From EBNF to PEG

Roman R. Redziejowski

Concurrency, Specification and Programming
Berlin 2012

EBNF: Extended Backus-Naur Form

A way to define grammar.

EBNF: Extended Backus-Naur Form

A way to define grammar.

$$
\begin{array}{lcl}
\mathit{Literal} & = & \mathit{Decimal} \;\; | \;\; \mathit{Binary} \\
\mathit{Decimal} & = & \text{[0-9]}^{+} \;\; \text{"."} \;\; \text{[0-9]}^{*} \\
\mathit{Binary} & = & \text{[01]}^{+} \;\; \text{"B"}
\end{array}
$$

Recursive-descent parsing

Parsing procedure for each equation and each terminal.

$$\begin{array}{lcl} \mathit{Literal} & = & \mathit{Decimal} \;\;|\;\; \mathit{Binary} \\ \mathit{Decimal} & = & \text{[0-9]}^{+} \text{ "." } \text{[0-9]}^{*} \\ \mathit{Binary} & = & \text{[01]}^{+} \text{ "B"} \end{array}$$

Literal calls *Decimal* or *Binary*.
Decimal calls repeatedly [0-9], then ".", then repeatedly [0-9].
Binary calls repeatedly [01], then "B".

Recursive-descent parsing

Parsing procedure for each equation and each terminal.

Literal	=	*Decimal* | *Binary*
Decimal	=	[0-9]$^+$ "." [0-9]*
Binary	=	[01]$^+$ "B"

Literal calls *Decimal* or *Binary*.
Decimal calls repeatedly [0-9], then ".", then repeatedly [0-9].
Binary calls repeatedly [01], then "B".

Problem: *Decimal* and *Binary* may start with any number of 0's and 1's.
Literal cannot choose which procedure to call by looking at any fixed distance ahead.

Solution: Backtracking

$$
\begin{aligned}
\mathit{Literal} &= \mathit{Decimal} \;|\; \mathit{Binary} \\
\mathit{Decimal} &= \text{[0-9]}^{+}\ \text{"."}\ \text{[0-9]}^{*} \\
\mathit{Binary} &= \text{[01]}^{+}\ \text{"B"}
\end{aligned}
$$

```
101B
^
```

Solution: Backtracking

$$
\begin{aligned}
\textit{Literal} &= \textit{Decimal} \mid \textit{Binary} \\
\textit{Decimal} &= [0\text{-}9]^{+}\ \text{"."}\ [0\text{-}9]^{*} \\
\textit{Binary} &= [01]^{+}\ \text{"B"}
\end{aligned}
$$

```
101B
^
```

Literal

Solution: Backtracking

$$
\begin{array}{lcl}
\mathit{Literal} & = & \mathit{Decimal} \;\; | \;\; \mathit{Binary} \\
\mathit{Decimal} & = & [0\text{-}9]^{+} \text{ "." } [0\text{-}9]^{*} \\
\mathit{Binary} & = & [01]^{+} \text{ "B"}
\end{array}
$$

```
101B
^
```

$\mathit{Literal} \rightarrow \mathit{Decimal}$

$$
\begin{aligned}
\textit{Literal} &= \textit{Decimal} \mid \textit{Binary} \\
\textit{Decimal} &= [0\text{-}9]^{+}\ \text{"."}\ [0\text{-}9]^{*} \\
\textit{Binary} &= [01]^{+}\ \text{"B"}
\end{aligned}
$$

```
101B
^
```

$\textit{Literal} \rightarrow \textit{Decimal} \rightarrow [0\text{-}9]$

Solution: Backtracking

$$\begin{array}{lcl} \textit{Literal} & = & \textit{Decimal} \mid \textit{Binary} \\ \textit{Decimal} & = & [0\text{-}9]^{+} \text{ "." } [0\text{-}9]^{*} \\ \textit{Binary} & = & [01]^{+} \text{ "B"} \end{array}$$

```
101B
   ^
```

Literal$\rightarrow$*Decimal*$\rightarrow$[0-9] : advance 3 times

Solution: Backtracking

$$\begin{array}{lcl} \textit{Literal} & = & \textit{Decimal} \ | \ \textit{Binary} \\ \textit{Decimal} & = & [0\text{-}9]^{+} \ "." \ [0\text{-}9]^{*} \\ \textit{Binary} & = & [01]^{+} \ "B" \end{array}$$

```
101B
   ^
```

Literal→*Decimal*→"."

