

# Parsing i unifikacja

Paweł Rychlikowski

Instytut Informatyki UWr

7 stycznia 2019

# RDP. Przypomnienie

**Recursive Descent Parser Application**

File Edit Apply View Animate Help

**Available Expansions**

- S -> NP VP
- NP -> Det N PP
- NP -> Det N
- VP -> V NP PP
- VP -> V NP
- VP -> V
- PP -> P NP
- NP -> 'I'
- Det -> 'the'
- Det -> 'a'
- N -> 'man'
- N -> 'park'
- N -> 'dog'
- N -> 'telescope'
- V -> 'ate'
- V -> 'saw'
- P -> 'in'
- P -> 'under'
- P -> 'with'

**Parse Tree:**

```
graph TD
    S --> NP1[NP]
    S --> VP1[VP]
    NP1 --> Det1[Det]
    NP1 --> N1[N]
    Det1 --> the1[the]
    N1 --> dog[dog]
    VP1 --> V1[V]
    VP1 --> NP2[NP]
    VP1 --> PP1[PP]
    V1 --> saw[saw]
    NP2 --> Det2[Det]
    NP2 --> N2[N]
    NP2 --> PP2[PP]
    Det2 --> the2[the]
    N2 --> man[man]
    PP2 --> P[P]
    PP2 --> NP3[NP]
    P --> in[in]
    NP3 --> the3[the]
    NP3 --> park[park]
```

the dog saw a man in the park

Last Operation: Expand: NP -> Det N PP

# Jakie gramatyki lubi RDP?

- Na pewno nie lubi lewostronnie rekurencyjnej! (bo się zapętla)
- Raczej fajne jest, jak może szybko „nawrócić”, jak widać że nic z danego rozbioru nie będzie.
- Czyli dobrze jest, żeby możliwie szybko pojawił się symbol terminalny.

## Idealna produkcja

Największą radość sprawimy parserowi RDP, dając mu regułę postaci:  $A \rightarrow wBCDE$ , gdzie  $w$  jest symbolem terminalnym.

# Postać normalna Greibach

- Wszystkie produkcje mają postać  $A \rightarrow wA_1 \dots A_n$  ( $n$  może być równe 0)
- Możemy założyć, że język nie zawiera słowa pustego (tak jak w postaci normalnej Chomsky'ego)

## Uwaga

Każdą gramatykę da się przekształcić do postaci normalnej Greibach przy zachowaniu akceptowanego języka.

Wyznaczmy postaci normalne dla następującej gramatyki (zielone są problematyczne):

$$NP \rightarrow Adj\ NP$$
$$NP \rightarrow NP\ Adj_2$$
$$Adj \rightarrow mały \mid głupi$$
$$Adj_2 \rightarrow brunatny \mid polarny$$
$$NP \rightarrow miś$$
$$NP \rightarrow NP\ i\ NP$$

Wyznaczmy postaci normalne dla następującej gramatyki (zielone są problematyczne):

$NP \rightarrow Adj\ NP$

$NP \rightarrow NP\ Adj_2$

$Adj \rightarrow mały \mid głupi$

$Adj_2 \rightarrow brunatny \mid polarny$

$NP \rightarrow miś$

$NP \rightarrow NP\ i\ NP$

# Top Down vs Bottom Up

Różne algorytmy mogą działać w stylu TopDown albo Bottom Up.

## Top down

Zaczynamy od symbolu startowego, próbujemy znaleźć drzewo rozbioru dla całego zdania. Nasze poszukiwania są ukierunkowane na sparsowanie zdania.

## Bottom up

Zaczynamy od pojedynczych wyrazów i próbujemy je łączyć w coraz większe struktury. Nawet, jak nie znajdziemy rozbioru dla zdania, to możemy znaleźć różne mniejsze, użyteczne rozbioru dla fraz (na przykład nominalnych).

