Podział metod error recovery (niezależnie od rodzaju parsingu)

- Metody lokalne są to metody, które próbują kontynuować parsing po wystąpieniu błędu przez modyfikowanie tokenów na wejściu parsera lub stanu stosu **w momencie wystąpienia błędu**
- Metody globalne mogą modyfikować tokeny, które zostały odczytane **przed** wystąpieniem błędu – tak aby błąd ten usunąć.

Lokalne metody error recovery dla parsingu top-down

W przypadku rozważanego tu typu parsera błąd wystąpi w sytuacji gdy na wierzchołku stosu znajduje się jakiś nieterminal A i jednocześnie aktualny symbol na wejściu parsera nie należy do zbioru FIRST(A) lub w przypadku istnienia produkcji $A \rightarrow \varepsilon$ do zbioru FOLLOW(A).

W przypadku wystąpienia takiego błędu można zastosować jedną z trzech dostępnych metod error recovery.

To którą metodę stosujemy zależy od tego w jakiej sytuacji wystąpił błąd.

W przypadku gdy tokenem na wejściu jest $ lub należy on do zbioru FOLLOW(A) stosujemy metodę polegającą na zdjęciu ze stosu nieterminala A.

Metodę tą oznaczę jako: pop

W przypadku gdy tokenem nie jest $ oraz nie należy on do zbioru FOLLOW(A) usuwamy tokeny z wejścia, aż do momentu natrafienia na token dla którego możemy kontynuować parsing.

Metotę tą oznaczę jako: scan

W przypadku gdy po zastosowaniu metody pop stos stał się pusty umieszczamy na stosie symbol początkowy gramatyki, a następnie stosujemy metodę scan.

Metotę tą oznaczę jako: push

Przykład

Wspomniane wcześniej metody zilustruje na przykładzie gramatyki opisującej proste wyrażenia arytmetyczne (wyrażenia złożone z identyfikatorów, symboli dodawnia i mnożenia oraz nawiasów).

Gramatyka taka może mieć postać:

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

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

Dla powyższej gramatyki zbiory FIRST i FOLLOW mają następującą postać:

	FIRST	FOLLOW
E	(,id	$,)
E'	+,ε	$,)
T	(,id	+,$,)
T'	*,ε	+,$,)
F	(,id	*,+,$,)

Po ponumerowaniu produkcji w następujący sposób:

(1) E → TE'
(2) E' → +TE' | ε
(3) T → FT'
(4) T' → *FT' | ε
(5) F → (E) | id

Otrzymujemy następującą tablice parsera:

	id	+	*	(	)	$
E	TE',1	err	err	TE',1	err	err
E'	err	+TE',2	err	err	ε,2	ε,2
T	FT',3	err	err	FT',3	err	err
T'	err	ε,4	*FT',4	err	ε,4	ε,4
F	id,5	err	err	(E),5	err	err

Na postawie przedstawionych wcześniej informacji oraz poniższej tabeli zawierającej m.in. elementy zbioru FOLLOW ...

	FIRST	FOLLOW
E	(,id	$,)
E'	+,ε	$,)
T	(,id	+,$,)
T'	*,ε	+,$,)
F	(,id	*,+,$,)

... można sporządzić tabele parsera uwzględniającą informacje którą metodeę (pop czy scan) należy stosować w przypadku napotkania błędu:

	id	+	*	(	)	$
E	TE',1	scan	scan	TE',1	pop	pop
E'	scan	+TE',2	scan	scan	ε,2	ε,2
T	FT',3	pop	scan	FT',3	pop	pop
T'	scan	ε,4	*FT',4	scan	ε,4	ε,4
F	id,5	pop	pop	(E),5	pop	pop

Działanie mechanizmu error recovery zaprezentuje na przykładzie następującego słowa: **(id*+id+)id**

	id	+	*	(	)	$
E	TE',1	scan	scan	TE',1	pop	pop
E'	scan	+TE',2	scan	scan	ε,2	ε,2
T	FT',3	pop	scan	FT',3	pop	pop
T'	scan	ε,4	*FT',4	scan	ε,4	ε,4
F	id,5	pop	pop	(E),5	pop	pop

