Metody kompilacji

Wykłady 9-10
Analiza wstępująca

Analiza wstępująca (*A Bottom-up Parse*)

- Analiza wstępująca jest bardziej ogólna niż zstępująca.
- Jest preferowana w praktyce, ale jest bardziej złożona niż zstępująca.

Analiza wstępująca

- Dalej będziemy korzystać z następującej gramatyki:

$E \rightarrow T + E \mid T$

$T \rightarrow \textbf{int} * T \mid \textbf{int} \mid (E)$

Analiza wstępująca

Redukcja

- *Redukcja* jest to krok odwrotny do wyprowadzenia.
- Rozważmy zdanie: **int * int + int**

  Korzystając z produkcji $T \rightarrow$ **int**, wynik redukcji jest następujący:

  int * T **+ int**

Idea

Analiza wstępująca *redukuje* zdanie do symbolu startowego:

int * int + int	$T \rightarrow$ **int**
int * T + **int**	$T \rightarrow$ **int** * T
T + **int**	$T \rightarrow$ **int**
$T + T$	$E \rightarrow T$
$T + E$	$E \rightarrow T + E$
E	

Kierunek analizy wstępującej: od zdania do symbolu startowego

Idea

Jeśli zastosujemy produkcje wykorzystane przez analizę wstępującą w kolejności odwrotnej, to uzyskamy wyprowadzenie prawostronne:

int * int + int	$T \rightarrow$ **int**
int * T + **int**	$T \rightarrow$ **int** * T
T + **int**	$T \rightarrow$ **int**
$T + T$	$E \rightarrow T$
$T + E$	$E \rightarrow T + E$
E	

