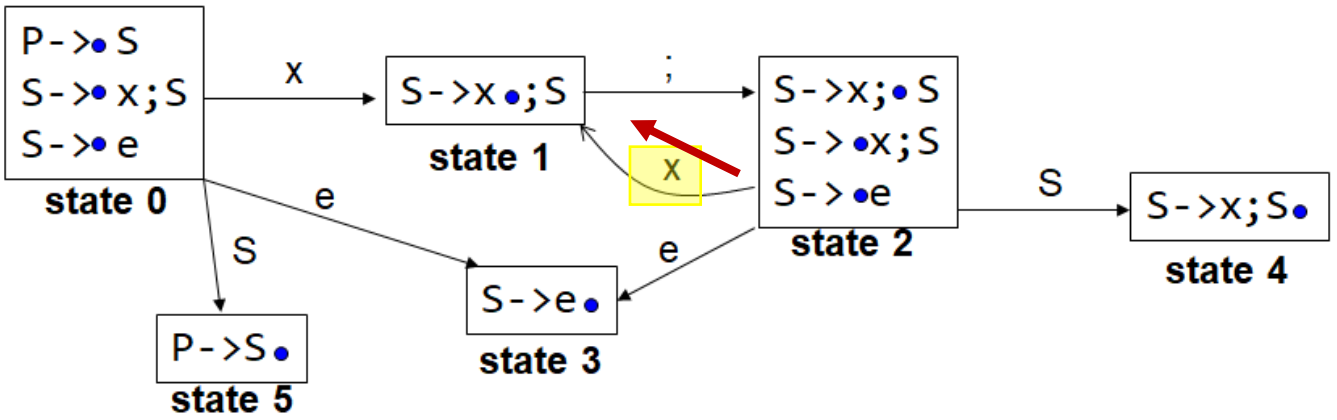


Go from state 1 to state 2 consuming ;

LR(0) Parsing

- Assume: Parse stack contains α == saying that a prefix of x;x is seen in the input string

Parse Stack
0
0 1
0 1 2
0 1 2 1



Go from state 2 to state 1 consuming x

LR(0) Parsing

- Assume: Parse stack contains α.

  => we are in some state s

LR(0) Parsing

- Assume: Parse stack contains α.

=> we are in some state s .

We reduce by X->β if state s contains X->β ●

- Note: reduction is done based solely on the current state.

LR(0) Parsing

- Assume: Parse stack contains α.

=> we are in some state s .

