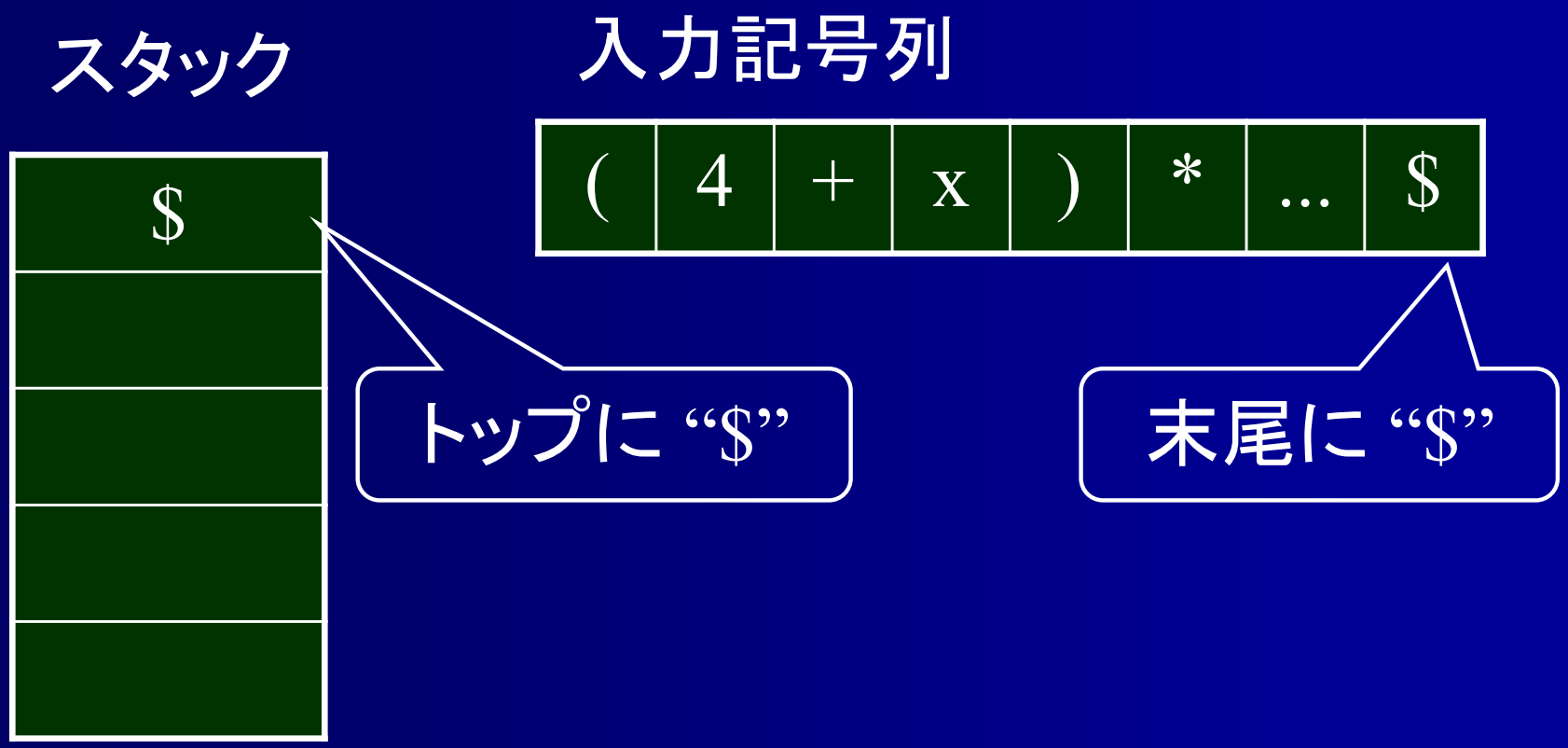
コンパイラ

第12回 上昇型構文解析(2)

http://www.info.kindai.ac.jp/compiler
E館3階E-331 内線5459
takasi-i@info.kindai.ac.jp

移動還元構文解析 (shift reduce parsing)

- 上昇型構文解析
- 構文解析表とスタックで解析



移動還元構文解析 (shift reduce parsing)

- 構文解析表とスタックで解析
  - 初期状態
    - 入力記号列末尾に “$”
    - スタックトップに “$”
  - if (スタックトップが生成規則の右辺に一致)
    生成規則の右辺をポップ, 左辺をプッシュ (還元)
    else 入力記号をプッシュ (移動)
  - if (スタックトップが “$” && 入力記号が “$”)
    解析終了

還元(reduce)

- 還元
  - 生成規則の右辺から左辺に戻す
- ハンドル
  - 右辺に一致する部分

<namelist> ::= <name> | <namelist> “,” <name>
<name> ::= “a” | “b” | “c”

“a” “,” “b”
→ <name> “,” “b”　　<name> → “a” の還元
→ <namelist> “,” “b”　　<namelist> → <name> の還元

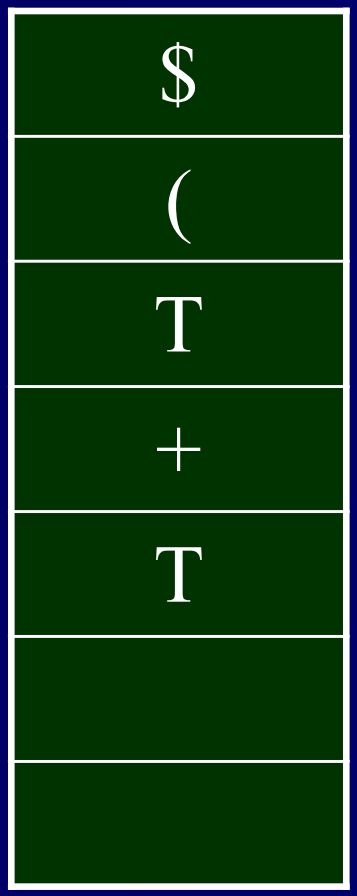
ハンドル

還元

生成規則

$\mathbf{P} = \{E \rightarrow T+T,\ T \rightarrow F*F,\ F \rightarrow i,\ F \rightarrow (E)\}$

スタック



E→T+T の
右辺と一致
＝ハンドル

右辺 T+T をポップ
左辺 E をプッシュ

出力

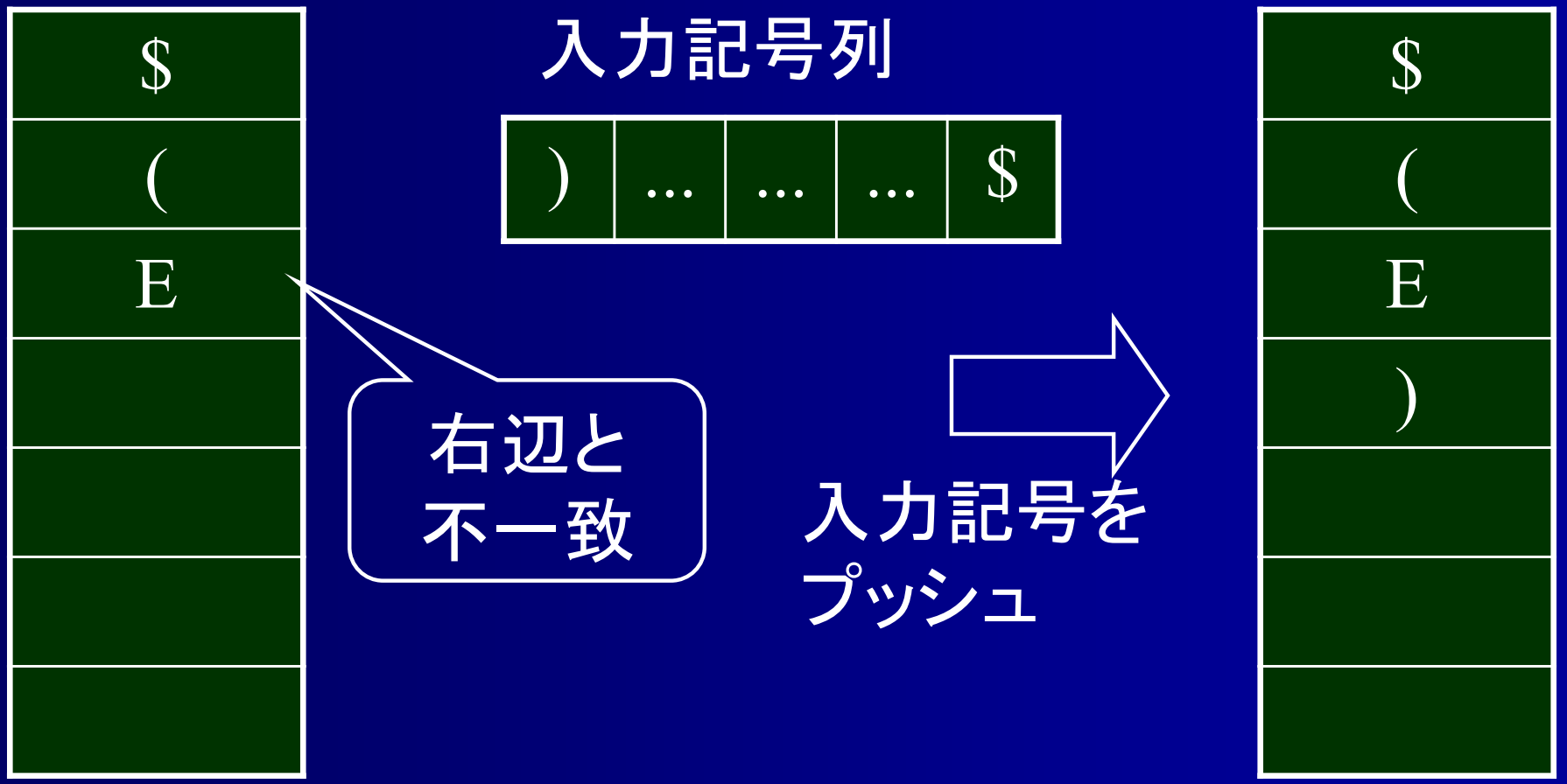
ADD

対応する
コードを出力

移動(shift)

生成規則

スタック　　$\mathbf{P} = \{E \rightarrow T+T,\ T \rightarrow F*F,\ F \rightarrow i,\ F \rightarrow (E)\}$



移動と還元

- 移動：右辺を読み込み途中
- 還元：右辺を読み込み完了

例：$\mathrm{E} \longrightarrow \mathrm{T} + \mathrm{T}$ ⇒ 読み込み位置を移動

$\mathrm{E} \longrightarrow \mathrm{T} + \mathrm{T}$ ⇒ 読み込み位置を移動

$\mathrm{E} \longrightarrow \mathrm{T} + \mathrm{T}$ ⇒ 読み込み位置を移動

$\mathrm{E} \longrightarrow \mathrm{T} + \mathrm{T}$ ⇒ 読み込み完了，還元する

移動還元構文解析の問題点

- 生成規則が複数ある場合
  - 例 : $\$ \rightarrow E\$$, $E \rightarrow T+T$, $E \rightarrow T$, $T \rightarrow i$, $T \rightarrow n$

$E \rightarrow T$ [$+T$] を解析

解析位置

$\Rightarrow E \rightarrow T+T$ の一部と見做して移動

$\Rightarrow E \rightarrow T$ と見做して還元

どちらの規則を使用する？

例：$ x + 1 $ { $→E$, E→T+T, E→T, T→*i*, T→*n* }

スタック	入力記号	操作
$	x + 1 $	移動
$ x	+ 1 $	還元 T → *n*
$ T	+ 1 $	還元 E → T
$ E	+ 1 $	移動
$ E +	1 $	移動
$ E + 1	$	還元 T → *i*
$ E + T	$	還元 E → T
$ E + E	$	移動
$ E + E $	ε	還元 $ → E$
$ E + $	ε	解析失敗

移動に変更

バックトラック

例：$ x + 1 $ { $→E$, E→T+T, E→T, T→*i*, T→*n* }

スタック	入力記号	操作
$	x + 1 $	移動
$ x	+ 1 $	還元 T → *n*
$ T	+ 1 $	移動
$ T +	1 $	移動
$ T + 1	$	還元 T → *i*
$ T + T	$	還元 E → T
$ T + E	$	移動
$ T + E $	ε	還元 $ → E$
$ T + $	ε	解析失敗

E→T+Tに変更

バックトラック

例 : $ x + 1 $ { $→E$, E→T+T, E→T, T→*i*, T→*n* }

スタック	入力記号	操作
$	x + 1 $	移動
$ x	+ 1 $	還元 T → *n*
$ T	+ 1 $	移動
$ T +	1 $	移動
$ T + 1	$	還元 T → *i*
$ T + T	$	還元 E → T+T
$ E	$	移動
$ E $	$	還元 $ → E$
$ $	ε	解析完了

移動還元構文解析の問題点

- 生成規則が複数ある場合
  - 例 : $\$ \rightarrow E\$$, $E \rightarrow T+T$, $E \rightarrow T$, $T \rightarrow i$, $T \rightarrow n$

E の後には必ず $

還元 $E \rightarrow T$ よりも $E \rightarrow T+T$ を優先 (最長一致)

Eへの還元はその後に “$” が来る場合のみ
⇒ 後に来る記号で判断

Follow 集合 を用いる

Follow集合 (後続終端記号)

- Follow (A) = {“b” | S⇒αAbβ}

  非終端記号<A>の直後に来る終端記号の集合

例 : <S> ::= “a” | <B> <C>

<B> ::= “b” | ε

<C> ::= “c”

