Compilation 2024

LR parsing

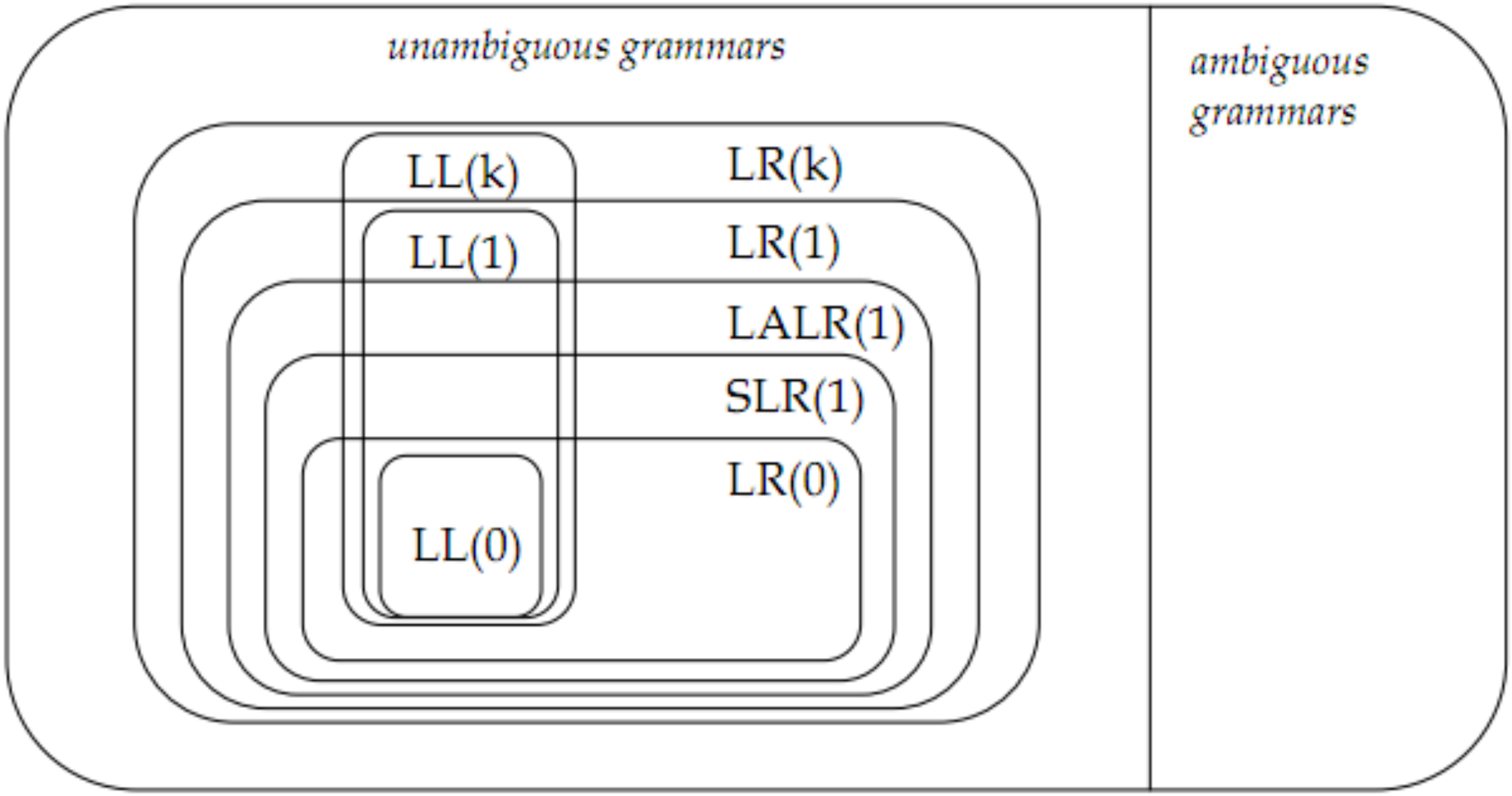
Amin Timany
timany@cs.au.dk

Revised from slides by Aslan Askarov

Recall

- The phase of compilation after lexing
- Using context-free grammars to convert a sequence of tokens in to an AST
- We focus on nice grammars (unambiguous)
  - We saw LL(1) parsing, i.e., recursive descent
- Today: LR parsing and parsing tools

