Parsing Data-Dependent Grammars using Derivatives

… in exponential time

Jaro Reinders - 14-08-2024

```
number >>= \n →
  fix (\p i → guard (i > 0) *> char '.' *> p (i - 1)
          <|> guard (i == 0)) n
```

```
fix (\p (a,m) →
      Add <$ guard a <*> p (0,1) <* char '+' <*> p (1,1)
  <|> Mul <$ guard m <*> p (0,0) <* char '*' <*> p (0,1)
  <|> Atom <$> number
  <|> char '(' *> p (1,1) <* char ')'
) (1,1)
```

Outline

- Context-Free Grammars
- Recognizing CFGs
- Parsing CFGs
- Data-Dependent Grammars
- Future work

Context-Free Grammars

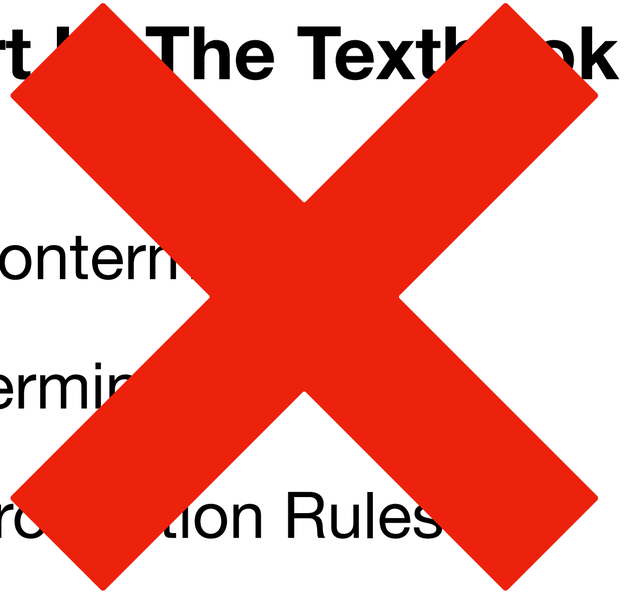
Context-Free Grammars

Part I - The Textbook Definition

- Nonterminals
- Terminals
- Production Rules

Context-Free Grammars

Part I: The Textbook Definition



- Nonterminals
- Terminals
- Production Rules

Context-Free Grammars

Part II - A Language for CFGs

```
data CFG where
  Fail : CFG
  Or   : CFG → CFG → CFG
  Done : CFG
  Seq  : CFG → CFG → CFG
  Char : Char → CFG
```

Context-Free Grammars

Part II - A Language for CFGs

```
data CFG n where
  Fail : CFG n
  Or   : CFG n → CFG n → CFG n
  Done : CFG n
  Seq  : CFG n → CFG n → CFG n
  Char : Char → CFG n
  Var  : Fin n → CFG n
  Fix  : CFG (1 + n) → CFG n
```

Recognizing

Brzozowski Derivative

Brzozowski Derivative

$$[\![\_]\!] : \mathrm{CFG} \to \mathrm{Set}\ \mathrm{String}$$

Brzozowski Derivative

```
⟦_⟧ : CFG → Set String

derivative : CFG → Char → CFG

 if ⟦g⟧ = {"","ab","acd","de"}

 then ⟦derivative g 'a'⟧ = {"b", "cd"}
```

Brzozowski Derivative

```
 ⟦_⟧ : CFG → Set String

derivative : CFG → Char → CFG

 if ⟦g⟧ = {"","ab","acd","de"}

 then ⟦derivative g 'a'⟧ = {"b", "cd"}

nullable : CFG → Bool

 nullable g = "" ∈ ⟦g⟧
```

Recognizing CFGs

Part I - Example

```
g = Fix (Or (Char ‘x’) (Seq (Var 0) (Var 0))
```

Recognizing CFGs

Part I - Example

```
g =CFG Or (Char ‘x’) (Seq g g)
```

Recognizing CFGs

Part I - Example

```
g =CFG Or (Char ‘x’) (Seq g g)
```

$\texttt{nullable g} = g^0 = \texttt{False}$

Recognizing CFGs

Part I - Example

```
g =CFG Or (Char ‘x’) (Seq g g)

nullable g = g⁰ = False

derivative g ‘x’ = g’ =CFG
          Or Done (Or (Seq g’ g) (if g⁰ then g’ else Fail))
```

Recognizing CFGs

Part I - Example

```
g =CFG Or (Char ‘x’) (Seq g g)

nullable g = g⁰ = False

derivative g ‘x’ = g’ =CFG
          Or Done (Or (Seq g’ g) (if g⁰ then g’ else Fail))

nullable g’ = True
```

Recognizing CFGs

Part II - Nullability

```
nullable : CFG n → Bool
nullable Fail = False
nullable (Or p q) = nullable p || nullable q
nullable Done = True
nullable (Seq p q) = nullable p && nullable q
nullable (Char _) = False
nullable (Var v) = False
nullable (Fix p) = nullable p
```

Recognizing CFGs

Part II - Nullability

```
nullable : CFG n → (Fin n → Bool) → Bool
nullable Fail _ = False
nullable (Or p q) ev = nullable p ev || nullable q ev
nullable Done _ = True
nullable (Seq p q) ev = nullable p ev && nullable q ev
nullable (Char _) _ = False
nullable (Var v) ev = ev v
nullable (Fix p) ev = nullable p [0 ⟼ False; ev]
```

Recognizing CFGs

Part III - Derivatives

```
derivative : CFG n → (Fin n → (Bool, Fin n)) → Char → CFG n
```

Recognizing CFGs

Part III - Derivatives

```
derivative : CFG n → (Fin n → (Bool, Fin n)) → Char → CFG n
derivative Fail _ _ = Fail
```