{ a, bc, c }

Follow (S) = {$}　($: 入力の末尾)

Follow (C) = {$}

Follow (B) = First (C) = {“c”}

Follow 集合の求め方

- Follow (A) の求め方 (A∈N)
  - 初期状態では Follow (A) = φ(空集合)

1. A = S のとき

   Follow (S) += “$”;

2. 生成規則 X → αAβがあるとき

   Follow (A) += (First (β) - {ε})

3. 生成規則 X → αA があるとき

   Follow (A) += Follow (X)

Follow集合を用いた移動還元構文解析

– 例 : $\$ \rightarrow E\$$, $E \rightarrow T+T$, $E \rightarrow T$, $T \rightarrow i$, $T \rightarrow n$

Follow (E) = {“$”}
Follow (T) = {“$”, “+”}

E の還元は次に “$” が来るときのみ
T の還元は次に “$”か“+”が来るときのみ

例：$ x + 1 $ { $→E$, E→T+T, E→T, T→*i*, T→*n* }

スタック	入力記号	Follow	操作
$	x + 1 $		移動
$ x	+ 1 $	+∈Follow (T)	還元 T → *n*

先読み記号：+
+∈Follow (T) なので
T → *n* で還元していい

Follow (E) = {“$”}
Follow (T) = {“$”, “+”}

例：$ x + 1 $ { $→E$, E→T+T, E→T, T→*i*, T→*n* }

スタック	入力記号	Follow	操作
$	x + 1 $		移動
$ x	+ 1 $	+∈Follow (T)	還元 T → *n*
$ T	+ 1 $	+∉Follow (E)	移動

先読み記号：+
+∉Follow (E) なので
E → T の還元はできない

Follow (E) = {“$”}
Follow (T) = {“$”, “+”}

例：$ x + 1 $ { $→E$, E→T+T, E→T, T→*i*, T→*n* }

スタック	入力記号	Follow	操作
$	x + 1 $		移動
$ x	+ 1 $	+∈Follow (T)	還元 T → *n*
$ T	+ 1 $	+∉Follow (E)	移動
$ T +	1 $		移動
$ T + 1	$	$∈Follow (T)	還元 T → *i*

先読み記号：$
$∈Follow (T) なので
T → *i* で還元していい

Follow (E) = {“$”}
Follow (T) = {“$”, “+”}

例：$ x + 1 $ { $→E$, E→T+T, E→T, T→*i*, T→*n* }

スタック	入力記号	Follow	操作
$	x + 1 $		移動
$ x	+ 1 $	+∈Follow (T)	還元 T → *n*
$ T	+ 1 $	+∉Follow (E)	移動
$ T +	1 $		移動
$ T + 1	$	$∈Follow (T)	還元 T → *i*
$ T + T	$	$∈Follow (E)	還元 E → T+T

Follow (E) = {"$

Follow (T) = {"$

先読み記号：$

$∈Follow (E) なので

E → T, E→ T+T で還元していい

E → T+T を優先

例：$ x + 1 $ { $→E$, E→T+T, E→T, T→*i*, T→*n* }

スタック	入力記号	Follow	操作
$	x + 1 $		移動
$ x	+ 1 $	+∈Follow (T)	還元 T → *n*
$ T	+ 1 $	+∉Follow (E)	移動
$ T +	1 $		移動
$ T + 1	$	$∈Follow (T)	還元 T → *i*
$ T + T	$	$∈Follow (E)	還元 E → T+T
$ E	$		移動
$ E $	ε		還元 $ → E$
$ $	ε		解析完了

LR構文解析

- Left to right scan & Right most derivation 解析
  - 上昇型解析
  - スタックと解析表を用いて解析
  - 演算順子位構文解析よりも広い文法を受理

演算子順位構文解析：式を解析
LR構文解析　　：全て解析

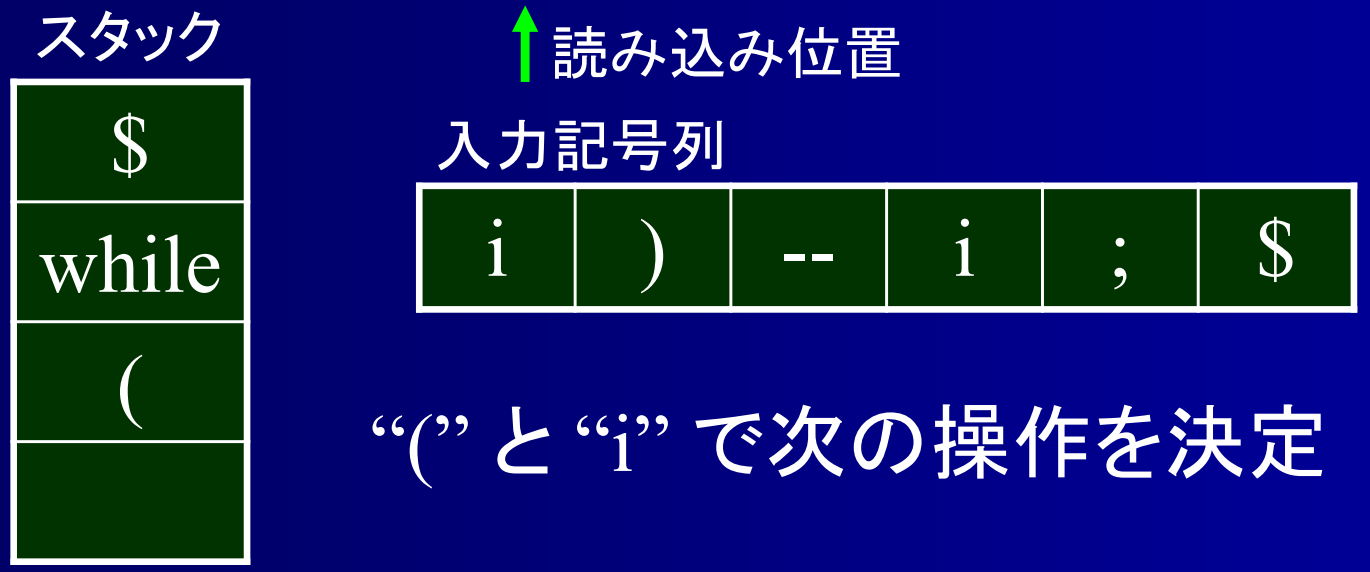
演算子順位構文解析の問題点

- 演算子順位構文解析の解析手順
  - スタックトップと入力記号で次の操作を決定

式の解析ならこれでOK
しかしプログラム全体だと？

演算子順位構文解析の問題点

例：“while” “(” “i” “)” “--” “i” “;”



“(” と “i” で次の操作を決定

“if” “(” “i” “)” “--” “i” “;”

読み込み位置

スタックトップと入力が同じだが異なる操作が必要

演算順位構文解析の問題点

- スタックトップと入力だけでは解析不能

状態を記憶する

状態：過去の入力の履歴

状態記号：それより下のスタックの内容を表す記号

例：$E \rightarrow T\,(\,+\,T \mid -\,T\,)$

s_0 s_1 s_2

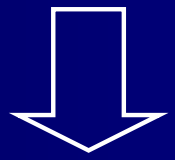
s_0：T まで読んだ
s_1：T+ まで読んだ
s_2：T- まで読んだ

状態の記憶

- スタックに状態を記憶する

X_i : 文法記号 ($\in \mathbf{N} \cup \mathbf{N}$)
s_j : 状態記号

文法記号と状態記号の対を
スタックに記憶



状態と入力で操作を決定

スタック

文法記号	状態記号
$	s_0
X_1	s_1
X_2	s_2
X_3	s_3
:	:
X_m	s_m

構文解析表

- action 表
  - 構文解析の動作を決定 (移動, 還元, 受理)
  - 状態 × 終端記号 → 動作(移動, 還元, 受理)
- goto 表
  - 還元時の状態遷移を決定
  - 状態 × 非終端記号 → 移動

action 表による動作

- s : 移動 (shift)
  - 入力記号と表で示された状態をプッシュ
- r : 還元 (reduce)
  - 表で示された規則によりスタックトップを還元
  - goto 表で示された状態をプッシュ
- a : 受理 (accept)
  - 解析完了

構文解析表

- 例 : $\$ \rightarrow E\$,\ E \rightarrow E*F,\ E \rightarrow F,\ F \rightarrow i$

	action			goto	
状態	i	*	$	E	F
0	移動 3			移動 1	移動 2
1		移動 4	受理		
2		還元 E→F	還元 E→F		
3		還元 F→i	還元 F→i		
4	移動 3				移動 5
5		還元 E→E*F	還元 E→E*F		

F を還元後
状態 0 ならば
状態 2 へ

状態 0 で
i を読めば
状態 3へ

空欄は構文解析エラー

状態 5 で
$ を読めば
E→E*F で還元

LR構文解析アルゴリズム

スタック : $(\$s_0)(X_1s_1)(X_2s_2)...(X_ms_m)$, 入力 : a_i のとき

- action $[s_m, a_i]$ =「移動 s」の場合
  - (a_i, s) をプッシュ, 次の入力を読み込む
- action $[s_m, a_i]$ =「還元 $A \rightarrow \beta$」の場合
  - $|\beta|$組の記号 (X_i, s_i) ($m-\beta+1 \leqq i \leqq m$) をポップ
  - (A, goto $[s_{m-|\beta|}, A]$) をプッシュ
  - $A \rightarrow \beta$に対応するコードを生成
- action $[s_m, a_i]$ =「受理」の場合
  - 解析完了

処理の手順 (初期状態)

入力末尾に $

入力記号

	1	*	2	*	3	$

還元

スタック

記号	状態
$	0

スタックトップには
(開始記号, 開始状態)

構文解析表

s : 移動

a : 受理

r : 還元

		action		goto	
状態	*i*	*	$	E	F
	s 3			1	2
		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

処理の手順 (移動)

入力記号

			2	*	3	$

還元

スタック

記号	状態
$	0
E	1
*	4

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F→*i*		
4	s 3				
5		r			

入力記号 2 と状態記号 3 をスタックへプッシュ

処理の手順 (移動)

入力記号

				*	3	$

還元

スタック

記号	状態
$	0
E	1
*	4
2	3

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

処理の手順 (還元)

入力記号

				*	3	$

還元

F

スタック

記号	状態
$	0
E	1
*	4
2	3

還元部分

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

F → *i* で還元

処理の手順 (還元)

入力記号

				*	3	$

還元

F

スタック

記号	状態
$	0
E	1
*	4

還元部分をポップ

構文解析表

	action			goto	
		*	$	E	F
				1	2
1		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F		
4	s 3				5
5		r E→E*F	r E→E*F		

F → *i* で還元

処理の手順 (還元)

入力記号

				*	3	$

還元

F

スタック

記号	状態
$	0
E	1
*	4

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s			
2		r			
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

記号 F と 状態記号 5 を
スタックへプッシュ

処理の手順 (還元)

入力記号

				*	3	$

還元

スタック

記号	状態
$	0
E	1
*	4
F	5

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

処理の手順 (還元)

入力記号

				*	3	$

還元

E

スタック

記号	状態
$	0
E	1
*	4
F	5

還元部分

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

E → E*F で還元

処理の手順 (還元)

入力記号

				*	3	$

還元

E

スタック

記号	状態
$	0

還元部分をポップ

構文解析表

	action			goto	
		*	$	E	F
				1	2
1		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

E → E*F で還元

処理の手順 (還元)

入力記号

				*	*i*	$

還元

E

スタック

記号	状態
$	0

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s 4			
2		r			
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

記号 E と 状態記号 1 を
スタックへプッシュ

処理の手順 (還元)

入力記号

				*	3	$

還元

スタック

記号	状態
$	0
E	1

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s 4	a		
2		r E→F	r E→F		
3		r F→*i*	r F→*i*		
4	s 3				5
5		r E→E*F	r E→E*F		

処理の手順 (受理)

入力記号

						$

還元

スタック

記号	状態
$	0
E	1

構文解析表

	action			goto	
状態	*i*	*	$	E	F
0	s 3			1	2
1		s 4	a		
2		r E→F	r E→F		
3					
4	s 3				5
5		r E→E*F	r E→E*F		

受理(a)なら解析完了

解析例

- $\$ \rightarrow E\$$, $E \rightarrow E+T$, $E \rightarrow T$, $T \rightarrow T*F$, $T \rightarrow F$, $F \rightarrow i$

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0

入力記号

	1	+	2	*	3	$

還元

初期状態ではスタックトップに
(開始記号, 初期状態)

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0

入力記号

	1	+	2	*	3	$

還元

action [0, *i*] = “s 4”

⇒ (1, 4) をプッシュ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
1	4

入力記号

		+	2	*	3	$

還元

F

action [4, +] = “r F→*i*”

⇒ (1, 4) をポップ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0

入力記号

		+	2	*	3	$

還元

F

action [4, +] = “r F→*i*”

⇒ (1, 4) をポップ

goto [0, F] = “3”

⇒ (F, 3) をプッシュ

コード生成 PUSHI 1

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
F	3

入力記号

		+	2	*	3	$

還元

T

action [3, +] = “r T→F”

⇒ (F, 3) をポップ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0

入力記号

		+	2	*	3	$

還元

T

action [3, +] = “r T→F”

⇒ (F, 3) をポップ

goto [0, T] = “2”

⇒ (T, 2) をプッシュ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
T	2

入力記号

		+	2	*	3	$

還元

E

action [2, +] = “r E→T”