Kolejność zastosowania produkcji

Kolorem czerwonym zaznaczone są wybierane symbole

Analiza wstępująca

int * int + int

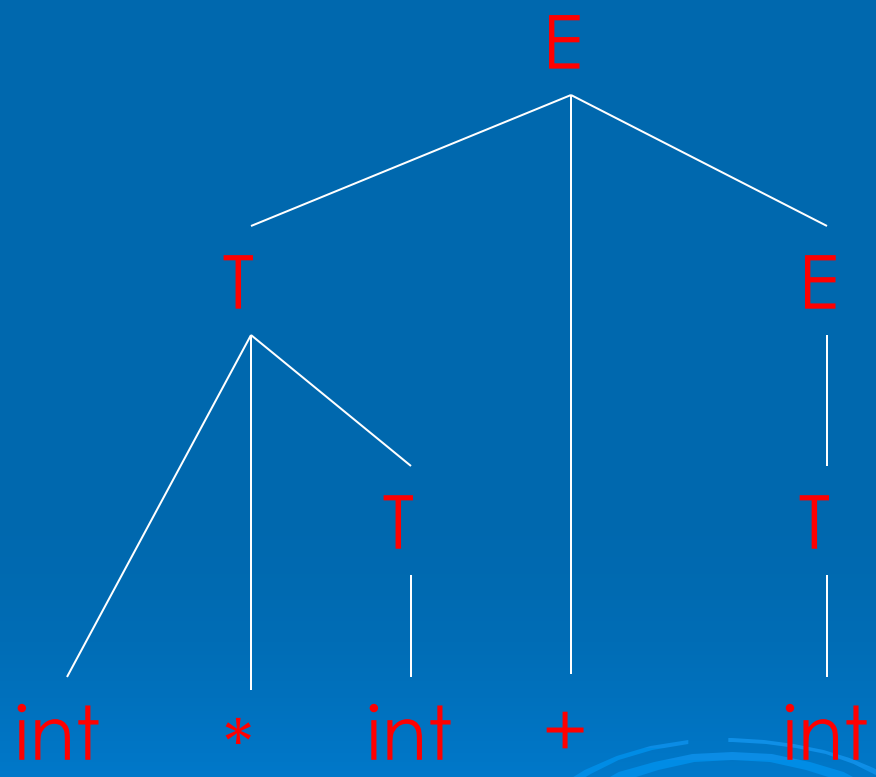
int * *T* + int

T + **int**

T + *T*

T + *E*

E



Analiza wstępująca

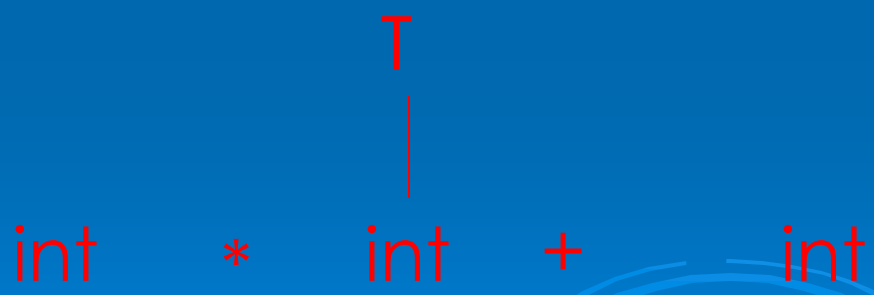
int * int + int

int * int + int

Analiza wstępująca

int * int + int

int * *T* + int

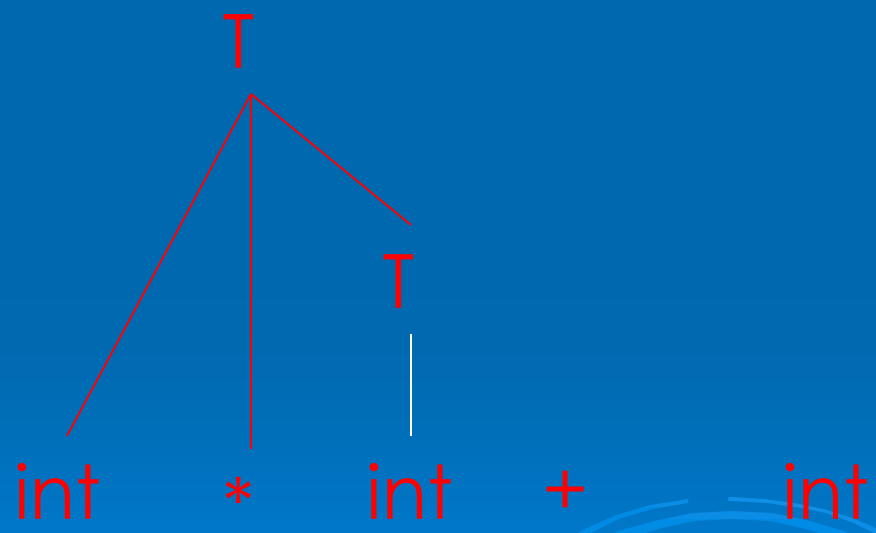


Analiza wstępująca

int * int + int

int * *T* + int

T + **int**



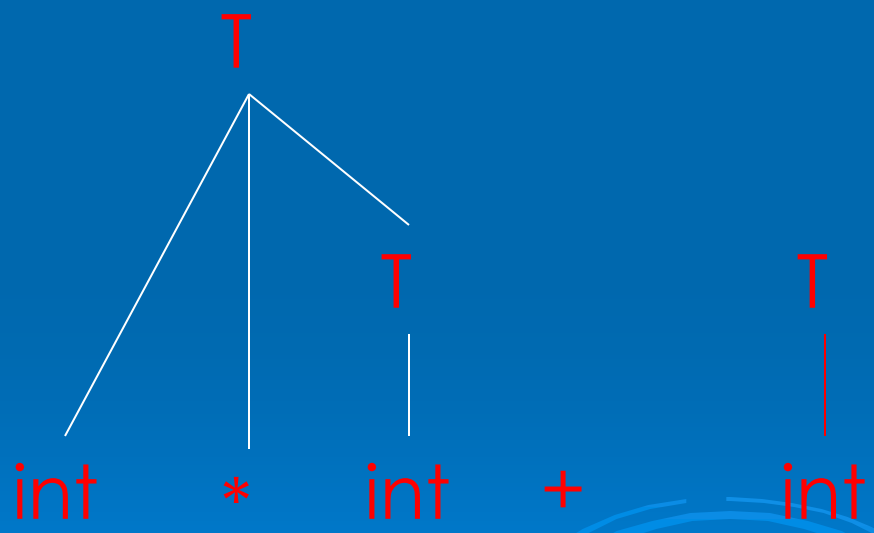
Analiza wstępująca

int * **int** + **int**

int * *T* + **int**

T + **int**

T + *T*



Analiza wstępująca

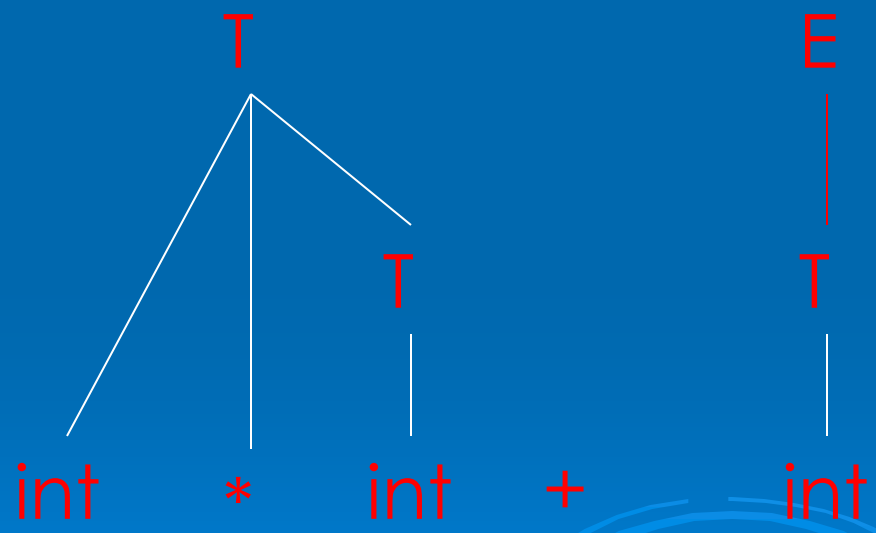
int * **int** + **int**

int * *T* + **int**

T + **int**

T + *T*

T + *E*



Analiza wstępująca

int * int + int

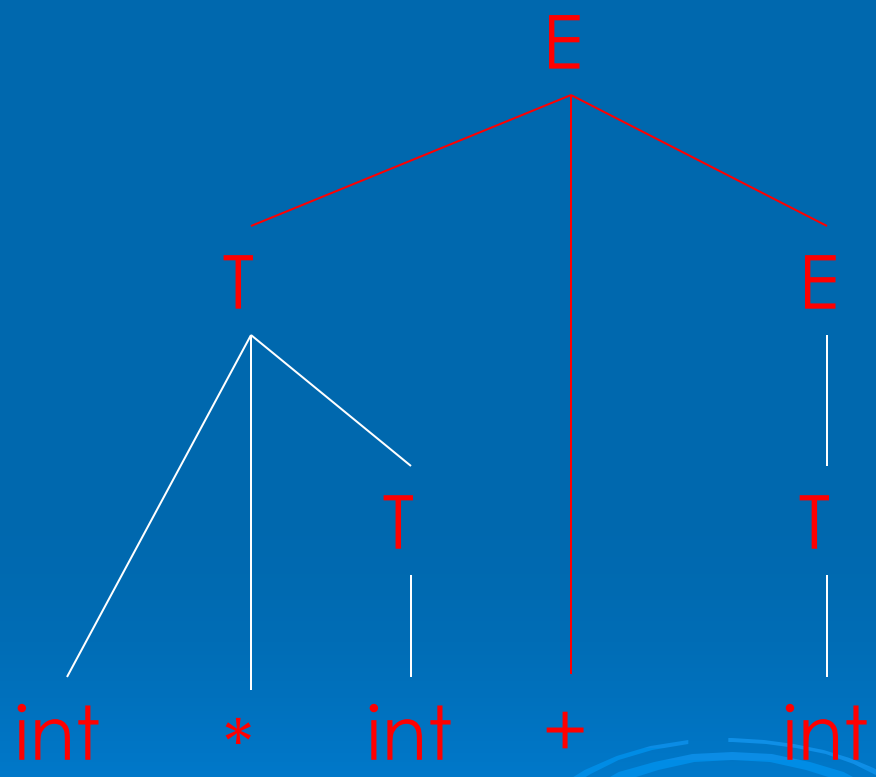
int * *T* + int

T + **int**

T + *T*

T + *E*

E



Analiza wstępująca
Algorytm parsowania

Wejście: zdanie I

Powtarzaj

w zdaniu I wybierz niepusty podciąg β,

gdzie β jest prawą stroną produkcji $X \rightarrow \beta$;

w zdaniu I zastąp β przez X;

Dopóki I != "S" (symbol startowy) lub wszystkie możliwości są wyczerpane.

Pytania

- Jak wybrać podciąg w każdym kroku?
- Czy ten algorytm zawsze się kończy?
- Jaka jest jego złożoność?
- Czy obsługuje dowolną gramatykę?

Analiza wstępująca

Ważny wniosek:

- Niech $\alpha\beta\omega$ będzie napisem bieżącym po jakiejś liczbie kroków analizy wstępującej.
- Załóżmy, że następna redukcja korzysta z produkcji $X \rightarrow \beta$
- Wtedy ω jest to podciąg terminali.

Dlaczego? Dlatego, że $\alpha X\omega \rightarrow \alpha\beta\omega$ jest to krok wyprowadzenia prawostronnego, czyli ω nie może zawierać żadnego nieterminala.

Analiza wstępująca

- Idea: Podziel napis na dwa podciągi:

  prawy podciąg jest jeszcze niezbadany przez parser (ciąg terminali),

  lewy podciąg zawiera tylko nieterminale.

- Punkt podziału jest oznaczony symbolem

  | , który nie jest częścią napisu.