(#E , (id*+id+)id$,)
(#E'T , (id*+id+)id$, 1)
(#E'T'F , (id*+id+)id$, 13)
(#E'T')E**(** , **(**id*+id+)id$, 135)
(#E'T')E , id*+id+)id$, 135) - dopasowano token (
(#E'T')E'T , id*+id+)id$, 1351)
(#E'T')E'T'F , id*+id+)id$, 13513)
(#E'T')E'T'**id** , **id***+id+)id$, 135135)

	id	+	*	(	)	$
E	TE',1	scan	scan	TE',1	pop	pop
E'	scan	+TE',2	scan	scan	ε,2	ε,2
T	FT',3	pop	scan	FT',3	pop	pop
T'	scan	ε,4	*FT',4	scan	ε,4	ε,4
F	id,5	pop	pop	(E),5	pop	pop

(#E'T')E'T' , *+id+)id$, 135135) – dopasowano token id
(#E'T')E'T'F* , *+id+)id$, 1351354)
(#E'T')E'T'F , +id+)id$, 1351354) – dopasowano token *
(#E'T')E'T' , +id+)id$, 1351354) – błąd, zastosowano metodę pop
(#E'T')E' , +id+)id$, 13513544)
(#E'T')E'T+ , +id+)id$, 135135442)
(#E'T')E'T , id+)id$, 135135442) – dopasowano token +
(#E'T')E'T'F , id+)id$, 1351354423)
(#E'T')E'T'**id** , **id**+)id$, 13513544235)

	id	+	*	(	)	$
E	TE',1	scan	scan	TE',1	pop	pop
E'	scan	+TE',2	scan	scan	ε,2	ε,2
T	FT',3	pop	scan	FT',3	pop	pop
T'	scan	ε,4	*FT',4	scan	ε,4	ε,4
F	id,5	pop	pop	(E),5	pop	pop

(#E'T')E'T' , +)id$, 13513544235) – dopasowano token id
(#E'T')E' , +)id$, 135135442354)
(#E'T')E'T+ , +)id$, 1351354423542)
(#E'T')E'T ,)id$, 1351354423542) – dopasowano token +
(#E'T')E' ,)id$, 1351354423542) – błąd, zastosowano metodę pop
(#E'T') ,)id$, 13513544235422)
(#E'T' , id$, 13513544235422) – dopasowano token)
(#E'T' , $, 13513544235422) – błąd, zastosowano metodę scan
(#E' , $, 135135442354224)
(# , $, 1351354423542242) – parsing został zakończony

Jak widać przy zastosowaniu metod error recovery pop i scan udało się doprowadzić parsing przykładowego słowa do końca pomimo wystąpienia w trakcie tego procesu 3 błędów.

Globalne metody error recovery dla parsingu top-down

Globalne error-recovery

Przykładowa gramatyka “if-else”:

S -> `if` S `then` S `else` S

S -> `a`

Wejście (z błędem):

`if a then ~~if~~ a then a else a else a`

Tablica parsera:

	if	***then***	***else***	***a***	***$***
S	if S then S else S			a	

Globalne error-recovery

Przebieg parsingu LL(1):

Wejście:

`if a then a then a else a else a`

Stos:

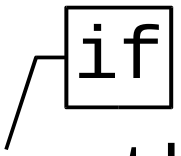
S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Globalne error-recovery

Przebieg parsingu LL(1):

Wejście:



`if a then a then a else a else a`

Stos:

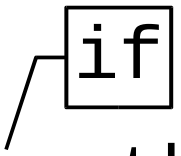
`# S else S then S if`

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Globalne error-recovery

Przebieg parsingu LL(1):

Wejście:



`if a then a then a else a else a`

Stos:

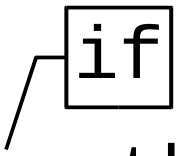
`# S else S then S`

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Globalne error-recovery

Przebieg parsingu LL(1):

Wejście:



`if a then a then a else a else a`

Stos:

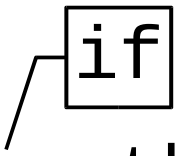
`# S else S then a`

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Globalne error-recovery

Przebieg parsingu LL(1):