⇒ (T, 2) をポップ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0

入力記号

		+	2	*	3	$

還元

E

action [2, +] = “r E→T”

⇒ (T, 2) をポップ

goto [0, E] = “1”

⇒ (E, 1) をプッシュ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1

入力記号

		+	2	*	3	$

還元

action [1, +] = “s 5”

⇒ (+, 5) をプッシュ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5

入力記号

			2	*	3	$

還元

action [5, *i*] = “s 4”

⇒ (2, 4) をプッシュ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5
2	4

入力記号

				*	3	$

還元

F

action [4, *] = “r F⟶*i*”

⇒ (2, 4) をポップ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5

入力記号

				*	3	$

還元

F

action [4, *] = “r F→*i*”

⇒ (2, 4) をポップ

goto [5, F] = “3”

⇒ (F, 3) をプッシュ

コード生成 PUSHI 2

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5
F	3

入力記号

				*	3	$

還元

T

action [3, *] = “r T→F”

⇒ (F, 3) をポップ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5

入力記号

				*	3	$

還元

T

action [3, *] = “r T→F”

⇒ (F, 3) をポップ

goto [5, T] = “7”

⇒ (T, 7) をプッシュ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5
T	7

入力記号

				*	3	$

還元

action [7, *] = “s 6”

⇒ (*, 6) をプッシュ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5
T	7
*	6

入力記号

					3	$

還元

action [6, *i*] = “s 4”

⇒ (3, 4) をプッシュ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5
T	7
*	6
3	4

入力記号

						$

還元

F

action [4, $] = “r F→*i*”

⇒ (3, 4) をポップ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5
T	7
*	6

入力記号

						$

還元

F

action [4, $] = “r F→*i*”

⇒ (3, 4) をポップ

goto [6, F] = “8”

⇒ (F, 8) をプッシュ

コード生成 PUSHI 3

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5
T	7
*	6
F	8

入力記号

						$

還元

T

action [8, $] = “r T→T*F”

⇒ (F, 8) (*, 6) (T, 7) をポップ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5

入力記号

						$

還元

T

action [8, $] = “r T→T*F”

⇒ (F, 8) (*, 6) (T, 7) をポップ

goto [5, T] = “7”

⇒ (T, 7) をプッシュ

コード生成 MUL

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1
+	5
T	7

入力記号

						$

還元

E

action [7, $] = “r E→E+T”

⇒ (T, 7) (+, 5) (E, 1) をポップ

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0

入力記号

						$

還元

E

action [7, $] = “r E→E+T”

⇒ (T, 7) (+, 5) (E, 1) をポップ

goto [0, E] = “1”

⇒ (E, 1) をプッシュ

コード生成 ADD

解析例

	action				goto		
状態	*i*	+	*	$	E	T	F
0	s 4				1	2	3
1		s 5		a			
2		r E→T	s 6	r E→T			
3		r T→F	r T→F	r T→F			
4		r F→*i*	r F→*i*	r F→*i*			
5	s 4					7	3
6	s 4						8
7		r E→E+T	s 6	r E→E+T			
8		r T→T*F	r T→T*F	r T→T*F			

スタック

記号	状態
$	0
E	1

入力記号

						$

還元

action [1, $] = “a”

⇒ 解析完了

スタック	入力列	参照	操作	出力
$0	1 + 2 * 3 $	a[0,*i*]	移動 4	
$0 14	+ 2 * 3 $	a[4,+], g[0,F]	還元 F→*i*, 3	PUSHI 1
$0 F3	+ 2 * 3 $	a[3,+], g[0,T]	還元 T→F, 2	
$0 T2	+ 2 * 3 $	a[2,+], g[0,E]	還元 E→T, 1	
$0 E1	+ 2 * 3 $	a[1,+]	移動 5	
$0 E1 +5	2 * 3 $	a[5,*i*]	移動 4	
$0 E1 +5 24	* 3 $	a[4,*], g[5,F]	還元 F→*i*, 3	PUSHI 2
$0 E1 +5 F3	* 3 $	a[3,*], g[5,T]	還元 T→F, 7	
$0 E1 +5 T7	* 3 $	a[7,*]	移動 6	
$0 E1 +5 T7 *6	3 $	a[6,*i*]	移動 4	
$0 E1 +5 T7 *6 34	$	a[4,$], g[6,F]	還元 F→*i*, 8	PUSHI 3
$0 E1 +5 T7 *6 F8	$	a[8,$], g[5,T]	還元 T→T*F, 7	MUL
$0 E1 +5 T7	$	a[7,$], g[0,E]	還元 E→E+T, 1	ADD
$0 E1	$	a[1,$]	受理	

LR(*k*) 文法

- LR(*k*) 文法
  - k 個のトークンを先読みすれば解析可能
- LR(1) 文法
  - 直後のトークン1個を先読みすれば解析可能
- LR(0) 文法
  - 先読み無しで解析可能

LR構文解析の作成

- LR構文解析
  - スタックと解析表で解析

⇒ 解析表の作成が必要

しかし解析表の作成は人力では不可能

表が大きくなり過ぎる
(終端記号数, 非終端記号数に対して
サイズが指数的に)

LR構文解析のクラス

クラス	特徴	解析表サイズ	受理文法	解析表の作成
LR(0)	先読み無し	小 ↑ ↓ 大	狭 ↑ ↓ 広	人力で可能
SLR(1) simple LR(1)	必要時に先読み			人力で可能
LALR(1) look ahead LR(1)	常に先読み 状態を一部統合			コンパイラコンパイラで作成
LR(1)	常に先読み 状態を完全分離			作成は難しい

コンパイラコンパイラ : コンパイラを自動生成するプログラム

LR(0) 構文解析の解析表作成

1. 各非終端記号のFollow集合を求める
2. LR(0) 項を求める
3. 閉包(=状態)を求める
4. 各状態からの遷移を求める

LR(0)項 (LR(0) item)

- LR(0)項
  - 生成規則の右辺に解析位置(・)を加えたもの

例 : A → BCD

[A → ・BCD][A → B ・CD][A → BC ・D][A → BCD ・]

解析前　Bまで解析　Cまで解析　全て解析

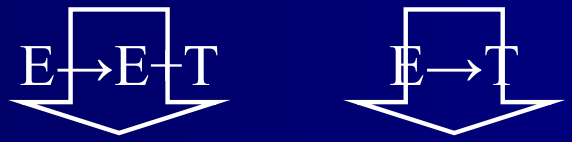
例 : A → ε

[A → ・]

LR(0)項

例 : { $\$ \rightarrow E\$$, $E \rightarrow E+T$, $E \rightarrow T$, $T \rightarrow T*F$, $T \rightarrow F$, $F \rightarrow i$}

項 $[\$ \rightarrow \cdot E\$]$



$\cdot E+T\$$　$\cdot T\$$

E$ 解析前 = E+T 解析前

= T 解析前

同じ状態として扱う必要あり

$[\$ \rightarrow \cdot E\$]$

$= [E \rightarrow \cdot E+T]$

$= [E \rightarrow \cdot T]$

$= [T \rightarrow \cdot T*F]$

$= [T \rightarrow \cdot F]$

$= [F \rightarrow \cdot i]$

閉包 (closure)

LR(0)項の閉包 (closure)

- LR(0)項集合 I の閉包　closure (I)
  - $i \in I \Rightarrow i \in$ closure (I)
    - I の項は closure (I) に含まれる
  - $[A \rightarrow u \cdot Bv] \in$ closure (I) $\wedge$ $B \rightarrow w \in \mathbf{P}$
    $\Rightarrow [B \rightarrow \cdot w] \in$ closure (I)
    - $[A \rightarrow u \cdot Bv]$ が closure (I) に含まれ
      $B \rightarrow w$ が生成規則 **P** に含まれるならば
      $[B \rightarrow \cdot w]$ は closure (I) に含まれる

LR(0)閉包の例

- 例 : $→E$, E→E+T, E→T, T→T*F, T→F, F→*i*

I = [$ →・E$] のとき

closure (I) = { [$ →・E$]　　I 自身

[E →・E+T]
[E →・T]　　E から生成

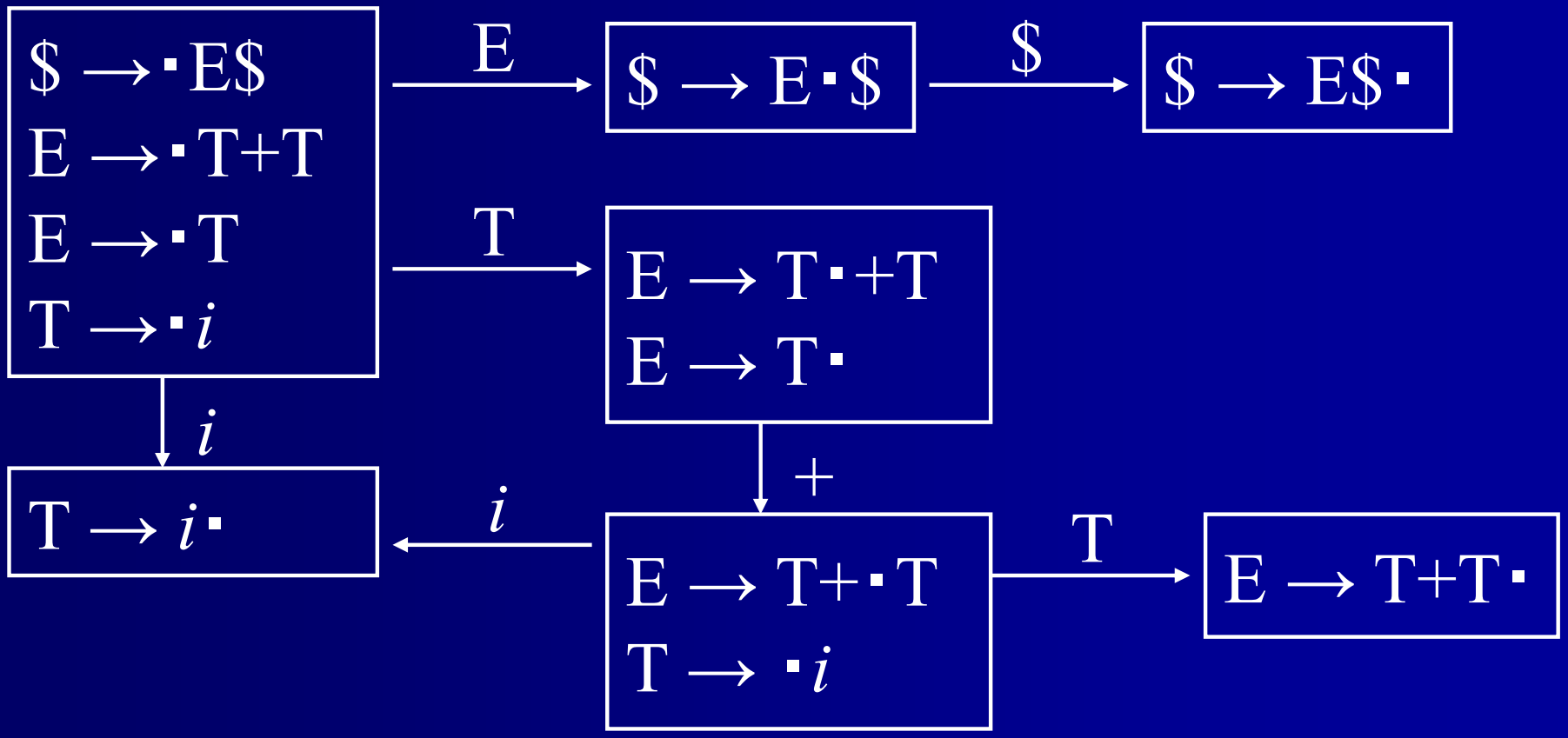
[T →・T*F]
[T →・F]　　T から生成

[F →・*i*] }　　F から生成

LR(0)閉包の例

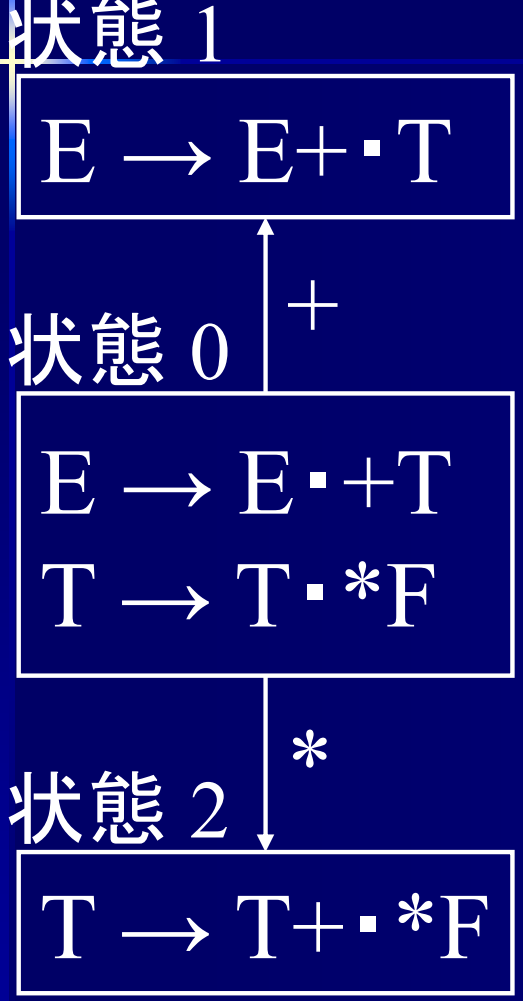
- 例 : $ → E$, E → T+T, E →T, T → *i*

Follow (E) = {$}, Follow (T) = {$, +}



各閉包を1つの状態として状態遷移を求める

構文解析表の作成 (移動)

