Crafting a Compiler with C (XI)

資科系
林偉川

LL(1) Grammar and parser

- LL(1) grammar is the class of CFG and is suitable for RDP. Define LL(1) parsers which use an LL(1) parse table rather than recursive procedures to control parsing
- A parser generator takes as input any of a class of grammars and produces as output a parser that will correctly parse the language defined by that grammar ➔ concept of a compiler is input the high-level definitions are translated into executable forms

2

LL(1) Grammar and parser

- Build a compiler – a grammar is written and fed to an automatic parser generator
- Changing a parser is easy to create from an updated grammar
- RDP use parsing procedures to match the tokens generated by non-terminals
- The main problem in building a parsing procedure is deciding which production to match

3

LL(1) Grammar and parser

- This decision can be formalized by defining a Predict function that examine the look-ahead symbol to deduce the production that must be used to expand each non-terminal
- Consider $A \rightarrow X_1X_2\ldots X_m$, $m \geq 0$. we need to compute the set of possible look-ahead tokens that might indicate that this production is to be matched.
- $First(X_1X_2\ldots X_m) = X_1$ if this is a terminal. If X_1 is a non-terminal, then compute a First set for each right-hand side corresponding to X_1. If X_1 can generate λ then $First(X_1X_2\ldots X_m)$ depend on X_2 etc.

4

Predict function

- What if $X_1X_2\ldots X_m$ can produce λ then the look-ahead is determined by those terminals that can follow the left-hand side Follow(A)
- Predice($A \rightarrow X_1X_2\ldots X_m$)=

  if $\lambda \in$ First($A \rightarrow X_1X_2\ldots X_m$)

  (First($A \rightarrow X_1X_2\ldots X_m$) - λ) U Follow(A)

  else

  First($A \rightarrow X_1X_2\ldots X_m$)

5

Conflict of predict function

- Any token that can be the first symbol produced by the right-hand side of a production will predict that production
- If the entire right-hand side can produce λ, then tokens that can immediately follow the left-hand side of a production will also predict that production. Because λ is not a terminal symbol, it cannot be a look-ahead and is not included in any Predict set ((First($A \rightarrow X_1X_2\ldots X_m$) - λ))

6

Conflict of predict function

- t ∈ predict(A➔$X_1X_2\ldots X_m$) and t ∈ predict(A➔$Y_1Y_2\ldots Y_n$), this conflict will exclude the grammar from the LL(1) grammar ➔ pair-wise disjoint!!!
- LL(1) contains exactly those grammar that have disjoint predict sets for productions that share a common left-hand side
- Not all grammars are LL(1). Many grammars that are not LL(1) belong to other grammar classes for which parsers can be automatically constructed

7

Compute the First set

To compute First(X) for all grammar symbols X, we can have the following algorithm:

1. **If X is terminal, then FIRST(X) is { X }**
2. **If X is non-terminal and X → aα is a production, add a to FIRST(X). If X → λ is a production, then add λ to FIRST(X).**
3. **If X→$Y_1Y_2\ldots Y_k$ is a production, then for all i ∋ $Y_1Y_2\ldots Y_{i-1}$ are non-terminals and FIRST(Y_j) contains λ for j=1,2,…i-1 (i.e. $Y_1Y_2\ldots Y_{i-1}$ =>* λ), add every non-λ symbol in FIRST(Y_j) to FIRST(X). If λ is in FIRST(Y_j) for j=1,2,…k, then add λ to FIRST(X).**

8

Compute the Follow set

To compute Follow(A) for all non-terminals A, we can have the following algorithm:

1. **λ is in Follow(A), where A is the start symbol**
2. **If there is a production A→ αBβ , β ≠ λ, then everything in First(β) but λ is in Follow(B)**
3. **If there is a production A→ αB, or a production A→ αBβ where First(β) contains λ (i.e. β =>* λ), then everything in Follow(A) is in Follow(B)**

9

The LL Grammar Class

E→E+T | T , T→T*F | F , F→ (E) | id

Can be converted into

E→TE' , E' → +TE' | λ
T→FT' , T' → *FT' | λ
F→(E) | id

Or

E→T | TE' , E'→+T | +TE'
T→F | FT' , T'→*F | *FT'
F→ (E) | id

10

Compute the Follow & First set

E→TE' **First(E)=First(T)=First(F)={ (, id }**

E' → +TE' | λ First(E')= { +, λ }, First(T')= { *, λ }

T→FT' Follow(E)=Follow(E')={), λ } by rule 1,2

T' → *FT' | λ Follow(T)=Follow(T')={+,), λ }

➔ First(E')-{λ} U Follow(E') by rule 3

F→(E) | id

Follow(F)={+, *,), λ }

➔First(T')-{λ} U Follow(T') by rule 3

11

EBNF Original Grammar

Start: <system goal> → <program> SCANEOF

1. <program> → begin <statement list> end
2. <statement list> → <statement> {<statement>}
3. <statement> → ID := <expression>;
4. <statement> → read (<id list>);
5. <statement> → write (<expr list>);
6. <id list> → ID {, ID}
7. <expr list> → <expression> {, <expression>}
8. <expression> → <primary> { <add op> <primary>}
9. <primary> → (<expression>)
10. <primary> → ID
11. <primary> → INTLITERAL
12. <add op> → PLUSOP
13. <add op> → MINUSOP
14. <system goal> → <program> SCANEOF

