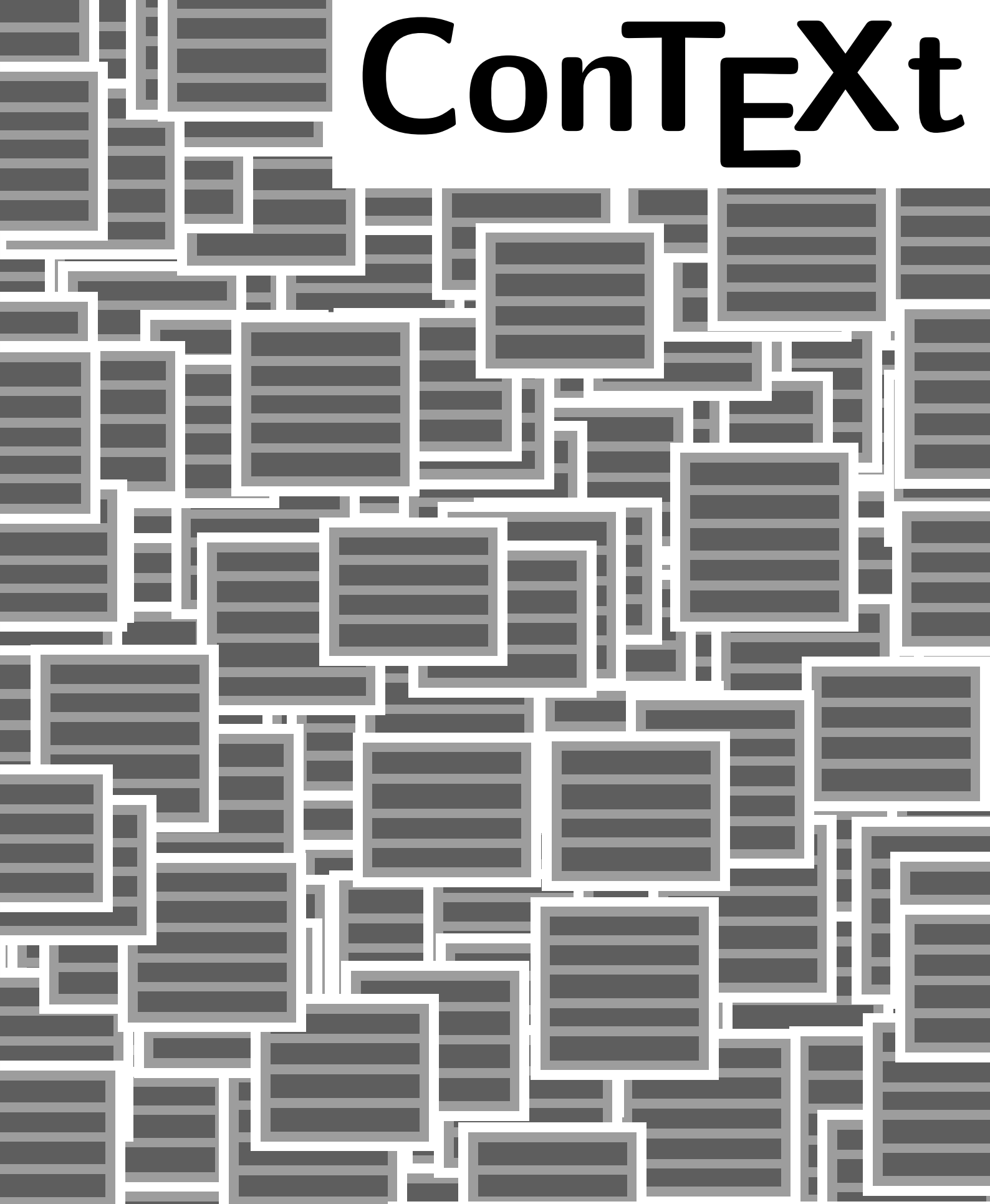

title : ConTEXt BNF Grammar Module
subtitle : Grammars
author : Nikolai Weibull
date : March 25, 2005

```
\writestatus{loading}{BNF Macros / Initialization}

\unprotect
```

We define a new system variable for our settings:

```
\definesystemvariable{bnf}
```

We need some constants for the multi–lingual interface,

```
\startconstants     english              dutch
     terminalstart: terminalstart        terminalstart
      terminalstop: terminalstop         terminalstop
  nonterminalstart: nonterminalstart     nonterminalstart
   nonterminalstop: nonterminalstop      nonterminalstop
                is: is                   worden
\stopconstants
```

and while we're at it, lets define some variables.

```
\startvariables     english              dutch
 bnfgrammar: bnfgrammar           bnfspraakleer
       bnfgrammars: bnfgrammars          bnfspraakleer
\stopvariables
```

Finally, we want the commands to be multi–lingually accessible, so we set that up as well:

```
\startcommands      english              dutch
   setupbnfgrammar: setupbnfgrammar      stelbnfspraakleer
   startbnfgrammar: startbnfgrammar      startbnfspraakleer
    stopbnfgrammar: stopbnfgrammar       startbnfspraakleer
\stopcommands
```

\startbnfg.. \stopbnfgr..