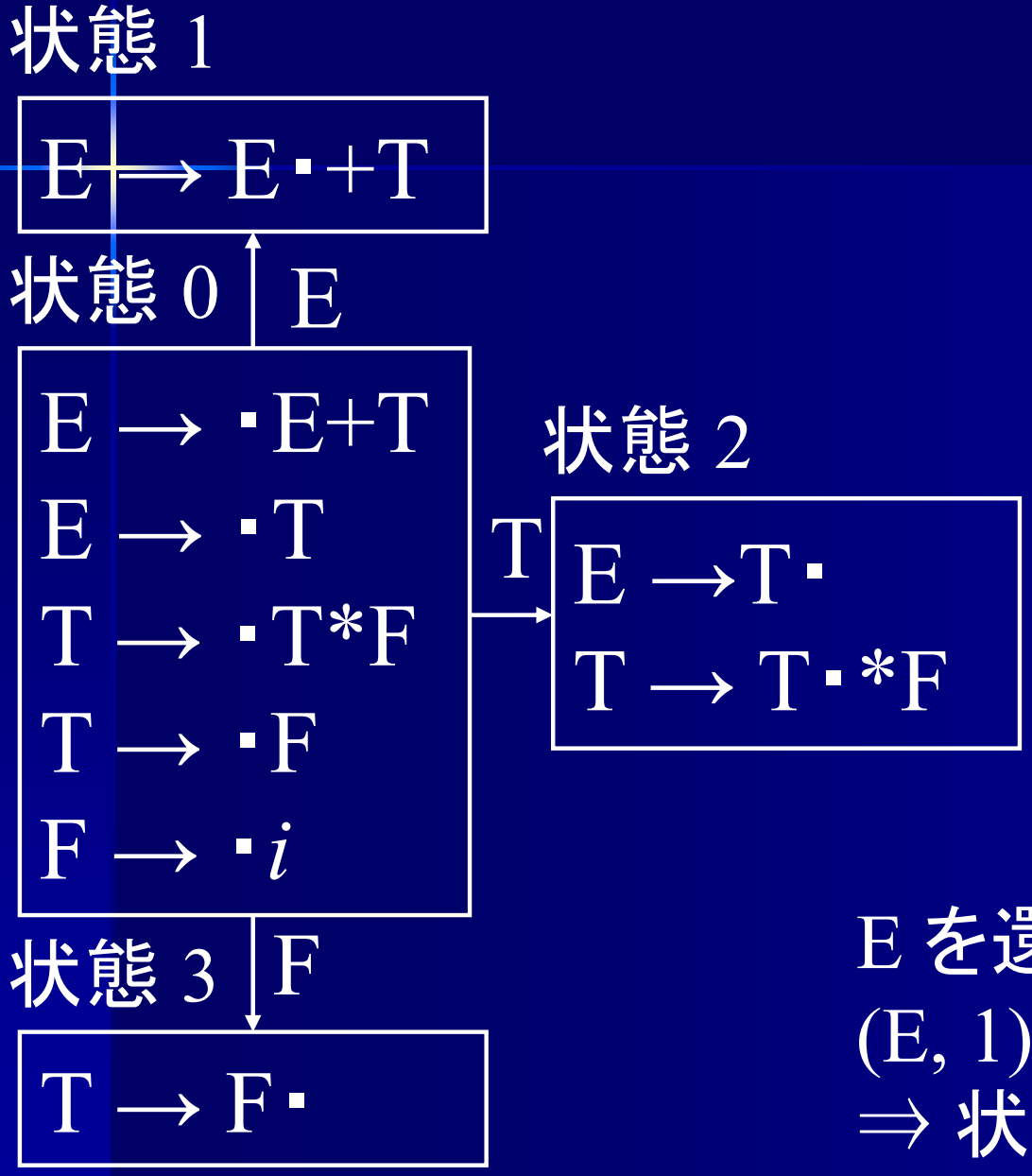


解析表

	action		
状態	+	*	$
0	s 1	s 2	
1			
2			

状態 0 で入力が + ならば
(+, 1) をスタックにプッシュ
⇒ 状態 1 へ移動

構文解析表の作成 (還元)



解析表

	goto		
状態	E	T	F
0	1	2	3
1			
2			
3			

E を還元後、状態 0 になれば
(E, 1) をスタックにプッシュ
⇒ 状態 1 へ移動

構文解析表の作成 (還元)

生成規則 E → E+T

Follow (E) = {+, $}

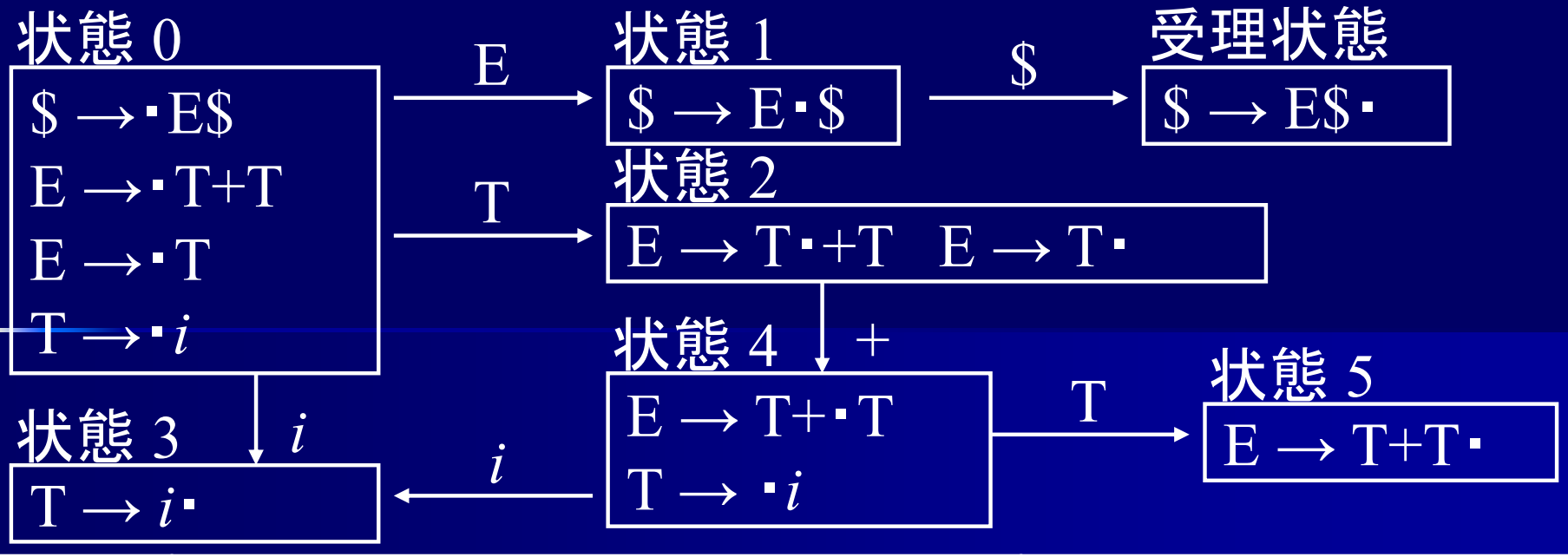
状態 0

E → E+T▪

解析表

	action		
状態	+	*	$
0	r E→E+T		r E→E+T
1			
2			

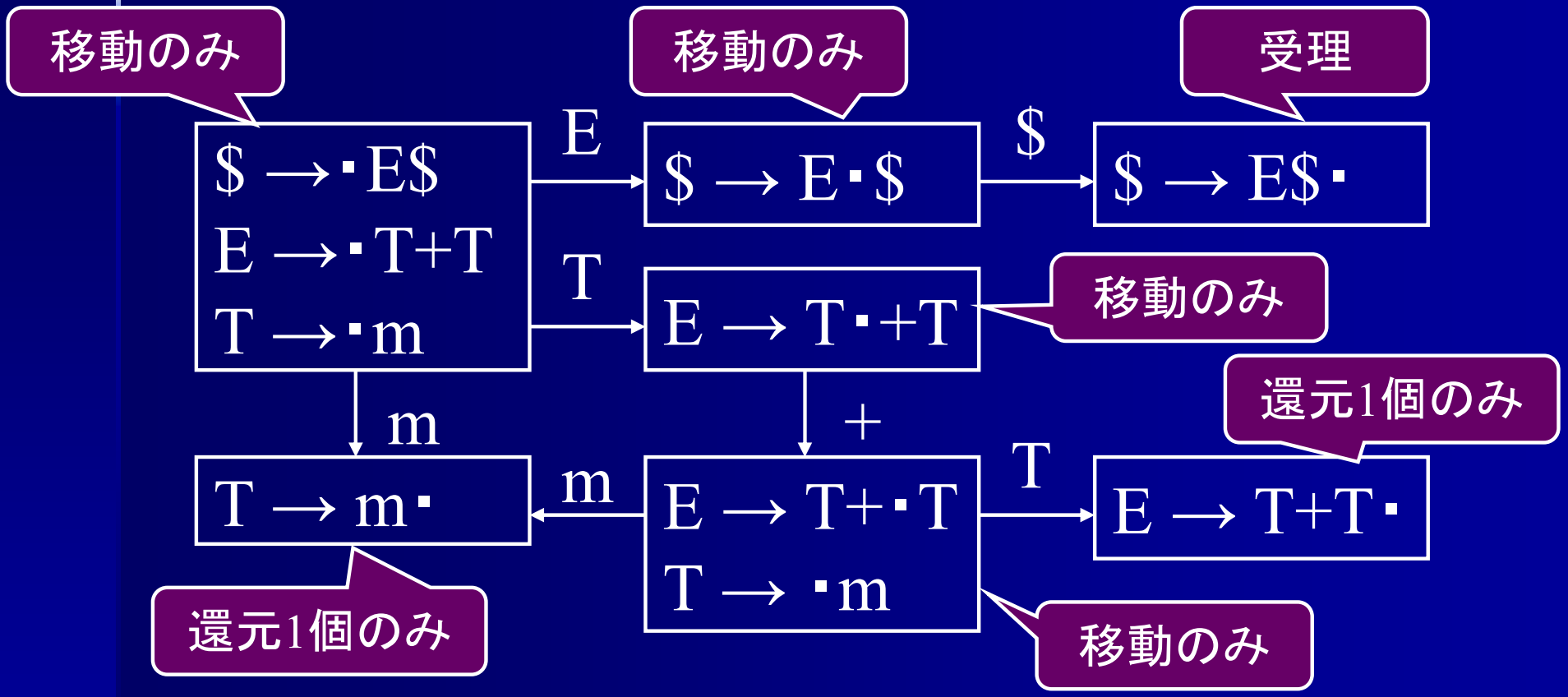
状態 0 で入力が +,$ ならば
E→E+T で還元



	action			goto	
状态	*i*	+	$	E	T
0	s 3			1	2
1			a		
2		s 4	r E→T		
3		r T→*i*	r T→*i*		
4	s 3				5
5			r E→T+T		

LR(0) 文法の例

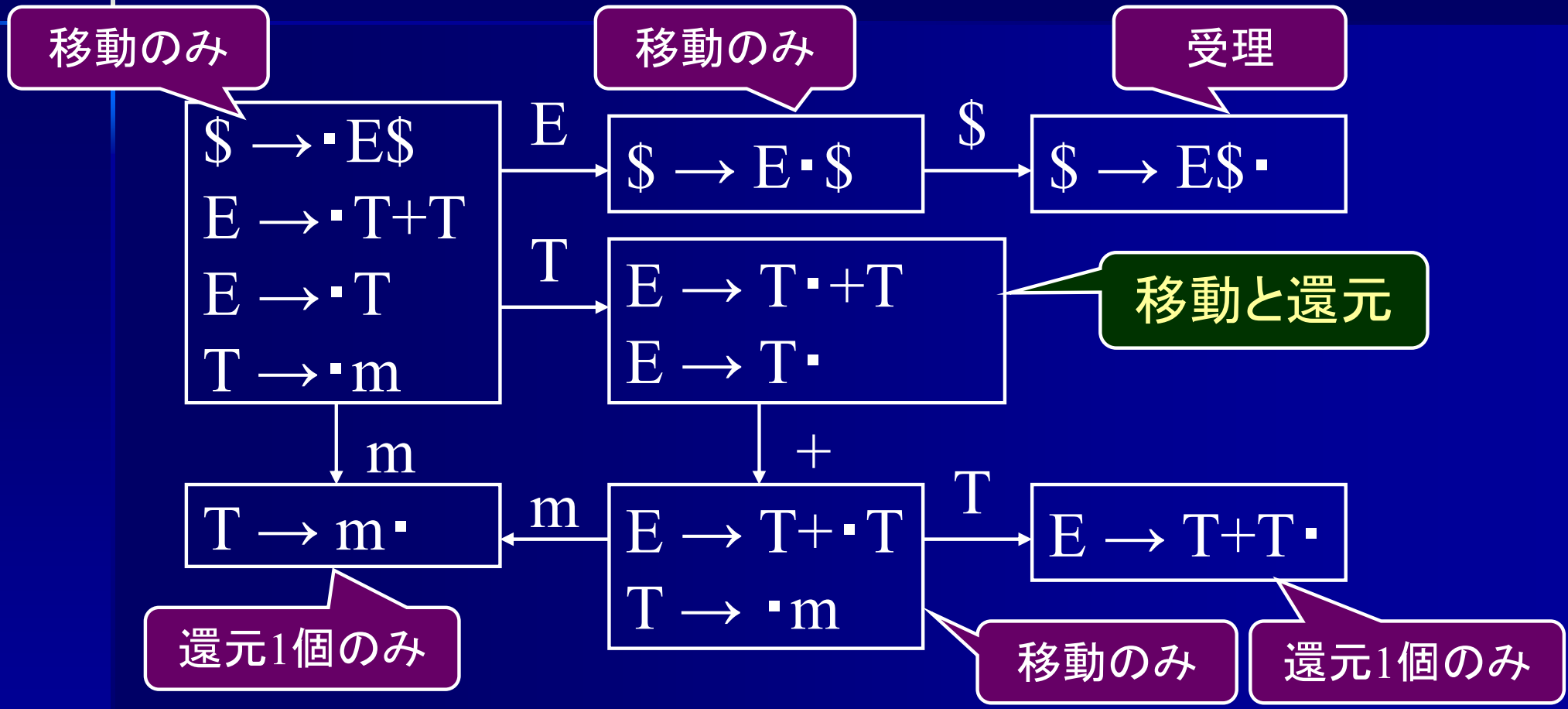
- $→E$, E→T+T, T→m



全ての状態が移動のみまたは還元1個のみ
⇒ LR(0) 文法

LR(0) 文法ではない例

- $→E$, E→T+T, E→T, T→m



移動と還元があるとどちらを適用するか分からない
⇒ LR(0) 文法ではない

SLR(1) (Simple LR) 構文解析

- SLR(1) 構文解析
  - 基本的には LR(0) と同様の処理
  - 不都合が起きた閉包に対して先読み
  - 先読みでFollow集合を使えば解析可能