12

Reduced LL(1) Grammar

Start: <system goal> → <program> $

1. **<program> → begin <statement list> end**
2. **<statement list> → <statement><statement tail>**
3. **<statement tail> → <statement><statement tail>**
4. **<statement tail> → λ**
5. **<statement> → ID := <expression>;**
6. **<statement> → read (<id list>);**
7. **<statement> → write (<expr list>);**
8. **<id list> → ID<id tail>**
9. **<id tail> → , ID<id tail>**
10. **<id tail> → λ**
11. **<expr list> → <expression> <expre tail>**
12. **<expr tail> → , <expression><expr tail>**
13. **<expr tail> → λ**
14. **<expression> → <primary><primary tail>**
15. **<primary tail> → <add op> <primary><primary tail>**
16. **<primary tail> λ**
17. **<primary> → (<expression>)**
18. **<primary> → ID**
19. **<primary> → INTLITERAL**
20. **<add op> → PLUSOP**
21. **<add op> → MINUSOP**
22. **<system goal> → <program>$**

13

Computed First Set

Nonterminal	First set
<program> production 1	**{begin}**
<statement list> production 2	**{ID, read, write}**
<statement> production 5,6,7	**{ID, read, write}**
<statement tail> production 3,4	**{ID, read, write, λ}**
<expression> production 14	**{ID, INTLIT, (}**
<id list> production 8	**{ID}**
<expr list> production 11	**{ID, INTLIT, (}**

14

Computed First Set

Nonterminal		First set
<id tail>	production 9,10	**{COMMA,λ}**
<expr tail>	production 12,13	**{COMMA,λ}**
<primary>	production 17,18,19	**{ID, INTLIT, (}**
<primay tail>	production 15,16	**{+, -, λ}**
<add op>	production 20,21	**{+, -}**
<system goal>	production 22	**{begin}**

15

Reduced LL(1) Grammar

Start: <system goal> → <program> $

1. **<program> → begin <statement list> end**
2. **<statement list> → <statement><statement tail>**
3. **<statement tail> → <statement><statement tail>**
4. **<statement tail> → λ**
5. **<statement> → ID := <expression>;**
6. **<statement> → read (<id list>);**
7. **<statement> → write (<expr list>);**
8. **<id list> → ID<id tail>**
9. **<id tail> → , ID<id tail>**
10. **<id tail> → λ**
11. **<expr list> → <expression> <expr tail>**
12. **<expr tail> → , <expression><expr tail>**
13. **<expr tail> → λ**
14. **<expression> → <primary><primary tail>**
15. **<primary tail> → <add op> <primary><primary tail>**
16. **<primary tail> →λ**
17. **<primary> → (<expression>)**
18. **<primary> → ID**
19. **<primary> → INTLITERAL**
20. **<add op> → PLUSOP**
21. **<add op> → MINUSOP**
22. **<system goal> → <program>$**

16

Computed Follow Set

Nonterminal	Follow set
<program> production 1	{$}
<statement list> production 2	{end} in production 1
<statement> production 5,6,7	{ID, read, write, end} **First(<stat. tail>) -{λ} U Follow(<stat. list>)**
<statement tail> production 3,4	{end} Follow(<stat. list>)
<expression> production 14	{COMMA, SEMICOLON,)} First(<expr. tail>) **-{λ}** U Follow(<expr. list>)
<id list> production 8	{)}

17

Computed Follow Set

Nonterminal	Follow set
<expr list> production 11	{)}
<id tail> production 9,10	{)}
<expr tail> production 12,13	{)}
<primary> production 17,18,19	{COMMA, SEMICOLON, +, -,)} First(<Prim. Tail>) **-{λ}** U Follow(<expr.>)
<primay tail> production 15,16	{COMMA, SEMICOLON,)} Follow(<expression>)
<add op> production 20,21	{ID, INTLIT, (} First(<primary>)
<system goal> production 22	{λ}