Innym algorytmem parsowania, który przedstawia dość istotne idee jest **Shift-Reduce**

- Mamy w algorytmie dwie struktury:
  - Listę drzew (las uporządkowany), dla sparsowanych fragmentów
  - Listę L zawierającą niesparsowany sufiks zdania
- Operacja **Shift** oznacza przesunięcie wyrazu z listy i utworzenie jednowęzłowego drzewka z tym wyrazem
- Operacja **Reduce** oznacza połączenie (zgodne z gramatyką) pewnej liczby drzew w nowe drzewo



# Gramatyka do prezentacji algorytmu SR

Interesuje nas gramatyka, która umożliwi sparsowanie zdania:

*The dog saw a man in the park*

Gramatyka:

NP -> Det N

Det -> a | the

N -> dog | man

PP -> Prep NP

VP -> V NP PP

S -> NP VP

V -> saw | read | killed

# Algorytm SR w działaniu

## 1. Initial state

Stack	Remaining Text
	the dog saw a man in the park

## 3. After reduce shift reduce

Stack	Remaining Text
Det N the dog	saw a man in the park

## 5. After building a complex NP

Stack	Remaining Text
<div>NP V NP PP Det N saw Det N P NP the dog a man in the park</div>	

## 2. After one shift

Stack	Remaining Text
the	dog saw a man in the park

## 4. After recognizing the second NP

Stack	Remaining Text
<div>NP V NP in Det N saw Det N the dog a man</div>	the park

## 6. Built a complete parse tree

Stack	Remaining Text
<div>S NP VP Det N V NP PP the dog saw NP PP Det N P NP a man in Det N the park</div>	

# Algorytm SR w działaniu

## 1. Initial state

Stack	Remaining Text
	the dog saw a man in the park

## 2. After one shift

Stack	Remaining Text
the	dog saw a man in the park

## 3. After reduce shift reduce

Stack	Remaining Text
Det N the dog	saw a man in the park

## 4. After recognizing the second NP

Stack	Remaining Text
NP V NP in Det N saw Det N the dog a man	the park

## 5. After building a complex NP

Stack	Remaining Text
NP V NP NP PP Det N saw Det N P NP the dog a man in Det N the park	

## 6. Built a complete parse tree

Stack	Remaining Text
S NP VP Det N V NP PP the dog saw NP PP Det N P NP a man in Det N the park	

# Algorytm SR w działaniu

## 1. Initial state

Stack	Remaining Text
	the dog saw a man in the park

## 3. After reduce shift reduce

Stack	Remaining Text
Det N the dog	saw a man in the park

## 2. After one shift

Stack	Remaining Text
the	dog saw a man in the park

## 4. After recognizing the second NP

Stack	Remaining Text
<div>NP V NP in</div> <div>Det N saw Det N</div> <div>the dog a man</div>	the park

## 5. After building a complex NP

Stack	Remaining Text
<div>NP V NP</div> <div>Det N saw NP PP</div> <div>the dog a man in Det N</div> <div>the park</div>	

## 6. Built a complete parse tree

Stack	Remaining Text
<div>S</div> <div>NP VP</div> <div>Det N V NP</div> <div>the dog saw NP PP</div> <div>Det N P NP</div> <div>a man in Det N</div> <div>the park</div>	

# Algorytm SR w działaniu

## 1. Initial state

Stack	Remaining Text
	the dog saw a man in the park

## 2. After one shift

Stack	Remaining Text
the	dog saw a man in the park

## 3. After reduce shift reduce

Stack	Remaining Text
Det N the dog	saw a man in the park

## 4. After recognizing the second NP

Stack	Remaining Text
NP V NP in Det N saw Det N the dog a man	the park

## 5. After building a complex NP

Stack	Remaining Text
NP V NP NP PP Det N saw Det N P NP the dog a man in Det N the park	

## 6. Built a complete parse tree

Stack	Remaining Text
S NP VP Det N V NP PP the dog saw NP PP Det N P NP a man in Det N the park	

# Algorytm SR w działaniu

## 1. Initial state

Stack	Remaining Text
	the dog saw a man in the park

## 2. After one shift

Stack	Remaining Text
the	dog saw a man in the park

## 3. After reduce shift reduce

Stack	Remaining Text
Det N the dog	saw a man in the park

## 4. After recognizing the second NP

Stack	Remaining Text
NP V NP in Det N saw Det N the dog a man	the park

## 5. After building a complex NP

Stack	Remaining Text
NP V NP NP PP Det N saw Det N P NP the dog a man in Det N the park	

## 6. Built a complete parse tree

Stack	Remaining Text
S NP VP Det N V NP PP the dog saw NP PP a man in Det N the park	

