The ottalt package

Jesse A. Tov
tov@eecs.harvard.edu

This document corresponds to ottalt v0.11, dated 2013/03/14.

Contents

1 Introduction

The purpose of this package is to provide an alternate style for laying out grammars and inference rules generated by Ott (`http://www.cl.cam.ac.uk/~pes20/ott/`). The goal is to produce high-quality output suitable for inclusion in articles or technical reports, while making it easier to access various commands that Ott

defines in its output. (This package makes no attempt to compress Ott-generated grammars to fit in space-limited situations, however.)

To use ottalt, one must generate standalone LaTeX definitions from an `.ott` source file. To make these suitable for inclusion from another LaTeX document, pass the `-tex_wrap false` option to Ott. For details, I've included a sample Makefile in §A. Then, supposing you've generated Ott definitions in ⟨*stem*⟩`.ott.tex`, load them in your LaTeX document with

`\inputott{`⟨*stem*⟩`.ott}`

Optionally, if you supply the `-tex_name_prefix` ⟨*prefix*⟩ option to Ott, you should also tell `\inputott` about the prefix, since that will tell it the names of some commands to redefine: `\inputott[`⟨*prefix*⟩`]{`⟨*stem*⟩`.ott}`.

Then, the easiest way to print Ott-defined grammars and rules is with the `\drules` and `\nonterms` commands, each of which takes a comma separated list of Ott rule or nonterminal names. For example, suppose your Ott file contains nonterminals `t` and `e`, like this:

```
1 ⟨*ott⟩
2 grammar
3 t {{ tex \nonterm t }} :: tp_ ::= {{ com types }}
4   | a             ::   :: var  {{ com type variable }}
5   | t1 -> t2      ::   :: arr  {{ com function }}
6   | all a . t     ::   :: all  {{ com univeral }}
7   | { t' / a } t :: M :: subst
8   | ( t )         :: S :: parens
9
10 e {{ tex \nonterm e }} :: tm_ ::= {{ com terms }}
11   | x             ::   :: var  {{ com variable }}
12   | \ x : t . e   ::   :: abs  {{ com abstraction }}
13   | e1 e2         ::   :: app  {{ com application }}
14   | /\ a . e      ::   :: tabs {{ com type abstraction }}
15   | e [ t ]       ::   :: tapp {{ com type application }}
16   | { t / a } e   :: M :: tsubst
17   | ( e )         :: S :: parens
18
19 G {{ tex \nonterm G }} :: env_ ::= {{ com typing contexts }}
20   | empty         ::   :: nil  {{ com empty }}
21   | G , x : t     ::   :: var  {{ com variable assumption }}
22   | G, a          ::   :: tvar {{ com type variable assumption }}
23 ⟨/ott⟩
```

Then you can print grammars for `t` and `e` like this:

`\nonterms{t,e}`

We could easily add `G` as well, or we could print it elsewhere. The result appears in figure 1.

Similarly, we can define inference rules in Ott and then easily include them in a document. Suppose we define these type rules using Ott:

τ	::=		types
	|	α	type variable
	|	$\tau_1 \to \tau_2$	function
	|	$\forall\, \alpha.\tau$	univeral
e	::=		terms
	|	x	variable
	|	$\lambda x : \tau.e$	abstraction
	|	$e_1\, e_2$	application
	|	$\Lambda \alpha.e$	type abstraction
	|	$e[\tau]$	type application

Figure 1: An example of `\nonterms`

```
⟨*ott⟩
defns Jtype :: '' ::=
defn G |- e : t :: :: typing :: T_ {{ com term typing }} by

x : t in G
---------- :: Var
G |- x : t

G, x:t1 |- e : t2
----------------------- :: Abs
G |- \x:t1.e : t1 -> t2

G |- e1 : t' -> t
G |- e2 : t'
----------------- :: App
G |- e1 e2 : t

G, a |- e : t
-------------------- :: TAbs
G |- /\a.e : all a.t

G |- e : all a.t'
G |- t ok
------------------- :: TApp
G |- e[t] : {t/a}t'
⟨/ott⟩
```