Grammar containment



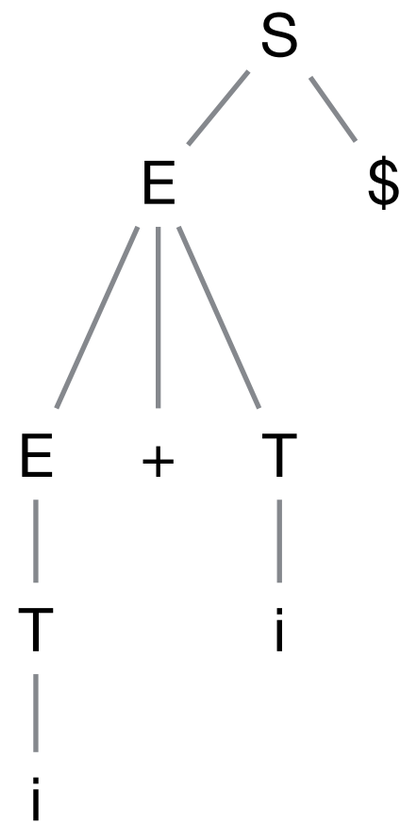
Quick example

- Let us consider the following grammar and string

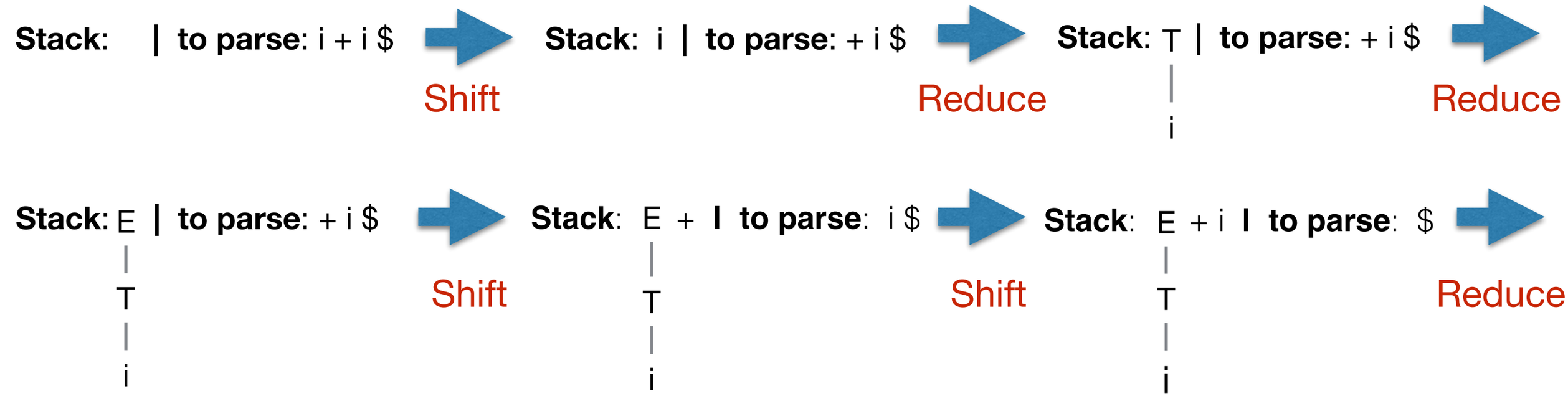
Grammar:

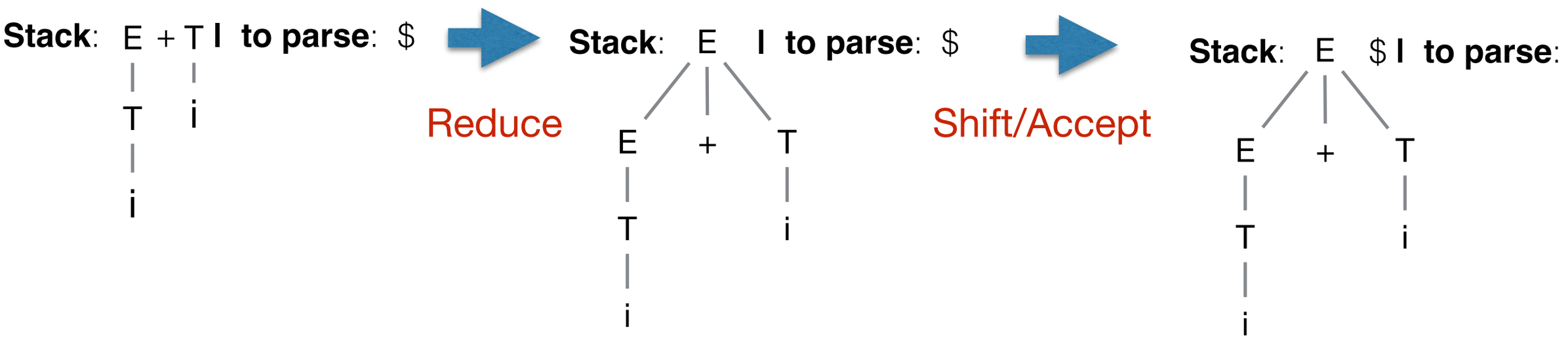
```
S -> E $
E -> T
E -> E + T
T -> i
T -> ( E )
```