SLR(1) 構文解析の解析表作成

1. 各非終端記号のFollow集合を求める
2. LR(0) 項を求める
3. 閉包(=状態)を求める
4. 各状態からの遷移を求める
5. 不都合が起きた閉包の項をLR(1)項に変換

ここまでLR(0)と共通

LR(1)項 (LR(1) item)

- LR(1)項
  - LR(0)項に先読み記号を加えたもの

例 : A → BCD, Follow (A) = {$}

[A → ・BCD, $][A → B ・CD, $]
[A → BC ・D, $][A → BCD ・, $]

LR(0)項　　先読み記号

LR(1)項の閉包 (closure)

- LR(1)項集合 I の閉包　closure (I)
  - $i \in I \Rightarrow i \in$ closure (I)
    - I の項は closure (I) に含まれる
  - $[A \rightarrow u \cdot Bv, a] \in$ closure (I) $\wedge$ $B \rightarrow w \in \mathbf{P}$
    $\Rightarrow [B \rightarrow \cdot w, First(va)] \in$ closure (I)
    - $[A \rightarrow u \cdot Bv, a]$ が closure (I) に含まれ
      $B \rightarrow w$ が生成規則 **P** に含まれるならば
      $[B \rightarrow \cdot w, First(va)]$ は closure (I) に含まれる