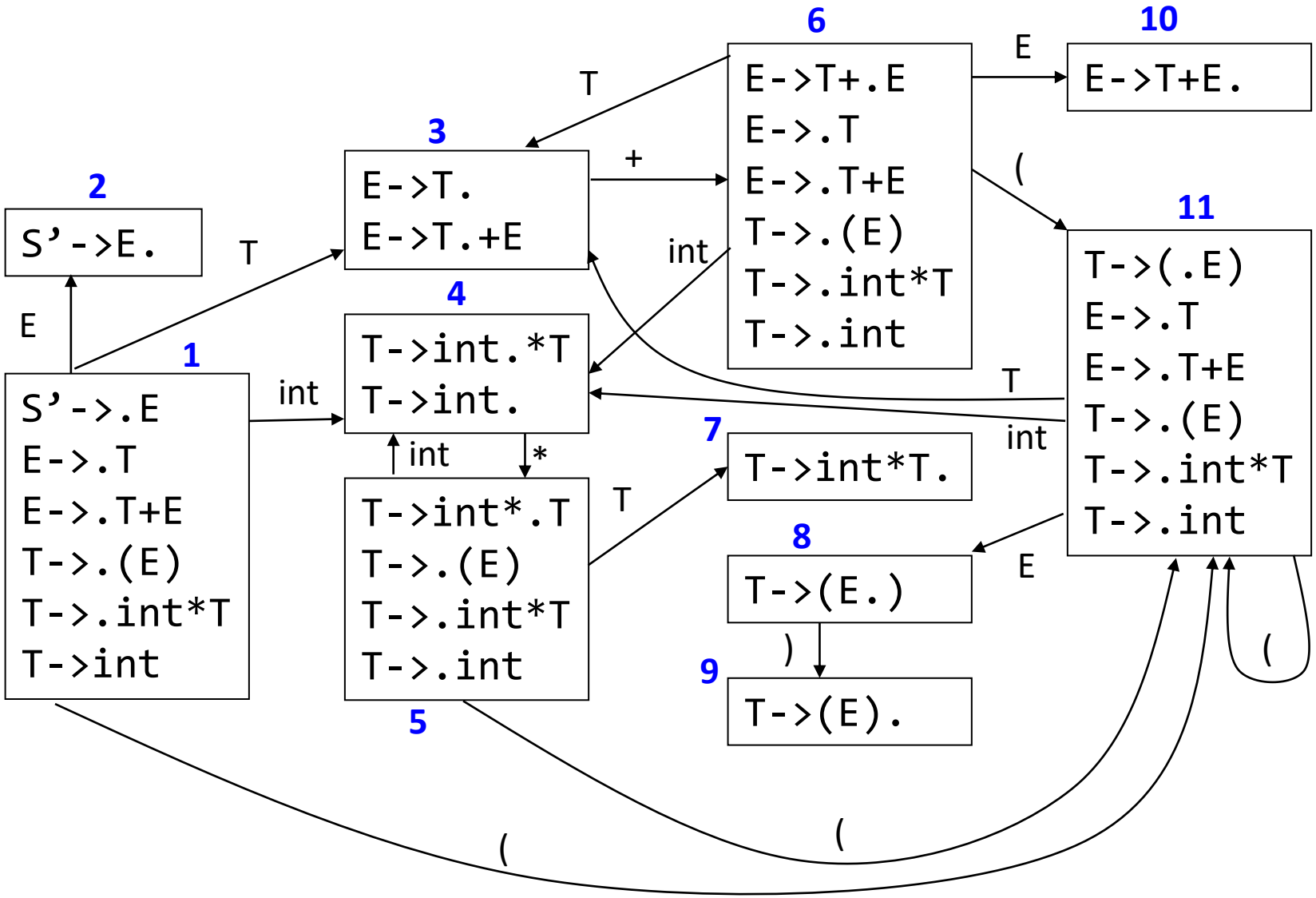
- Assume: Next input is t

We shift if s contains X->β ● tω

== s has a transition labelled t

LR(0) Parsing

- What if s contains X->β • tω and X->β• ?