Solution: Backtracking

$$\begin{aligned} \mathit{Literal} &= \mathit{Decimal} \;\; | \;\; \mathit{Binary} \\ \mathit{Decimal} &= [0\text{-}9]^{+} \; \text{"."} \; [0\text{-}9]^{*} \\ \mathit{Binary} &= [01]^{+} \; \text{"B"} \end{aligned}$$

```
101B
^
```

$\mathit{Literal} \rightarrow \mathit{Decimal} \rightarrow$ "." : fail, backtrack

Solution: Backtracking

$$
\begin{aligned}
\mathit{Literal} &= \mathit{Decimal} \mid \mathit{Binary} \\
\mathit{Decimal} &= [0\text{-}9]^{+}\ \text{"."}\ [0\text{-}9]^{*} \\
\mathit{Binary} &= [01]^{+}\ \text{"B"}
\end{aligned}
$$

```
101B
^
```

Literal

Solution: Backtracking

$$
\begin{aligned}
\textit{Literal} &= \textit{Decimal} \mid \textit{Binary} \\
\textit{Decimal} &= [0\text{-}9]^{+}\ \text{"."}\ [0\text{-}9]^{*} \\
\textit{Binary} &= [01]^{+}\ \text{"B"}
\end{aligned}
$$

```
101B
^
```

$\textit{Literal} \rightarrow \textit{Binary}$

Solution: Backtracking

$$
\begin{aligned}
\mathit{Literal} &= \mathit{Decimal} \mid \mathit{Binary} \\
\mathit{Decimal} &= [0\text{-}9]^{+} \text{ "." } [0\text{-}9]^{*} \\
\mathit{Binary} &= [01]^{+} \text{ "B"}
\end{aligned}
$$

```
101B
^
```

$\mathit{Literal} \rightarrow \mathit{Binary} \rightarrow [01]$

Solution: Backtracking

$$
\begin{array}{lcl}
\mathit{Literal} & = & \mathit{Decimal} \;\;|\;\; \mathit{Binary} \\
\mathit{Decimal} & = & [0\text{-}9]^{+} \text{ "." } [0\text{-}9]^{*} \\
\mathit{Binary} & = & [01]^{+} \text{ "B"}
\end{array}
$$

```
101B
   ^
```

$\mathit{Literal} \rightarrow \mathit{Binary} \rightarrow [01]$: advance 3 times

Solution: Backtracking

$$
\begin{aligned}
\mathit{Literal} &= \mathit{Decimal} \;\;|\;\; \mathit{Binary} \\
\mathit{Decimal} &= \text{[0-9]}^{+}\ \text{"."}\ \text{[0-9]}^{*} \\
\mathit{Binary} &= \text{[01]}^{+}\ \text{"B"}
\end{aligned}
$$

```
101B
   ^
```

Literal→*Binary*→"B"

Solution: Backtracking

$$
\begin{array}{lcl}
\mathit{Literal} & = & \mathit{Decimal} \mid \mathit{Binary} \\
\mathit{Decimal} & = & [0\text{-}9]^{+}\ \text{"."}\ [0\text{-}9]^{*} \\
\mathit{Binary} & = & [01]^{+}\ \text{"B"}
\end{array}
$$

```
101B
    ^
```

Literal→*Binary*→"B" : advance, return

Solution: Backtracking

$$
\begin{aligned}
\textit{Literal} &= \textit{Decimal} \;\mid\; \textit{Binary} \\
\textit{Decimal} &= \text{[0-9]}^{+} \; \text{"."} \; \text{[0-9]}^{*} \\
\textit{Binary} &= \text{[01]}^{+} \; \text{"B"}
\end{aligned}
$$

Limited backtracking

Backtracking solves the problem,
but may take exponential time.

Limited backtracking

Backtracking solves the problem,
but may take exponential time.

Solution: limited backtracking.
Never go back after one alternative succeeded.

Limited backtracking

Backtracking solves the problem,
but may take exponential time.

Solution: limited backtracking.
Never go back after one alternative succeeded.

- 1961 Brooker & Morris - Altas Compiler Compiler
- 1965 McClure - TransMoGrifier (TMG)
- 1972 Aho & Ullman - Top-Down Parsing Language (TDPL)
- ...
- 2004 Ford - Parsing Expression Grammar (PEG)

Limited backtracking

Backtracking solves the problem,
but may take exponential time.

Solution: limited backtracking.
Never go back after one alternative succeeded.

- 1961 Brooker & Morris - Altas Compiler Compiler
- 1965 McClure - TransMoGrifier (TMG)
- 1972 Aho & Ullman - Top-Down Parsing Language (TDPL)
- ...
- 2004 Ford - Parsing Expression Grammar (PEG)

It can work in linear time.

PEG - Parsing Expression Grammar

Looks exactly like EBNF:

$$
\begin{array}{lcl}
\textit{Literal} & = & \textit{Decimal} \;\; / \;\; \textit{Binary} \\
\textit{Decimal} & = & \text{[0-9]}^{+} \;\; \text{"."} \;\; \text{[0-9]}^{*} \\
\textit{Binary} & = & \text{[01]}^{+} \;\; \text{"B"}
\end{array}
$$

PEG - Parsing Expression Grammar

Looks exactly like EBNF:

$$
\begin{aligned}
\textit{Literal} &= \textit{Decimal} \ \ / \ \ \textit{Binary} \\
\textit{Decimal} &= [\text{0-9}]^{+} \ \text{"."} \ [\text{0-9}]^{*} \\
\textit{Binary} &= [\text{01}]^{+} \ \text{"B"}
\end{aligned}
$$

Specification of a recursive-descent parser with limited backtracking, where "/" means an **ordered no-return choice**.