Now to the interesting parts, those that are actually useful to the outside world. First we have the `\startbnfgrammar` and `\stopbnfgrammar` pairs, which are of course used to delimit BNF grammars. We would like to define `\startbnfgrammar` as `\def\startbnfgrammar[#1]`, but a bug in ConTeXt prevents us from doing this, as the first character in the grammar may be active, for example `<`, but while checking for the presence of `[`, it gets ruined. A way around it is of course to require that the user pass an empty `[]` pair, and we will use this method at the moment.

```
\def\complexstartbnfgrammar[#1]%
  {\endgraf\nobreak\medskip
   \begingroup
   \setupbnfgrammar[#1]%
   \chardef\bnfsinglequote=‘’
   \defineactivecharacter : {\@@bnfis}
   \defineactivecharacter | {\@@bnfoption}
   \defineactivecharacter " %
     {\thinspace\bgroup\@@bnfterminalstart\setupinlineverbatim%
      \defineactivecharacter " {\@@bnfterminalstop\egroup\thinspace}}
   \defineactivecharacter ’ %
     {\thinspace\bgroup\@@bnfterminalstart\setupinlineverbatim%
      \defineactivecharacter ’ {\@@bnfterminalstop\egroup\thinspace}}
   \catcode‘<=13
   \let\par=\bnfgrammarline
   \obeylines}
```

```
\def\stopbnfgrammar{\medbreak\noindent}

\definecomplexorsimpleempty\startbnfgrammar
```

\<> \bnfgramma.. We need a couple more macros to deal with the interior of a BNF grammar. **\<>** is used for non–terminals, and **\bnfgrammarrule** is used later on in **\bnfgrammarswitch** for continuing a line.

```
\def\<#1>{\leavevmode\hbox{\@@bnfnonterminalstart#1\/\@@bnfnonterminalstop}}

\bgroup
  \catcode`<=13
  \global\let<=\<
  \gdef\bnfgrammarrule<#1>{\endgraf\indent\<#1>}
\egroup
```

\bnfgramma.. \bnfgramma.. \bnfgramma.. These macros deal with the ending of a line in a grammar. **\bnfgrammarline** is called whenever a new line begins, and invokes **\bnfgrammarswitch** to determine what to do next. If the next token is **\<**, we will call upon **\bnfgrammarrule** to deal with the new rule. If it is **\stopbnfgrammar**, we end the top–level group, and let it process **\stopbnfgrammar** afterwards. Otherwise we invoke **\bnfgrammarcont**, which will end the line and add some indentation to the continuing line.

```
\def\bnfgrammarline{\futurelet\next\bnfgrammarswitch}
\def\bnfgrammarswitch%
  {\ifx\next\<
     \let\next=\bnfgrammarrule
   \else\ifx\next\stopbnfgrammar
     \let\next=\endgroup
   \else
     \let\next=\bnfgrammarcont
   \fi\fi
   \next}
\def\bnfgrammarcont{\hfil\break\indent\qquad}
```

\setupbnfg.. We want to allow our users to change the way the BNF grammars are typeset, so we define a setup command for them to use.

It allows you to define the start and stop sequence for terminals and non–terminals, as well as colons (lhs / rhs separator) and vertical bars (alternative), and commas. This has been multi–lingualized above, so choose your language.

```
\def\dosetupbnfgrammar[#1]%
  {\getparameters[\??bnf][#1]}

\def\setupbnfgrammar%
  {\dosingleargument\dosetupbnfgrammar}

\setupbnfgrammar
  [\c!terminalstart=\tttf,
   \c!terminalstop=,
   \c!nonterminalstart=\mathematics{\langle},
   \c!nonterminalstop=\mathematics{\rangle},
   \c!is={ \mathematics{\longrightarrow}},
   \c!option=\mathematics{\vert}]
```

\BNF We also define a useful abbreviation to be used for header texts and labels.

```
16  \logo[BNF]{bnf}
```

And we use it here:

```
17  \setupheadtext[\s!en][\v!bnfgrammar=\BNF\ Grammar]
    \setupheadtext[\s!en][\v!bnfgrammars=\BNF\ Grammars]
    \setuplabeltext[\s!en][\v!bnfgrammar=\BNF\ Grammar ]
```

Finally we define a float to be use with BNF grammars, so that we can finish off with something like this:

```
\placebnfgrammar
  [][]
  {An example of a placed grammar.}
  {\startbnfgrammar[]
     <exp>: <num> | <num> "+" <num>
     <num>: "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
   \stopbnfgrammar}
```

⟨exp⟩ ⟶ ⟨num⟩ | ⟨num⟩ + ⟨num⟩

⟨num⟩ ⟶ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

BNF Grammar 1 An example of a placed grammar.

which looks kind of nice.

```
18  \definefloat
      [\v!bnfgrammar]
      [\v!bnfgrammars]
```

```
19  \protect \endinput
```

Grammars