# Algorytm SR w działaniu

## 1. Initial state

Stack	Remaining Text
	the dog saw a man in the park

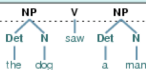
## 3. After reduce shift reduce

Stack	Remaining Text
Det N the dog	saw a man in the park

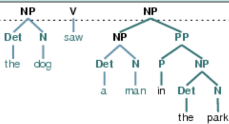
## 2. After one shift

Stack	Remaining Text
the	dog saw a man in the park

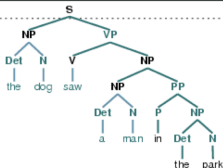
## 4. After recognizing the second NP

Stack	Remaining Text
 NP V NP in the dog saw a man the park	the park

## 5. After building a complex NP

Stack	Remaining Text
 NP V NP NP PP the dog saw a man in the park	

## 6. Built a complete parse tree

Stack	Remaining Text
 S NP VP the dog saw NP PP a man in Det N the park	

- Nie mówiliśmy nic o nawracaniu, prosta implementacja SR parsera może nie znaleźć rozbioru!
- Można zaimplementować jakąś strategię decydowania o S i R. Na przykład:
  - Preferować reduce, jeżeli możliwe
  - Mając do wyboru wiele redukcji wybierać tę, która „usunie” więcej drzew

## Uwaga

Jest dużo możliwości „uczenia” algorytmu parsingu: wynikiem tego uczenia miałyby być metoda wyboru S/R w zależności od danych. Mając rozbiór bowiem wiemy, dokładnie, jak powinien działać optymalny S/R parser, możemy zatem uczyć takiej strategii. Jak?



- Dynamiczny algorytm parsingu dla gramatyk w dowolnej postaci.
- Łatwo go uogólniać i opracowywać różne warianty
- Zaczniemy od podstawowej konstrukcji

# Struktura danych w Algorytmie Earleya

Algorytm Earleya (AE) tworzy w każdym z  $N + 1$  punktów zdania o długości  $N$  tablicę informacji, wyglądających następująco:

$$A \rightarrow \alpha \bullet \beta [a:b]$$

gdzie  $A \rightarrow \alpha\beta$  jest produkcją gramatyki, natomiast  $0 \leq a \leq b \leq N + 1$ , są pozycjami w tekście (jak w Pythonie)

## Interpretacja

Każdy z powyższych napisów jest zdaniem, mówiącym:

- W miejscu  $a$  próbuję znaleźć fragment zdania pasujący do nieterminala  $A$ ,
- używając do tego produkcji  $A \rightarrow \alpha\beta$ ,
- udało mi się dojść do miejsca  $b$
- zjadając część produkcji przed znakiem •

# Przykłady. Zgadnijmy, co oznaczają?

- $S \rightarrow \bullet NP VP$  [0:0]  
Zaczynamy analizę zdania.
- $NP \rightarrow AP \bullet NP$  [5:7]  
Pierwsze 2 wyrazy na pozycji piątej z sukcesem sparsowaliśmy jako frazę przymiotnikową, próbujemy dalej znaleźć frazę rzeczownikową.
- $N \rightarrow w \bullet$  [k:k+1]  
Na pozycji  $k$  znajduje się słowo  $w$  (o tagu  $N$ )
- $S \rightarrow NP VP \bullet$  [0:10]  
Sparsowaliśmy z sukcesem zdanie o długości 10.

- Dla każdej produkcji  $S \rightarrow \alpha$  dodaje element do tablicy  $\text{Chart}[0]$   
 $\text{Chart}[i]$  oznacza *zadania parsingu rozpoczynające się od pozycji i*
- Elementem tym jest oczywiście:

$$S \rightarrow \bullet \alpha \ [0:0]$$

## Definicja

Stan jest **kompletny** jeżeli na końcu jest • (czyli przetworzyliśmy wszystko).