PEG is not EBNF

	EBNF:	PEG:
A = ("a" / "aa") "b"	{ab, aab}	{ab}
A = ("aa" / "a") "ab"	{aaab, aab}	{aaab}
A = ("a" / "b"?) "a"	{aa, ba, a}	{aa, ba}

PEG is not EBNF

	EBNF:	PEG:
A = ("a" / "aa") "b"	{ab, aab}	{ab}
A = ("aa" / "a") "ab"	{aaab, aab}	{aaab}
A = ("a" / "b"?) "a"	{aa, ba, a}	{aa, ba}

Backtracking may examine input far ahead
so result may depend on context in front.

PEG is not EBNF

	EBNF:	PEG:
A = ("a" / "aa") "b"	{ab, aab}	{ab}
A = ("aa" / "a") "ab"	{aaab, aab}	{aaab}
A = ("a" / "b"?) "a"	{aa, ba, a}	{aa, ba}

Backtracking may examine input far ahead so result may depend on context in front.

A = "a" A "a" / "aa"   EBNF: a^{2n} PEG: a^{2^n}

Sometimes PEG is EBNF

In this case PEG = EBNF:

$$\begin{array}{lcl} \mathit{Literal} & = & \mathit{Decimal} \;\; / \;\; \mathit{Binary} \\ \mathit{Decimal} & = & \text{[0-9]}^{+} \; \text{"."} \; \text{[0-9]}^{*} \\ \mathit{Binary} & = & \text{[01]}^{+} \; \text{"B"} \end{array}$$

Sometimes PEG is EBNF

In this case PEG = EBNF:

$$
\begin{aligned}
\mathit{Literal} &= \mathit{Decimal}\ /\ \mathit{Binary} \\
\mathit{Decimal} &= \text{[0-9]}^{+}\ \text{"."}\ \text{[0-9]}^{*} \\
\mathit{Binary} &= \text{[01]}^{+}\ \text{"B"}
\end{aligned}
$$

When does it happen?

When PEG = EBNF?

Sérgio Queiroz de Medeiros

Correspondência entre PEGs e Classes de Gramáticas Livres de Contexto.

Ph.D. Thesis

Pontifícia Universidade Católica do Rio deJaneiro (2010).

When PEG = EBNF?

Sérgio Queiroz de Medeiros

Correspondência entre PEGs e Classes de Gramáticas Livres de Contexto.

Ph.D. Thesis

Pontifícia Universidade Católica do Rio deJaneiro (2010).

If EBNF has LL(1) property then PEG = EBNF

When PEG = EBNF?

But this is not LL(1):

$$
\begin{aligned}
\mathit{Literal} &= \mathit{Decimal}\ /\ \mathit{Binary}\\
\mathit{Decimal} &= \text{[0-9]}^{+}\ \text{"."}\ \text{[0-9]}^{*}\\
\mathit{Binary} &= \text{[01]}^{+}\ \text{"B"}
\end{aligned}
$$

When PEG = EBNF?

But this is not LL(1):

$$\begin{aligned} \textit{Literal} &= \textit{Decimal} \;\; / \;\; \textit{Binary} \\ \textit{Decimal} &= [0\text{-}9]^{+} \; \text{"."} \; [0\text{-}9]^{*} \\ \textit{Binary} &= [01]^{+} \; \text{"B"} \end{aligned}$$

Which means PEG = EBNF for a wider class.

When PEG = EBNF?

But this is not LL(1):

$$
\begin{array}{lcl}
\mathit{Literal} & = & \mathit{Decimal} \;\; / \;\; \mathit{Binary} \\
\mathit{Decimal} & = & [0\text{-}9]^{+} \;\; \text{"."} \;\; [0\text{-}9]^{*} \\
\mathit{Binary} & = & [01]^{+} \;\; \text{"B"}
\end{array}
$$

Which means PEG = EBNF for a wider class.
Let us find more about it.

Simple grammar

Alphabet Σ (the "terminals").

Set N of names (the "nonterminals").

For each $A \in N$ one rule of the form:

- $A = e_1\, e_2$ (*Sequence*) or
- $A = e_1 \,|\, e_2$ (*Choice*)

where $e_1, e_2 \in N \cup \Sigma \cup \{\varepsilon\}$.

Start symbol $S \in A$.

"Syntax expressions": $\mathbb{E} = N \cup \Sigma \cup \{\varepsilon\}$.

Simple grammar

Alphabet Σ (the "terminals").

Set N of names (the "nonterminals").