Parsowanie Shift-Reduce

Analiza wstępująca korzysta z dwóch akcji :

Przesunięcie*(Shift)*

Redukcja*(Reduce)*

Shift

- *Shift:* Przesuń symbol | o jedno miejsce w prawo
  - Przesuwa terminal do lewego podciągu

$$ABC|xyz \Rightarrow ABCx|yz$$

Reduce

- Niech $A \rightarrow xy$ będzie produkcją
- Reduce: Zastąp podciąg xy (prawa strona produkcji) przez A (lewa strona produkcji)
  - Przykład:

$$Cbxy|ijk \Rightarrow CbA|ijk$$

Parsowanie Shift-Reduce

|int * int + int	shift
int | * int + int	shift
int * | int + int	shift
int * int | + int	reduce $T \rightarrow$ **int**
int * *T* | + int	reduce $T \rightarrow$ **int** * T
T **| + int**	shift
T **+ | int**	shift
T **+ int |**	reduce $T \rightarrow$ **int**
T **+ *T* |**	reduce $E \rightarrow T$
T **+ *E* |**	reduce $E \rightarrow T + E$
E |	

Parsowanie Shift-Reduce

int * int + int

Parsowanie Shift-Reduce

int * int + int

int | * int + int

int * int + int

↑

Parsowanie Shift-Reduce

|int * int + int

int | * int + int

int * | int + int

int * int + int

↑

Parsowanie Shift-Reduce

|int * int + int

int | * int + int

int * | int + int

int * int | + int

Parsowanie Shift-Reduce

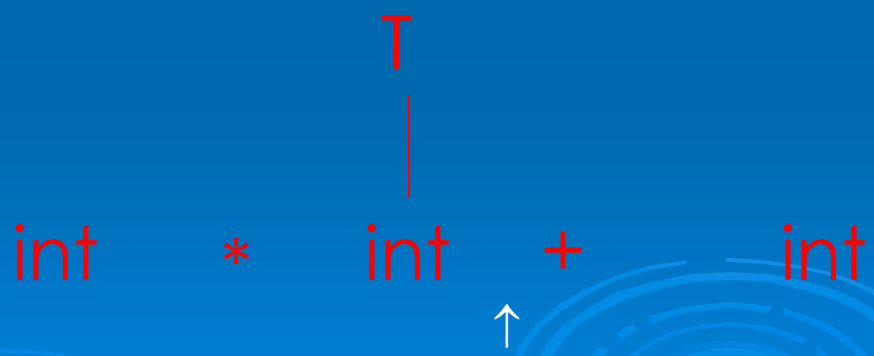
|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * *T* | + int