Wejście:



`if a then a then a else a else a`

Stos:

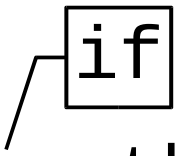
`# S else S then`

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Globalne error-recovery

Przebieg parsingu LL(1):

Wejście:



`if a then a then a else a else a`

Stos:

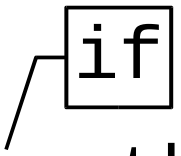
`# S else S`

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Globalne error-recovery

Przebieg parsingu LL(1):

Wejście:



`if a then a then a else a else a`

Stos:

`# S else a`

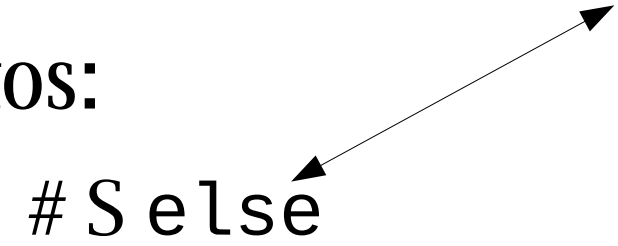
	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Globalne error-recovery

Przebieg parsingu LL(1):

Wejście:

Stos:



Niezależnie od zastosowanej metody lokalnego naprawiania błędów, parser nie może “nadrobić” brakującego tokenu `if` – jest na to za późno. Przy parsowaniu dalszych tokenów wystąpią nowe błędy

Globalne error-recovery

- W wielu przypadkach błędny token prowadzi do wystąpienia błędu parsera dopiero kilka tokenów później.
- Lokalne error-recovery operuje tylko na bieżącym tokenie, więc nie może usunąć rzeczywistego źródła błędu – najczęściej wywołuje kolejne błędy.
- Globalne error-recovery może modyfikować tokeny już wczytane tak aby usunąć błąd w miejscu, w którym powstał.

Algorytm Burke-Fishera

- Parser przechowuje w buforze ***k*** ostatnio przeczytanych tokenów
- W momencie wystąpienia błędu, sprawdzna jest każda możliwość dodania, usunięcia lub zmiany jednego tokenu (maksymalnie do ***k*** tokenów wstecz)
- Wybierana jest taka modyfikacja, która pozwala najdłużej kontynuować poprawne parsowanie

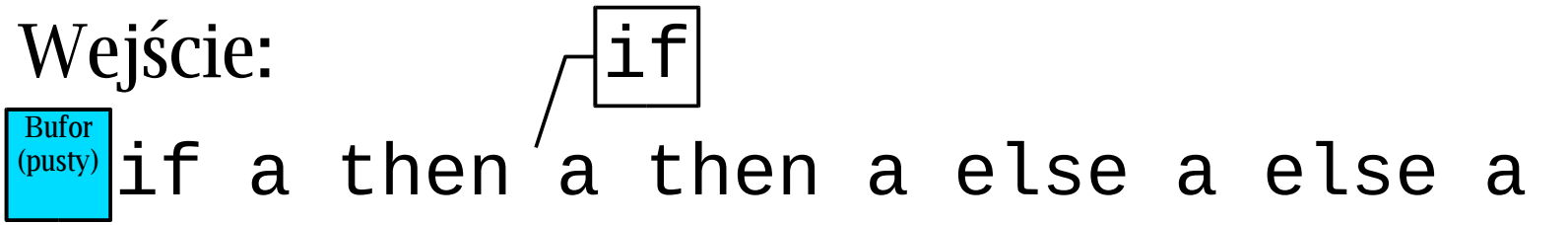
Algorytm Burke-Fishera

- Przy sprawdzaniu możliwości naprawienia błędu musi być możliwe efektywne “wycofanie” parsera o ***k*** tokenów
- Przechowywany jest drugi stos parsera, “opóźniony” w stosunku do pierwszego o ***k*** tokenów. W przypadku wycofywania można z niego odtworzyć stan parsera.