Dla każdego  $i$  i dla każdego elementu w  $\text{Chart}[i]$  zastosuj jedną z 3 procedur

- Scanner (jeżeli stan jest niekompletny i mamy do przetworzenia POS)
- Predictor (jeżeli stan jest niekompletny i mamy do przetworzenia nie POS)
- Completer (jeżeli stan jest kompletny)

- Stanem jest  $A \rightarrow \alpha \bullet B\beta$   $[i:j]$
- $B$  to POS słowa na pozycji  $j$
- Czyli możemy je skonsumować i przesunąć się o 1 krok
- Dodajemy do  $\text{Chart}[j+1]$  stan

$$B \rightarrow \text{word}[j] \bullet [j:j+1]$$

- Podobnie jak Scanner, ale następnym symbolem jest nieterminal, zatem musimy rozpocząć jego analizę.
- Stanem jest  $A \rightarrow \alpha \bullet B\beta$  [i:j]
- Dla każdej produkcji  $B \rightarrow \gamma$  dodamy do Chart[j]
- stan  $B \rightarrow \bullet\gamma$  [j:j]

- Przetwarza reguły, które są już zakończone przesuając kropkę w innych stanach.
- Stanem jest  $B \rightarrow \gamma \bullet [j:k]$
- Dla każdego stanu

$$A \rightarrow \alpha \bullet B\beta [i:j]$$

w Chart[j] dodaj do Chart[k] stan

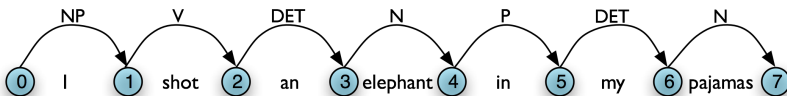
$$A \rightarrow \alpha B \bullet \beta [i:k]$$



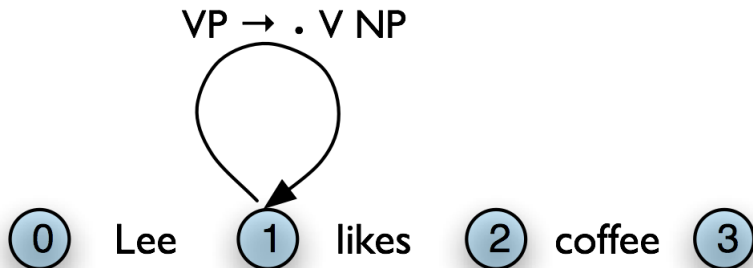
- Inspirowany parserem Earleya, używa podobnej notacji (z kropką)
- Można o nim myśleć jako o uogólnieniu parsera Earleya.
- Używa pojęć „grafowych”, w grafie węzłami są pozycje między słowami, krawędzie etykietowane są produkcjami z kropką.

# Graf dla zdania. Wersja 0

Myślmy o tym, że graf opisuje zdanie wraz z częściowym parsyniem:



# Pętla w grafie wprowadzająca produkcje



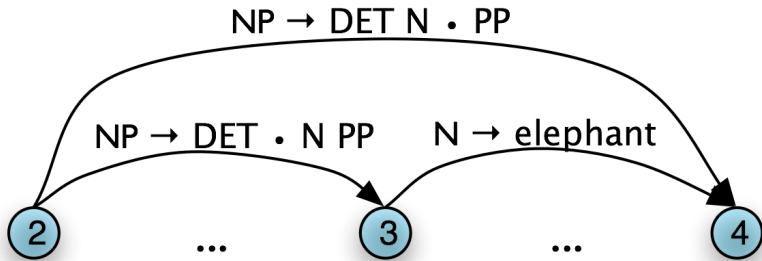
# Fundamental Rule

Jeżeli w grafie znajdują się obie poniższe krawędzie

- $A \rightarrow \alpha \bullet B\beta$  [i:j]
- $B \rightarrow \gamma \bullet$  [j:k]

Wówczas dodaj krawędź:

$$A \rightarrow \alpha B \bullet \beta$$



# Sposoby wykorzystania idei Chart parsera. Przykładowe reguły

## Bottom Up Predict

Dla każdej reguły  $A \rightarrow \alpha \bullet [i:j]$  i produkcji  $B \rightarrow A\beta$  dodaj krawędź:

$$B \rightarrow \bullet A\beta [i:i]$$

## Top Down Predict

- Mamy  $A \rightarrow \alpha \bullet B\beta [i:j]$
- Dodajemy  $B \rightarrow \bullet \gamma [j:j]$ , dla produkcji gramatyki  $B \rightarrow \gamma$

Szczegóły na ćwiczeniach.