This defines macros such as `\ottdruleTXXVar` for typesetting rule T-Var. We can display some or all of them using `\drules`. For example, we may want to show only three of them:

```
\drules[T]{$[[G |- e : t]]$}
          {term typing, selected rules}
          {Var,Abs,App}
```

$\boxed{\Gamma \vdash e : \tau}$ *(term typing, selected rules)*

$$\text{T-Var}\ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \text{T-Abs}\ \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1 . e : \tau_1 \to \tau_2} \qquad \text{T-App}\ \frac{\Gamma \vdash e_1 : \tau' \to \tau \qquad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\, e_2 : \tau}$$

Figure 2: An example of `\drules`

The result appears in figure 2.[1]

The examples above use this package's facility for managing nonterminal symbols. For example, the Ott code gives a `tex` morphism of `\nonterm t` for nonterminal `t`. This is set up in two stages. First, we declare a new group of nonterminals using `\newNTclass`, naming the new class `nonterm`:

```
\newNTclass{nonterm}
```

This results in the definition of three new macros: `\newnonterm` and `\newnonterms` for declaring new nonterminal symbols belonging to the `nonterm` class, and `\nonterm` for referring to them. We then declare five nonterminal symbols in the new class:

```
\newnonterm t \tau
\newnonterm e e
\newnonterm a \alpha
\newnonterm x x
\newnonterm G \Gamma
```

This associates a short name, such as `t`, with the actual code for printing that nonterminal, in this case `\tau`. Then, we can refer to it as `\nonterm t`. This makes it convenient to configure how nonterminal symbols appear in Ott output. It includes support for switching between two versions of the nonterminals, which is useful, for example, to have color and black-and-white modes.

1.1 The Line Break Hack

The "line break hack" is a method for specifying line breaks between premises in Ott inference rules so that the line breaks appear in in the final LaTeX document. The idea is to emit a TeX control sequence, `\ottlinebreakhack`, after any premises where we want to force a break. Then, the definition of `\ottpremise` from this package looks for that and uses it to force a break.

One way to do this is to add a production the Ott's `formula` nonterminal, which allows specifying additional formulas that may be used as premises in judgments. I add a production that looks like this:

[1] In the sample code, we use the double bracket notation that requires passing the LaTeX document through Ott as a preprocessor. This documentation is not actually processed in that way, but the `Makefile` in §A gives a rule for doing that preprocessing.

```
| formula \\\\ :: :: lbh {{ tex [[formula]] \ottlinebreakhack }}
```

The effect of this is to allow writing `\\\\` after any premise in an Ott inference rule (on the same line). This yields a formula that translates to LaTeX just like the original formula, but with `\ottlinebreakhack` after it.

Interpretation of the line break hack is *not enabled by default*. To turn it on, you need to pass the `lineBreakHack` option when loading the package. See §2.1 for information on the other two options related to line breaking.

1.2 Requirements

The ottalt package depends on five other packages. Three are part of the standard LaTeX distribution, but two others must be downloaded.

ifthen Standard LaTeX package

keyval Standard LaTeX package

listproc Non-standard, available at http://www.eecs.harvard.edu/~tov/code/latex/

mathpartir Non-standard, available at http://cristal.inria.fr/~remy/latex/

Additionally, ottalt will cooperate with the hyperref package, if it is loaded, to provide hyperlinks from inference rule references to their definitions. There is also an option to load the supertabular package and use it for presenting grammars.

2 Command Reference

2.1 Loading & Options

`\usepackage[`⟨*ottalt-options*⟩`]{ottalt}`

The package supports these options:

`alternateNonterms`

> This tells the package to use the alternate styling for nonterminal symbols defined using the system described in §2.4.1. When a new nonterminal is declared with the `\new`⟨*class*⟩`s{`⟨*name*⟩`}{`⟨*text*⟩`}{`⟨*alt-text*⟩`}`, then it normally prints as ⟨*text*⟩, but this option causes it to print as ⟨*alt-text*⟩ instead.