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:



Stos:

S

Stary stos:

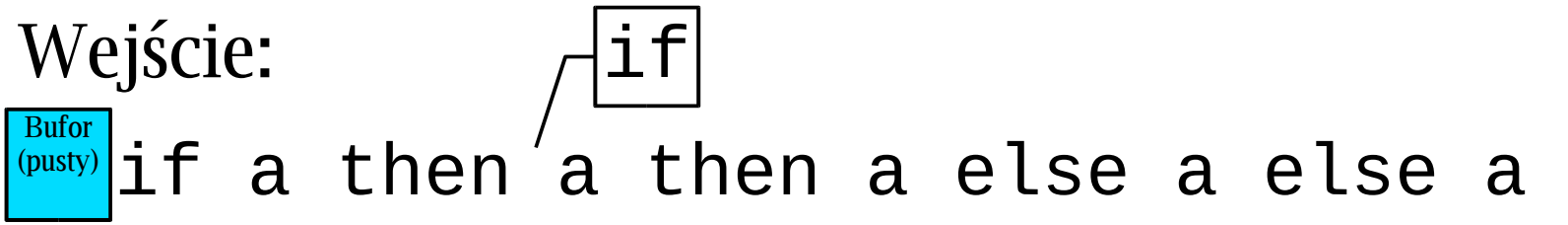
S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:



Stos:

S else S then S if

Stary stos:

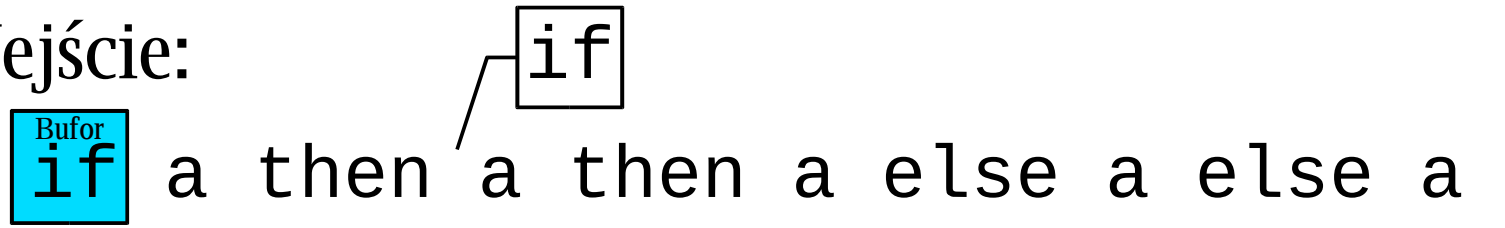
S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:



Stos:

S else S then S

Stary stos:

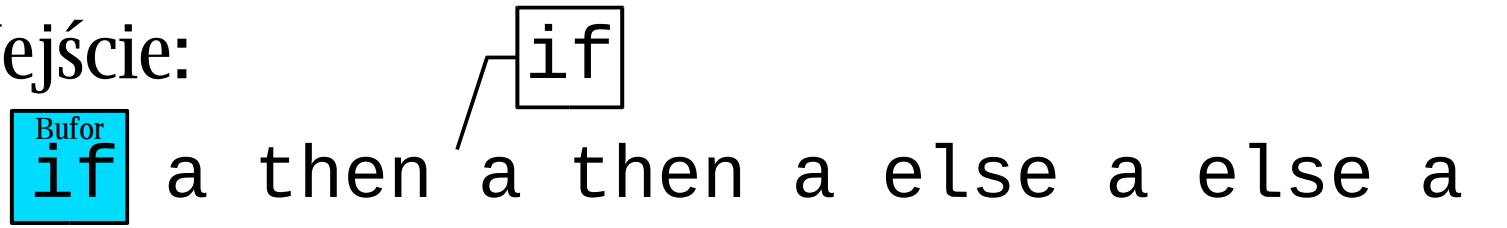
S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:



Stos:

S else S then a

Stary stos:

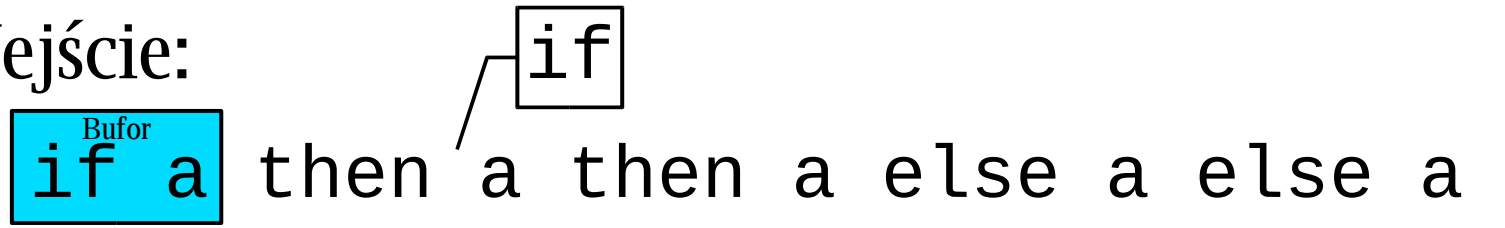
S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:



Stos:

S else S then

Stary stos:

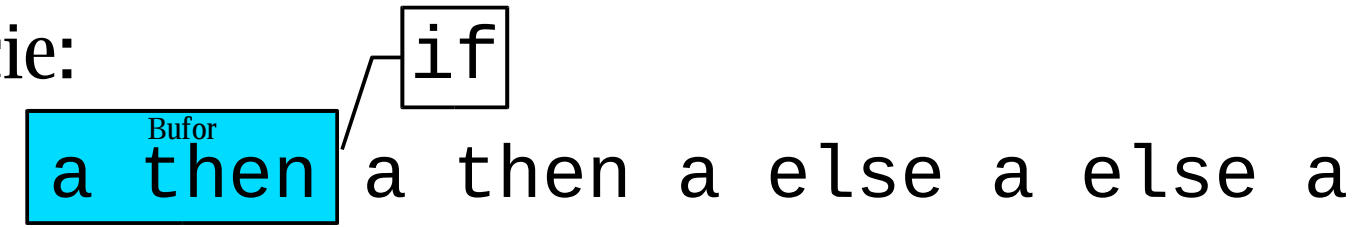
S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:



Stos:

S else S

Stary stos:

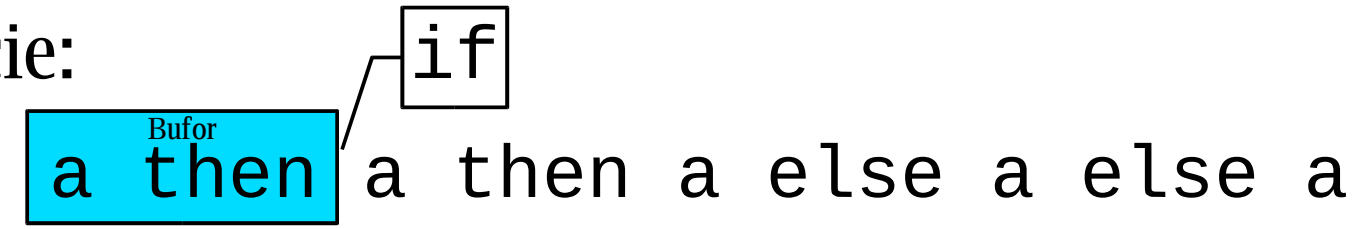
S else S then S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:



Stos:

S else a

Stary stos:

S else S then S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:

Stos:

`# S else`

Stary stos:

`# S else S then`

Parser może dodać brakujący token `if`, a następnie poprawnie sparsować dalszy ciąg programu.

Algorytm Burke-Fishera

Przykład dla k=2

Wejście:

if a then Bufor[if a] then a else a else a

Stos:

S else S else S then

Stary stos:

S else S

	if	*then*	*else*	*a*	*$*
S	if S then S else S			a	

Algorytm Burke-Fishera

- Jeśli żadna ze sprawdzonych “poprawek” nie pozwala kontynuować parsowania, stosowane sa metody lokalne (np. usuwanie tokenów z wejścia aż do napotkania pasującego)
- Ponieważ tokeny znajdujące się w buforze mogą być w przypadku usuwania błędów wielokrotnie parsowane, akcje semantyczne związane z produkcjami wywołuje się dopiero przy usuwaniu tokenów z bufora (i aktualizowaniu “starego” stosu)