Polynomial algorithms for CFGs via semiring embeddings

Piotr Mikołajczyk

Theoretical Computer Science Department of Jagiellonian University

November 4, 2020

Sources

- **Parsing with Derivatives** (2011) – by M. Might, D. Darais, D. Spiewak
- **On the Complexity and Performance of Parsing with Derivatives** (2016) – by M. Adams, C. Hollenbeck, M. Might
- **A C++ implementation of Parsing With Derivatives** (2019) – ☺

Regular languages

- $\varnothing$ and $\{\varepsilon\}$ are regular
- $\forall_{c \in \Sigma}\ \{c\}$ is regular
- If A and B are regular, then $A \cup B$, $A \circ B$ and $A^{\star}$ are regular
- No other language is regular

Regular languages

- $\varnothing$ and $\{\varepsilon\}$ are regular
- $\forall_{c \in \Sigma}$ $\{c\}$ is regular
- If A and B are regular, then $A \cup B$, $A \circ B$ and $A^\star$ are regular
- No other language is regular

$$R = \{\epsilon\} \cup \{a\} \cdot (\{a\} \cup \{b\})^\star$$

Context-free languages

- They are just like recursive regular languages

Context-free languages

- They are just like recursive regular languages
- As $L^\star \equiv \varepsilon \cup (L \circ L^\star)$, we can give up using Kleene's star operator

Context-free languages

A context-free grammar is a tuple $\mathcal{G} = (\mathcal{N}, \Sigma, S, P)$. A language which it generates is denoted as $\mathcal{L}(\mathcal{G})$.

Context-free languages

A context-free grammar is a tuple $\mathcal{G} = (\mathcal{N}, \Sigma, S, P)$. A language which it generates is denoted as $\mathcal{L}(\mathcal{G})$.

Example:

- $\mathcal{N} = \{S\}$
- $S \to aSa|bSb|\varepsilon$
- $\mathcal{L}(\mathcal{G}) = \{\varepsilon, aa, bb, abba, aaaa, \dots\}$

Binarization

Each grammar $\mathcal{G}$ can be transformed (in polynomial time) to an equivalent *binarized* grammar $\mathcal{G}'$ ($\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$) – operators (concatenation and alternative) are considered as purely binary; operands must be from $\Sigma \cup \mathcal{N}$.

Binarization

Each grammar $\mathcal{G}$ can be transformed (in polynomial time) to an equivalent *binarized* grammar $\mathcal{G}'$ ($\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$) – operators (concatenation and alternative) are considered as purely binary; operands must be from $\Sigma \cup \mathcal{N}$.

- $C \to ccc$
- $B \to b|A$
- $A \to \varepsilon|Bb|C$

Binarization

Each grammar $\mathcal{G}$ can be transformed (in polynomial time) to an equivalent *binarized* grammar $\mathcal{G}'$ ($\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$) – operators (concatenation and alternative) are considered as purely binary; operands must be from $\Sigma \cup \mathcal{N}$.

- $C \to ccc$
- $B \to b|A$
- $A \to \varepsilon|Bb|C$

- $C \to C'c$
- $C' \to cc$
- $B \to b|A$
- $A \to \varepsilon|A'$
- $A' \to B'|C$
- $B' \to Bb$

Grammar graph

Given a binarized grammar $\mathcal{G}$, we can consider its **graph** $G(\mathcal{G}) = (\mathbb{V}, \mathbb{E})$, where: $\mathbb{V} = (\mathcal{N} \cup \Sigma \cup \{\varnothing, \varepsilon\})$ and $(u, v) \in \mathbb{E}$ if and only if there is a production in $\mathcal{P}$ which has u on its left side and v on the right side.

The resulting graph is directed and has ordered edges, i.e. we distinguish the left child from the right one.