String: i + i $



Quick example





How do we know when to shift and when to reduce? LR parsing states and automaton

LR Terminology

- Item — a *hypothesis* about the sub-derivation: N is the hypothesis, α has been already confirmed to be parsed, β is to be confirmed $N \to \alpha \,.\, \beta$
- Item is reducible if β is empty
- Item set — a collection of items

- Closure of an item set: add new hypotheses to the set if expecting a non-terminal; repeat until we reach a fixed point

- We start parsing in the state that is the closure of hypotheses corresponding to the start symbol (see example)

LR (0)

- Stack based: obs the stack of alternating components: item sets and derivations
- Reductions do not depend on the input
- Shifting: updating items
- Reducing: popping from the stack and replacing by the new (larger) sub derivation
- Conflicts
  - shift/reduce
  - reduce/reduce

LR (0)

Grammar:

```
S -> E $
E -> T
E -> E + T
T -> i
T -> ( E )
```

Start state: `Closure of {S -> .E $}`

Which is:

```
{S -> .E $,
 E -> .T
 E -> .E + T
 T -> .i
 T -> .( E )}
```

LR(0) example

http://users-cs.au.dk/askarov/dovs/lr/lr0basic.html

Question: perform LR(0) parsing on the following:

Grammar:

```
S -> E $
E -> T
E -> E + T
T -> i
T -> ( E )
T -> i [ E ]
```