For each $A \in N$ one rule of the form:

- $A = e_1\, e_2$ (*Sequence*) or
- $A = e_1 \mid e_2$ (*Choice*)

where $e_1, e_2 \in N \cup \Sigma \cup \{\varepsilon\}$.

Start symbol $S \in A$.

"Syntax expressions": $\mathbb{E} = N \cup \Sigma \cup \{\varepsilon\}$.

Will consider two interpretations: EBNF and PEG.

EBNF interpretation

$\mathcal{L}(e)$ – language of expression $e \in \mathbb{E}$.

- $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
- $\mathcal{L}(a) = \{a\}$ for $a \in \Sigma$
- $\mathcal{L}(A) = \mathcal{L}(e_1)\mathcal{L}(e_2)$ for $A = e_1\, e_2$
- $\mathcal{L}(A) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$ for $A = e_1 \,|\, e_2$

Language defined by the grammar: $\mathcal{L}(S)$.

"Natural semantics" (after Medeiros)

Relation $\overset{\text{BNF}}{\leadsto} \subseteq \mathbb{E} \times \Sigma^* \times \Sigma^*$, written $[e]\; x \overset{\text{BNF}}{\leadsto} y$.

$[e]\; xy \overset{\text{BNF}}{\leadsto} y$ means "xy has prefix $x \in \mathcal{L}(e)$".

Or: parsing procedure for e, applied to xy consumes x".

$w \in \mathcal{L}(S) \;\Leftrightarrow\; [S]\; w\$ \overset{\text{BNF}}{\leadsto} \$$

where \$ is "end of text" marker.

"Natural semantics" (after Medeiros)

$[e]\ x \overset{\text{BNF}}{\rightsquigarrow} y$ holds if and only if
it can be proved using these inference rules:

$$\frac{}{[\varepsilon]\ x \overset{\text{BNF}}{\rightsquigarrow} x} \qquad \frac{}{[a]\ ax \overset{\text{BNF}}{\rightsquigarrow} x}$$

$$\frac{A = e_1 e_2 \quad [e_1]\ xyz \overset{\text{BNF}}{\rightsquigarrow} yz \quad [e_2]\ yz \overset{\text{BNF}}{\rightsquigarrow} z}{[A]\ xyz \overset{\text{BNF}}{\rightsquigarrow} z}$$

$$\frac{A = e_1|e_2 \quad [e_1]\ xy \overset{\text{BNF}}{\rightsquigarrow} y}{[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y}$$

$$\frac{A = e_1|e_2 \quad [e_2]\ xy \overset{\text{BNF}}{\rightsquigarrow} y}{[A]\ xy \overset{\text{BNF}}{\rightsquigarrow} y}$$

Example of proof

Grammar: $S = aX, \quad X = S|b$ Proof of $aab \in \mathcal{L}(S)$

$$\frac{}{[b]\ b\$ \overset{BNF}{\leadsto} \$}$$

$$\frac{}{[a]\ ab\$ \overset{BNF}{\leadsto} b\$} \qquad \frac{X = S|b \quad [b]\ b\$ \overset{BNF}{\leadsto} \$}{[X]\ b\$ \overset{BNF}{\leadsto} \$}$$

$$\frac{S = aX \quad [a]\ ab\$ \overset{BNF}{\leadsto} b\$ \quad [X]\ b\$ \overset{BNF}{\leadsto} \$}{[S]\ ab\$ \overset{BNF}{\leadsto} \$}$$

$$\frac{}{[a]\ aab\$ \overset{BNF}{\leadsto} ab\$} \qquad \frac{X = S|b \quad [S]\ ab\$ \overset{BNF}{\leadsto} \$}{[X]\ ab\$ \overset{BNF}{\leadsto} \$}$$

$$\frac{S = aX \quad [a]\ aab\$ \overset{BNF}{\leadsto} ab\$ \quad [X]\ ab\$ \overset{BNF}{\leadsto} \$}{[S]\ aab\$ \overset{BNF}{\leadsto} \$}$$

PEG interpretation

Elements of $\mathbb{E}$ are parsing procedures
that consume input or return "failure".

- ε returns success without consuming input.
- a consumes a if input starts with a.
  Otherwise returns failure.
- $A = e_1\, e_2$ calls e_1 then e_2.
  If any of them failed, backtracks and returns failure.
- $A = e_1 \mid e_2$ calls e_1.
  If e_1 succeeded, returns success.
  If e_1 failed, calls e_2 and returns its result.

"Natural semantics" (after Medeiros)

Relation $\overset{\text{PEG}}{\rightsquigarrow} \subseteq \mathbb{E} \times \Sigma^* \times (\Sigma^* \cup \mathsf{fail})$, written $[e]\ x \overset{\text{PEG}}{\rightsquigarrow} y$.