Parsowanie Shift-Reduce

|int * int + int

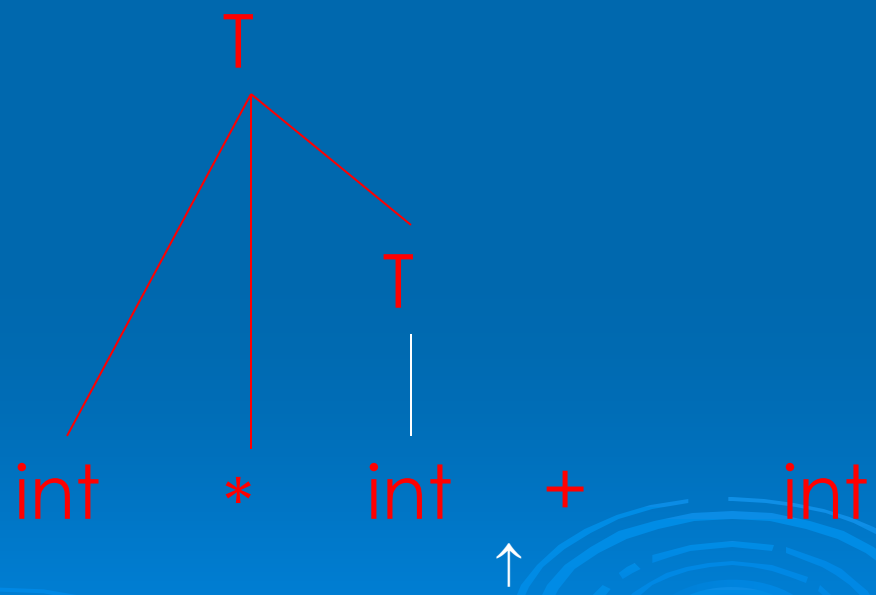
int | * int + int

int * | int + int

int * int | + int

int * *T* | + int

T | + int



Parsowanie Shift-Reduce

|int * int + int

int | * int + int

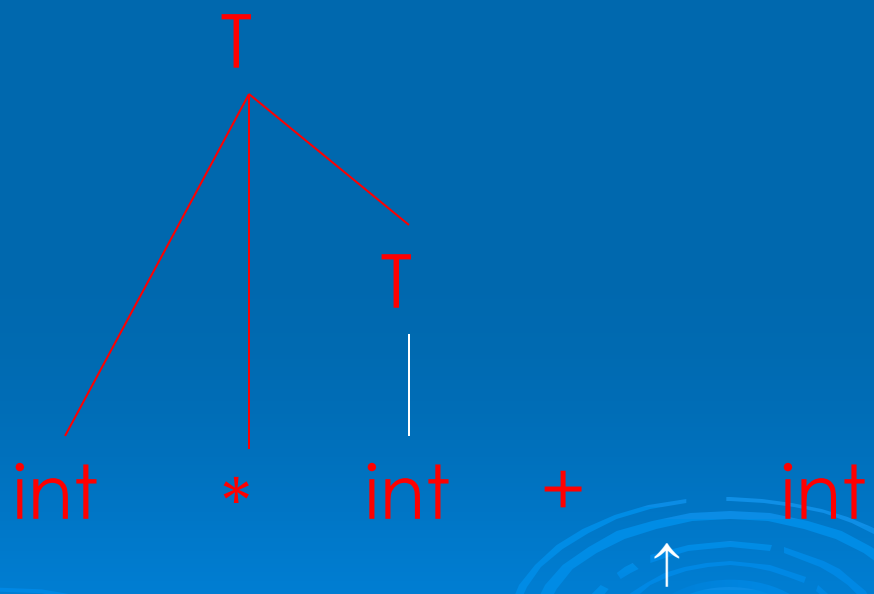
int * | int + int

int * int | + int

int * *T* | + int

T **| + int**

T **+ | int**



Parsowanie Shift-Reduce

|int * int + int

int | * int + int

int * | int + int

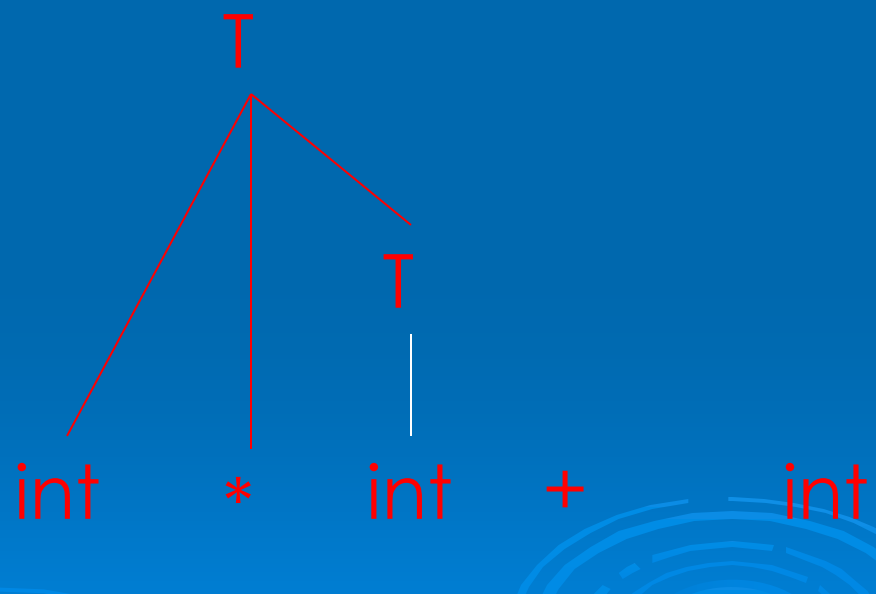
int * int | + int

int * *T* | + int

T **| + int**

T **+ | int**

T **+ int |**



Parsowanie Shift-Reduce

|int * int + int

int | * int + int

int * | int + int

int * int | + int

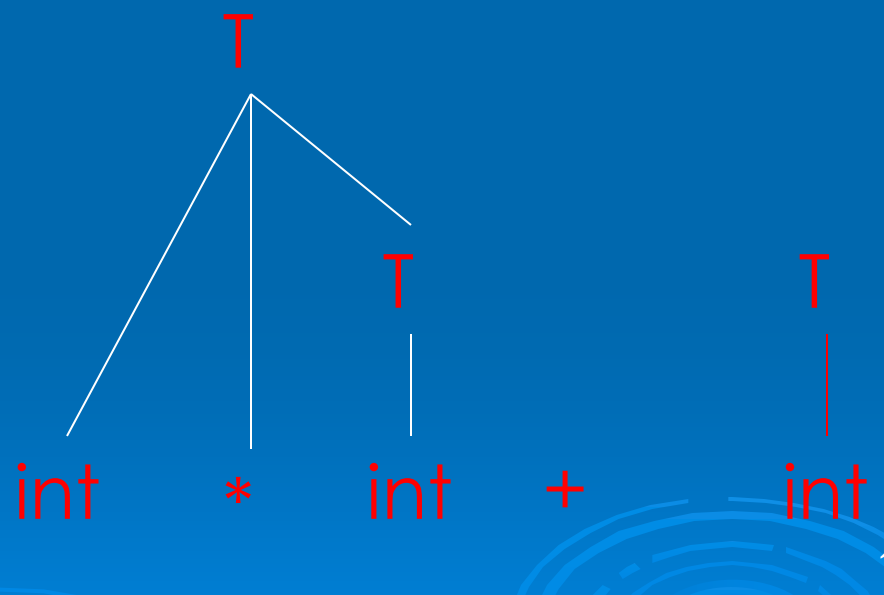
int * *T* | + int

T **| + int**

T **+ | int**

T **+ int |**

T **+ *T* |**



Parsowanie Shift-Reduce

|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * *T* | + int

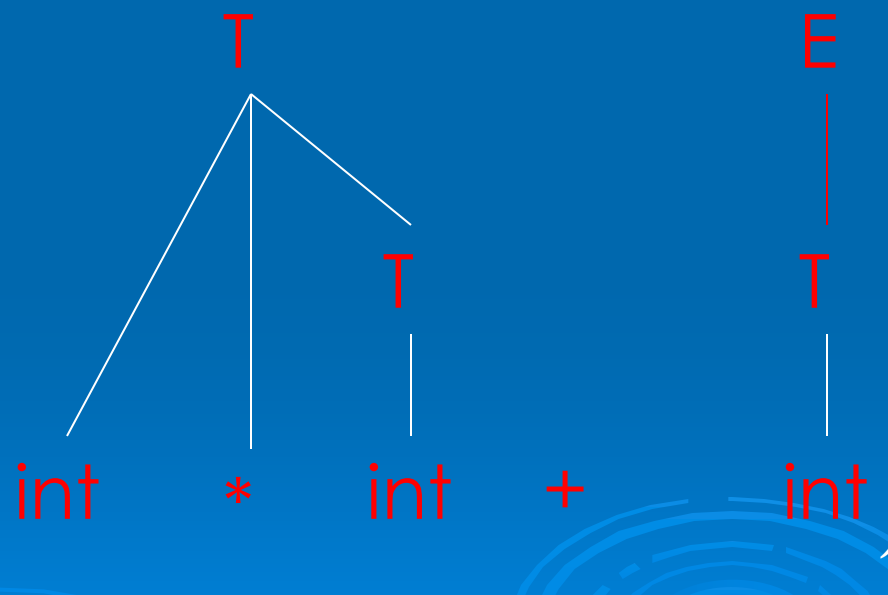
T **| + int**

T **+ | int**

T **+ int |**

T **+ *T* |**

T **+ *E* |**



Parsowanie Shift-Reduce

|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * *T* | + int

T **| + int**

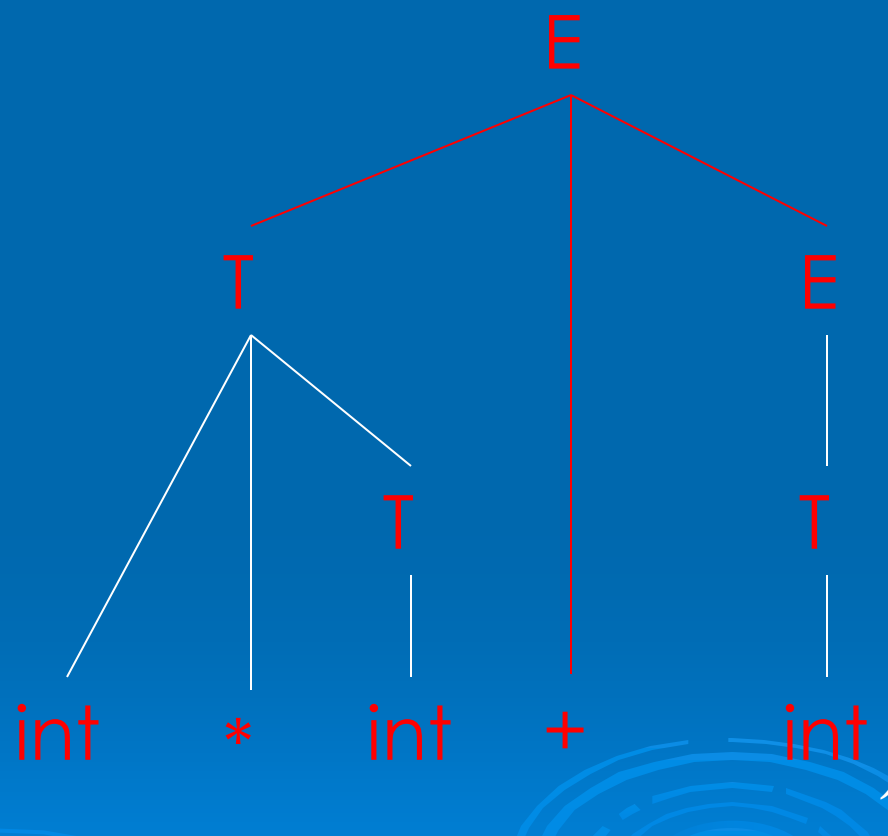
T **+ | int**

T **+ int |**

T **+ *T* |**

T **+ *E* |**

E **|**



Zastosowanie stosu

- Lewy ciąg może być obsługiwany za pomocą stosu:
  - przed parsowaniem na szczyt stosu dodajemy symbol |
  - shift: dodaje terminal na stos
  - reduce: zdejmuje 0 lub więcej symboli(prawa strona zastosowanej produkcji) ze stosu i dodaje nieterminal na stos(lewa strona zastosowanej produkcji)

Parsowanie Shift-Reduce

- Kiedy shift, kiedy reduce?
  - Rozważmy napis: **int * int + int**
  - Stosując $T \rightarrow$ **int** możemy zredukować napis jak wyżej do

    T | * **int + int**
  - Fatalny błąd: Nigdy w ten sposób nie zredukujemy zdania do symbolu startowego E.