## Uwaga

Opisując język polski korzystaliśmy z gramatyki atrybutowej (czyli takiej, w której symbole nieterminalne mają jakąś strukturę), nie definiując jej specjalnie.

Zastanówmy się zatem, jak wyglądałaby definicja?

- Symbole nieterminalne moglibyśmy definiować jako pływkie termy, postaci  $f(X_1, \dots, X_n)$ , gdzie  $X_i$  są zmiennymi albo stałymi ze skończonego zbioru.
- Składnikiem wyprowadzenia byłoby podstawienie.

## Wyprowadzenie w jednym kroku

Powiemy, że  $(xAy, \theta) \Rightarrow (x\beta y, \theta')$ , jeżeli:

- $A' \rightarrow \beta$  jest świeżym wariantem produkcji gramatyki
- $A\theta\theta_2 = A'\theta_2$
- $\theta' = \theta\theta_2$

## Gramatyki atrybutowe (2)

- Tak zdefiniowane GA nie wyprowadzają poza języki bezkontekstowe (bo?)
- Teoretycznie parsing staje się trudniejszy, dokładniej NP-trudny.

## Uwaga

W „świecie NLP” nie używa się termów, tylko tak zwane **struktury atrybutowe**.

## Definicja

Strukturą atrybutową jest:

- 1 Stała (ze skończonego zbioru stałych)
- 2 Odwzorowanie, zawierające skończenie wiele atrybutów (stałych z innego, też skończonego zbioru). Wartościami dla tych atrybutów są Struktury atrybutowe.

## Uwaga

Strukturę atrybutową możemy kodować jako słownik, w którym kluczem jest nazwa atrybutu, a wartością napis lub struktura atrybutowa.



# Przykłady na gramatykę atrybutową w NLTK

Prosta gramatyka pozwalająca na konstruowanie zdań: **this dog runs**, **these dogs run**.

```
Det[NUM=sg] -> 'this'  
Det[NUM=pl] -> 'these'
```

```
N[NUM=sg] -> 'dog'  
N[NUM=pl] -> 'dogs'  
V[NUM=sg] -> 'runs'  
V[NUM=pl] -> 'run'
```

```
S -> NP[NUM=?n] VP[NUM=?n]  
NP[NUM=?n] -> Det[NUM=?n] N[NUM=?n]  
VP[NUM=?n] -> V[NUM=?n]
```

## Przykłady na gramatykę atrybutową w NLTK (2)

Możemy dodać inne rodzajniki wraz ze zmienną nieukonkretnioną (jak w Prologu `_`)

```
Det[NUM=?n] -> 'the' | 'some' | 'any'
```

To powyższe jest równoważne:

```
Det -> 'the' | 'some' | 'any'
```

# Parsing dla gramatyk atrybutowych

Chart Parsing łatwo uogólnia się na gramatyki z atrybutami.

[przykład na tablicy dla zdania jaś widzi ładny stół]

- Czasami nie potrzebujemy informacji o pełnej strukturze zdania, a tylko chcemy wyłować interesujące nas fragmenty.
- Przykładowo osoby, daty, miejsca, jakieś części poleceń, ...
- Wszystkie mechanizmy, w których zgadzamy się, że nasza informacja będzie (potencjalnie) częściowa nazywamy płytkim parsingiem.

# Przybliżanie głębokiego parsingu

- Można definiować proste (regularne) gramatyki i otrzymywać przybliżone drzewo rozbioru o zadanej wysokości za pomocą kaskady takich gramatyk.
- Przykładowo:

$\text{Np}(\text{Case}) \rightarrow \text{Adj}(\text{Case}) * \text{N}(\text{Case})$

$\text{Np}(\text{Case}) \rightarrow \text{Adj}(\text{Case}) * \text{N}(\text{Case}) \text{ Adj}(\text{Case})$

$\text{Np2}(\text{Case}) \rightarrow \text{Np}(\text{Case}) \text{ Np}(\text{gen}) *$

$\text{Np3}(\text{Case}) \rightarrow \text{Np2}(\text{Case}) \text{ Prep}(\text{C2}) \text{ Np}(\text{C2})$