- $[e]\ xy \overset{\text{PEG}}{\rightsquigarrow} y$ means "e consumes prefix x of xy".
- $[e]\ x \overset{\text{PEG}}{\rightsquigarrow}$ fail means "e applied to x returns failure".

w accepted by the grammar iff $[S]\ w\$ \overset{\text{PEG}}{\rightsquigarrow} \$$.

"Natural semantics" (after Medeiros)

$[e]\ x \overset{\text{PEG}}{\leadsto} Y$ holds if and only if
it can be proved using these inference rules:

$$\frac{}{[\varepsilon]\ x \overset{\text{PEG}}{\leadsto} x} \qquad \frac{}{[a]\ ax \overset{\text{PEG}}{\leadsto} x} \qquad \frac{b \neq a}{[b]\ ax \overset{\text{PEG}}{\leadsto} \text{fail}} \qquad \frac{}{[a]\ \varepsilon \overset{\text{PEG}}{\leadsto} \text{fail}}$$

$$\frac{A = e_1 e_2 \quad [e_1]\ xyz \overset{\text{PEG}}{\leadsto} yz \quad [e_2]\ yz \overset{\text{PEG}}{\leadsto} Z}{[A]\ xyz \overset{\text{PEG}}{\leadsto} Z}$$

$$\frac{A = e_1 e_2 \quad [e_1]\ x \overset{\text{PEG}}{\leadsto} \text{fail}}{[A]\ x \overset{\text{PEG}}{\leadsto} \text{fail}} \qquad \frac{A = e_1 | e_2 \quad [e_1]\ xy \overset{\text{PEG}}{\leadsto} y}{[A]\ xy \overset{\text{PEG}}{\leadsto} y}$$

$$\frac{A = e_1 | e_2 \quad [e_1]\ x \overset{\text{PEG}}{\leadsto} \text{fail} \quad [e_2]\ xy \overset{\text{PEG}}{\leadsto} Y}{[A]\ xy \overset{\text{PEG}}{\leadsto} Y}$$

where Y is y or fail and Z is z or fail.

When PEG = EBNF?

By induction on the height of proof trees for
$[S]\ w\$ \overset{\text{PEG}}{\leadsto} \$$ and $[S]\ w\$ \overset{\text{BNF}}{\leadsto} \$$:

- $[S]\ w\$ \overset{\text{PEG}}{\leadsto} \$ \;\Rightarrow\; [S]\ w\$ \overset{\text{BNF}}{\leadsto} \$$. (Medeiros)

When PEG = EBNF?

By induction on the height of proof trees for
$[S]\ w\$ \overset{\text{PEG}}{\rightsquigarrow} \$$ and $[S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$$:

- $[S]\ w\$ \overset{\text{PEG}}{\rightsquigarrow} \$ \ \Rightarrow\ [S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$$. (Medeiros)
- $[S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$ \ \Rightarrow\ [S]\ w\$ \overset{\text{PEG}}{\rightsquigarrow} \$$
  if for every Choice $A = e_1|e_2$ holds
  $\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(A)) = \varnothing$.

When PEG = EBNF?

By induction on the height of proof trees for
$[S]\ w\$ \overset{\text{PEG}}{\rightsquigarrow} \$$ and $[S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$$:

- $[S]\ w\$ \overset{\text{PEG}}{\rightsquigarrow} \$ \;\Rightarrow\; [S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$$. (Medeiros)
- $[S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$ \;\Rightarrow\; [S]\ w\$ \overset{\text{PEG}}{\rightsquigarrow} \$$
  if for every Choice $A = e_1 | e_2$ holds
  $\mathcal{L}(e_1) \cap \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(A)) = \varnothing$.

(Tail(A) is any possible continuation after A:
$y \in \mathrm{Tail}(A)$ iff proof tree of $[S]\ w\$ \overset{\text{BNF}}{\rightsquigarrow} \$$ for some w
contains partial result $[A]\ xy\$ \overset{\text{BNF}}{\rightsquigarrow} y\$$.)

When PEG = EBNF?

Let us say that Choice $A = e_1|e_2$ is "safe" to mean $\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(A)) = \varnothing$.

When PEG = EBNF?

Let us say that Choice $A = e_1|e_2$ is "safe" to mean
$\mathcal{L}(e_1) \cap \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(A)) = \varnothing$.

The two interpretations are equivalent if every Choice in the grammar is safe.

Notes

$$\boxed{\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(A)) = \varnothing}$$

Notes

$$\boxed{\mathcal{L}(e_1) \cap \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(A)) = \varnothing}$$

Requires $\varepsilon \notin \mathcal{L}(e_1)$.

Notes

$$\boxed{\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(A)) = \varnothing}$$

Requires $\varepsilon \notin \mathcal{L}(e_1)$.

Depends on context.

Notes

$$\boxed{\mathcal{L}(e_1) \cap \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(A)) = \varnothing}$$

Requires $\varepsilon \notin \mathcal{L}(e_1)$.

Depends on context.