Grammar graph

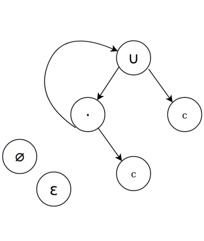
- $\mathcal{L} = (\mathcal{L} \cdot c) \cup c$

- $L = L'|c$
- $L' = Lc$

Grammar graph

- $\mathcal{L} = (\mathcal{L} \cdot c) \cup c$

- $L = L' | c$
- $L' = Lc$



Derivative of a language

Derivative by a symbol – b – from alphabet:

Derivative of a language

Derivative by a symbol – b – from alphabet:

- take a language: $\{foo, bar, bee\}$

Derivative of a language

Derivative by a symbol – b – from alphabet:

- take a language: $\{foo, bar, bee\}$
- leave only these words, which start with the chosen symbol: $\{bar, bee\}$

Derivative of a language

Derivative by a symbol – b – from alphabet:

- take a language: $\{foo, bar, bee\}$
- leave only these words, which start with the chosen symbol: $\{bar, bee\}$
- remove first symbol: $\{ar, ee\}$

Derivative of a language

Derivative by a symbol – b – from alphabet:

- take a language: $\{foo, bar, bee\}$
- leave only these words, which start with the chosen symbol: $\{bar, bee\}$
- remove first symbol: $\{ar, ee\}$
- voilà

Derivative of a language

Derivative by a symbol – b – from alphabet:

- take a language: $\{foo, bar, bee\}$
- leave only these words, which start with the chosen symbol: $\{bar, bee\}$
- remove first symbol: $\{ar, ee\}$
- voilà ($D_b(\{foo, bar, bee\}) = \{ar, ee\}$)

Derivative of a language

Derivative by a symbol – b – from alphabet:

- take a language: $\{foo, bar, bee\}$
- leave only these words, which start with the chosen symbol: $\{bar, bee\}$
- remove first symbol: $\{ar, ee\}$
- voilà $(D_b(\{foo, bar, bee\}) = \{ar, ee\})$

Derivative of a language

$D_c(L) = \{w \mid cw \in L\}$

Derivative of a language

Derivative of a language

$D_c(L) = \{w \mid cw \in L\}$

So $cw \in L \iff w \in D_c(L)$

$w = a_1 \dots a_k \implies D_w(L) = D_{a_k}(\dots D_{a_1}(L)\dots)$
So $w \in L \iff \varepsilon \in D_w(L)$

Derivatives

- $D_c(\varnothing) = \varnothing$
- $D_c(\varepsilon) = \varnothing$
- $D_c(a) = \begin{cases} \varnothing & a \neq c \\ \varepsilon & a = c \end{cases}$
- $D_c(L_1 \cup L_2) = D_c(L_1) \cup D_c(L_2)$
- $\left(D_c(L^\star) = D_c(L) \circ L^\star\right)$
- $D_c(L_1 \circ L_2) = \begin{cases} D_c(L_1) \circ L_2 & \varepsilon \notin L_1 \\ (D_c(L_1) \circ L_2) \cup D_c(L_2) & \varepsilon \in L_1 \end{cases}$

Nullability function

$$\delta : \mathcal{P}(\Sigma^\star) \to \{\varnothing, \{\varepsilon\}\}$$

$$\delta(L) = \begin{cases} \varnothing & \varepsilon \notin L \\ \{\varepsilon\} & \varepsilon \in L \end{cases}$$

Nullability function

$$\delta : \mathcal{P}(\Sigma^\star) \to \{\varnothing, \{\varepsilon\}\}$$

$$\delta(L) = \begin{cases} \varnothing & \varepsilon \notin L \\ \{\varepsilon\} & \varepsilon \in L \end{cases}$$

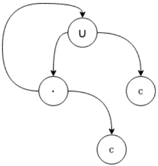
With this, we have: $D_c(L_1 \circ L_2) = (D_c(L_1) \circ L_2) \cup (\delta(L_1) \circ D_c(L_2))$

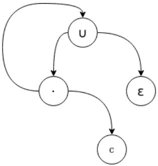
Nullability function

- $\delta(\varnothing) = \varnothing$
- $\delta(\varepsilon) = \varepsilon$
- $\delta(a) = \varnothing$
- $\delta(L_1 \cup L_2) = \delta(L_1) \cup \delta(L_2)$
- $\delta(L_1 \circ L_2) = \delta(L_1) \circ \delta(L_2)$
- $\left(\delta(L^\star) = \varepsilon\right)$

Derivative on graphs

$$\mathcal{L} = (\mathcal{L} \cdot c) \cup c$$
$$D_c(\mathcal{L}) = (D_c(\mathcal{L}) \cdot c) \cup \varepsilon$$





