EBNF to Ti*k*Z Converter

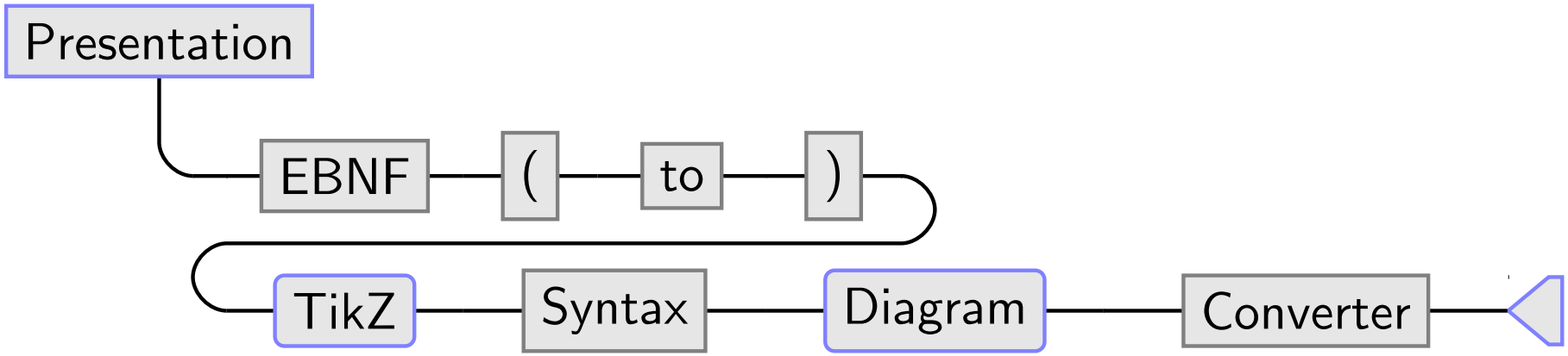
Generating syntax diagrams from EBNF with AG

Reinier Vis

Center for Software Technology, Utrecht University
http://www.cs.uu.nl/groups/ST/

November 8, 2007

Example



Outline

1. Attribute Grammars
2. EBNF
3. Calculations
4. Diagram generation
5. HTML generation
6. Limitations
7. Conclusions and future work

Attribute Grammars

Course in Software Technology.
Has anyone not followed the IPL course yet?
Course starts next monday..

Assignment in 1st year

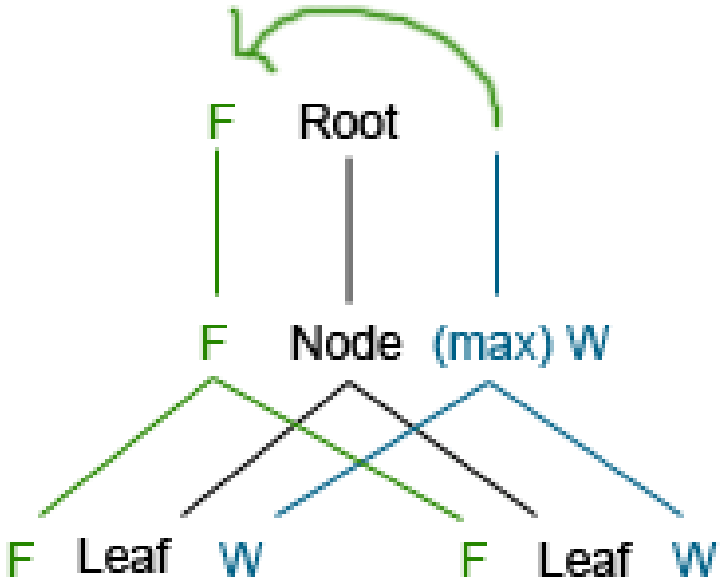
```
type Table :: [[String]]
top10 :: Table
top10 = [ [ "Nr", "Artist", "Clip" ]
        , [ "1", "Shakira", "Whenever Wherever" ]
        , [ "2", "Marco en Sita", "Lopen op het water" ]
        , [ "3", "Pink", "Get the Party Started" ]
        ]
```

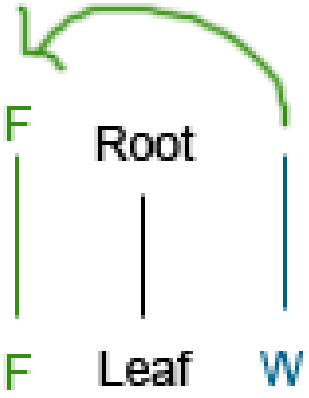
```
-----------------------------------------------
| Nr | Artist         | Clip                   |
===============================================
| 1  | Shakira        | Whenever Wherever      |
| 2  | Marco en Sita  | Lopen op het water     |
| 3  | Pink           | Get the Party Started  |
-----------------------------------------------
```

Attributing a datatype

With attribute grammars, you attribute a haskell datatype.

```
DATA Root | Root  Tree
DATA Tree | Node  Tree  Tree
          | Leaf  String
ATTR Tree [ finalWidth : Int |   | width : Int ]
```