Zastosowanie stosu

Strategia wyboru akcji:

- Jeśli uchwyt jest na szczycie stosu, to wykonaj reduce
- Inaczej wykonaj shift

Parsowanie Shift-Reduce

Uchwyt

- Nieformalnie, "uchwyt" jest to podciąg pasujący do prawej strony jakiejś produkcji.

- Podstawowe pytanie: jak w sposób formalny rozpoznać uchwyt?

Parsowanie Shift-Reduce

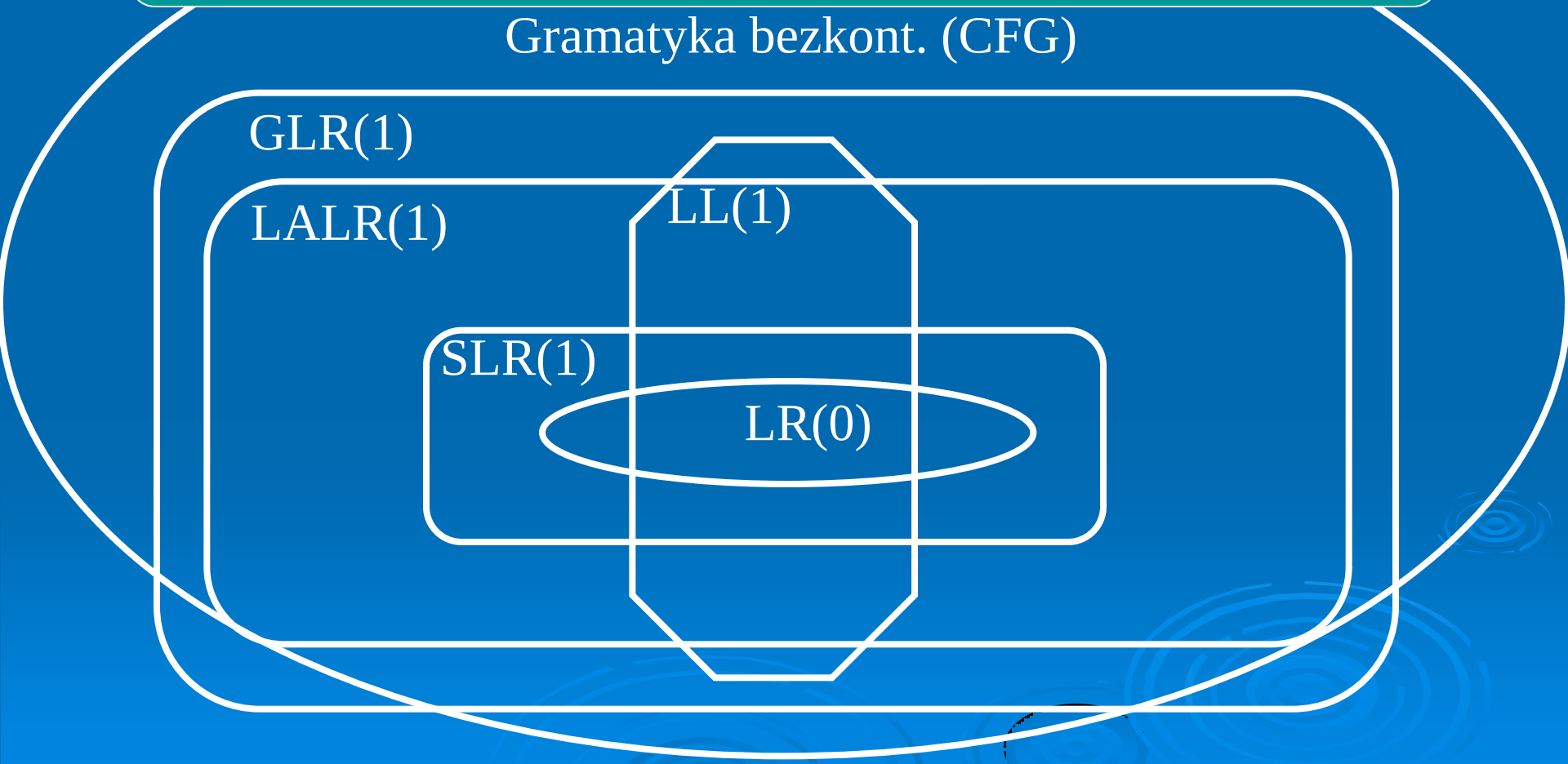
Uchwyt

- Zostało opracowanych wiele metod do rozpoznania uchwytu w sposób formalny.

- Wiele metod daje nam wiele gramatyk

Gramatyki

Im większa jest powierzchnia, zaznaczona dla jakiejś gramatyki, tym większy jest zakres jej zastosowania.



Parser LR (k)

„L” oznacza przeglądanie wejścia od lewej do prawej,

„R” oznacza budowę zdania wejściowego przez zastosowanie wyprowadzenia prawostronnego w oparciu o produkcje zwrócone przez parser w kolejności od ostatniej do pierwszej,

„k” oznacza liczbę symboli podglądanych podczas analizy.

Parser LR (k)

Im większa jest wartość k, tym większy jest zakres stosowalności parsera –może on parsować więcej gramatyk.

Ale jednocześnie ze wzrostem k rośnie złożoność procesu tworzenia parsera oraz czas parsowania.

Parser LR (k)

- Można zbudować analizatory LR do prawie wszystkich konstrukcji z języków programowania.

Parser LR (k)

- Jest wiele modyfikacji parserów LR(k).
- Każda modyfikacja ma taką samą architekturę parsera i taki sam sposób jego działania.
- Jedyna różnica – to sposób tworzenia tablic parsowania.
- Dalej rozważamy najbardziej popularne modyfikacje parsera LR(k).

Parser SLR (k)

- Simple LR(k) parser – prosty LR(k) parser.
- Jest najsłabszy ze względu na liczbę gramatyk, dla których działa, ale jest najprostszy w implementacji.

LALR (1)

- LookAhed LR(1) - podglądający LR(1) parser
- Tablice uzyskiwane przy jego zastosowaniu są znacznie mniejsze niż tablice LR(1).
- Narzędzia YACC i Bison są oparte na LALR(1).

Parser GLR

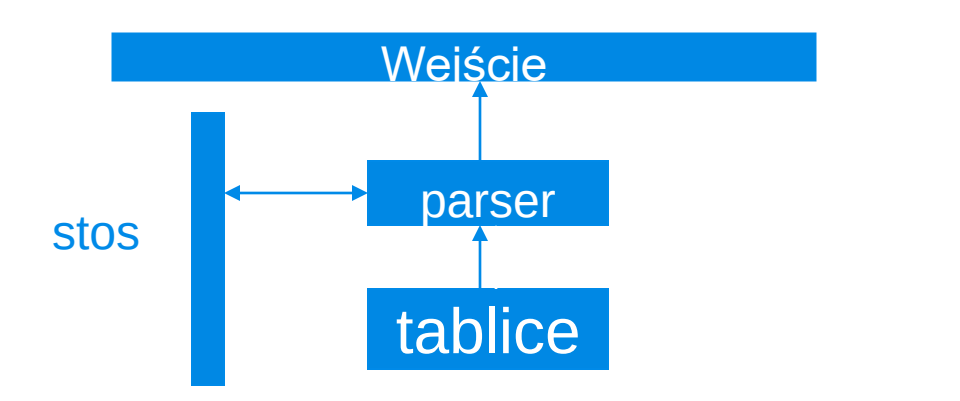
Generalized (Uogólniony) LR parser jest najbardziej zaawansowaną modyfikacją parsera LR – może parsować prawie każdą gramatykę.