String: `i [ i ] $`

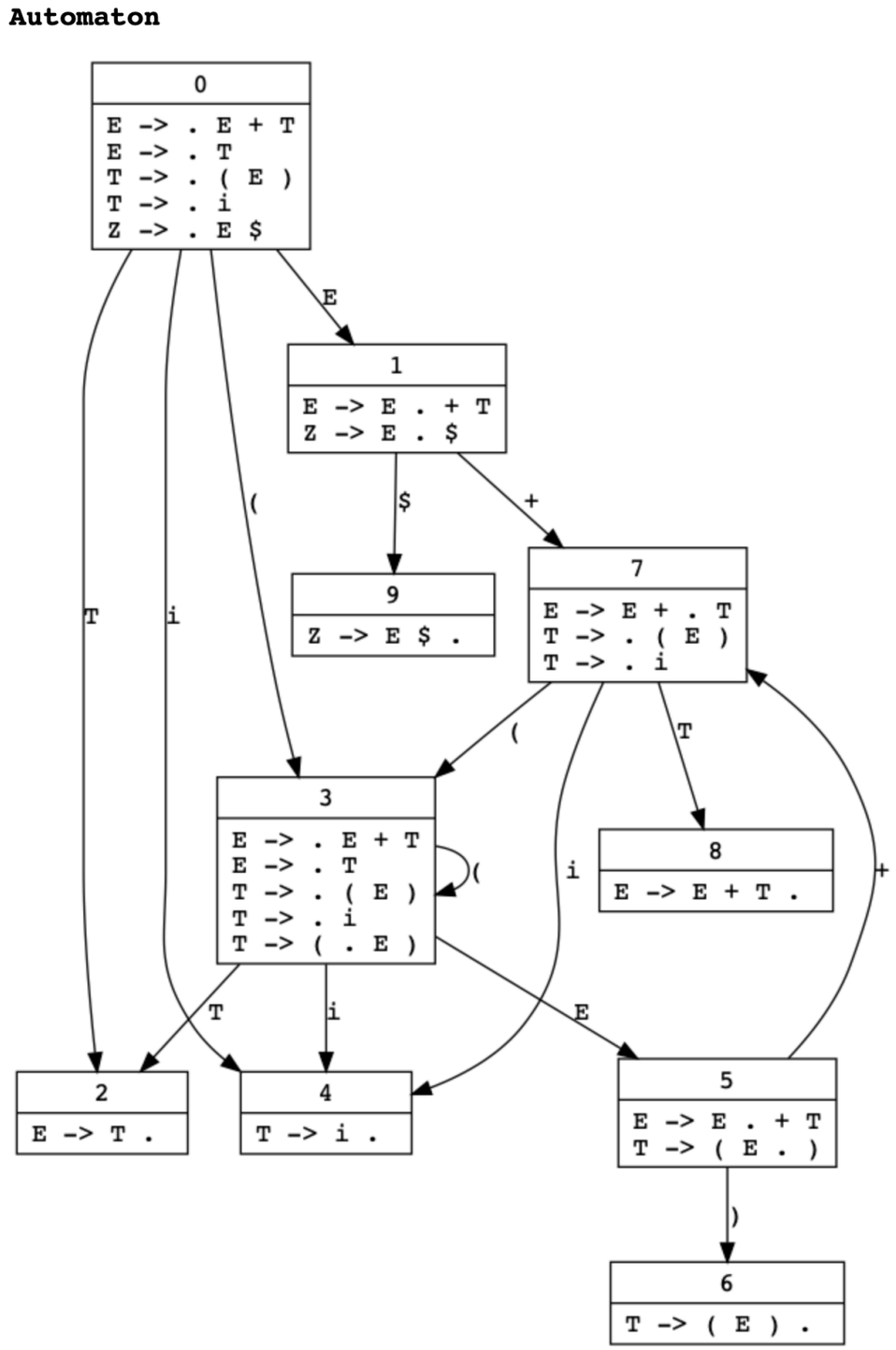
LR(0) example with conflict

http://users-cs.au.dk/askarov/dovs/lr/lr0shift_reduce.html

LR(0) automaton

- Idea: precompute all the states (item sets)
- Conflicts discovered at the time of automaton creation
- A grammar is LR(0) if there are no conflicts
- There are few useful grammars that are LR(0)

Question: Which states are reduce states and what do they have in common?



LR (1)

- Reduction based on a lookahead
- Fewer shift/reduce conflicts

- Idea: Items are a pair of hypothesis and *a lookahead set*

- We start with the same item set as LR(0), *with the empty lookahead set*

- The closure, operation, when processing hypothesis $(N \rightarrow \alpha \, . \, A\beta, L)$, adds all the hypotheses corresponding to non-terminal A with the lookahead set $first(\beta L)$

Let us LR(1) parse the following

Grammar:

```
E -> T
E -> E + T
T -> i
T -> ( E )
T -> i [ E ]
```

String: `i [ i ] $`

LR(1) examples

http://users-cs.au.dk/askarov/dovs/lr/

Compilation 2024

Parsing Tools

Amin Timany
timany@cs.au.dk

Revised from slides by Aslan Askarov

Language of arithmetic expressions

Example: 1 + (2*x- 3)

CFG:

```
expr -> id | num | expr op expr | ( expr )
op   -> plus | minus | times | div
```