LR(1)閉包の例

- 例 : $→E$, E→T+T, T→F*F, F→ *i*

Follow(E) = {$}, Follow(T) = {+, $}, Follow(F) = {+,*,$}

I = [$ →・E$, ε] のとき

closure (I) = { [$ →・E$, ε]　I 自身

[E →・T+T, $]　E から生成

[T →・F*F, +]　T から生成

[F →・*i*, *], }　F から生成

このTの後には + のみ来る

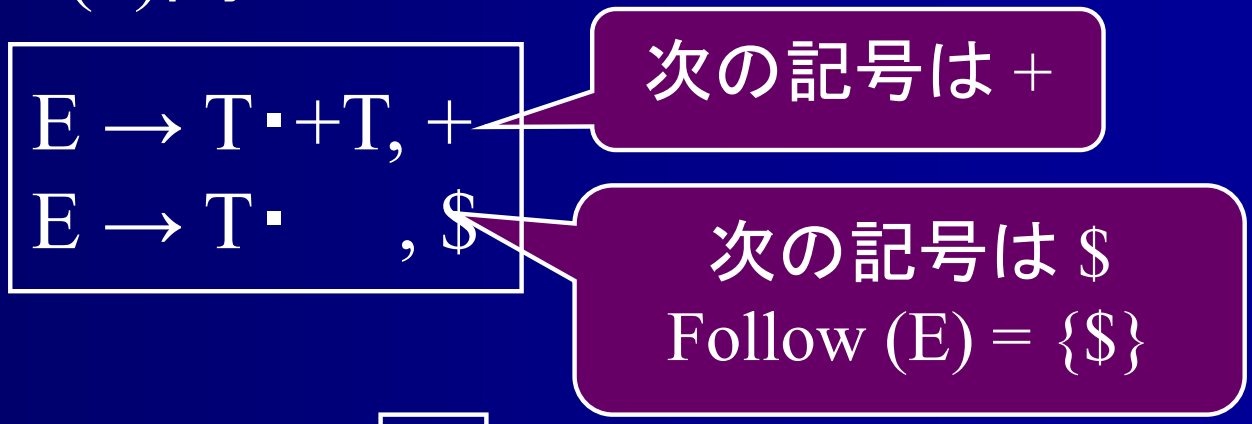
このFの後には * のみ来る

LR(1)閉包の例

- 例 : $ → E$, E → T+T, E →T, T → *i*

  Follow (E) = {$}, Follow (T) = {$, +}

LR(1)閉包



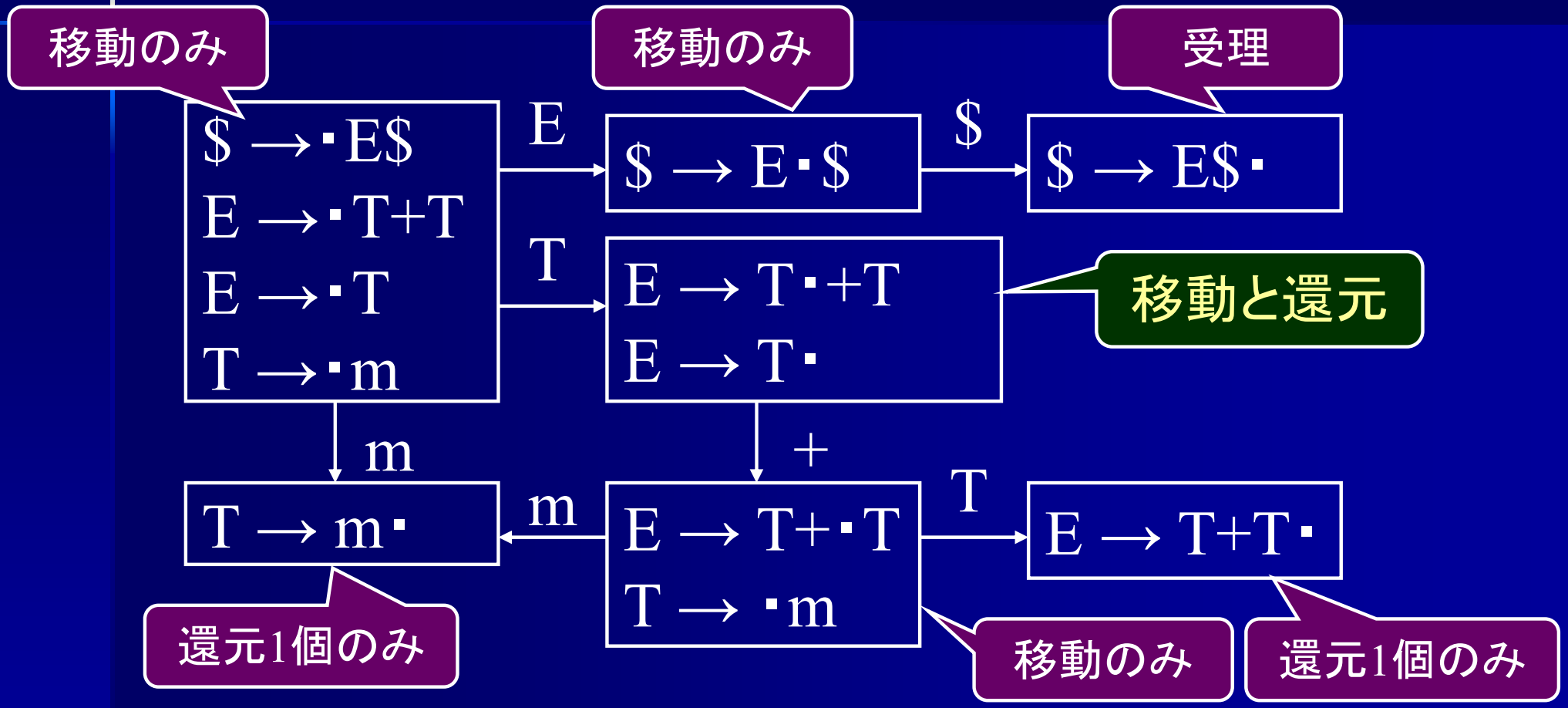
次の記号が + ⇒ E → T+T の解析

次の記号が $ ⇒ E → T の解析

SLR(1) 文法の例

① LR(0)と
同様に処理

- $→E$, E→T+T, E→T, T→m

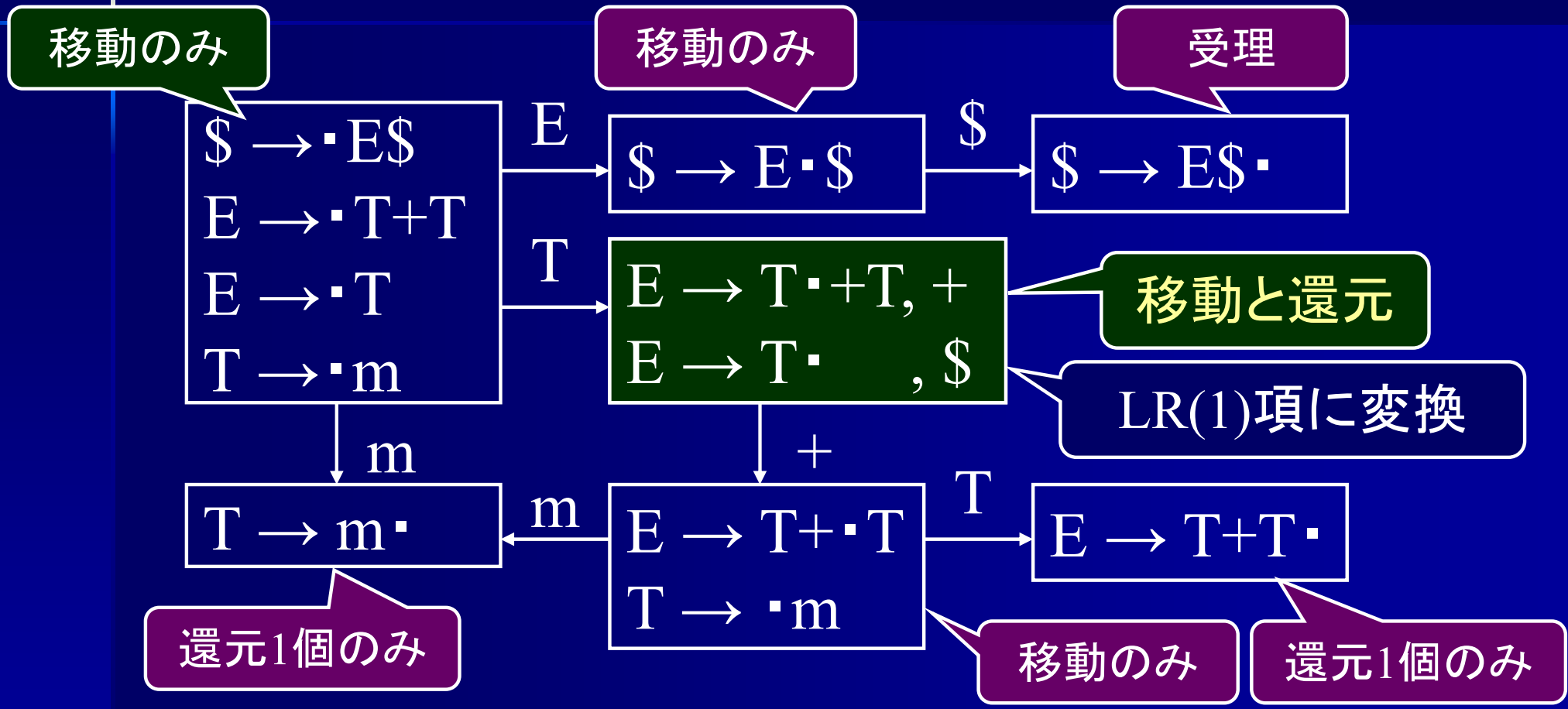


移動と還元があるとどちらを適用するか分からない
⇒ LR(0) 文法ではない

SLR(1) 文法の例

② 不都合な項を LA(1)に変換

- $→E$, E→T+T, E→T, T→m



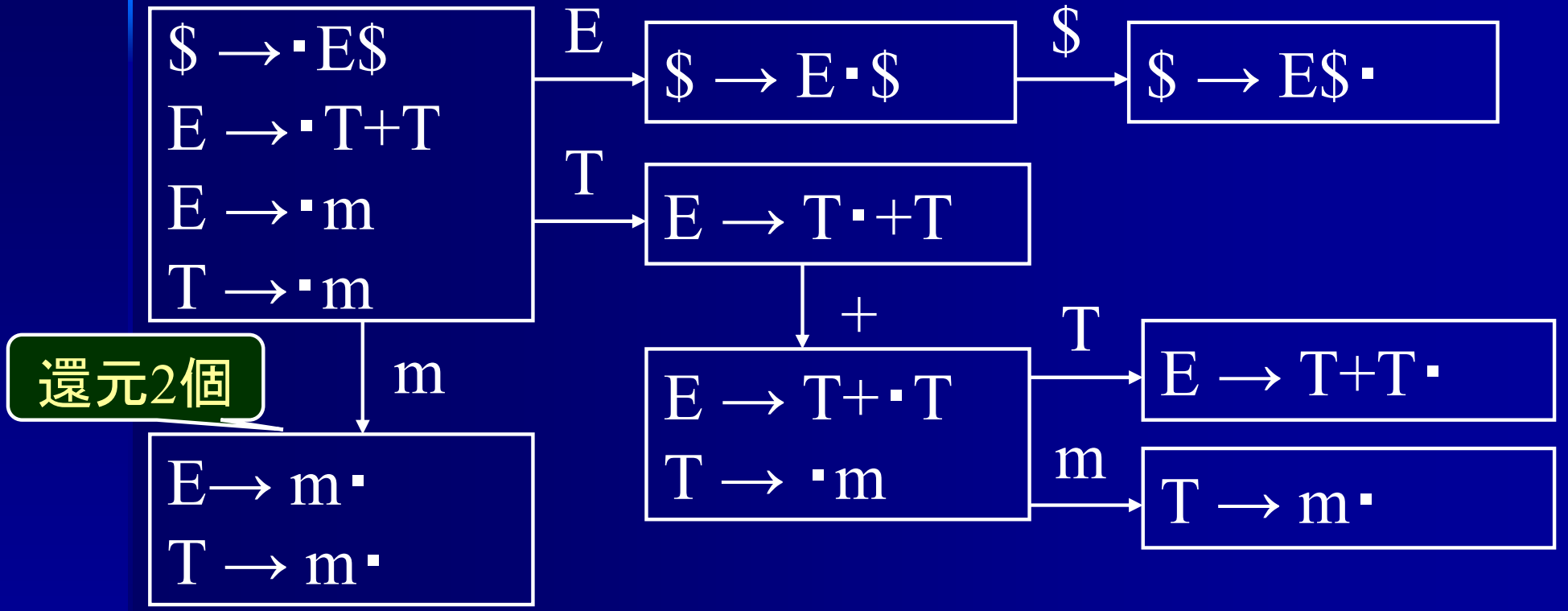
次の記号が + か $ かで移動か還元か判定可能

⇒ SLR(1)文法

SLR(1) 文法ではない例

- $→E$, E→T+T, E→m, T→m

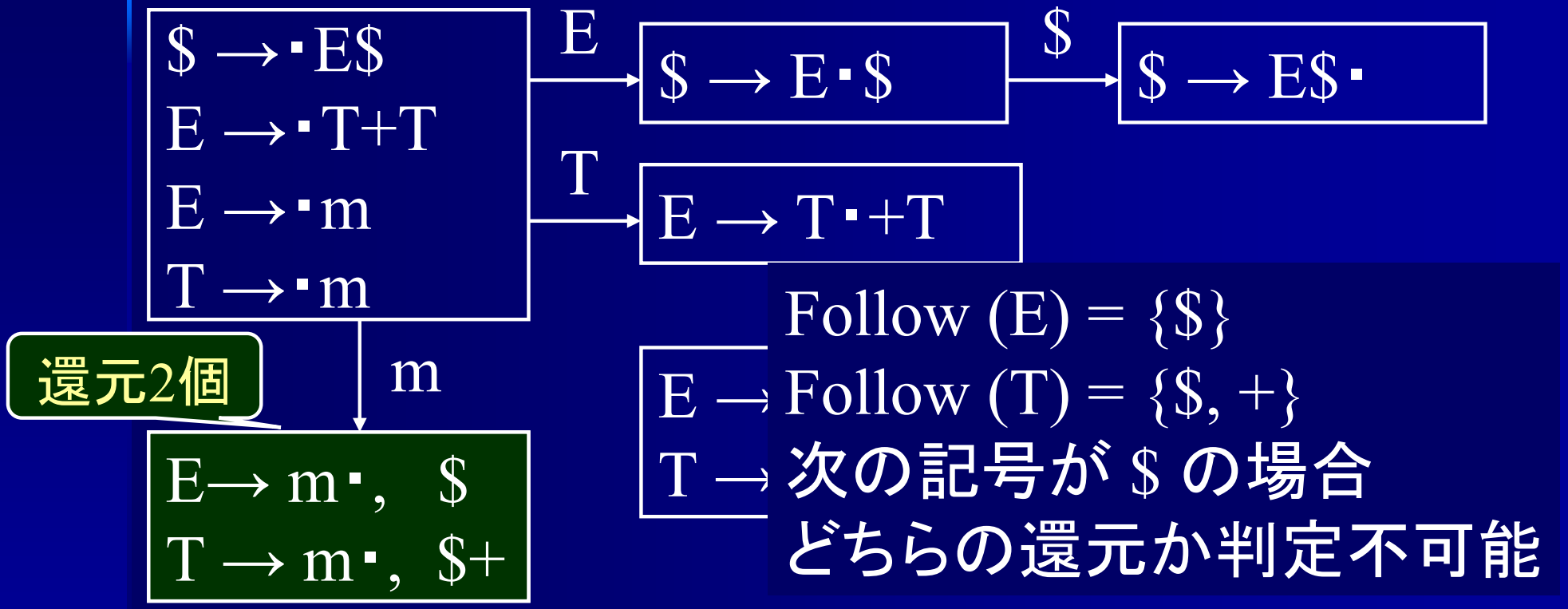
Follow (E) = {$}, Follow (T) = {$, +}



SLR(1) 文法ではない例

- $→E$, E→T+T, E→m, T→m

Follow (E) = {$}, Follow (T) = {$, +}



不都合発生部分をLR(1)項に変換しても解析不可能
⇒ SLR(1) 文法ではない

LALR(1) (Look Ahead LR) 構文解析

- LALR(1) 構文解析
  - まずLR(0) と同様の処理
  - その後全ての閉包に対して先読み
  - 先読みでFollow集合を使えば解析可能

LALR(1) 構文解析の解析表作成

1. 各非終端記号のFollow集合を求める
2. LR(0) 項を求める
3. 閉包(=状態)を求める
4. 各状態からの遷移を求める
5. 初期状態から順に全ての閉包の項をLR(1)項に変換

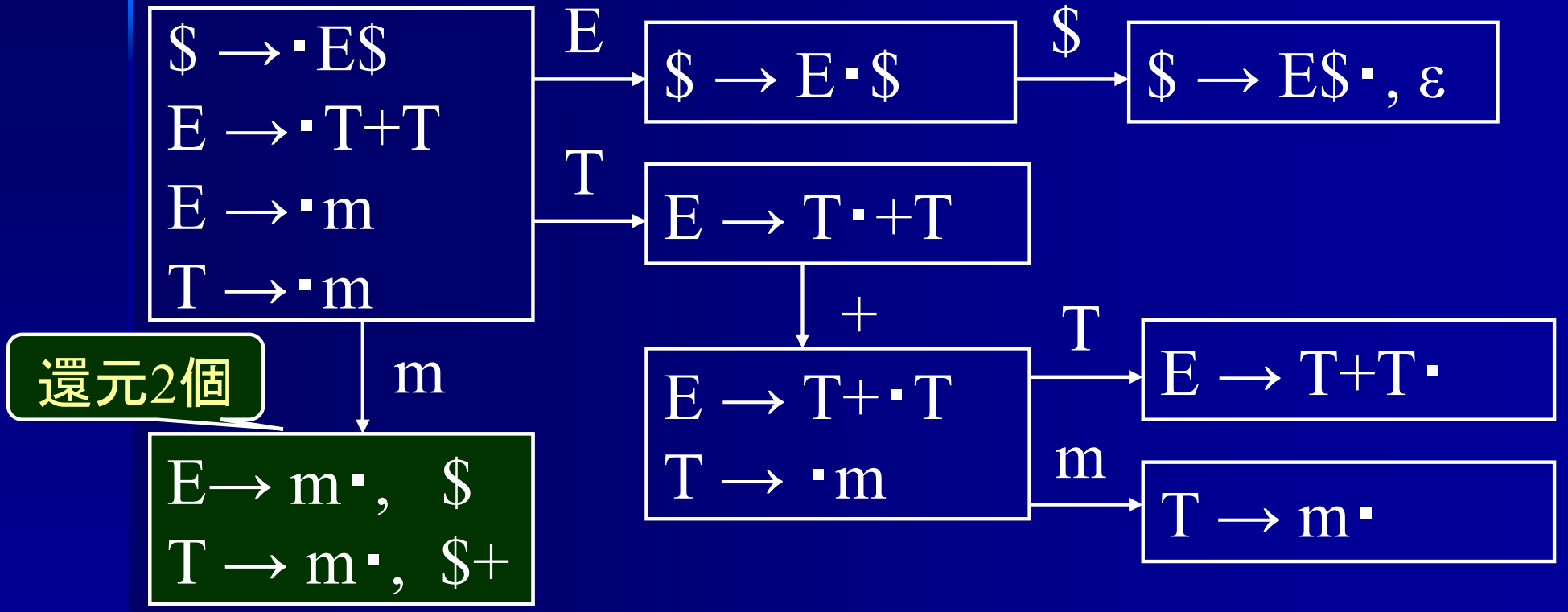
ここまでLR(0)と共通

LALR(1) 文法の例

① LR(0)と
同様に処理

- $→E$, E→T+T, E→m, T→m

Follow (E) = {$}, Follow (T) = {+, $}



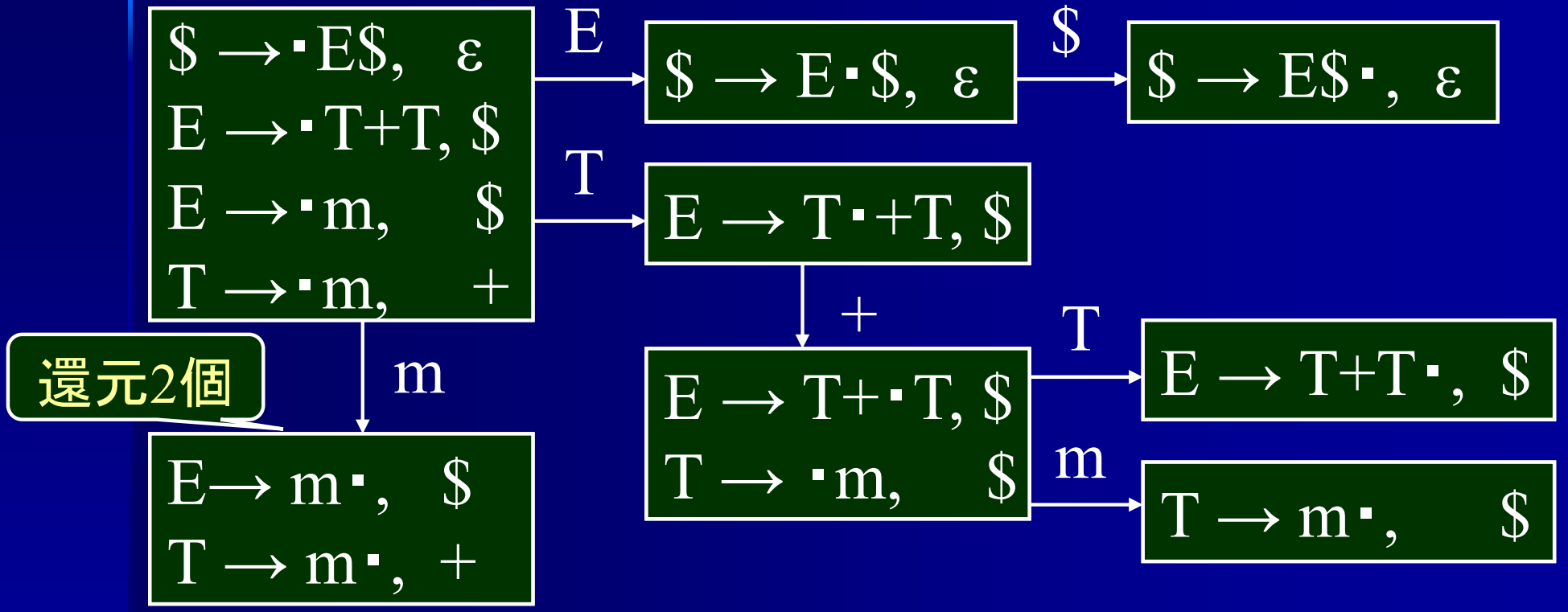
不都合発生部分をLR(1)に変換しても解析不可能
⇒ SLR(1) 文法ではない