The EBNF to TikZ Converter

- The EBNF converter is constructed by using Attribute Grammars
- Input is a grammar specification
- Output is an image in TikZ

```
\tikzstyle{terminal}   =[rectangle,draw=black]
\tikzstyle{nonterminal}=[rectangle,rounded corners=2pt]
\tikzstyle{rulename}   =[rectangle,draw=blue]
\begin{tikzpicture}
   \node at (0.0, 0.0) [rulename]    { Foo };
   \draw [thick] (0.0,0.0) -- (0.0,-2.0);
   \node at (0.0,-2.0) [nonterminal] { Bar };
\end{tikzpicture}
```

Extended Backus Naur Form

Used to specify languages Structured computer languages in particular
Example:

```
IfThenElse ::= "if" CondExpr "then" Expr ("else" Expr)? .
CondExpr   ::= Expr "==" Expr
             | ...
             | Expr ">" Expr .
Expr       ::= Integer "+" Integer
             | IfThenElse
             | True
             | False .
```

Extended Backus Naur Form

Used to specify languages Structured computer languages in particular
Example:

```
IfThenElse ::= "if" CondExpr "then" Expr ("else" Expr)? .
CondExpr   ::= Expr "==" Expr
             | ...
             | Expr ">" Expr .
Expr       ::= Integer "+" Integer
             | IfThenElse
             | True
             | False .
```

? = Optional
* = Many
+ = Many1

Extended Backus Naur Form (2)

Grammar description of EBNF in EBNF:

```
Grammar      ::= Rule* .
Rule         ::= Identifier "::=" Disjunction "." .
Disjunction  ::= Alternation ( "|" Disjunction )* .
Alternation  ::= Conjunction ( "//" Conjunction )* .
Conjunction  ::= BasicExpr ( " " Conjunction )* .
BasicExpr    ::= "(" Disjunction ")"  Qualifier?
               |     Identifier       Qualifier?
               |     Terminal         Qualifier? .
Terminal     ::= "'" Identifier "'"
               | '"' Identifier '"' .
Qualifier    ::= '*' | '+' | '?' .
```

Extra productions in EBNF grammar

I would like to accept more centences as a grammar. To have influence in the directions in which the image is drawn:

```
Disjunction  ::= Alternation ( OrSeparator  Disjunction )* .
Alternation  ::= Conjunction ( AltSeparator Conjunction )* .
Conjunction  ::= BasicExpr   ( AndSeparator Conjunction )* .
OrSeparator  ::= "|"       -- Below eachother
               | "<|>"     -- Besides eachother
               | "<<|>"    -- Below, left aligned
               | "<|>>" .  -- Below, right aligned
AltSeparator ::= "\\"      -- Below
               | "//" .    -- Above
AndSeparator ::= " "       -- Beside eachother
               | "<&>"     -- Below eachother
               | "<<&>"    -- Below, left aligned
               | "<&>>" .  -- Below, right aligned
```

Extra productions in EBNF grammar

A sentence that adheres to the following grammar production

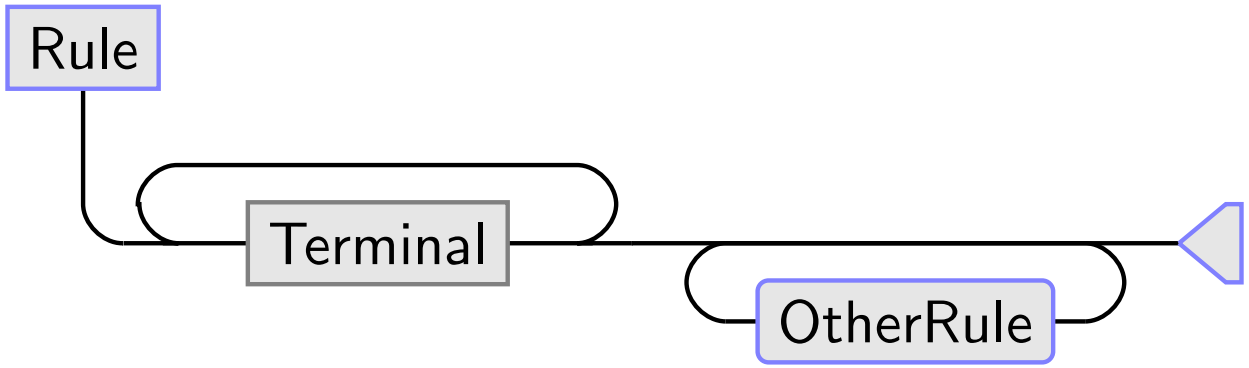
```
Disjunction  ::= Alternation ( OrSeparator  Disjunction )*
```

parses to one of the following Disjunctions:

```
data Disjunction = Single     Alternation
                 | Below      Alternation Disjunction
                 | Besides    Alternation Disjunction
                 | BelowLeft  Alternation Disjunction
                 | BelowRight Alternation Disjunction
```

Attribute the Datatypes

- Thus far, we have seen the grammar of an EBNF grammar.
- All sentences are parsed to a Haskell datatype with the UU Parser Combinator Library.
- Next, we are going to create diagrams like the one below:



Attribute the Datatypes

- The constructors and fields in the datatype are labeled, so we can reference them in the AG:

```
...
DATA Conjunction
  | Besides        basic:BasicExpr   conj:Conjunction
  | Below          basic:BasicExpr   conj:Conjunction
  | BelowLeft      basic:BasicExpr   conj:Conjunction
  | BelowRight     basic:BasicExpr   conj:Conjunction
  | Single         basic:BasicExpr
DATA BasicExpr
  | Dis          disj:Disjunctions card:{Maybe OccSymbol}
  | NonTerm      name:String       card:{Maybe OccSymbol}
  | Term         term:Terminal     card:{Maybe OccSymbol}
DATA Terminal
  | Terminal       name:String
```

Determine widths and heights

- Calculate all widths and heights of the datatypes

```
ATTR Terminal BasicExpr Conjunction Alternation Disjunction
     Rule Grammar [ | | width : Double  height : Double ]
```

Determine widths and heights

- Calculate all widths and heights of the datatypes

```
ATTR Terminal BasicExpr Conjunction Alternation Disjunction
     Rule Grammar [ | | width : Double  height : Double ]

SEM Terminal
  | Terminal lhs.width  = getStringBoxWidth @name
             lhs.height = 0.5
```

Determine widths and heights

- Calculate all widths and heights of the datatypes

```
ATTR Terminal BasicExpr Conjunction Alternation Disjunction
     Rule Grammar [ | | width : Double  height : Double ]

SEM Terminal
  | Terminal lhs.width  = getStringBoxWidth @name
             lhs.height = 0.5

SEM BasicExpr
  | Terminal lhs.width = case @card of
                           Just Question -> @term.width + 1
                           Just _        -> @term.width + 0.5
                           Nothing       -> @term.width
             lhs.height = case @card of
                            Just _  -> @term.height + 0.25
                            Nothing -> @term.height
```

Determine Left-top, entry- and exit-point

- We can calculate for every production its absolute placement

```
type Point = (Double, Double)
ATTR Terminal BasicExpr Conjunction Alternation
     Disjunction Rule Grammar
     [ leftTop : Point | | startPoint : Point
                           endPoint : Point ]

-- an example
SEM Root
  | Root  grammar.leftTop = (0,0)
SEM Conjunction
  | Below alt.leftTop  = @lhs.leftTop
          conj.leftTop = let (x,y) = @lhs.leftTop
                         in (x, y + @alt.height + 0.5 )
```

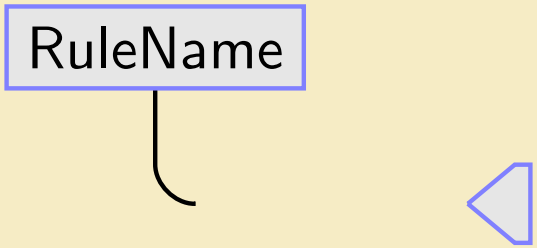
Synthesizing the image

- All fixed widths, heights, startpoints and endpoints are calculated
- Every constructor in the datatype is responsible to generate its own picture representation
- Image is combined in a synthesized way
- Ti*k*Z image is of type PP_Doc

```
type TikzPicture = PP_Doc
ATTR Terminal BasicExpr Conjunction Alternation
     Disjunction Rule Grammar
     [ | | picture : TikzPicture ]
```

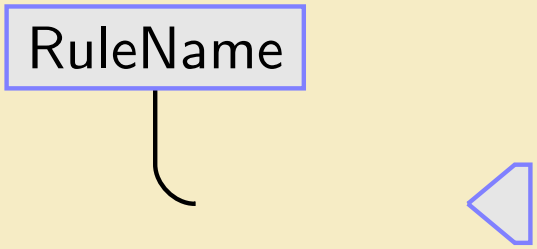
Rule, terminal & nonterminal

RuleName ::= .

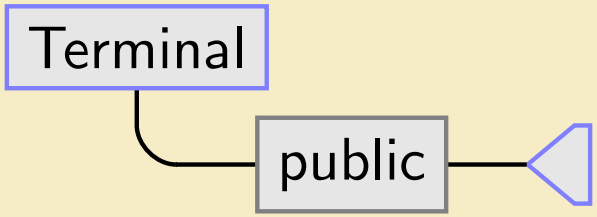


Rule, terminal & nonterminal

RuleName ::= .

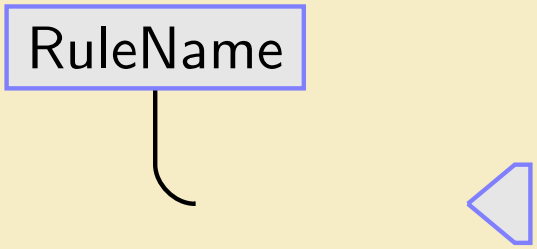


Terminal ::= "public" .