Recognizing algorithm

```
def recognize(G, w):
    for c ∈ w:
        G = D_c(G)
    return δ(G)
```

Parsing

Some problems:

Parsing

Some problems:

- forgetting about parsed symbols (tokens) – $D_c(\{c\}) = \varepsilon$

Parsing

Some problems:

- forgetting about parsed symbols (tokens) – $D_c(\{c\}) = \varepsilon$
- loosing information by using δ – $D_c(L_1 \circ L_2) = (D_c(L_1) \circ L_2) \cup (\delta(L_1) \circ D_c(L_2))$

Parsing

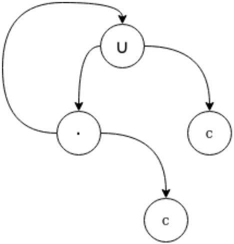
Some problems:

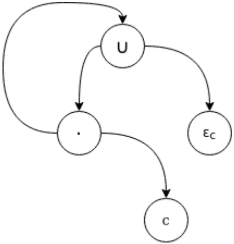
- forgetting about parsed symbols (tokens) – $D_c(\{c\}) = \varepsilon$
- loosing information by using δ –
  $D_c(L_1 \circ L_2) = (D_c(L_1) \circ L_2) \cup (\delta(L_1) \circ D_c(L_2))$
- skewing parse tree (by operators associativity)

Parsed tokens

$$\varepsilon_\Sigma = \{\varepsilon_a : a \in \Sigma\}$$

$$D_a(c) = \begin{cases} \varepsilon_a & a = c \\ \varnothing & a \neq c \end{cases}$$





Delta nodes

$$D_c(L_1 \circ L_2) = (D_c(L_1) \circ L_2) \cup (\Delta(L_1) \circ D_c(L_2))$$

$$D_a(\Delta(P)) = \varnothing$$

Markers

$$\mathcal{M} = \{\dashv_i\}_{i \in \mathbb{N}}$$

- $C \to ccc \dashv_1$
- $B \to b \dashv_2 \ \mid A \dashv_3$
- $A \to \varepsilon \dashv_4 \ \mid Bb \dashv_5 \ \mid C \dashv_6$

Markers

$$\mathcal{M} = \{\dashv_i\}_{i \in \mathbb{N}}$$

- $C \to ccc \dashv_1$
- $B \to b \dashv_2 \ \mid A \dashv_3$
- $A \to \varepsilon \dashv_4 \ \mid Bb \dashv_5 \ \mid C \dashv_6$

- $C \to C'C''$
- $C'' \to c \dashv_1$
- $C' \to cc$
- $B \to B' \mid B''$
- $B'' \to A \dashv_3$
- $B' \to b \dashv_2$
- $\ldots$

Semirings

A semiring $\mathcal{R}$ is a triple $(R, +_{\mathcal{R}}, \cdot_{\mathcal{R}}, 0_{\mathcal{R}}, 1_{\mathcal{R}})$, where:

- R is a set of semiring's elements
- $(R, +_{\mathcal{R}})$ is a commutative monoid with $0_{\mathcal{R}}$ as an identity element
- $(R, \cdot_{\mathcal{R}})$ is a monoid with $1_{\mathcal{R}}$ as an identity element and 0 as an annihilator
- $\cdot_{\mathcal{R}}$ is distributive over $+_{\mathcal{R}}$ (on both sides)

Semirings

A semiring $\mathcal{R}$ is a triple $(R, +_{\mathcal{R}}, \cdot_{\mathcal{R}}, 0_{\mathcal{R}}, 1_{\mathcal{R}})$, where:

- R is a set of semiring's elements
- $(R, +_{\mathcal{R}})$ is a commutative monoid with $0_{\mathcal{R}}$ as an identity element
- $(R, \cdot_{\mathcal{R}})$ is a monoid with $1_{\mathcal{R}}$ as an identity element and 0 as an annihilator
- $\cdot_{\mathcal{R}}$ is distributive over $+_{\mathcal{R}}$ (on both sides)

We skip the $_{\mathcal{R}}$ subscript next to the operators and elements whenever possible.