`supertabular`

> Load the supertabular package and use the `supertabular` environment for grammars.

`implicitPremiseBreaks`, `lineBreakHack`, `implicitLineBreakHack`

These three options specify different linebreaking behavior in inference rules, all of which involve the "line break hack" (§1.1). If option `implicitPremiseBreaks` is enabled, then any line breaks specified by the line break hack are ignored, and all `\inferrule*` premises are separated by `\\`, which allows `\inferrule*` to decide where to break lines between premises. If option `lineBreakHack` is enabled, then there are no implicit breaks: requested line breaks are passed to `\inferrule*` as `\\\\`, which forces a break, and other premises are separated by `\and`, which prohibits a break. Finally, option `implicitLineBreakHack` respected requested line breaks by passing them as `\\\\`, but allows automatic line breaks elsewhere by using `\\` to separate remaining premises.

`\inputott [`⟨*prefix*⟩`] {`⟨*ott-tex-src*⟩`}`

Load the Ott definitions from `{`⟨*ott-tex-src*⟩`}` and then redefine several Ott macros. The default names for Ott macros all begin with `ott`, but if the prefix has been changed by providing the `-tex_name_prefix` ⟨*prefix*⟩ option to Ott, then passing the same prefix to `\inputott` will cause it to redefine the correct macros.

`\renewottcommands [`⟨*prefix*⟩`]`

Redefine the Ott commands (using ⟨*prefix*⟩ if supplied or `ott` by default). This is ordinarily called by `\inputott`, but can be used directly if the Ott definitions are loaded some other way first.

2.2 Rule References

This package provides a system for hyperlinking rule references to rule definitions. The links will be generated only if the hyperref package is loaded.

`\ranchor{`⟨*rule-name*⟩`}`

Print ⟨*rule-name*⟩ as a rule name and create a hypertarget with that name.

`\rref[*]{`⟨*rule-name*⟩$_1$`,...,`⟨*rule-name*⟩$_k$`}`
`\Rref[*]{`⟨*rule-name*⟩$_1$`,...,`⟨*rule-name*⟩$_k$`}`

Print ⟨*rule-name*⟩$_1$ through ⟨*rule-name*⟩$_k$ as a nice list, with hyperlinks to the corresponding `\ranchor`s if hyperref is loaded. For example,

```
\rref{T-Var,T-Abs,T-App}
```

prints

rules T-Var, T-Abs, and T-App

The uppercase variant `\Rref` is for the beginning as a sentence, as it capitalizes the introduction word ("Rule[s]") by default. The starred variants `\rref*` and `\Rref*`

do not print the introduction word, but merely a bare list of rule references. The style of the references and text used to introduce them is configurable using by renewing several commands:

command	default value
\rrefruletext	rule
\Rrefruletext	\expandafter\MakeUppercase\rrefruletext
\rrefrulestext	\rrefruletext s
\Rrefrulestext	\Rrefruletext s
\ranchorstyle	\rrefstyle
\rrefstyle	\normalfont\scshape
\wraparoundrref	\relax

Commands `\rrefruletext` and `\Rrefruletext` are used to introduce single rules made by `\rref` and `\Rref`, respectively. Commands `\rrefrulestext` and `\Rrefrulestext` are the plural versions, used to introduce lists rules made by `\rref` and `\Rref`. To change the style of rule anchors and rule references, redefine `\ranchorstyle` and `\rrefstyle`. Finally, `\wraparoundrref` is wrapped around each rule reference, which can be used to prevent line and page breaks within the rule name, among other things. (Older versions of hyperref crash when a link breaks between pages, so to prevent this, use `\renewcommand\wrapparoundrref{\mbox}`.)

To redefine how lists are separated (as in "," and "and"), see the documentation for the listproc package. The relevant commands to redefine are `\FormatListSepTwo`, `\FormatListSepMore`, and `\FormatListSepLast`.