Difficult to check: $\mathcal{L}(e_1)$, $\mathcal{L}(e_2)$, and $\mathrm{Tail}(A)$ can be any context-free languages.

Intersection of context-free languages is in general undecidable.

Notes

$$\boxed{\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(A)) = \varnothing}$$

Requires $\varepsilon \notin \mathcal{L}(e_1)$.

Depends on context.

Difficult to check: $\mathcal{L}(e_1)$, $\mathcal{L}(e_2)$, and $\text{Tail}(A)$ can be any context-free languages.

Intersection of context-free languages is in general undecidable.

Can be "approximated" by stronger conditions.

Approximation by first letters

Consider $A = e_1 | e_2$.

$\textsc{First}(e_1)$, $\textsc{First}(e_2)$:
sets of possible first letters
of words in $\mathcal{L}(e_1)$ respectively $\mathcal{L}(e_2)$.

Approximation by first letters

Consider $A = e_1 | e_2$.

$\textsc{First}(e_1)$, $\textsc{First}(e_2)$:
sets of possible first letters
of words in $\mathcal{L}(e_1)$ respectively $\mathcal{L}(e_2)$.

If $\mathcal{L}(e_1), \mathcal{L}(e_2)$, do not contain ε,
$\textsc{First}(e_1) \cap \textsc{First}(e_2) = \varnothing$
implies $\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(A)) = \varnothing$.

Approximation by first letters

Consider $A = e_1|e_2$.

$\textsc{First}(e_1)$, $\textsc{First}(e_2)$:
sets of possible first letters
of words in $\mathcal{L}(e_1)$ respectively $\mathcal{L}(e_2)$.

If $\mathcal{L}(e_1), \mathcal{L}(e_2)$, do not contain ε,
$\textsc{First}(e_1) \cap \textsc{First}(e_2) = \varnothing$
implies $\mathcal{L}(e_1) \cap \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(A)) = \varnothing$.

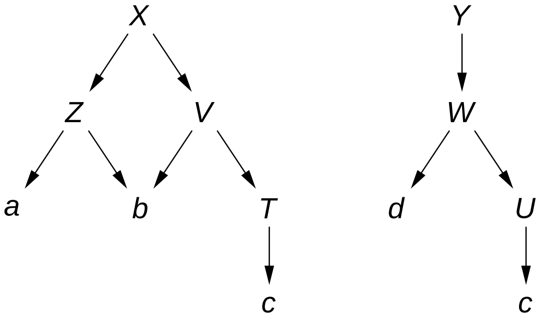
This is LL(1) for grammar without ε.
Each choice in such grammar is safe.
The two interpretations are equivalent.

Approximation by first expressions

To go beyond LL(1), we shall look at
first *expressions* rather than first *letters*.

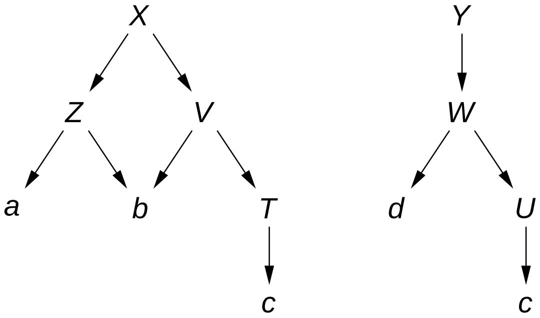
Computing FIRST

$$S = X \mid Y$$
$$X = Z \mid V$$
$$Y = W\,X$$
$$Z = a \mid b$$
$$V = b \mid T$$
$$W = d \mid U$$
$$T = c\,V$$
$$U = c\,W$$



Computing FIRST

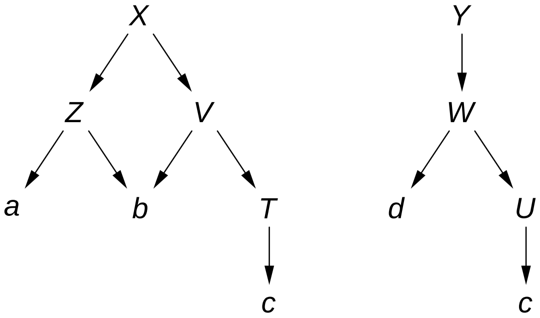
$$S = X \mid Y$$
$$X = Z \mid V$$
$$Y = W\,X$$
$$Z = a \mid b$$
$$V = b \mid T$$
$$W = d \mid U$$
$$T = c\,V$$
$$U = c\,W$$



$$\text{FIRST}(X) = \{a, b, c\}$$

Computing FIRST

$$S = X \mid Y$$
$$X = Z \mid V$$
$$Y = W\, X$$
$$Z = a \mid b$$
$$V = b \mid T$$
$$W = d \mid U$$
$$T = c\, V$$
$$U = c\, W$$