LALR(1) 文法の例

② 全ての項を LA(1)に変換

- $→E$, E→T+T, E→m, T→m

Follow (E) = {$}, Follow (T) = {+, $}

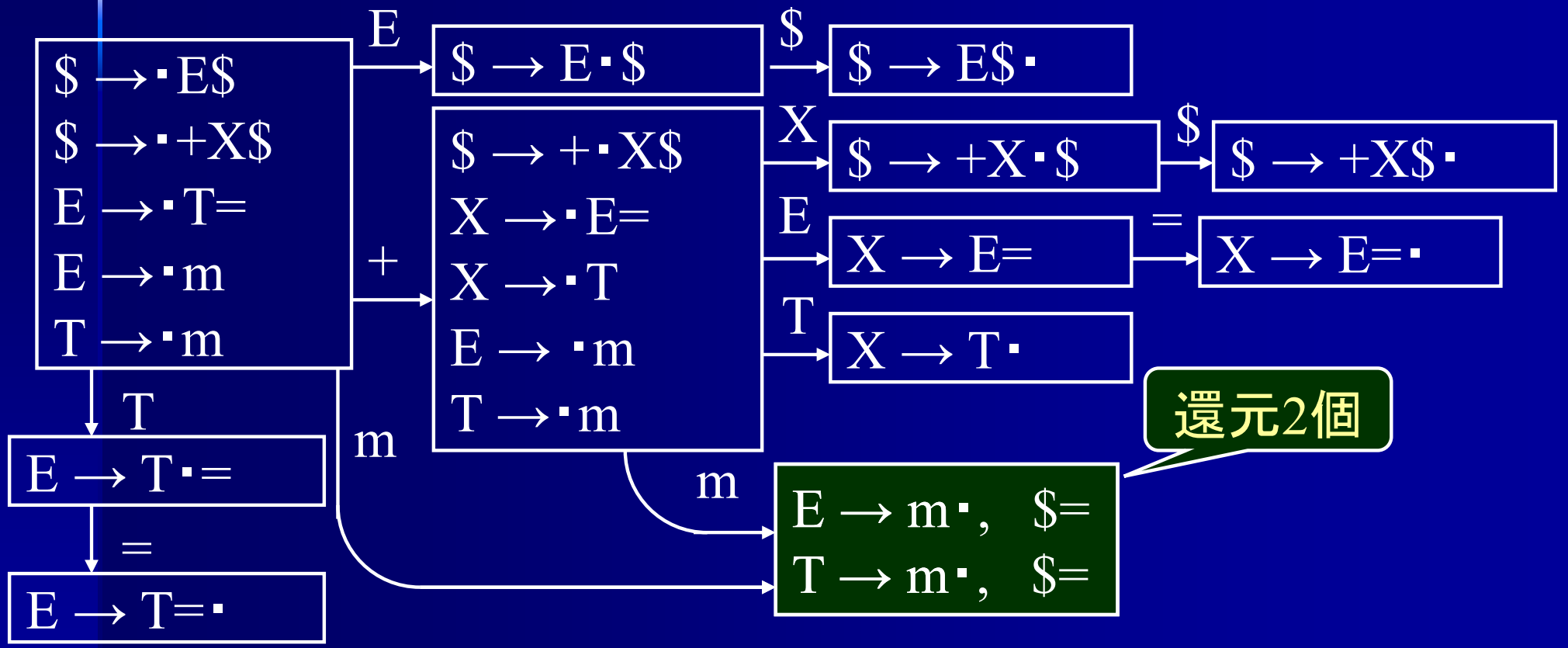


次の記号が $ か + かでどちらの還元か判定可能
⇒ LALR(1) 文法

LALR(1) 文法ではない例

- $→E$, $→+X$, E→T=, X→E=, X→T, E→m, T→m

Follow (E) = {$, =}, Follow(X) ={$}, Follow(T) = {$, =}

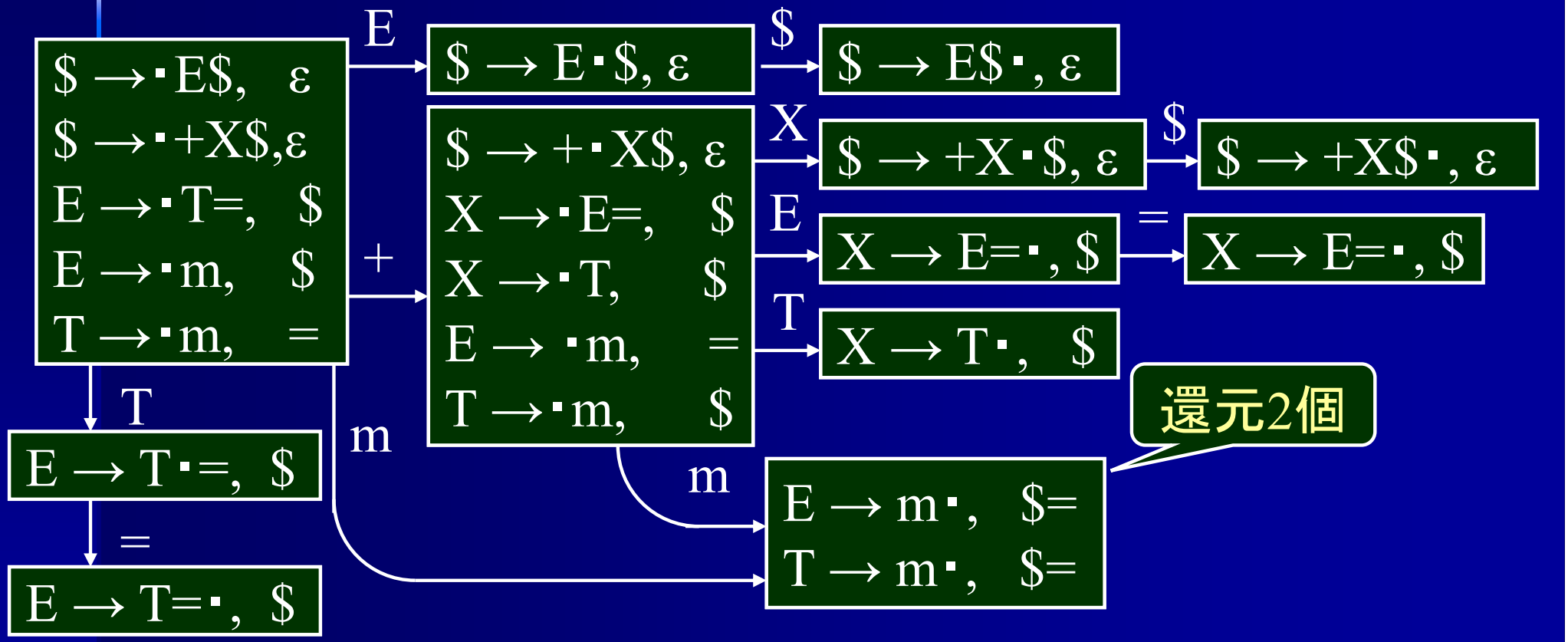


不都合発生部分をLA(1)に変換しても解析不可能
⇒ SLR(1) 文法ではない

LALR(1) 文法ではない例

- $→E$, $→+X$, E→T=, X→E=, X→T, E→m, T→m

Follow (E) = {$, =}, Follow(X) ={$}, Follow(T) = {$, =}



全ての閉包をLR(1)項に変換しても解析不可能
⇒ LALR(1) 文法ではない

LR(1) 構文解析

- LR(1) 構文解析
  - 最初からLR(1)項を使って閉包を求める
  - 先読みでFollow集合を使えば解析可能

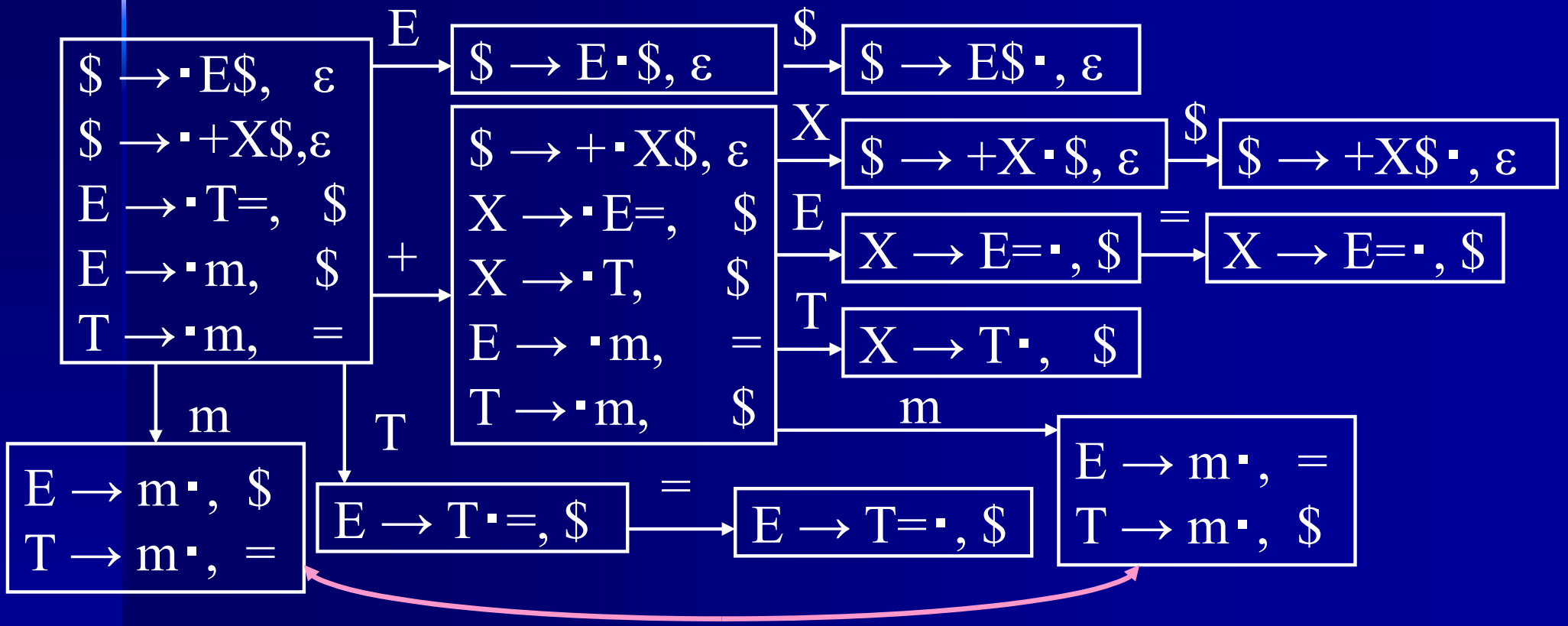
LR(1) 構文解析の解析表作成

1. 各非終端記号のFollow集合を求める
2. LR(1) 項を求める
3. 閉包(=状態)を求める
4. 各状態からの遷移を求める

LR(1) 文法の例

- $→E$, $→+X$, E→T=, X→E=, X→T, E→m, T→m

Follow (E) = {$, =}, Follow(X) ={$}, Follow(T) = {$, =}



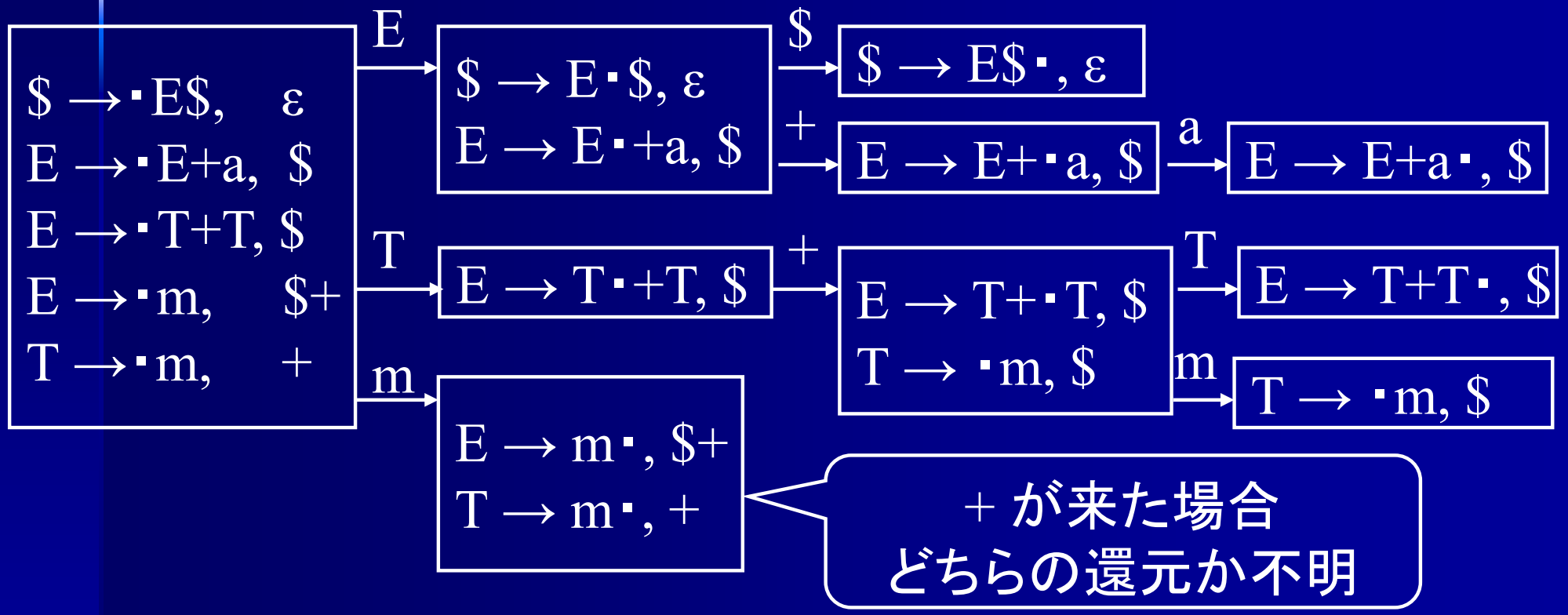
どちらの閉包も次の記号でどちらの還元か判定可能

⇒ LR(1)文法

LR(1)文法ではない例

- $→E$, E→E+a, E→T+T, E→m, T→m

Follow (E) = {$, +}, Follow (T) = {$, +}

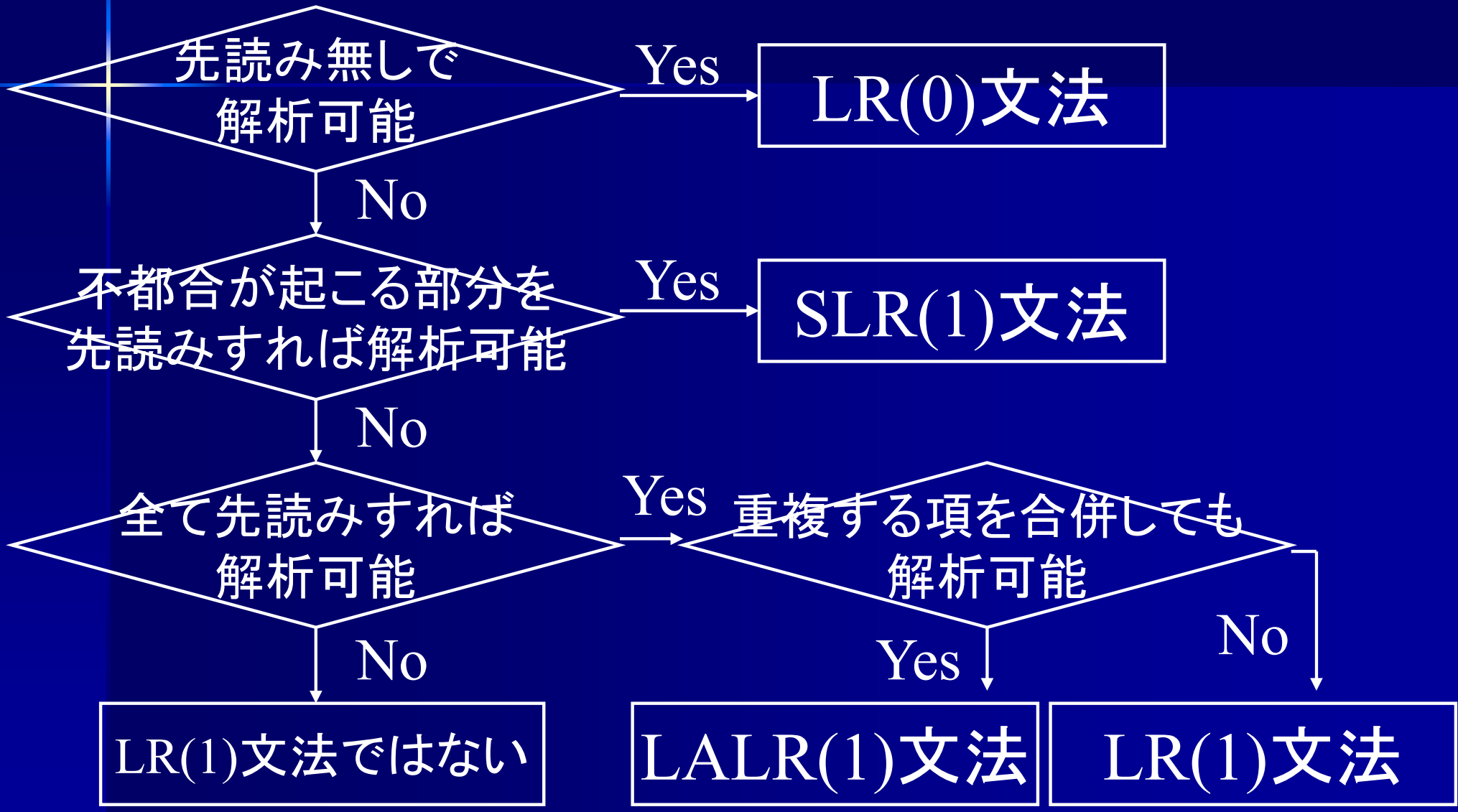


1個先読みしても解析不可能
⇒ LR(1)文法ではない

LR構文解析のクラス

クラス	先読み	状態	解析表サイズ	受理文法	その他の特徴
LR(0)	無し	LR(0)項	小 ↑ ↓ 大	狭 ↑ ↓ 広	
SLR(1)	有り (必要時のみ)	LR(0)項 ⇒LR(1)項 (必要部分のみ)			