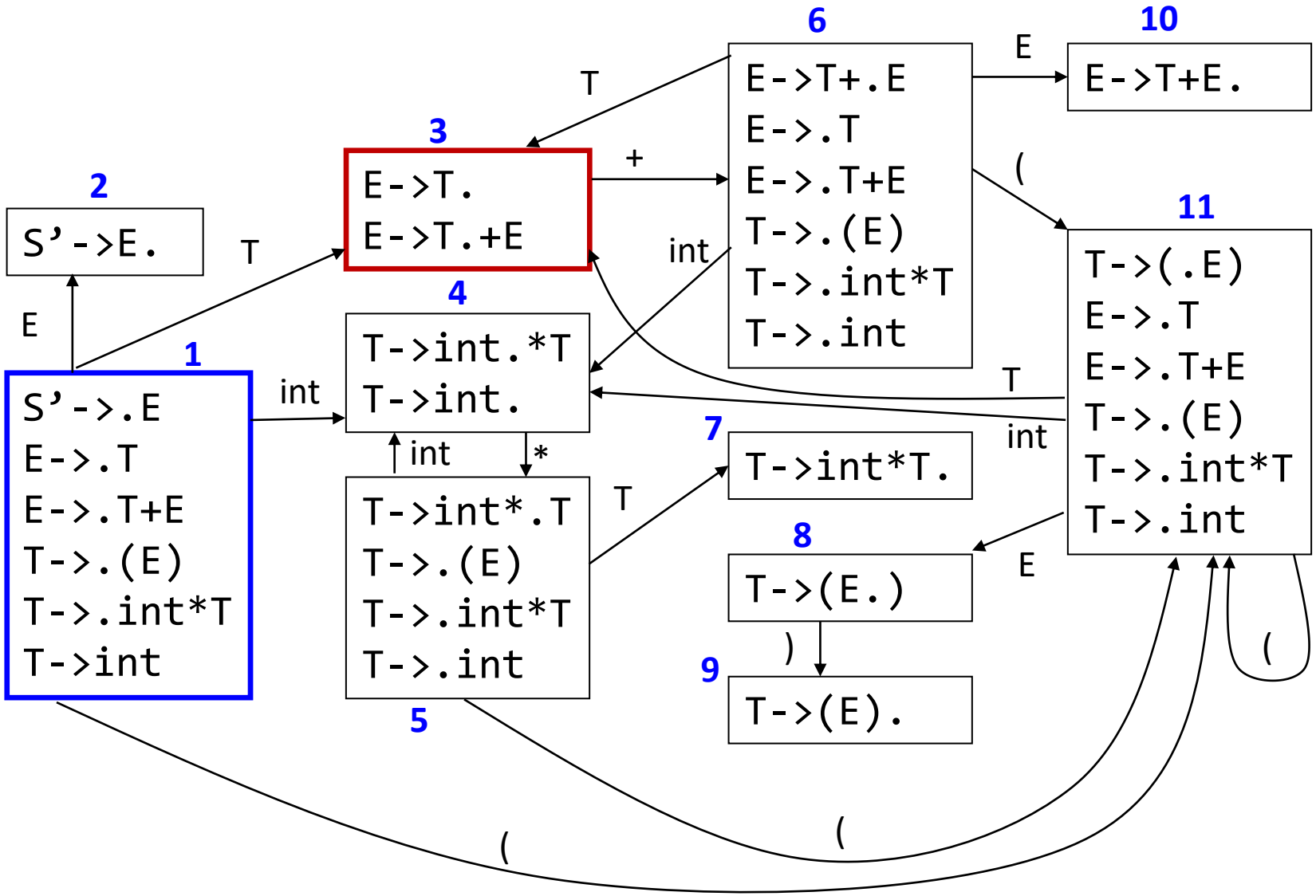
Conflicts or not?

SLR Parsing

- SLR Parsing improves the shift-reduce conflict states of LR(0):