$$\text{FIRST}(X) = \{a, b, c\}$$
$$\text{FIRST}(Y) = \{c, d\}$$

Computing FIRST

$S = X \mid Y$

$X = Z \mid V$

$Y = W\,X$

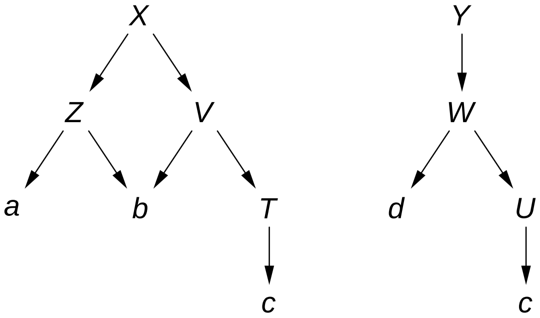
$Z = a \mid b$

$V = b \mid T$

$W = d \mid U$

$T = c\,V$

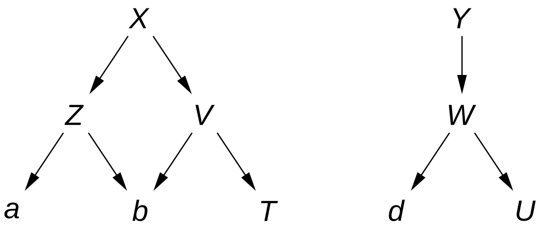
$U = c\,W$



$\text{FIRST}(X) = \{a, b, c\}$
$\text{FIRST}(Y) = \{c, d\}$
$\{a, b, c\} \cap \{c, d\} \neq \varnothing$: $S = X|Y$ is not LL(1).

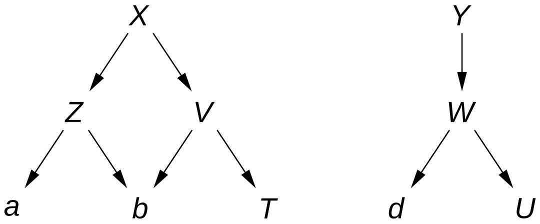
Truncated computation of FIRST

$$S = X \mid Y$$
$$X = Z \mid V$$
$$Y = W\, X$$
$$Z = a \mid b$$
$$V = b \mid T$$
$$W = d \mid U$$
$$T = c\, V$$
$$U = c\, W$$



Truncated computation of FIRST

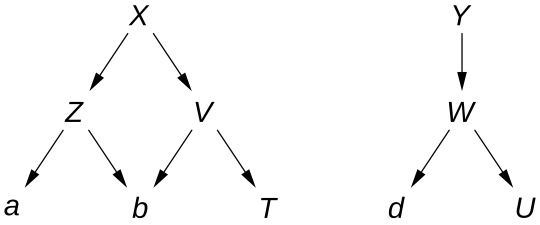
$$S = X \mid Y$$
$$X = Z \mid V$$
$$Y = W\,X$$
$$Z = a \mid b$$
$$V = b \mid T$$
$$W = d \mid U$$
$$T = c\,V$$
$$U = c\,W$$



Each word in $\mathcal{L}(X)$ has a prefix in $\{a, b\} \cup \mathcal{L}(T) = a \cup c^* b$.

Truncated computation of FIRST

$$
\begin{aligned}
S &= X \mid Y \\
X &= Z \mid V \\
Y &= W\, X \\
Z &= a \mid b \\
V &= b \mid T \\
W &= d \mid U \\
T &= c\, V \\
U &= c\, W
\end{aligned}
$$



Each word in $\mathcal{L}(X)$ has a prefix in $\{a, b\} \cup \mathcal{L}(T) = a \cup c^* b$.
Each word in $\mathcal{L}(Y)$ has a prefix in $\{d\} \cup \mathcal{L}(U) = d^* b$.

Approximation by first expressions

Each word in $\mathcal{L}(X)$ has a prefix in $a \cup c^*b$.
Each word in $\mathcal{L}(Y)$ has a prefix in d^*b.

$$\mathcal{L}(X) = (a \cup c^*b)(\ldots)$$
$$\mathcal{L}(Y) = (c^*d)(\ldots)$$

Approximation by first expressions

Each word in $\mathcal{L}(X)$ has a prefix in $a \cup c^*b$.
Each word in $\mathcal{L}(Y)$ has a prefix in d^*b.

$$\mathcal{L}(X) = (a \cup c^*b)(\ldots)$$
$$\mathcal{L}(Y) = (c^*d)(\ldots)$$

$$\mathcal{L}(X) \cap \mathrm{Pref}(\mathcal{L}(Y)\,\mathrm{Tail}(S))$$
$$= (a \cup c^*b)(\ldots) \;\cap\; \mathrm{Pref}((c^*d)(\ldots)(\ldots))$$

Approximation by first expressions

Each word in $\mathcal{L}(X)$ has a prefix in $a \cup c^*b$.
Each word in $\mathcal{L}(Y)$ has a prefix in d^*b.