LALR(1)	有り	LR(0)項 ⇒LR(1)項 (全体)			コンパイラコンパイラで作成
LR(1)	有り	LR(1)項			

LR文法のクラス


先読み無しで
解析可能
Yes
LR(0)文法
No
不都合が起こる部分を
先読みすれば解析可能
Yes
SLR(1)文法
No
全て先読みすれば
解析可能
Yes
重複する項を合併しても
解析可能
No
Yes
No
LR(1)文法ではない
LALR(1)文法
LR(1)文法

構文解析の種類

下降型解析 (top-down parsing)	再帰下降解析 (recursive descent parsing)
	LL解析 (Left to right scan & Left most derivation)
上昇型解析 (bottom-up parsing)	演算子順位構文解析 (operator precedence parsing)
	LR解析 (Left to right scan & Right most derivation)

構文解析の長所と短所

文法	解析表と文法の対応	解析表サイズ	適用可能文法範囲	解析効率	その他の長所
再帰下降	高	小	狭	高	構文エラー発見時のエラーメッセージを作り易い コード生成部を埋め込み易い
LL					
演算子順位	式:高 全体:低	式:小 全体:大	狭	式:最高 全体:低	
LR	低	大	広	低	構文エラー発見が速い

LL解析とLR解析の解析表

- LL解析
  - 文法から直接解析表を作成可能
  - サイズが小さい　$LL(k) : |\mathbf{N}| \times |\mathbf{T}|^k$

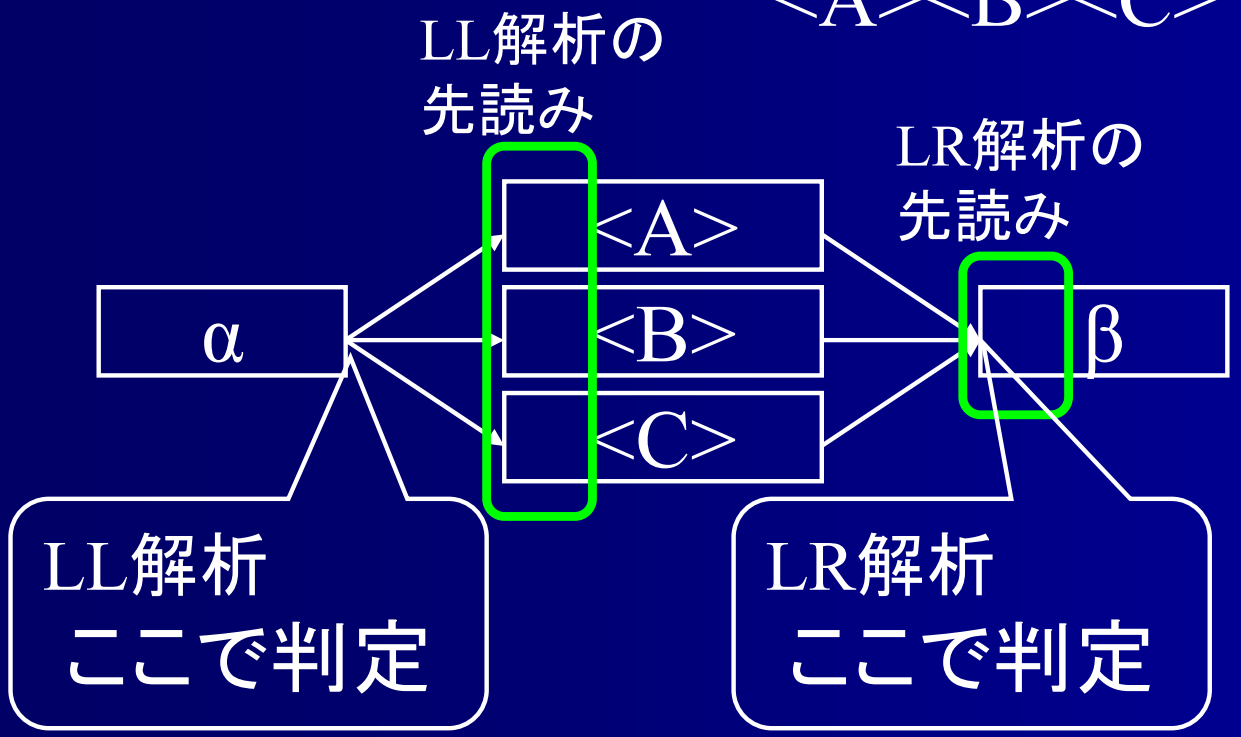
  ⇒解析表の作成は容易
- LR解析
  - 解析表の導出作業が必要
  - サイズが大きい　$LR(k) : 2^{|\mathbf{P}|} \times (|\mathbf{N}|+|\mathbf{T}|) \times |\mathbf{T}|^k$

  ⇒解析表の作成は困難

LL解析とLR解析の能力差

例：<X> ::= α (<A> | <B> | <C>) β の解析

<A><B><C> のどれが来る？

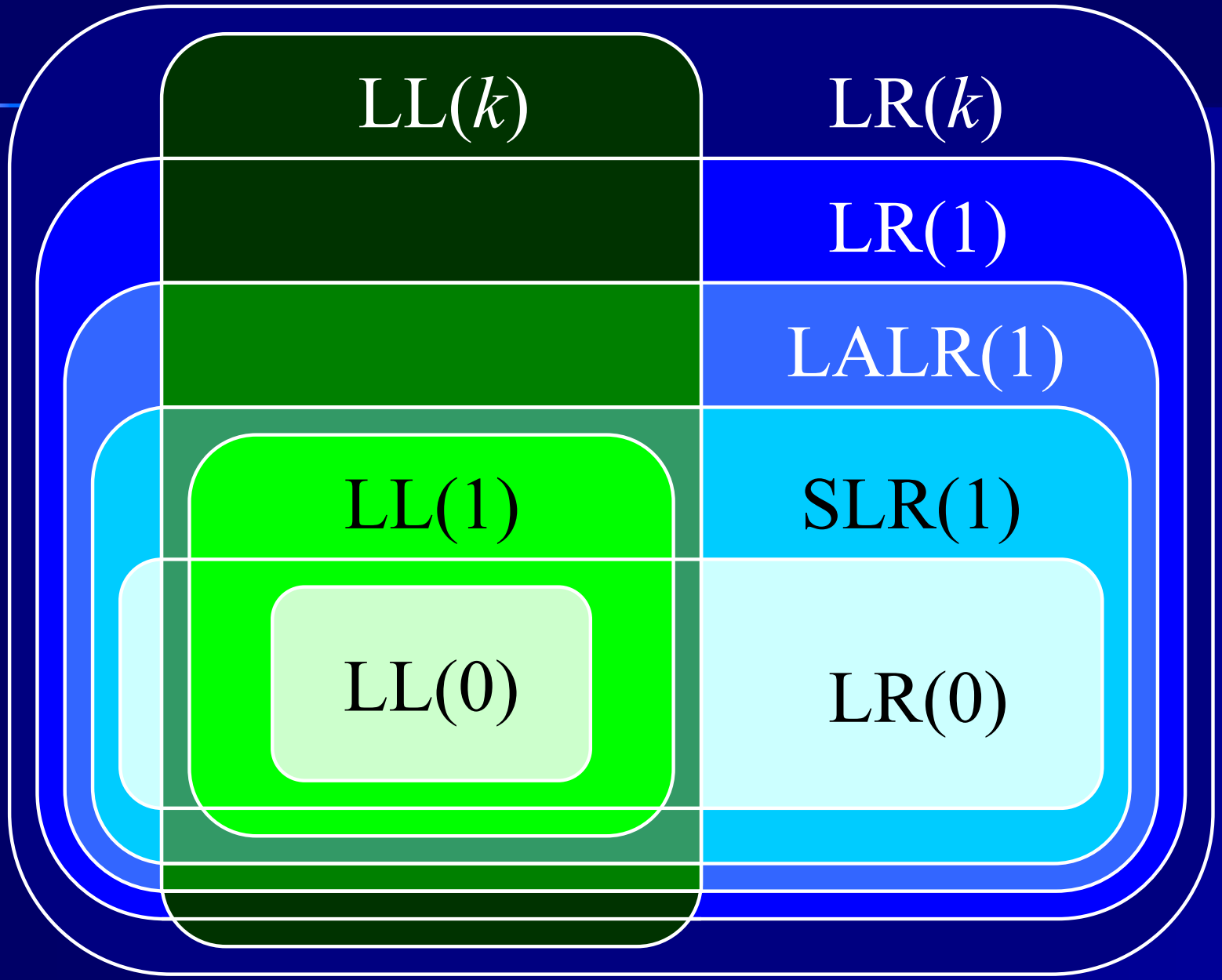


LL解析：読み込む前に判定

LR解析：読み込み後に判定

⇒ LR 解析の方がより広い文法を受理できる

LL解析とLR解析の能力差



オンライン試験

- 試験日：7月26日(水)
- 試験時間：60分
- 試験範囲：第1～14回
- 配点：70点満点
- 持ち込み：全て可
  - ただし外部との通信は禁止