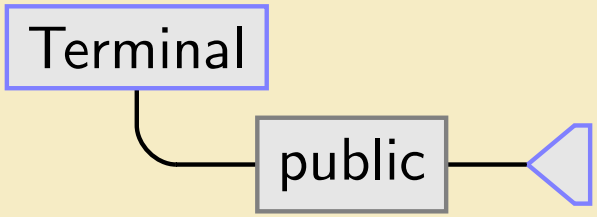


Rule, terminal & nonterminal

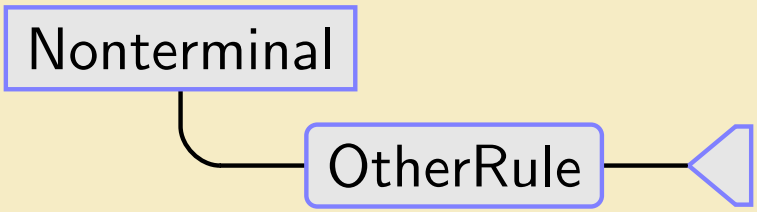
RuleName ::= .



Terminal ::= "public" .

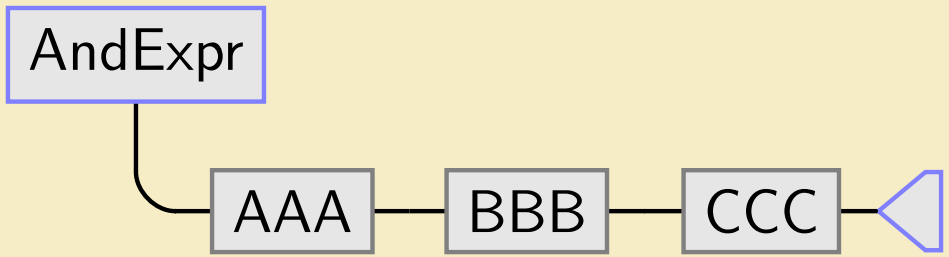


NonTerminal ::= OtherRule .



Conjunctions

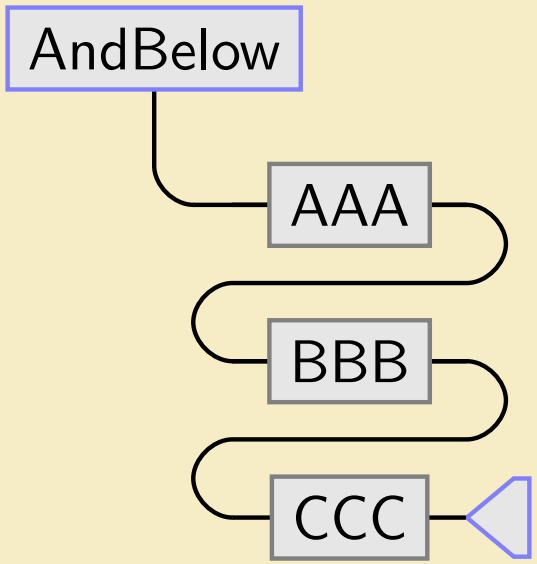
Conjunction ::= "AAA" "BBB" "CCC" .



- Extra combinators were introduced to determine the direction of the drawing, and the alignment
- < & > = Below each-other
- << & > = Below each-other, left aligned
- < & >> = Below each-other, right aligned

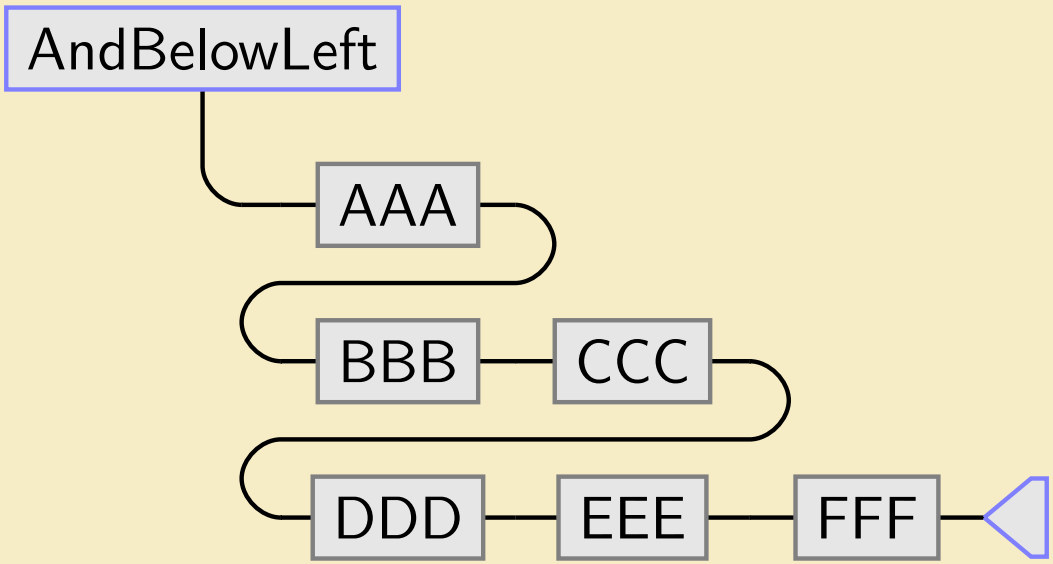
Conjunctions below each-other

Conjunction ::= "AAA" < & > "BBB" < & > "CCC" .