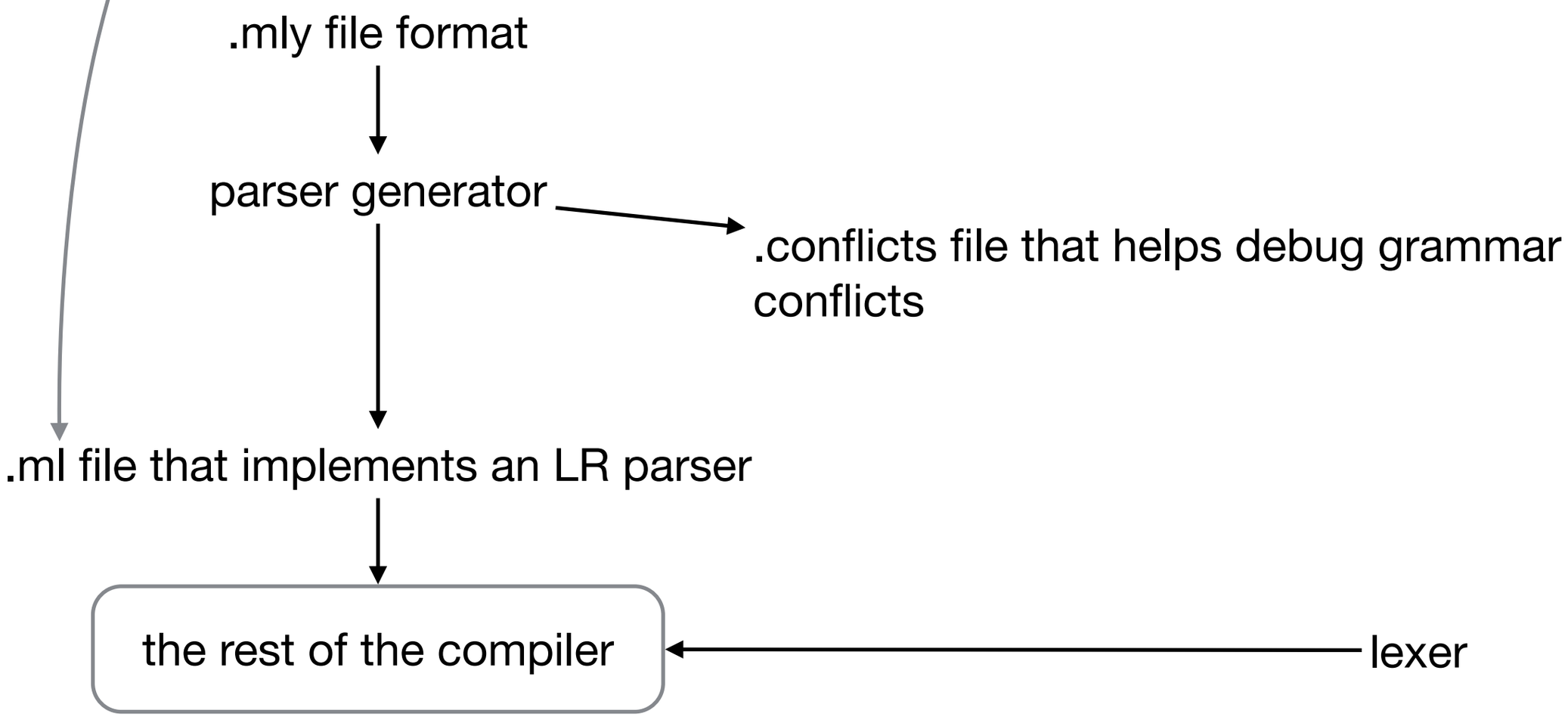
ML code:

```
type aexp = Id of string
          | Number of int
          | Op of binop * aexp * aexp
and binop = Plus | Minus | Times | Div
```

?

Using menhir

```
%{
ML declarations to be copied verbatim into the generated parser
%}
parser declarations (nonterminals, terminals, precedence, etc)
%%
grammar rules
```



Operator associativity

- Consider expression 7 - 5 - 3
- What should be the default interpretation?

(7 - 5) - 3

Say that operator MINUS has *left associativity*

7 - (5 - 3)