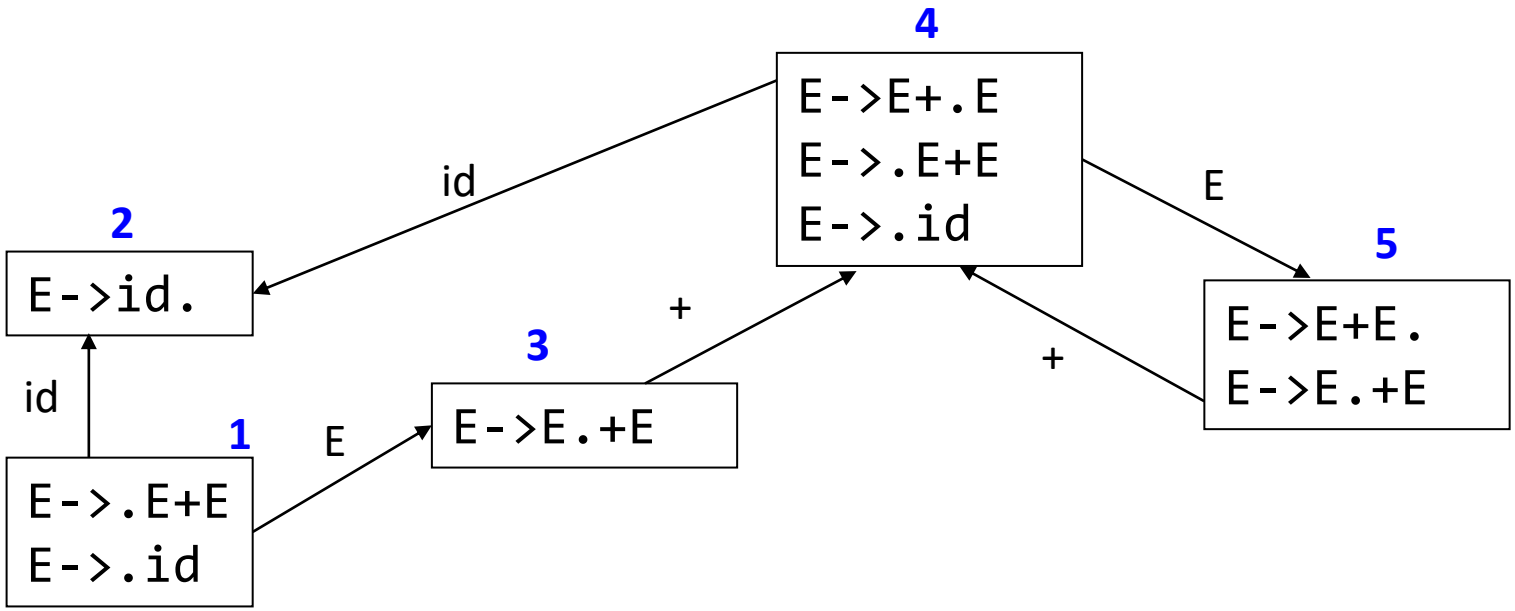
Reduce X->β● only if

t ∈ Follow(X)



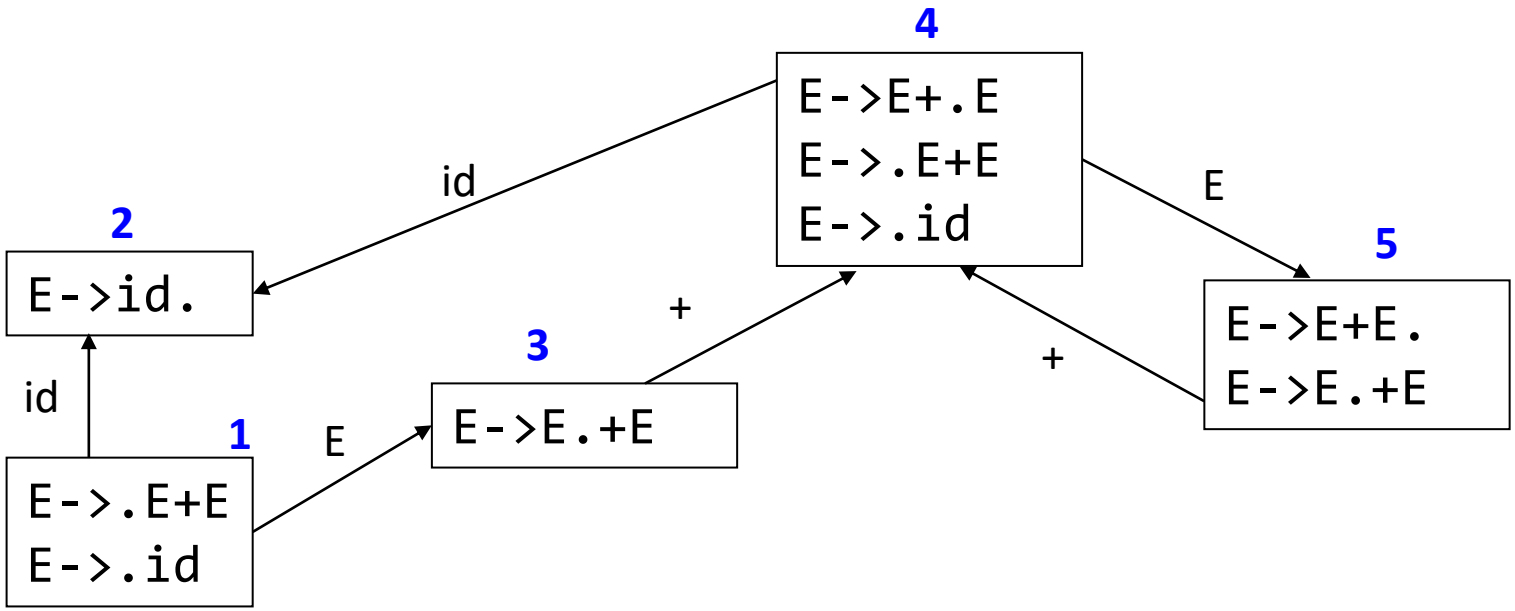
Follow(E) = { $,) } => reduce by E->T. only if next input is $ or)