2.3 Including Inference Rules

`\drule [`⟨*opt*⟩`] {`⟨*rule-name*⟩`}`

Include the Ott rule with name ⟨*rule-name*⟩. This translates a name like `T-Var` to the name that Ott uses for the rule, `\`⟨*prefix*⟩`druleTXXVar`. If the optional argument is given, then that is passed to the Ott rule command; otherwise we pass the empty argument.

This macro is primarily intended to be used in the `drulepar` environment. Multiple uses of `\drule` will try to space themselves using `\and`, which will be issued by all but the first rule in a scope. Thus, a group of rules should appear in a TEX group. To suppress automatic spacing altogether, wrap each rule in its own group.

\begin{drulepar} [⟨*rule-prefix*⟩] {⟨*judgment*⟩} {⟨*descr*⟩}   ⟨*math*⟩ \end{drulepar}
\begin{drulepar*} [⟨*rule-prefix*⟩] {⟨*judgments*⟩} {⟨*descr*⟩}   ⟨*math*⟩ \end{drulepar*}

Create a block for setting rules. Argument ⟨*judgment*⟩ will be set as a heading in a framed box on the left; or for the starred variant, ⟨*judgments*⟩ should be a comma-separated list, each of which will be set in its own framed box. The description, ⟨*descr*⟩, is right aligned in text mode. Then the contents of the environment are set as a math paragraph using mathpartir's `mathparpagebreakable` environment.

If the optional ⟨*rule-prefix*⟩ is specified, then any uses of `\drule` within the group will have ⟨*rule-prefix*⟩`-` prepended to their rule name.

\drules [⟨*rule-prefix*⟩] {⟨*judgment*⟩} {⟨*descr*⟩} {⟨*rule-name*⟩[,...]}

Create a block with the specified rules, which are the comma-separated argument. This is essentially a shortcut for using the `drulepar` environment and `\drule` command. The above sample syntax is equivalent to

```
\begin{drulepar} [⟨rule-prefix⟩] {⟨judgment⟩} {⟨descr⟩}
  \drule{⟨rule-name⟩} [...]
\end{drulepar}
```

\begin{rulesection} [⟨*rule-prefix*⟩] {⟨*judgment*⟩} {⟨*descr*⟩}   ⟨*text*⟩ \end{rulesection}
\begin{rulesection*} [⟨*rule-prefix*⟩] {⟨*judgments*⟩} {⟨*descr*⟩}   ⟨*text*⟩ \end{rulesection*}

Create a block with ⟨*judgment*⟩ (or ⟨*judgments*⟩) and ⟨*descr*⟩ heading, like `drulepar`, but the contents of the environment are (initial) in text mode. Set the optional ⟨*rule-prefix*⟩ for its duration if specified.

2.3.1 Configuration Macros

These commands are used by the commands above, and may be redefined to change their behavior:

\begin{ottaltgrammar} [⟨*dimen*⟩]
  \nt {⟨*nonterm-name*⟩}
  ...
\end{ottaltgrammar}

Build an Ott grammar. The optional argument ⟨*dimen*⟩ specifies the default vertical spacing between grammar items. Within the grammar, the command `\nt`

is defined, which takes an Ott nonterminal name and adds its productions to the grammar.

`\nonterms [`⟨*dimen*⟩`] {`⟨*nonterm-name*⟩`[,...]}`

This is a convenience macro for typesetting a grammar with the specified nonterminals. It's easily defined in terms of the `ottaltgrammar` environment.

`\ottaltinferrule {`⟨*rule-name*⟩`} {`⟨*infer-opts*⟩`} {`⟨*premises*⟩`} {`⟨*conclusion*⟩`}`

For setting inference rules, by default this delegates to mathpartir's `\inferrule*`. The ⟨*rule-name*⟩ is passed to `\inferrule*` using key `lab`, and the remaining options ⟨*infer-opts*⟩ are passed as is. This command may be redefined in order to set inference rules with different options or to use a different mechanism than `\inferrule*`.