Parser LR (1)

- Parsowanie jest wykonywane w oparciu o automat skończony.
- Parsowanie jest niezależne od języka.
- Automat skończony jest generowany na podstawie gramatyki wejściowej.

Parser LR (1)

- Ogólna architektura parsera LR(1)



Automat PDA

- Automat skończony, który korzysta ze stosu, jest nazywany automatem PDA (*pushdown automaton).*

- Działanie PDA jest określone przez jego obecny stan przechowywany na szczycie stosu oraz bieżący symbol wejściowy.

Parser LR (1)

- Parser LR wykorzystuje tablicę parsowania, bufor wejściowy i stos stanów.
- Wykonuje trzy akcje:
  - Przesuwa (*Shift*) token z bufora wejściowego na stos.
  - Redukuje (*Reduce*) zawartość stosu przez zastosowanie produkcji.
  - Przechodzi do nowego stanu (*Go to*) .

Sytuacje LR(0)

- Żeby zbudować tablicę parsowania, musimy najpierw znaleźć sytuacje (*items*) LR(0).

- Sytuacją LR(0) nazywamy produkcję z kropką (•) w jakimś miejscu jej prawej strony.

Sytuacje LR(0)

- Dla produkcji

$$E \rightarrow E + T,$$

sytuacje LR(0) są jak niżej:

$E \rightarrow \bullet\, E + T$

$E \rightarrow E \bullet + T$

$E \rightarrow E + \bullet\, T$

$E \rightarrow E + T \bullet$

Kropka (•) jest we wszystkich możliwych miejscach prawej strony produkcji

Sytuacje LR(0)

- Interpretacja sytuacji $A \rightarrow \alpha \bullet \beta$:

  "napis α już został przetworzony, więc możemy przetwarzać β".

- Czy rzeczywiście będziemy przetwarzać β jest uzależnione od kolejnych symboli wejścia.

Parser LR

- Zbiór sytuacji LR (0) reprezentuje pojedynczy stan PDA.
- W parsowaniu LR potrzebujemy wzbogacić gramatykę przez dodanie produkcji:

  $S' \rightarrow S,$

  gdzie S' jest to nowy symbol startowy.
- Gramatyka wzbogacona gwarantuje to, że nowy symbol startowy nie spowoduje rekurencji.

Stany PDA

- Stan początkowy nazywa się I_0 (sytuacja 0).
- Stan I_0 jest to domknięcie zbioru zawierającego jedną sytuację:

$$\{S' \rightarrow \bullet\, S\}.$$

Stany PDA

- **Sposób obliczenia domknięcia zbioru sytuacji**:
  - Dla każdej sytuacji $A \rightarrow \alpha \bullet B\beta$ w zbiorze oraz dla każdej produkcji $B \rightarrow \gamma$ gramatyki, dodaj sytuację $B \rightarrow \bullet \gamma$ do zbioru,
  - Kontynuuj, dopóki nie będzie żadnych nowych elementów do dodania do zbioru.

Przykład

- Dla gramatyki

$$E' \rightarrow E$$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$$

Przykład

stan I_0 zawiera sytuacje, które należą do domknięcia sytuacji $E' \to \bullet\ E$:

$E' \to \bullet\ E$

$E \to \bullet\ E + T$

$E \to \bullet\ T$

$T \to \bullet\ T * F$

$T \to \bullet\ F$

$F \to \bullet\ (E)$

$F \to \bullet$ **id**

$F \to \bullet$ **num**

$E' \to E$

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to (E) \mid$ **id** $\mid$ **num**

Przejścia

- Jeśli $A \rightarrow \alpha \bullet X\beta$ jest to sytuacja opisująca (razem z innymi sytuacjami) jakiś stan, to
  - Przejście z tego stanu może nastąpić, gdy symbol X jest przetwarzany.
  - Przejście następuje do stanu, który jest reprezentowany domknięciem sytuacji $A \rightarrow \alpha X \bullet \beta$.