lookahead 1



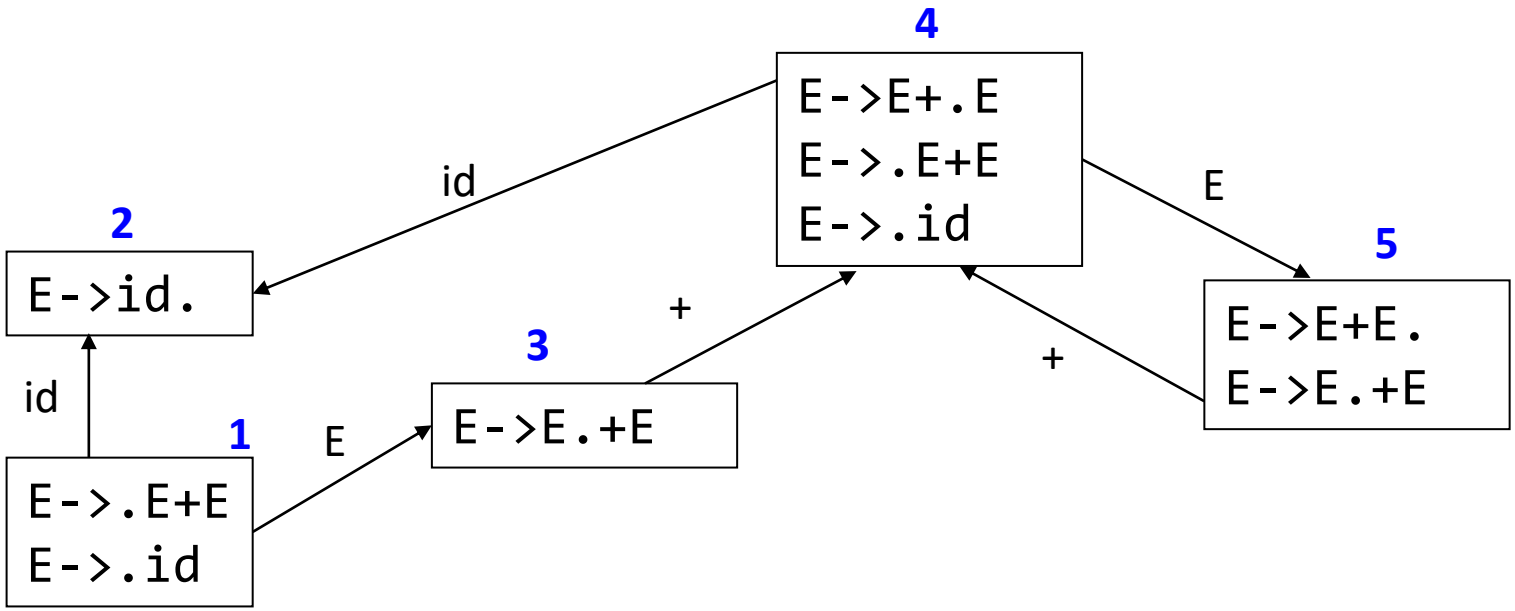
What about the grammar **E-> E + E | id** ?

LR(0)?



What about the grammar **E-> E + E | id** ?