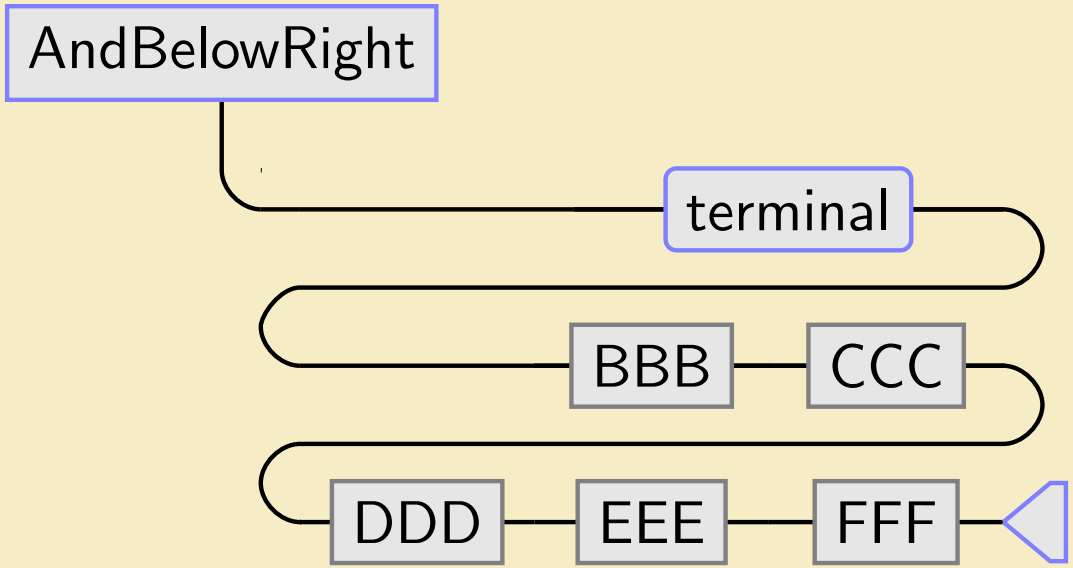
Conjunctions left- aligned

Conjunction ::= "AAA" << & > "BBB" << & > "CCC" .



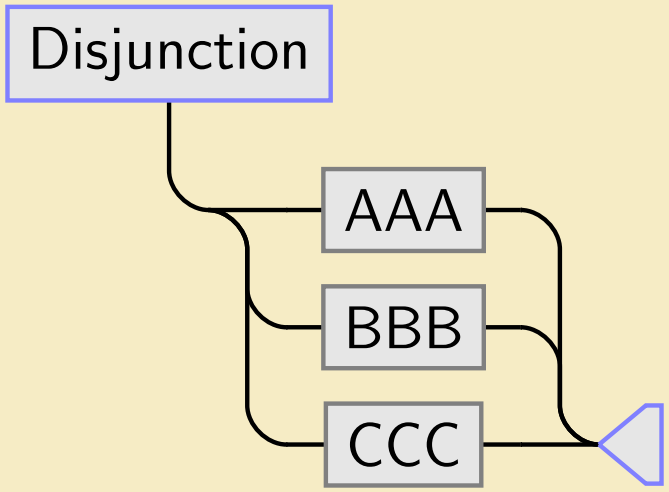
Conjunctions right- aligned

Conjunction ::= "AAA" < & >> "BBB" < & >> "CCC" .



Disjunctions

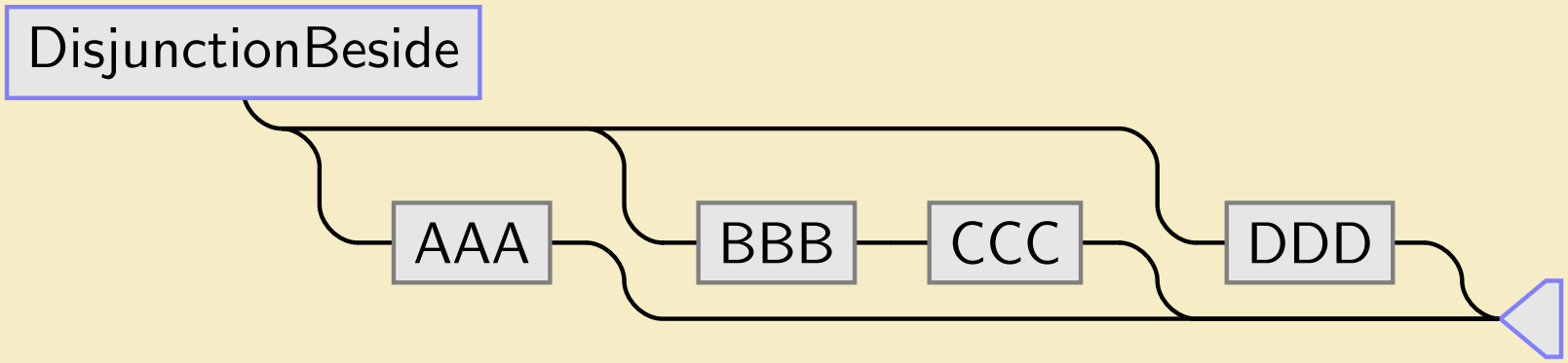
Disjunction ::= "AAA" | "BBB" | "CCC" .