# Przybliżanie głębokiego parsingu

$Np(Case) \rightarrow Adj(Case)^* N(Case)$

$Np(Case) \rightarrow Adj(Case)^* N(Case) Adj(Case)$

$Np2(Case) \rightarrow Np(Case) Np(gen)^*$

$Np3(Case) \rightarrow Np2(Case) Prep(C2) Np(C2)$

Jakie frazy ta gramatyka może sparsować:

*samotny biały żagiel*

*niepokojący widok samotnego białego żagla*

*obraz olejny Matejki z niepokojącym widokiem*

*samotnego białego żagla*

*nadmiar parówek z wieprzowimy*

*parówka z wieprzowiny grubego Stefana*

*radość z książki o smokach*

# Przybliżanie głębokiego parsingu

```
np(Case) -> adj(Case)* N(Case)
np(Case) -> adj(Case)* N(Case) adj(Case)
np2(Case) -> np(Case) np(gen)*
np3(Case) -> np2(Case) prep(C2) np(C2)
```

Jakie frazy ta gramatyka może sparsować:

*samotny biały żagiel*  
*niepokojący widok samotnego białego żagla*  
*obraz olejny Matejki z niepokojącym widokiem*  
*samotnego białego żagla*  
*nadmiar parówek z wieprzowimy*  
*parówka z wieprzowiny grubego Stefana*  
*radość z książki o smokach*

- Często wystarcza nam jeszcze bardziej płytka (jednopoziomowa) struktura, zawierająca jedynie informacje o typach rozłącznych podfraz danego zdania.
- Przykładowo w zdaniu:

*Ta firma dostarczy odzież sportową dla uczestników Narodowego Święta Biegania, które odbędzie się 24 kwietnia 2016 roku.*

- Moglibyśmy znaleźć takie frazy: **nominalna**, **nazwa własna**, **data**.

*Ta firma dostarczy odzież sportową dla uczestników Narodowego Święta Biegania, które odbędzie się 24 kwietnia 2016 roku.*



# Chunking (2)

- Trzeba mieć świadomość, że to nie jest parsing, więc nie pojawiają się w nim frazy:
  - uczestników Narodowego Święta Biegania
  - Święta Biegania
- Jak decydować, jaki podział na frazy jest właściwy?

# Dwie zasady chunkingu

## Zasada minimalnej frazy

Fraza jakiegoś typu nie ma w sobie (*sensownej*) podfrazy tego samego typu o długości większej niż 1.

## Zasada maksymalnej frazy

Interesuje nas maksymalny fragment tekstu o określonym typie (na przykład będący frazą nominalną).

# Maksymalna vs minimalna fraza

## Uwaga

Oczywiście te dwie zasady są ze sobą sprzeczne. Gdy robimy **np-chunking** jak powinniśmy potraktować zdanie:

*chodziłem do III Liceum Ogólnokształcącego im. Adama Mickiewicza*

:

## Wariant maksymalny

chodziłem do **III Liceum Ogólnokształcącego im. Adama Mickiewicza**

## Wariant minimalny (z wątpliwościami)

chodziłem do **III Liceum Ogólnokształcącego im. Adama Mickiewicza**

# Maksymalna vs minimalna fraza

## Uwaga

Oczywiście te dwie zasady są ze sobą sprzeczne. Gdy robimy **np-chunking** jak powinniśmy potraktować zdanie:

*chodziłem do III Liceum Ogólnokształcącego im. Adama Mickiewicza*

## Wariant maksymalny

chodziłem do **III Liceum Ogólnokształcącego im. Adama Mickiewicza**

## Wariant minimalny (z wątpliwościami)

chodziłem do **III Liceum Ogólnokształcącego im. Adama Mickiewicza**

# Maksymalna vs minimalna fraza (2)

## Pytanie

Jakie frazy mogłyby być bardziej użyteczne?

## Odpowiedź

To zależy:

- Gdy myślimy o wyszukiwaniu informacji, to mniejsze frazy wydają się być bardziej użyteczne (bo odpowiadają naturalnym pojęciom, takim jak zmienna losowa, białe wino, ołowiany żołnierz, dom starców)
- Gdy chcemy wykorzystać Chunking w rozbiórce, to możemy preferować duże frazy (najpierw je znajdujemy, a potem każdą z osobna analizujemy gramatyką)

Dla języka polskiego istnieją korpusy np. fraz nominalnych korzystające zasady maksymalnej frazy.