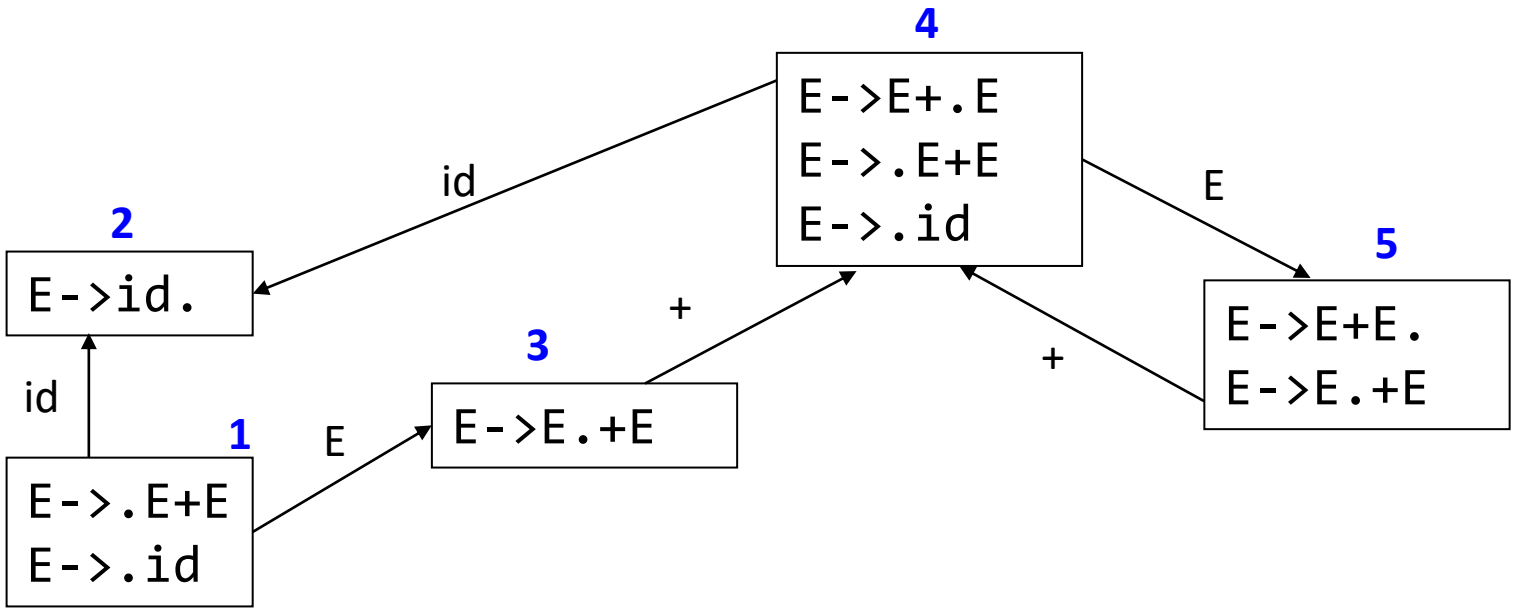
LR(0)? SLR(1)?



What about the grammar **E-> E + E | id** ?

LR(0)? SLR(1)?

Follow(E) = {+,$} => in state 5, reduce by E->T. only if <u>next input</u> is $ or +



What about the grammar **E-> E + E | id** ?

LR(0)? SLR(1)?