- Extra combinators were introduced to determine the direction of the drawing, and the alignment
- $< | >$ = Beside each-other
- $<< | >$ = Below each-other, left aligned
- $< | >>$ = Below each-other, right aligned

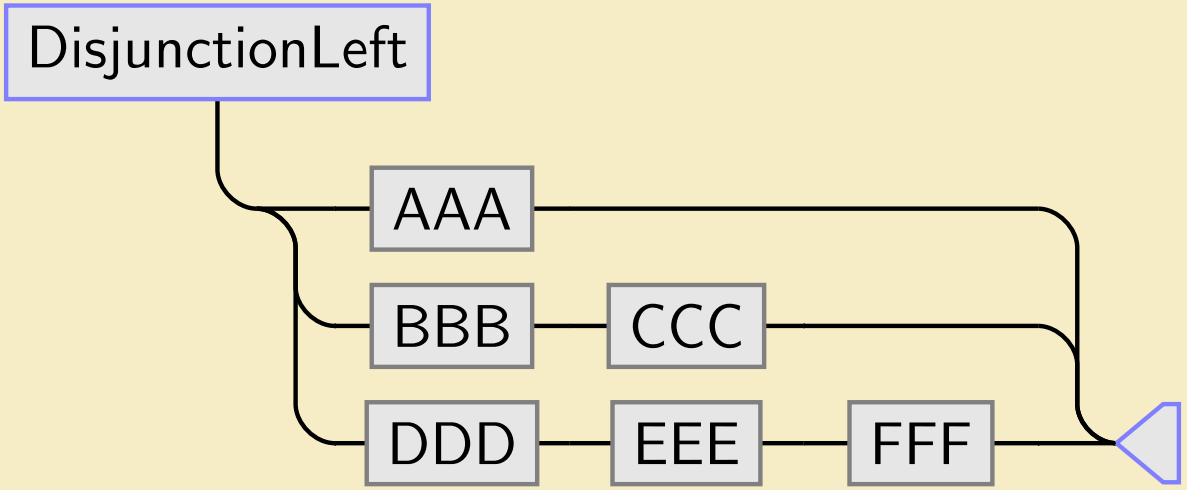
Disjunction beside each-other

Disjunction ::= "AAA" < | > "BBB" "CCC" < | > "DDD" .



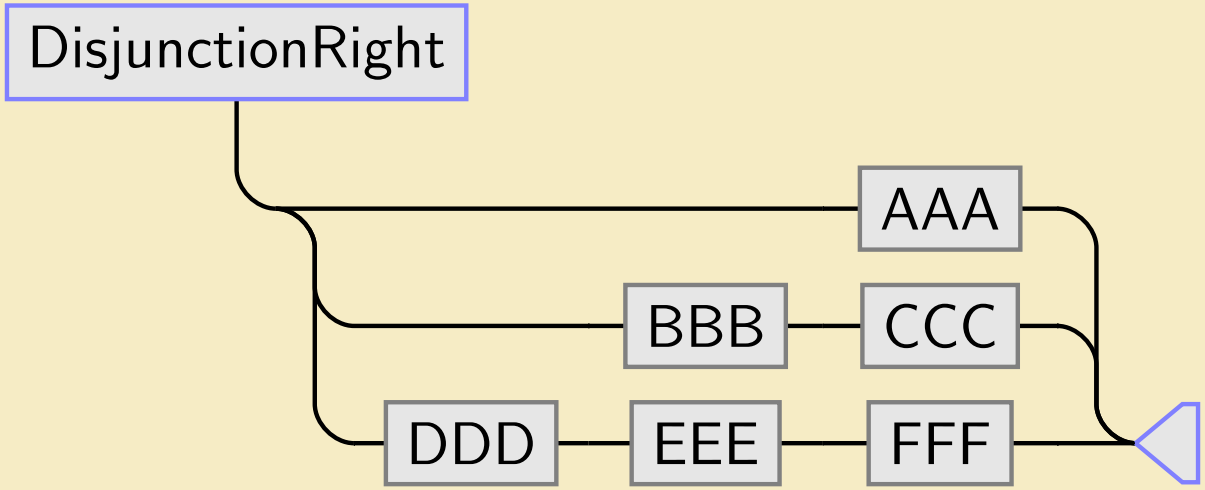
Disjunction left- aligned

Disjunction ::= "AAA" << | > "BBB" << | > "CCC" .