`\drulesectionhead {`⟨*judgment*⟩`} {`⟨*descr*⟩`}`
`\drulesectionhead* {`⟨*judgments*⟩`} {`⟨*descr*⟩`}`

These commands are used to set the heading for the `drulepar` and `rulesection` environments. Argument ⟨*judgment*⟩ is the judgment to set in a box on the left, and ⟨*descr*⟩ is the judgment description to set on the right. Or, ⟨*judgments*⟩ is a comma-separated list of judgments, each of which should be set in its own box.

command	**default value**
`\FormatDruleSectionHead[1]`	`\fbox{#1}`
`\FormatDruleSectionHeads[1]`	`\fbox{\strut#1}`
`\FormatDruleSectionHeadRight[1]`	`\emph{(#1)}`
`\FormatDruleSepTwo`	`\,,~`
`\FormatDruleSepMore`	`\FormatDruleSepTwo`
`\FormatDruleSepLast`	`\FormatDruleSepTwo`

These are used to customize the output of `\drulesectionhead` and `\drulesectionhead*`, and by extension, the `drulepar` and `rulesection` environments. `\FormatDruleSectionHead` and `\FormatDruleSectionHeads` specify how to format the left-side head (usually the form of a rule) for `\drulesectionhead` and `\drulesectionhead*`, respectively; by default, they use a framed box, and the latter adds a `\strut` so that several boxes have the same height. `\FormatDruleSectionHeadRight` is used to format the right-side head (usually a description). `\FormatDruleSepTwo`, `\FormatDruleSepMore`, and `\FormatListSepLast` are used by `\drulesectionhead*` to separate several left-side heads. `\FormatDruleSepTwo` is used if there are only two heads to separate. If there are more, then `\FormatDruleSepMore` is used for all but the last separator, and `\FormatDruleSepLast` is used between the last two.

2.4 Nonterminal Styling

2.4.1 Nonterminal Classes

Ott allows specifying the LaTeX code for printing nonterminal symbols. For example, we may use the concrete syntax `G` in Ott for typing contexts, but want it to appear as Γ in typeset output. We can do this in Ott by specifying a "morphism" after declaring the nonterminal, like this:

```
G {{ tex \Gamma }} :: env_ ::= {{ com typing contexts }}
```

I've found this method of specifying how nonterminals should be printed to be difficult to manage in a large Ott development, because it's scattered throughout the Ott source, and because I often want to parameterize how nonterminals are printed, for example, to produce both color and black-and-white versions of the same document. This package provides a facility for managing nonterminal symbols so that instead, one writes something like

```
G {{ tex \nonterm G }} :: env_ ::= {{ com typing contexts }}
```

in the Ott source and then can configure what `\nonterm G` means from the LaTeX side.

To do this, we declare an "NT class" and then define the meaning of nonterminal symbols within it:

`\newNTclass [`⟨*mode-if*⟩`] {`⟨*class-name*⟩`}`

Declare a new nonterminal class named ⟨*class-name*⟩. If ⟨*mode-if*⟩ is supplied, then it is used as a conditional to choose between regular and alternate styles for the nonterminals, as explained below. The conditional should take two arguments, and return the first for *true* and the second for *false*. (The default for ⟨*mode-if*⟩ is `\ifnotalternateNonterms`, which is set by the `alternateNonterms` package option.)

This command results in the definition of three new commands:

`\new`⟨*class-name*⟩ `[`⟨*style*⟩`] {`⟨*nt-name*⟩`} {`⟨*defn*⟩`}`

Declare nonterminal name ⟨*nt-name*⟩ to be printed as ⟨*style*⟩`{`⟨*defn*⟩`}` in the regular case (when ⟨*mode-if*⟩ is true) and ⟨*defn*⟩ in the alternate case (when ⟨*mode-if*⟩ is false).

`\new`⟨*class-name*⟩`s [`⟨*style*⟩`] {`⟨*nt-name*⟩`} {`⟨*defn*⟩`} {`⟨*alt-defn*⟩`}`

Declare nonterminal name ⟨*nt-name*⟩ to be printed as ⟨*style*⟩`{`⟨*defn*⟩`}` in the regular case (when ⟨*mode-if*⟩ is true) and ⟨*alt-defn*⟩ in the alternate case (when ⟨*mode-if*⟩ is false).