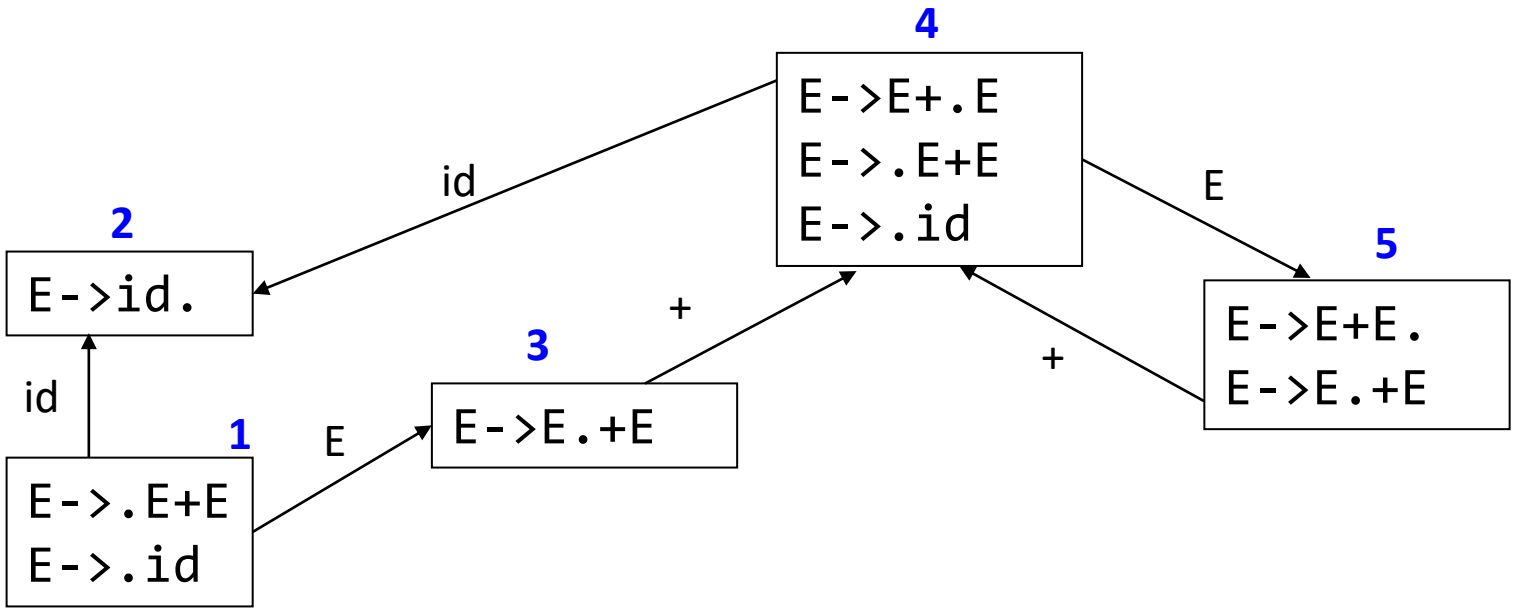
Follow(E) = {+,$} => in state 5, reduce by E->T. only if next input is $ or +

But state 5 has E->E.+E (shift if next input is +)

Shift-reduce conflict!

LR(k) parsers

- LR(0) parsers
  - No lookahead
  - Predict which action to take by looking only at the symbols currently on the stack
- LR(k) parsers
  - Can look ahead *k* symbols
  - Most powerful class of deterministic bottom-up parsers
  - LR(1) and variants are the most common parsers



What about the grammar **E-> E + E | id** ?

LR(0)? SLR(1)?

Follow(E) = {+,$} => in state 5, reduce by E->T. only if next input is $ or +

But state 5 has E->E.+E (shift if next input is +)
Shift-reduce conflict!