Disjunction right- aligned

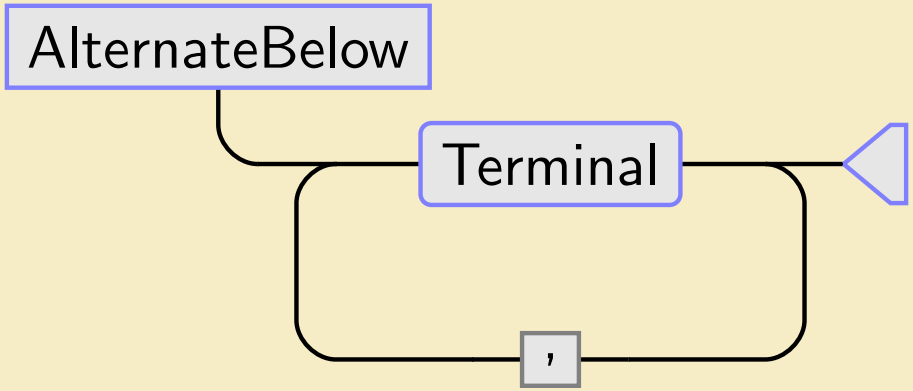
Disjunction ::= "AAA" <|>> "BBB" <|>> "CCC" .



Alternation

- An alternation can be used to draw an 'X' separated list
- Two combinators to influence appearance
  - \\ = Alternate below
  - // = Alternate above

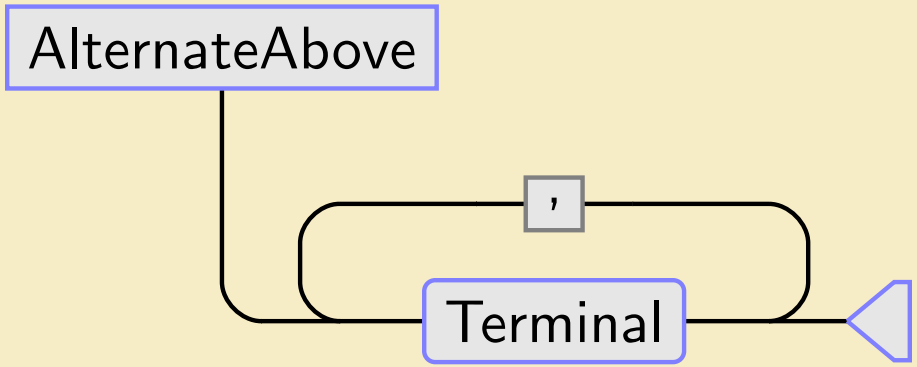
AlternateBelow ::= "Terminal" \\ ',' .



Alternation

- An alternation can be used to draw an 'X' separated list
- Two combinators to influence appearance
  - \\ = Alternate below
  - // = Alternate above

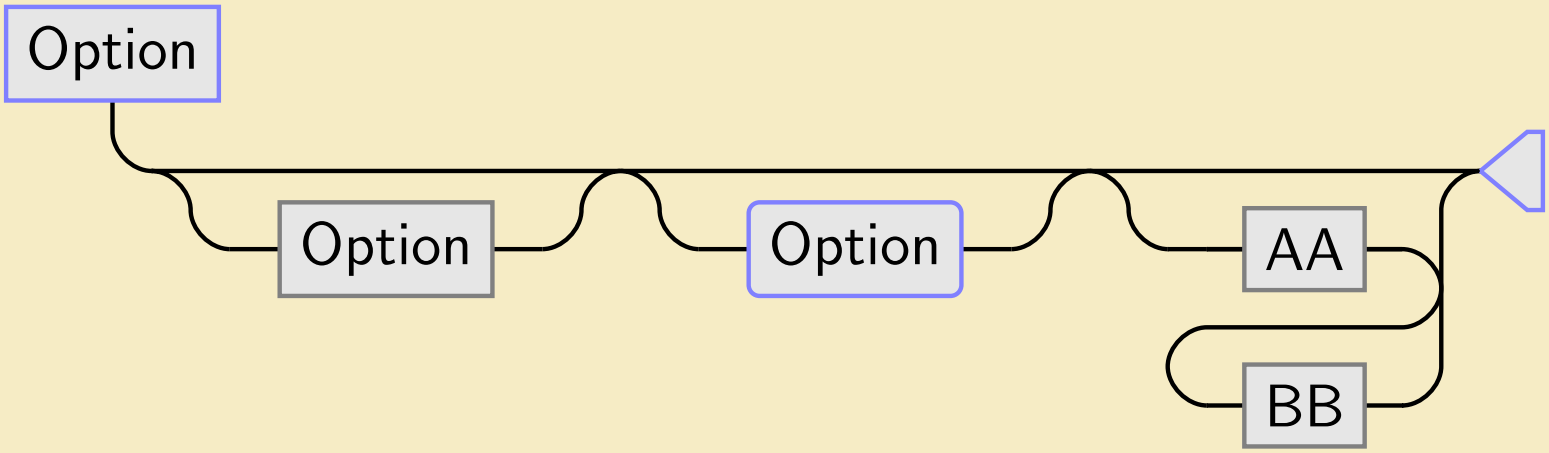
AlternateAbove ::= "Terminal" // ',' .



