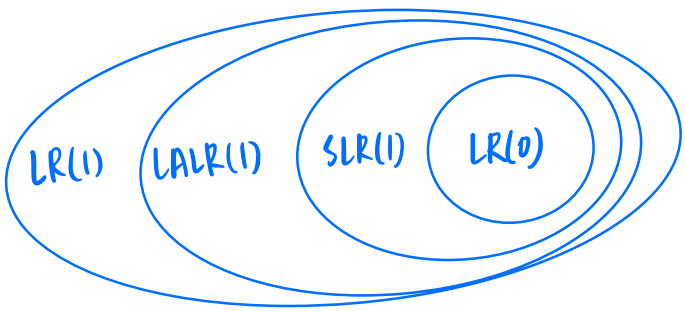
ch6

bottom-up	LR parser	rightmost derivation	leaf → root	左 → 右推導	
top-down	LL parser	leftmost derivation	root → leaf	左 → 右推導	

- handle：下一次會被reduce成non-terminal的部分 ⇒ 不ambiguous時唯一
- parsing stack：

$ parsing stack	input stack $	action (Shift/Reduce)

- kernel item：① initial item：$S' \to \cdot S$ ② 其他·不在最左的
- non-kernel item：除了kernel item以外的

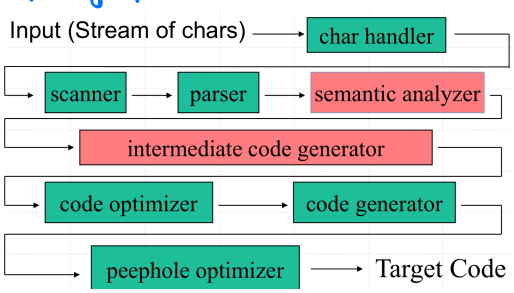


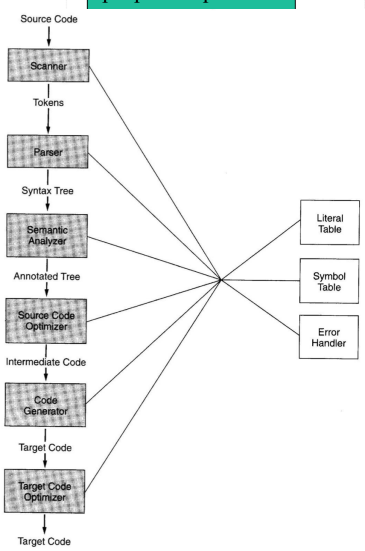
- LR(0)：整行reduce ⇒ 不能解 rr/rs
- SLR, LALR：只在左式的follow reduce ⇒ 不能解 rr
- LR(1) 多了 offset：$[A \to \alpha \cdot B\gamma, w]$ —closure→ $[B \to \cdot\delta, first(\gamma w)]$ 花用

goto時的offset照抄，closure不要算 只在有offset地方reduce

ch7

- Compiling process





- Attribute grammer example　虛線：parser tree edge　實線：dependency

①

Grammar Rule	Semantic Rules
$exp_1 \to exp_2$ + term	$exp_1.val = exp_2.val + term.val$
$exp_1 \to exp_2$ - term	$exp_1.val = exp_2.val - term.val$
$exp \to term$	$exp.val = term.val$
$term_1 \to term_2$ * factor	$term_1.val = term_2.val * factor.val$
$term \to factor$	$term.val = factor.val$
$factor \to$ (exp)	$factor.val = exp.val$
$factor \to$ **number**	$factor.val =$ **number**.val

②

Grammar Rule	Semantic Rules
based-num → num basechar	based-num.val = num.val num.base = basechar.base
basechar → o	basechar.base = 8
basechar → d	basechar.base = 10
$num_1 \to num_2$ digit	$num_1.val =$ if digit.val = error or $num_2.val$ = error then error else $num_2.val * num_1.base + digit.val$ $num_2.base = num_1.base$ $digit.base = num_1.base$
num → digit	num.val = digit.val digit.base = num.base
digit → 0	digit.val = 0
digit → 1	digit.val = 1
...	...
digit → 7	digit.val = 7
digit → 8	digit.val = if digit.base = 8 then error else 8
digit → 9	digit.val = if digit.base = 8 then error else 9

- 綜合屬性 synthesize attribute：Postorder，由下往上 ⇒ 若所有屬性皆為綜合屬性，則稱為S-attribute grammer
- 繼承屬性 inherited attribute：其他非綜合的都算

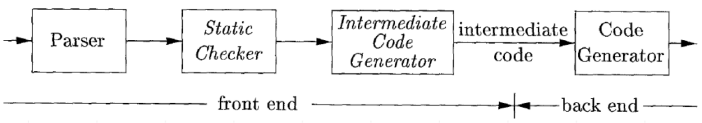
Table 6.8
Parsing and semantic actions for the expression 3*4+5 during an LR parse

↱ bottom-up：reduce時作語意動作

	Parsing Stack	Input	Parsing Action	Value Stack	Semantic Action
1	$	3*4+5 $	shift	$	
2	$ **n**	*4+5 $	reduce $E \to$ **n**	$ **n**	$E.val =$ **n**$.val$
3	$ E	*4+5 $	shift	$ 3	
4	$ E *	4+5 $	shift	$ 3 *	
5	$ E * **n**	+5 $	reduce $E \to$ **n**	$ 3 * **n**	$E.val =$ **n**$.val$
6	$ E * E	+5 $	reduce $E \to E * E$	$ 3 * 4	$E_1.val = E_2.val * E_3.val$
7	$ E	+5 $	shift	$ 12	
8	$ E +	5 $	shift	$ 12 +	
9	$ E + **n**	$	reduce $E \to$ **n**	$ 12 + **n**	$E.val =$ **n**$.val$
10	$ E + E	$	reduce $E \to E + E$	$ 12 + 5	$E_1.val = E_2.val + E_3.val$
11	$ E	$		$ 17	

9Lh10

- Compiler 流程：



- 三位址碼 three address code：可和① Array ② DAG圖 ③ Quadruple互換

ex：do $i = i + 1$; while $(a[i] < v)$; ⇒ do-while的組譯

```
L:  t1 = i + 1                100:  t1 = i + 1
    i = t1                    101:  i = t1
    t2 = i * 8                102:  t2 = i * 8
    t3 = a [ t2 ]             103:  t3 = a [ t2 ]
    if t3 < v goto L          104:  if t3 < v goto 100
```

(a) Symbolic labels. (b) Position numbers.

- Quadruple：

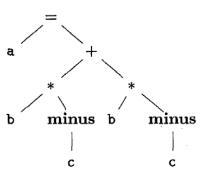
operator	arg1	arg2	result

ex：

	Operator	Arg1	Arg2	Result
A = B op(1) C	op(1)	B	C	A
A = op(2) B	op(2)	B		A
goto L	goto			L
if A relop B goto L	relopgoto	A	B	L
param A and call P,n	param	A		
	call	P	n	
A = B [i]	=[]	B	i	A
A[i] = B	[]=	B	i	A
A = &B	=&	B		A
A = *B	=*	B		A
*A = B	*=	B		A

- Triples：比Quadruple少了result項

ex：



(a) Syntax tree

	op	arg_1	arg_2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)
項		...	

第0項

(b) Triples