%left +

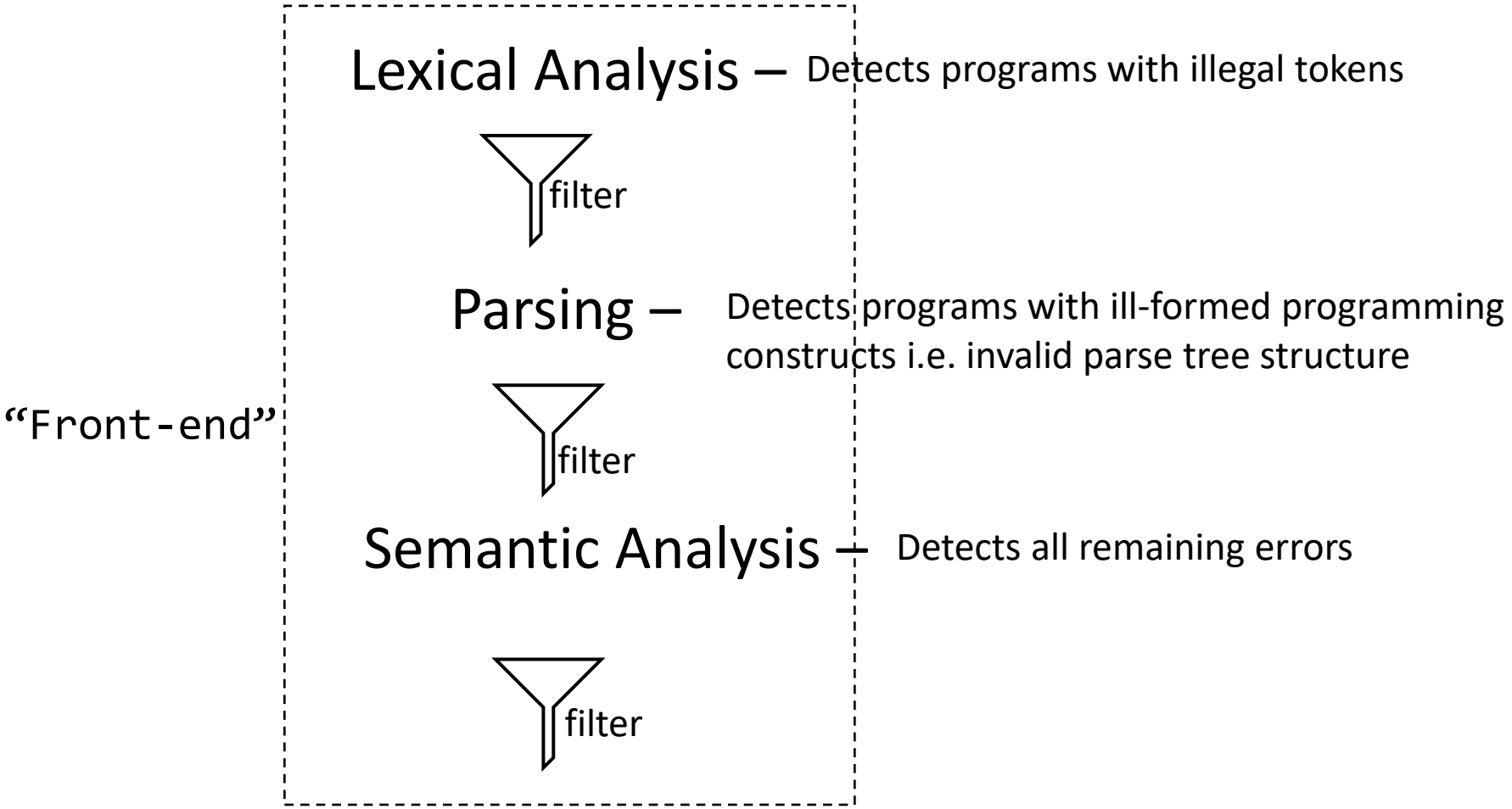
says reduce if the next input symbol is + i.e. prioritize rule E+E. over E.+E

Top-down vs. Bottom-up parsers

- Top-down parsers expand the parse tree in *pre-order*
  - Identify parent nodes before the children
- Bottom-up parsers expand the parse tree in *post-order*
  - Identify children before the parents
- Notation:
  - LL(1): Top-down derivation with 1 symbol lookahead
  - LL(k): Top-down derivation with k symbols lookahead
  - LR(1): Bottom-up derivation with 1 symbol lookahead

slide courtesy: Milind Kulkarni

Semantic Analysis



Why Semantic Analysis?

- Context-free grammars cannot specify all requirements of a language
  - Identifiers declared before use
  - Type checks

    STRING str:= “Hello”;

    str := str + 2;
  - Misuse of keywords
  - A Class is declared only once in a OO language
  - etc.