18

Reduced LL(1) Grammar

Start: <system goal> → <program> $

1. <program> → begin <statement list> end
2. <statement list> → <statement><statement tail>
3. <statement tail> → <statement><statement tail>
4. <statement tail> → λ
5. <statement> → ID := <expression>;
6. <statement> → read (<id list>);
7. <statement> → write (<expr list>);
8. <id list> → ID<id tail>
9. <id tail> → , ID<id tail>
10. <id tail> → λ
11. <expr list> → <expression> <expr tail>
12. <expr tail> → , <expression><expr tail>
13. <expr tail> → λ
14. <expression> → <primary><primary tail>
15. <primary tail> → <add op> <primary><primary tail>
16. <primary tail> →λ
17. <primary> → (<expression>)
18. <primary> → ID
19. <primary> → INTLITERAL
20. <add op> → PLUSOP
21. <add op> → MINUSOP
22. <system goal> → <program>$

19

Computed predict Set

prod	Predict set		
1	First(begin <statement list> end)	First(begin)=	{begin}
2	First(<statement><statement tail>)	First(<statement>)=	{ID,read,write}
3	First(<statement><statement tail>)	First(<statement>)=	{ID,read,write}
4	(First(λ)-λ) ∪ Follow(<statement tail>)	Follow(<statement tail>)=	{end}
5	First(ID := <expression>;)	First(ID)=	{ID}
6	First(read (<id list>);)	First(read)=	{read}
7	First(write (<expr list>);)	First(write)=	{write}
8	First(ID<id tail>)	First(ID)=	{ID}
9	First(, ID<id tail>)	First(,)=	{ , }
10	(First(λ)-λ) ∪ Follow(<id tail>)	Follow(<id tail>)=	{) }
11	First(<expression> <expre tail>)	First(<statement>)=	{ID,INTLIT,(}
12	First(,<expression> <expre tail>)	First(,)=	{ , }

20

Computed predict Set

prod	Predict set		
13	**(First(λ)-λ) ∪ Follow(<expr tail>)**	**Follow(<expr tail>)=**	**{) }**
14	**First(<primary><primary tail>)**	**First(<primary>)=**	**{ID,INTLIT,(}**
15	**First(<add op><primary><primary tail>)**	**First(<add op>)=**	**{ +,- }**
16	**(First(λ)-λ) ∪ Follow(<primary tail>)**	**Follow(<primary tail>)=**	**{Comma, ; ,)}**
17	**First(<expression>;)**	**First(( )=**	**{ (}**
18	**First(read (ID)**		**{ID}**
19	**First(write (INTLIT)**		**{INTLIT}**
20	**First(+)**		**{ + }**
21	**First(-)**		**{ - }**
22	**First(<program>$)**	**First(<program>)=**	**{ begin }**

21

LL(1) parse table

- Predict function can be represented in a LL(1) parse table T : $V_n * V_t \rightarrow P \cup \{Error\}$, P is the set of all production
- If A is the non-terminal to be matched and t is a look-ahead token, T[A][t] tells what production to predict.
- If no production is appropriate, T[A][t] yields an Error to indicate the syntax error

$T[A][t]=A \rightarrow X_1X_2\ldots X_m$ [production]

if $t \in$ Predict($A \rightarrow X_1X_2\ldots X_m$)

T[A][t]=Error otherwise

22

LL(1) grammar test

- A grammar G is LL(1) iff all entries in T contain a unique prediction or an error flag
- For any non-terminal A and look-ahead symbol t, it cannot be the case that t ∈ Predict(A→α) and t ∈ Predict(A→β) where A→α and A→β are distinct productions (pairwise disjoint!!!)
- A grammar is LL(1), it is guaranteed a unique prediction or an error flag for any combination of non-terminal and look-ahead symbol➔ well suited to top-down parsing

23

Homework Which is LL(1)

E→Prefix (E) | V Tail
Prefix→F | λ
Tail→+ E | λ

S→ABc
A→a | λ
B→b | λ

S→aSe | B
B→bBe | C
C→cCe | d

S→ABBA
A→a | λ
B→b | λ

S→Ab
A→a | B | λ
B→b | λ

24

LL(1) grammar test

- This allows us to build a parse tree for any valid input string, starting at the start symbol and working downward toward the input symbols that are the leave of the tree
- We will take Predict sets and transform them into the LL(1) parse table. This table can be used to build the parsing procedures that constitute a recursive descent parser or the tables can be used to drive a LL(1) parser

25

Construct PT by the Follow & First set

Why do we need First and Follow?