Semirings

For our purposes, we will also require the semirings to have an additional element $\infty_{\mathcal{R}}$ (or simply ∞) with the following properties:

$$
\begin{aligned}
\forall_{e \in R} \quad e + \infty &= \infty, \\
0 \cdot \infty = \infty \cdot 0 &= 0, \\
\forall_{e \in R - \{0\}} \quad e \cdot \infty = \infty \cdot e &= \infty.
\end{aligned}
$$

Embedding

Every nonterminal P can be associated with the language $\mathcal{L}(P) \subseteq \Sigma^\star$. Thus we can perceive alternative and concatenation as respective operators in the semiring $\mathcal{R}_\Sigma = (\wp(\Sigma^\star), \cup, \cdot, \varnothing, \{\varepsilon\})$.

Embedding

Every nonterminal P can be associated with the language $\mathcal{L}(P) \subseteq \Sigma^\star$. Thus we can perceive alternative and concatenation as respective operators in the semiring $\mathcal{R}_\Sigma = (\wp(\Sigma^\star), \cup, \cdot, \varnothing, \{\varepsilon\})$.

Now we can generalize the function δ introduced previously. From now, $\delta_\mathcal{R}$ will represent any homomorphism between $\mathcal{R}_\Sigma$ and an arbitrary semiring $\mathcal{R}$.

Generic algorithm

```
def recognize<𝓡>(G, w):
    for c ∈ w:
        G = D_c(G)
    return δ_𝓡(G)
```

Back to recognition

For recognition we can work with the Boolean semiring, i.e. $\mathcal{R}_{\mathbb{B}} = (\mathbb{B}, \vee, \wedge, 0, 1)$ with $\mathbb{B} = \{0, 1\}$, $\infty = 1$.

Back to recognition

For recognition we can work with the Boolean semiring, i.e. $\mathcal{R}_{\mathbb{B}} = (\mathbb{B}, \vee, \wedge, 0, 1)$ with $\mathbb{B} = \{0, 1\}$, $\infty = 1$.

$$\begin{aligned} \delta_{\mathbb{B}}(\varnothing) &= 0, \\ \delta_{\mathbb{B}}(\epsilon) &= 1, \\ \forall_{a \in \Sigma} \quad \delta_{\mathbb{B}}(\varepsilon_a) &= 1, \\ \forall_{a \in \Sigma} \quad \delta_{\mathbb{B}}(a) &= 0. \end{aligned}$$

Counting parse trees

For counting parse trees we can work with $\mathcal{R}_{\mathbb{N}} = (\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$, which is the standard semiring of non-negative integers, enriched with a special element ∞ behaving "naturally", except that $\infty \cdot 0 = 0$.

Counting parse trees

For counting parse trees we can work with $\mathcal{R}_{\mathbb{N}} = (\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$, which is the standard semiring of non-negative integers, enriched with a special element ∞ behaving "naturally", except that $\infty \cdot 0 = 0$.

$$\begin{aligned} \delta_{\mathbb{N}}(\varnothing) &= 0 \\ \delta_{\mathbb{N}}(\varepsilon) &= 1 \\ \forall_{a \in \Sigma \cup \mathcal{M}} \quad \delta_{\mathbb{N}}(\varepsilon_a) &= 1 \\ \forall_{a \in \Sigma} \quad \delta_{\mathbb{N}}(a) &= 0 \end{aligned}$$

Parsing

For parsing we can work with $\mathcal{R}_{\aleph} = (\mathcal{Q} \times (\epsilon_{\Sigma} \cup \mathcal{M})^{\star}, \oplus, \otimes)$, where the first coordinate, an element from $\mathcal{Q} = \{$ NONE, UNIQUE, FINITELY_MANY, INFINITELY_MANY$\}$ is one of the quantity categories.