Przykład

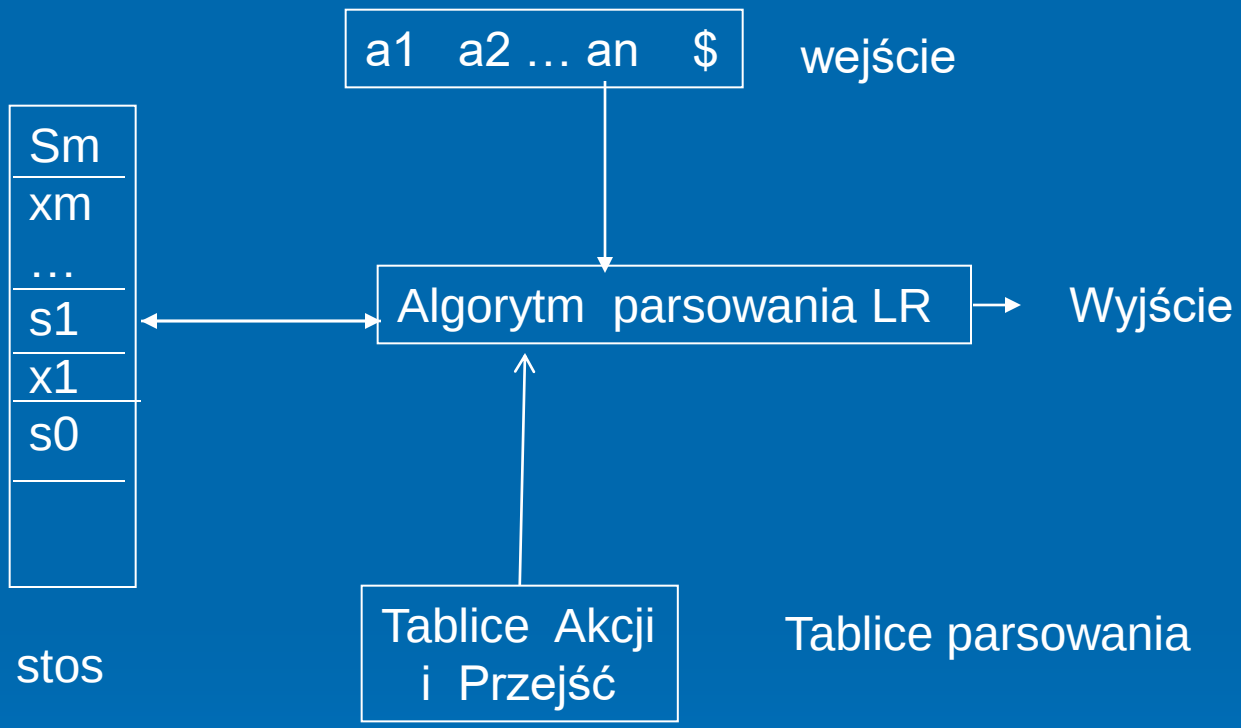
Na przykład, dla E, ze zbioru sytuacji

$$E' \to \bullet\, E,\ \ E \to \bullet\, E + T$$

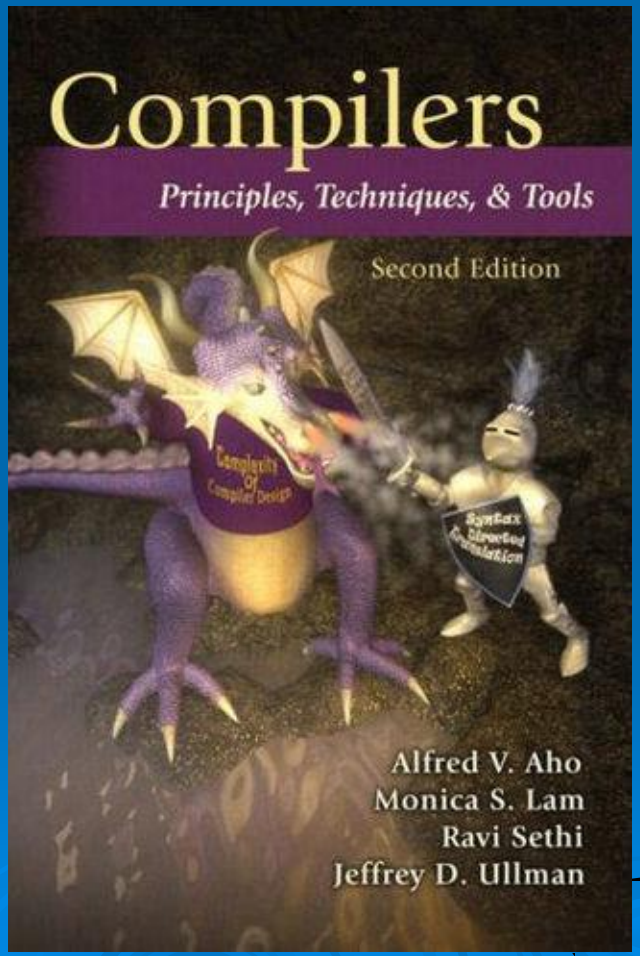
mamy przejścia do

$$E' \to E\, \bullet,\ \ E \to E\, \bullet + T.$$

Architektura parsera LR(k)



- Dla każdej modyfikacji parsera LR(k) zostały opracowane algorytmy tworzenia tablic parsowania, które można znaleźć w książce:



Tablice parsowania

- Tablica parsowania ma dwie części: tablicę akcji i tablicę przejść.
- Wpisy w tablicy akcji [s, t] mogą mieć cztery wartości:
  - Przesunięcie (shift) si, gdzie si oznacza stan, do którego należy przejść.

Tablice parsowania

- Redukcja (reduce) z zastosowaniem produkcji (zazwyczaj podany jest numer produkcji).
- Akceptacja (accept)
- Błąd (error)

Tablice parsowania

- Wpis w tablicy przejść(*go to*) [s, T] = si oznacza przejście ze stanu s i symbolu nieterminalnego T do stanu si.
- Z tablicy przejść korzystamy po każdej redukcji: bierzemy bieżący stan oraz symbol na szczycie stosu(lewa strona zastosowanej produkcji) jako wejście do tablicy i dodajemy stan, który zwraca tablica, na szczyt stosu.

Gramatyki LR(k)

- Gramatyka, która pozwala na parsowanie za pomocą analizatora LR, podglądającego co najwyżej k symboli wejściowych, nazywana jest gramatyką LR(k).

Konflikty w parserach LR(k)

- O konflikcie mówimy wtedy, gdy jakiś wpis w tablicy parsowania zawiera dwie lub więcej akcji: przesunięcie/redukcja lub redukcja/redukcja.

Konflikty w parserach LR(k)

- Konflikty mogą być rozwiązywane automatycznie jeśli korzystamy z narzędzi do tworzenia parserów, na przykład YACC'a,

  lub wymagają ingerencji twórcy parsera celem wskazania akcji właściwej.

Akcje parsera LR(1)

- Parser LR(1) korzysta z następujących akcji:

if akcja[s, **a**]= shift s' **then**

wstaw **a**, a następnie s' na szczyt stosu,

przejdź do następnego symbolu

wejściowego;

Akcje parsera LR(1)

if akcja [**s, a**] = reduce (*A*-> β) **then**

| β | -długość napisu β

zdejmij ze stosu 2*| β | symboli;

niech s' będzie stanem, który znalazł się na wierzchołku stosu (po usunięciu ze stosu 2*| β | symboli);

wstaw *A* i wartość, którą zwraca tablica przejść(s', *A*), na wierzchołek stosu;

przekaż na wyjście produkcję *A*-> β ;

Akcje parsera LR(1)

if akcja[s, **a**] = accept **then** koniec;

else błąd().

Przykład działania parsera LR(1)

Kolejne slajdy pokazują na przykładzie sposób działania parsera LR(1) dla gramatyki:

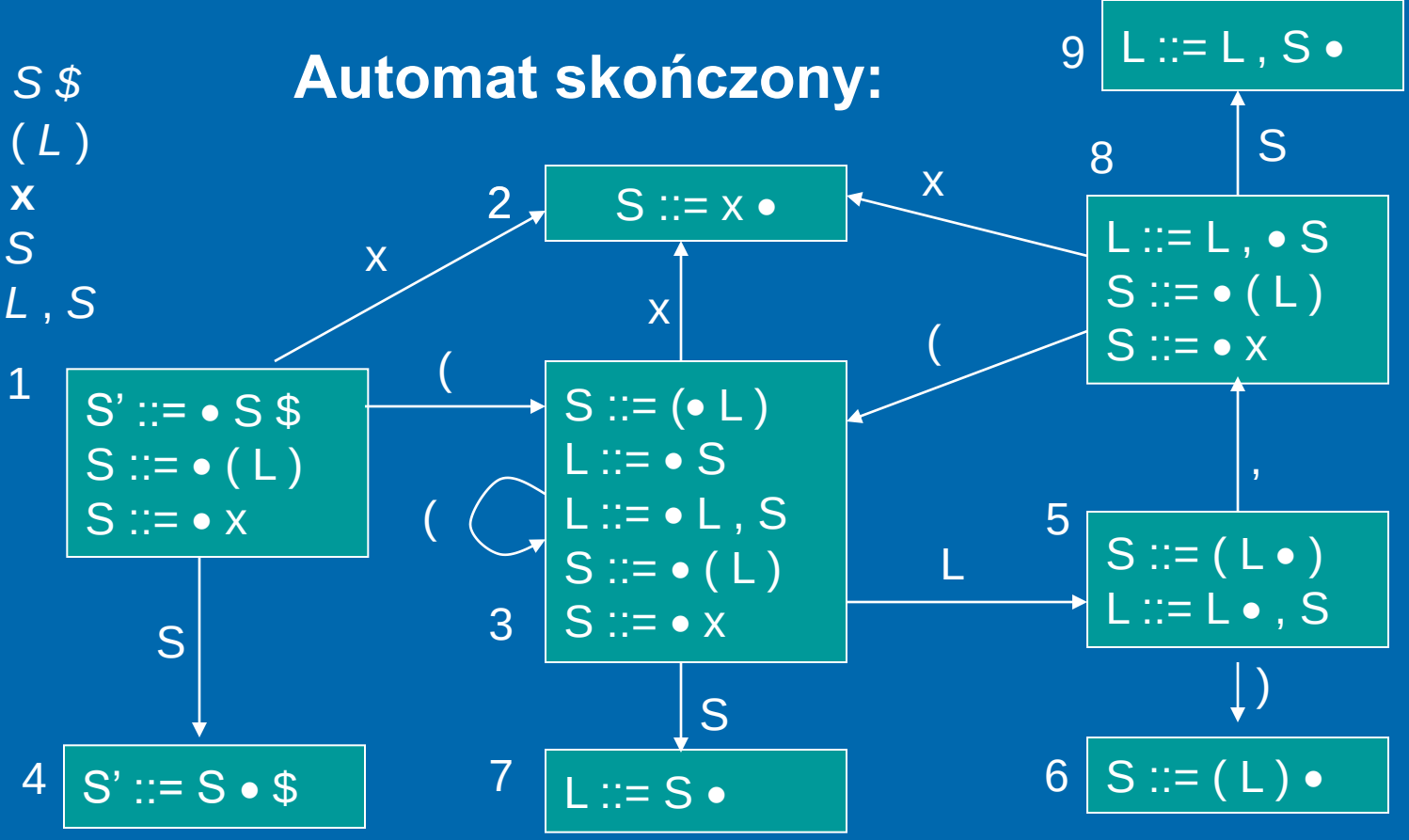
0. *S'* -> *S* $
1. *S* -> (*L*)
2. *S* -> **x**
3. *L* -> *S*
4. *L* -> *L* , *S*