➔ construct a parsing table

Input: Grammar G Output: Parsing table M

Algorithm: Constructing the parsing table

1. For each terminal a in First(α), add A → α (production) to M[A][a]

2. If λ is in First(α), add A → α to M[A][b] for each terminal b in Follow(A). If λ is in First(α), and $ is in Follow(A), add A → α to M[A][$]

3. Make each undefined entry of M as error!!!

26

Reduced LL(1) Grammar

Start: <system goal> → <program> $

1. <program> → begin <statement list> end
2. <statement list> → <statement><statement tail>
3. <statement tail> → <statement><statement tail>
4. <statement tail> → λ
5. <statement> → ID := <expression>;
6. <statement> → read (<id list>);
7. <statement> → write (<expr list>);
8. <id list> → ID<id tail>
9. <id tail> → , ID<id tail>
10. <id tail> → λ
11. <expr list> → <expression> <expr tail>
12. <expr tail> → , <expression><expr tail>
13. <expr tail> → λ
14. <expression> → <primary><primary tail>
15. <primary tail> → <add op> <primary><primary tail>
16. <primary tail> →λ
17. <primary> → (<expression>)
18. <primary> → ID
19. <primary> → INTLITERAL
20. <add op> → PLUSOP
21. <add op> → MINUSOP
22. <system goal> → <program>$

27

Computed predict Set

prod	Predict set		
1	First(begin <statement list> end)	First(begin)=	{begin}
2	First(<statement><statement tail>)	First(<statement>)=	{ID,read,write}
3	First(<statement><statement tail>)	First(<statement>)=	{ID,read,write}
4	(First(λ)-λ) ∪ Follow(<statement tail>)	Follow(<statement tail>)=	{end}
5	First(ID := <expression>;)	First(ID)=	{ID}
6	First(read (<id list>);)	First(read)=	{read}
7	First(write (<expr list>);)	First(write)=	{write}
8	First(ID<id tail>)	First(ID)=	{ID}
9	First(, ID<id tail>)	First(,)=	{ , }
10	(First(λ)-λ) ∪ Follow(<id tail>)	Follow(<id tail>)=	{) }
11	First(<expression> <expre tail>)	First(<statement>)=	{ID,INTLIT,(}
12	First(,<expression> <expre tail>)	First(,)=	{ , }

28

Computed predict Set

prod	Predict set		
13	**(First(λ)-λ) ∪ Follow(<expr tail>)**	**Follow(<expr tail>)=**	**{) }**
14	**First(<primary><primary tail>)**	**First(<primary>)=**	**{ID,INTLIT,(}**
15	**First(<add op><primary><primary tail>)**	**First(<add op>)=**	**{ +,- }**
16	**(First(λ)-λ) ∪ Follow(<primary tail>)**	**Follow(<primary tail>)=**	**{Comma, ; ,)}**
17	**First(<expression>;)**	**First(( )=**	**{ (}**
18	**First(read (ID)**		**{ID}**
19	**First(write (INTLIT)**		**{INTLIT}**
20	**First(+)**		**{ + }**
21	**First(-)**		**{ - }**
22	**First(<program>$)**	**First(<program>)=**	**{ begin }**

29

Construct Parsing Table using Predict

	ID	INTLIT	:=	,	;	+	-	(	)	Begin	End	Read	Write	$
<program>										**1**				
<statement list>	**2**											**2**	**2**	
<statement>	**5**											**6**	**7**	
<statement tail>	**3**										**4**	**3**	**3**	
<expression>	**14**	**14**						**14**						
<id list>	**8**													
<expr list>	**11**	**11**						**11**						
<id tail>				**9**					**10**					
<expr tail>				**12**					**13**					
<primary>	**18**	**19**						**17**						
<primay tail>				**16**	**16**	**15**	**15**		**16**					
<add op>						**20**	**21**							
<system goal>										**22**				

30

Homework to construct parsing table

S→AbB | d
A→CAb | B
B→cSd | λ
C→a | ed

<program> → **begin** <stmts> **end** $
<stmts> → <stmt>; <stmts>
<stmts> → λ
<stmt> → **simplestmt**
<stmt> → **begin** <stmts> **end**

<Assign> → <id> := <expr>
<id> → A<id_tail> | B<id_tail> | … | Z<id_tail>
<id_tail> A<id_tail> | B<id_tail> | … | Z<id_tail> |
0<id_tail> | … | 9<id_tail> | λ
<expr> → <expr> + <term> |
<expr> − <term> | <term>
<term> → <term> * <factor> |
<term> / <factor> | <factor>
<factor> → (<expr>)| <id>

31

Homework to construct parsing table

E→Prefix (E) | V Tail
Prefix→F | λ
Tail→+ E | λ

S→aSe | B
B→bBe | C
C→cCe | d

S→ABc
A→a | λ
B→b | λ

32

Homework to construct parsing table

Expr→- Expr | (Expr) | Var ExprTail
Var →ID VarTail
VarTail → (Expr) | λ
ExprTail→- Expr | λ

33