Parsing

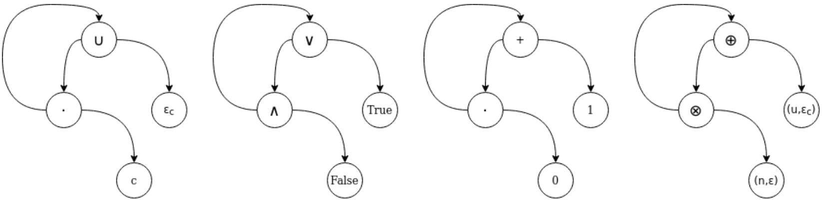
For parsing we can work with $\mathcal{R}_{\aleph} = (\mathcal{Q} \times (\epsilon_{\Sigma} \cup \mathcal{M})^{\star}, \oplus, \otimes)$, where the first coordinate, an element from $\mathcal{Q} = \{$ NONE, UNIQUE, FINITELY_MANY, INFINITELY_MANY$\}$ is one of the quantity categories.

- if the first coordinate is UNIQUE, the second one is the postorder of this unique parse tree; otherwise there could be anything - we do not care
- both operators $\oplus, \otimes$ when given two elements from $R_{\aleph}$ firstly look at the quantity coordinates and depending on them determine the resulting quantity. Then, if the resulting quantity is UNIQUE, they combine the second coordinates.
- 0-element is (NONE, ϵ) (in fact the second coordinate can contain anything)
- 1-element is $(\text{UNIQUE}, \epsilon)$
- ∞-element is $(\text{INFINITELY\_MANY}, \epsilon)$ (in fact the second coordinate can contain anything)

Parsing

$$
\begin{aligned}
& \delta_{\aleph}(\varnothing) = (\text{NONE}, \varepsilon) \\
& \delta_{\aleph}(\varepsilon) = (\text{UNIQUE}, \varepsilon) \\
\forall_{a \in \Sigma \cup \mathcal{M}} \quad & \delta_{\aleph}(\varepsilon_a) = (\text{UNIQUE}, \varepsilon_a) \\
\forall_{a \in \Sigma} \quad & \delta_{\aleph}(a) = (\text{NONE}, \varepsilon)
\end{aligned}
$$

All of them



Computing embedding

$$\delta \; : \; \mathbb{V}(\mathcal{G}) \; \rightarrow \; R$$

$$\delta(X) = \begin{cases} 0 & \text{if } X \; : \; \mathit{Empty} \\ 0 & \text{if } X \; : \; \mathit{Token} \\ 1 & \text{if } X \; : \; \mathit{Epsilon} \\ e_X \in R & \text{if } X \; : \; \mathit{ParsedToken} \\ \delta(X.\mathit{ref}) & \text{if } X \; : \; \mathit{Delta} \\ \delta(X.\mathit{left}) \; +_{\mathcal{R}} \; \delta(X.\mathit{right}) & \text{if } X \; : \; \mathit{Alternative} \\ \delta(X.\mathit{left}) \cdot_{\mathcal{R}} \delta(X.\mathit{right}) & \text{if } X \; : \; \mathit{Concatenation} \end{cases}$$

Disambiguation

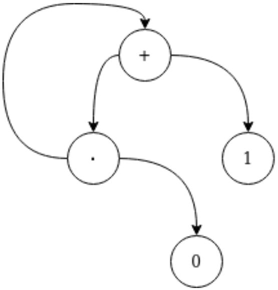
Any ambiguities (that may appear due to cycles) can be dealt with by the following two rules:

$$\delta(X) = \alpha_1 \cdot \delta(X) \cdot \alpha_2 \implies \delta(X) = 0$$

$$\delta(X) = \alpha_1 \cdot \delta(X) \cdot \alpha_2 + \beta \ \wedge \ \alpha_i, \beta \neq 0 \ \wedge \ \delta(X) \notin \beta \implies \delta(X) = \infty$$

Algorithm

1. find all nodes X for which $\delta_{\mathcal{R}}(X) = 0_{\mathcal{R}}$,
2. propagate all "finite" values from $\mathcal{R}$ as far as it is possible
3. what remains, should be equal to $\infty_{\mathcal{R}}$.



Pseudocode

See in the full version.

Correctness

Correctness

- finding all zeros (induction on the number of connected components)

Correctness

- finding all zeros (induction on the number of connected components)
- after marking all 0s and propagating finite values as far as possible, any value not yet calculated must be ∞ (by *recursive alternatives*)

Correctness

- finding all zeros (induction on the number of connected components)
- after marking all 0s and propagating finite values as far as possible, any value not yet calculated must be ∞ (by *recursive alternatives*)
- sum disjointness (by *(semi-)parse-words*)