$\mathcal{L}(X) = (a \cup c^*b)(\ldots)$
$\mathcal{L}(Y) = (c^*d)(\ldots)$

$$\mathcal{L}(X) \cap \text{Pref}(\mathcal{L}(Y)\,\text{Tail}(S))$$
$$= (a \cup c^*b)(\ldots) \;\cap\; \text{Pref}((c^*d)(\ldots)(\ldots))$$

No word in $a \cup c^*b$ is a prefix of word in c^*d and vice-versa.

Approximation by first expressions

Each word in $\mathcal{L}(X)$ has a prefix in $a \cup c^*b$.
Each word in $\mathcal{L}(Y)$ has a prefix in d^*b.

$\mathcal{L}(X) = (a \cup c^*b)(\dots)$
$\mathcal{L}(Y) = (c^*d)(\dots)$

$$\mathcal{L}(X) \cap \mathrm{Pref}(\mathcal{L}(Y)\,\mathrm{Tail}(S)) \\ = (a \cup c^*b)(\dots) \;\cap\; \mathrm{Pref}((c^*d)(\dots)(\dots))$$

No word in $a \cup c^*b$ is a prefix of word in c^*d and vice-versa.

The intersection is empty: $S = X|Y$ is safe.

Some terminology

X starts with a, b, or T:
"X has a, b, and T as possible first expressions".

$\{a, b, T\} \sqsubseteq X$

Some terminology

X starts with a, b, or T:
"X has a, b, and T as possible first expressions".

$\{a, b, T\} \sqsubseteq X$

No word in a, b, or T is a prefix of a word in d or U and vice-versa:
"$\{a, b, T\}$ and $\{d, U\}$ are exclusive".

$\{a, b, T\} \asymp \{d, U\}$

One can easily see that...

If $\varepsilon \notin e_1$ and $\varepsilon \notin e_2$
and there exist $\text{First}_1 \sqsubseteq e_1$, $\text{First}_2 \sqsubseteq e_2$
such that $\text{First}_1 \asymp \text{First}_2$
then $A = e_1 | e_2$ is safe.

When PEG = EBNF?

The two interpretations of an ε-free grammar are equivalent if for every Choice $A = e_1|e_2$, e_1 and e_2 have exclusive sets of first expressions.

Final remarks

- (Good news) Grammar with ε is easy to handle. This involves first expressions of Tail(A), that are obtained using the classical computation of FOLLOW.

Final remarks

- (Good news) Grammar with ε is easy to handle. This involves first expressions of Tail(A), that are obtained using the classical computation of FOLLOW.
- (Good news) The results for simple grammar are easily extended to full EBNF / PEG.

Final remarks

- (Good news) Grammar with ε is easy to handle. This involves first expressions of Tail(A), that are obtained using the classical computation of FOLLOW.
- (Good news) The results for simple grammar are easily extended to full EBNF / PEG.
- (Good news) The possible sets of first expressions are easily obtained in a mechanical way.

Final remarks

- (Good news) Grammar with ε is easy to handle. This involves first expressions of Tail(A), that are obtained using the classical computation of FOLLOW.
- (Good news) The results for simple grammar are easily extended to full EBNF / PEG.
- (Good news) The possible sets of first expressions are easily obtained in a mechanical way.
- (Bad news) Checking that they are exclusive is not easy: it is undecidable in general case (but we may hope first expressions are simple enough to be decidable.)

Final final remark

S = (aa|a)b (that is: S = Xb, X = aa|a.)

Final final remark

S = (aa|a)b (that is: S = Xb, X = aa|a.)

$\mathcal{L}(e_1) \cap \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(X)) = \mathrm{aa} \cap \mathrm{Pref}(\mathrm{ab}) = \varnothing$.

Final final remark

S = (aa|a)b   (that is: S = Xb, X = aa|a.)

$\mathcal{L}(e_1) \cap \mathrm{Pref}(\mathcal{L}(e_2)\,\mathrm{Tail}(X)) = \mathrm{aa} \cap \mathrm{Pref}(\mathrm{ab}) = \varnothing$.

X is safe. Both interpretations accept {aab,ab}.

Final final remark

S = (aa|a)b   (that is: S = Xb, X = aa|a.)

$\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(X)) = \text{aa} \cap \text{Pref}(\text{ab}) = \varnothing$.

X is safe. Both interpretations accept {aab,ab}.

Sets of first expressions in X: {aa} and {a}. Not exclusive!

Final final remark

S = (aa|a)b   (that is: S = Xb, X = aa|a.)

$\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(\text{X})) = \text{aa} \cap \text{Pref}(\text{ab}) = \varnothing$.

X is safe. Both interpretations accept {aab,ab}.

Sets of first expressions in X: {aa} and {a}. Not exclusive!

There is more to squeeze out of $\mathcal{L}(e_1) \cap \text{Pref}(\mathcal{L}(e_2)\,\text{Tail}(A))$.

That's all

Thanks for your attention!