Recognizing CFGs

Part III - Derivatives

```
derivative : CFG n → (Fin n → (Bool, Fin n)) → Char → CFG n
derivative Fail _ _ = Fail
derivative (Or p q) ev c = Or (derivative p ev c) (derivative q ev c)
```

Recognizing CFGs

Part III - Derivatives

```
derivative : CFG n → (Fin n → (Bool, Fin n)) → Char → CFG n
derivative Fail _ _ = Fail
derivative (Or p q) ev c = Or (derivative p ev c) (derivative q ev c)
derivative Done _ _ = Fail
```

Recognizing CFGs

Part III - Derivatives

```
derivative : CFG n → (Fin n → (Bool, Fin n)) → Char → CFG n
derivative Fail _ _ = Fail
derivative (Or p q) ev c = Or (derivative p ev c) (derivative q ev c)
derivative Done _ _ = Fail
derivative (Seq p q) c ev = Or (Seq (derivative p ev c) q)
  (if nullable p (π₁ . ev) then derivative q ev c else Fail)
```

Recognizing CFGs

Part III - Derivatives

```
derivative : CFG n → (Fin n → (Bool, Fin n)) → Char → CFG n
derivative Fail _ _ = Fail
derivative (Or p q) ev c = Or (derivative p ev c) (derivative q ev c)
derivative Done _ _ = Fail
derivative (Seq p q) c ev = Or (Seq (derivative p ev c) q)
  (if nullable p (π₁ . ev) then derivative q ev c else Fail)
derivative (Char c') _ c = guard (c == c')
```

Recognizing CFGs

Part III - Derivatives

```
derivative : CFG n → (Fin n → (Bool, Fin n)) → Char → CFG n
derivative Fail _ _ = Fail
derivative (Or p q) ev c = Or (derivative p ev c) (derivative q ev c)
derivative Done _ _ = Fail
derivative (Seq p q) c ev = Or (Seq (derivative p ev c) q)
  (if nullable p (π₁ . ev) then derivative q ev c else Fail)
derivative (Char c') _ c = guard (c == c')
derivative (Fix p) ev c =
  Fix (derivative (1 + p)
         [1 ⟼ (nullable (Fix p) (π₁ . ev), 0); ev] c)
    [0 ⟼ Fix p]
derivative (Var v) ev _ = Var (π₂ (ev v))
```

Parsing

Parsing CFGs

Part I - A Language for Parsing CFGs

```
data CFG a where
  Fail : CFG a
  Or   : CFG a → CFG a → CFG a
  Done : a → CFG a
  Seq  : CFG a → (a → CFG b) → CFG b
  Char : Char → CFG ()
```

Parsing CFGs

Part I - A Language for Parsing CFGs

```
data CFG Γ a where
  Fail : CFG Γ a
  Or   : CFG Γ a → CFG Γ a → CFG Γ a
  Done : a → CFG Γ a
  Seq  : CFG Γ x → (x → CFG Γ a) → CFG Γ a
  Char : Char → CFG Γ ()
  Var  : a ∈ Γ → CFG Γ a
  Fix  : CFG (a :: Γ) a → CFG Γ a

Γ : List Type
_∈_ : Type → List Type → Type
```

Parsing CFGs

Part II - Nullability

```
nullable : CFG Γ a → (∀x. x ∈ Γ → [x]) → [a]
nullable Fail _ = []
nullable (Or p q) ev = nullable p ev ++ nullable q ev
nullable (Done x) _ = [x]
nullable (Seq p q) ev =
  [ y | x ← nullable p ev, y ← nullable (q x) ev ]
nullable (Char _) _ = []
nullable (Var v) ev = ev v
nullable (Fix p) ev = nullable p [0 ⟼ []; ev]
```

Parsing CFGs

Part III - Derivatives

```
derivative : CFG Γ a → (∀x. x ∈ Γ → ([x], x ∈ Γ))
           → Char → CFG Γ a

derivative (Seq p q) c ev = Or (Seq (derivative p ev c) q)
  (foldr Or Fail
     [derivative (q x) ev c | x ← nullable (π₁ . ev) p])
```

Data-Dependent Grammars

Data-Dependent Grammars

```
number >>= \n →
  fix (\p i → guard (i > 0) *> char '.' *> p (i - 1)
            <|> guard (i == 0)) n
```

Parsing DDGs

Part I - A Language for DDGs

```
data DDG Γ a where
  Fail : DDG Γ a
  Or   : DDG Γ a → DDG Γ a → DDG Γ a
  Done : a →  Γ a
  Seq  : DDG Γ x → (x → DDG Γ a) → DDG Γ a
  Char : Char → DDG Γ ()
  Var  : Fun x a ∈ Γ → x → DDG Γ a
  Fix  : (x → DDG (Fun x a :: Γ) a) → x → DDG Γ a

type Fun a b = a → [b]
```

Parsing DDGs

Part II - Nullability

```
nullable : CFG Γ a → (∀x. x ∈ Γ → x) → [a]

nullable (Var v x) ev = ev v x
nullable (Fix p x) ev = nullable (p x) [0 ⟼ const []; ev]
```

Parsing DDGs

Part III - Derivatives

```
derivative : CFG Γ a → (∀x. x ∈ Γ → (x, x ∈ Γ))
           → Char → CFG Γ a

derivative (Fix p x) ev c =
  (Fix (\x′ → derivative (1 + p x′)
         [1 ⟼ (\x″ → nullable (Fix p x″) ev, 0); ev] c)
    x)
    [0 ⟼ Fix p]
derivative (Var v x) ev _ = Var (π₂ (ev v)) x
```

Future Work

Future Work

- Explain nullability semantics of variables
- Performance (memoization)
- Abstract over common disambiguation strategies