, jest to terminal

dla której symbole terminalne to są: ( ) , **x** natomiast nieterminalami są *L* i *S*.

Automat skończony:

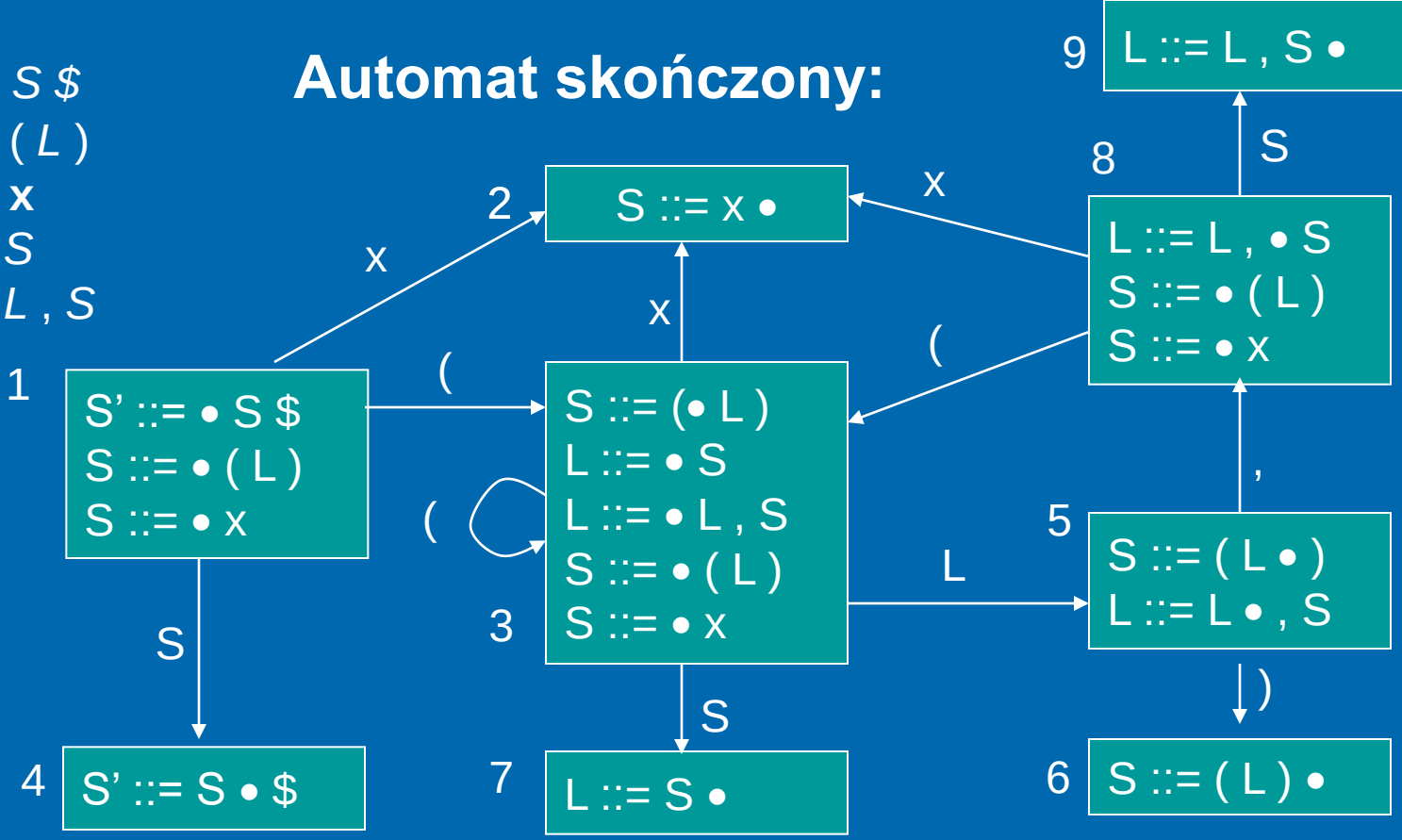
0. *S'* -> *S* $
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*



Dla wejścia (x , x) $ parser wykona następujące akcje:
1 s3 s2 r2g7 r3g5 s8 s2 r2g9 r4g5 s6 r1g4 accept
gdzie si oznacza shift do stanu i, rigj oznacza reduce za pomocą produkcji i i przejście do stanu j; 1 jest stanem startowym

Automat skończony:

0. *S'* -> *S* $
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*



stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
...							

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos 1

Akcja: s3

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

↓

wejście: (x , x) $

Stos 1 (3

Akcja: s2

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

↓

wejście: (x , x) $

Stos 1 (3 x 2
Akcja: r2

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos 1 (3 S
Akcja: g7

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

↓

wejście (x , x) $

stos: 1 (3 S 7

Akcja: r3

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

↓

wejście: (x , x) $

stos: 1 (3 L

Akcja: g5

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście (x , x) $

Stos 1 (3 L 5

Akcja: s8

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście (x , x) $

Stos: 1 (3 L 5 , 8

Akcja: s2

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* $
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos: 1 (3 L 5 , 8 x 2

Akcja: r2

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos: 1 (3 L 5 , 8 S

Akcja: g9

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* $
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos: 1 (3 L 5 , 8 S 9

Akcja: r4

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos: 1 (3 L

Akcja: g5

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos: 1 (3 L 5
Akcja: s6

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos: 1 (3 L 5) 6
Akcja: r1

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos: 1 *S*

Akcja: g4

stan	(	)	x	,	$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

0. *S'* -> *S* *$*
- 1. *S* -> (*L*)
- 2. *S* -> **x**
- 3. *L* -> *S*
- 4. *L* -> *L* , *S*

wejście: (x , x) $

Stos: 1 S 4
Akcja: accept

Dziękuję za uwagę