… *right associativity*

error

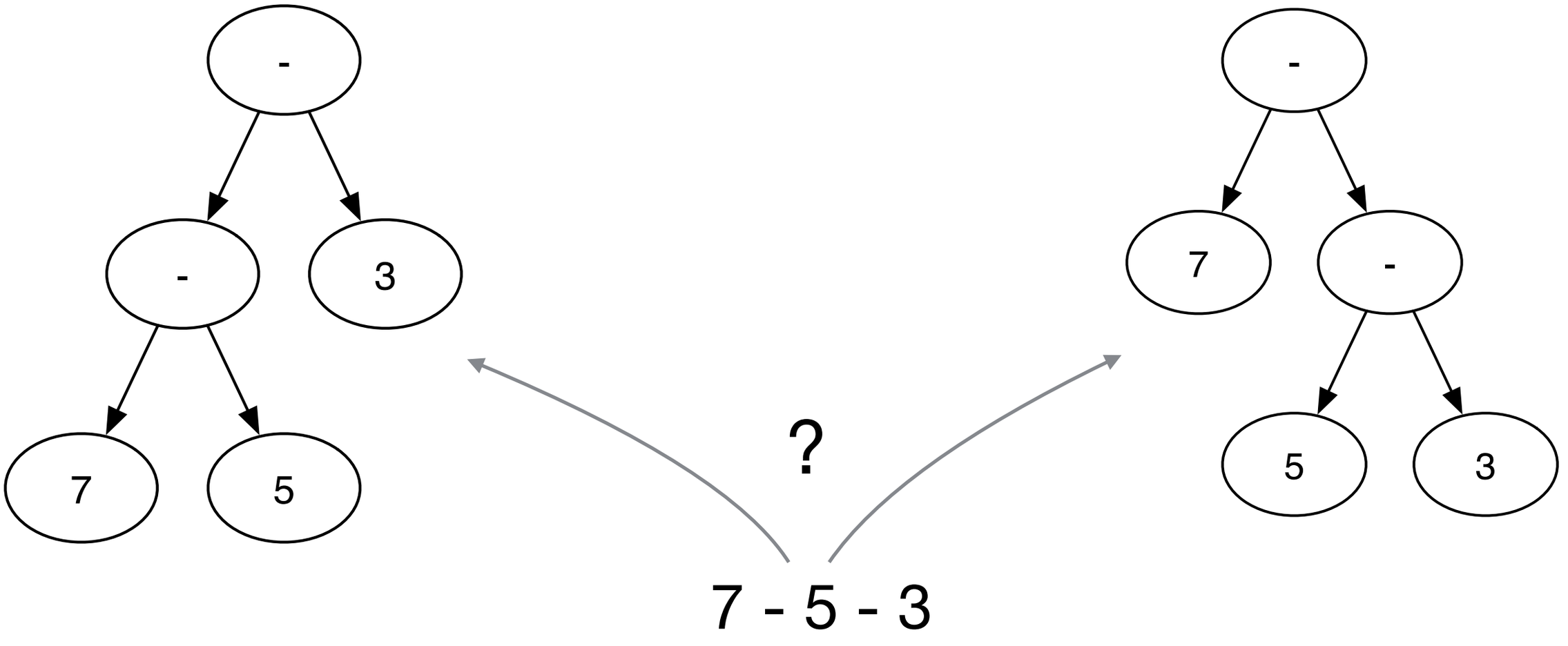
also a possibility

- For arithmetic MINUS the *correct* choices are either left associativity or error, but not right associativity
  - The *desired* choice is left associativity

Examples of associativity

Operators	**Associativity**	
Arithmetic minus, division	Left	
Arithmetic addition, multiplication	Left/Right	*a+b+c = (a+b)+c = a+(b+c)*
Assignment in C: x = y = 5	Right	*y is assigned the value of 5; x is assigned the updated value of y*
Arithmetic exponentiation 2^1^3	Right	*2^1^3 = 2^ (1^3) = 2* *not the same as (2^1)^3 = 8*
Comparison operators 2 < 3 < 4	Non-associative	*parse error*

What can parse trees tell about associativity?



left-associative parse tree

right-associative parse tree

Grammar conflicts

- Shift/reduce conflicts
  - Typical fixes
    - try specifying associativity/precedence
    - otherwise rewrite grammar
- Reduce/reduce conflicts
  - Rarely a good sign: must rewrite grammar

Example shift/reduce conflict

```
exp : exp . MINUS exp
exp : exp MINUS exp .  (reduce by rule 4)
```

Scenario 1

stack:	*input string*
exp MINUS exp	MINUS exp

Example shift/reduce conflict

```
exp : exp . MINUS exp
exp : exp MINUS exp .  (reduce by rule 4)
```

Scenario 1

stack:	*input string*	
exp MINUS exp MINUS	exp	*shift action*

Scenario 2

stack:	*input string*
exp MINUS exp	MINUS exp

Example shift/reduce conflict

```
exp : exp . MINUS exp
exp : exp MINUS exp .  (reduce by rule 4)
```

Scenario 1

stack:	*input string*	
exp MINUS exp MINUS	exp	*shift action*

Scenario 2

stack:	*input string*	
exp	MINUS exp	*reduce action*

Example shift/reduce conflict

- Possible fix: specifying associativity using%left or %right directive
  - %right favors shifting
  - %left favors reducing

- **Q:** What should we do for MINUS?

- Other fixes: *precedence* is given by the order of parser directives in the .mly file

- If not enough: rework the grammar!