Occurrences

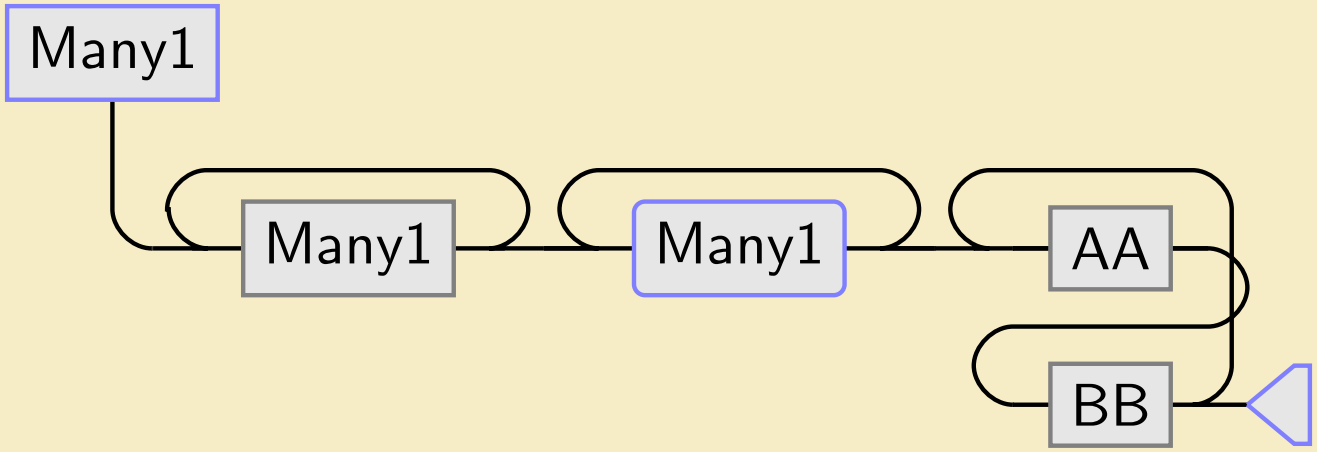
- Productions can be specified to be optional, appear 0 or more times, or 1 or more times

Option := "Option"? Option? ("AA" < & > "BB")?.



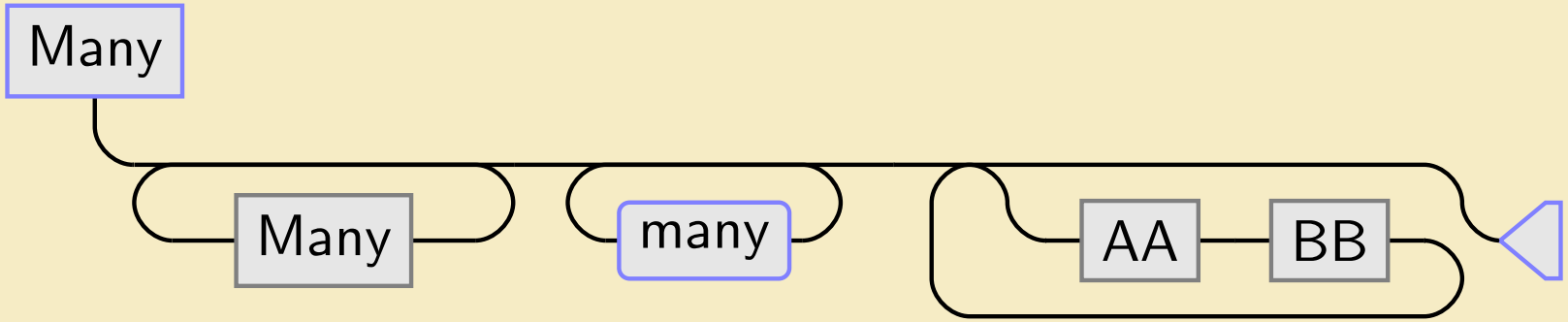
1 or more

Many1 := "Many1"+ Many1+ ("AA" < & > "BB")+.



0 or more

Many := "Many"* Many* ("AA" "BB")*.



Generate a hyperlinked grammar

- Idea posted on the Haskell mailing list by Peter Hercek
- Very easy to implement

```
type Html = PP_Doc
ATTR Root Grammar Rule Disjunction Alternation
     Conjunction BasicExpr Terminal [ | | html : Html ]
ATTR Disjunction [ nested : Bool | | ]
```

Limitations

- TikZ images included in LaTeX as text
  - Special characters can screw up the image
  - Examples: &, \
- Parsing nested productions can increase processing time drastically
- No command-line compile options
  - Only enter one ebnf grammar at a time
  - All outputs are generated, no option to specify one
- Images do not scale automatically to page width-height

Conclusions & Future Work

- Inline rules
  - Rules with a "=" instead of "::=" can be inlined everywhere the rule is used
- Use the EBNF converter to generate parsers for you
  - Specify which tokens can be deleted from the result (like braces/parentheses/etc)
  - Let the converter infer a datatype for you
  - Generate parsers that construct the datatypes from input sentences

Conclusion:

The EBNF Converter is a nice tool to process grammars and generate all kinds of output.