`\`⟨*class-name*⟩ `{`⟨*nt-name*⟩`}`

Print nonterminal ⟨*nt-name*⟩.

Note that the mode selector ⟨*mode-if*⟩ takes effect when nonterminals are declared, not when they are used.

`\ifnotalternateNonterms {`⟨*regular*⟩`} {`⟨*alternate*⟩`}`

Select between regular and alternate presentations. By default, this is defined to return ⟨*regular*⟩, but if the `alternateNonterms` package option is given when loading the package, then it returns ⟨*alternate*⟩ instead. This is used by `\newNTclass` as the mode selector if none is explicitly supplied.

`\@ifToif {`⟨*LaTeX-style-if*⟩`}`
`\ifTo@if {`⟨*TeX-style-if*⟩`}`

Convert between TEX-style conditional, which expects to see `\else` (optionally) and `\fi`, and LATEX-style conditional, which takes two arguments. For example, we can convert the TEX-style `\ifnum\dim<5` to LATEX-style and give it two arguments, like so:

`\ifTo@if {\ifnum\dim<5} {`⟨*then-text*⟩`} {`⟨*else-text*⟩`}`

Or, we can convert LATEX-style `\@ifundefined{somecs}` to TEX-style:

`\@ifToif {\@undefined{somecs}}` ⟨*then-text*⟩`\else` ⟨*else-text*⟩`\fi`

The former is useful to convert a TEX-style conditional declared with `\newif` for passing to `\newNTclass`. Here's an example. We begin by declaring a new TEX conditional, which we'll use to switch color on and off:

```
\newif\ifcolor \ifcolortrue
```

Later, we declare an nonterminal class for "calculus A", whose presentation mode is controlled by `\ifcolor`:

```
\newNTclass[\ifTo@if\ifcolor]{calcA}
```

Then we can add nonterminal symbols to calculus A. For example, we declare a nonterminal named `t`, which prints as τ when `\ifcolor` was true when it was declared and as τ when `\ifcolor` was false:

```
\newcalcA[\textcolor{teal}] t \tau
```

2.4.2 Subscript-and-Prime Helpers

Ott allows writing nonterminals and metavariables followed by digits and apostrophes to produce numeric subscripts and primes. For example, if `t` is set to print τ, then `t12'` prints τ'_{12}. However, if we style τ, it will not automatically style the subscripts and primes as well. For example, if we give a declare `t` as above to be `\calcA t`, which expands to `\textcolor{teal}{\tau}`, then `t12'` prints as τ'_{12}, which is unsatisfactory.

In this section, we describe commands that capture subscripts and primes that follow a nonterminal symbol so that we can style them along with the nonterminal. For example, you could define `\calcA t` as

```
\newcalcA[\NTCAPTURE{\textcolor{teal}}] t \tau,
```

and then `t12'` appears as τ'_{12}.

`\NTCAPTURE {`⟨*style*⟩`} {`⟨*text*⟩`} `⟨*capture-seq*⟩

where ⟨*capture-seq*⟩ ::= ⟨*empty*⟩
| `_{`⟨*subscript*⟩`}` ⟨*capture-seq*⟩
| `'` ⟨*capture-seq*⟩

Style ⟨*text*⟩ and any immediately following ⟨*subscript*⟩s and 's (primes) using ⟨*style*⟩. That is, it expands to

⟨*style*⟩`{`⟨*text*⟩⟨*capture-seq*⟩`}`

`\NTCAPTURELOW {`⟨*styler*⟩`} {`⟨*text*⟩`} `⟨*capture-seq*⟩

Like `\NTCAPTURE`, it captures subscripts and primes following `{`⟨*text*⟩`}`, but it passes the ⟨*text*⟩, primes, and subscripts to ⟨*styler*⟩ as separate arguments, like this:

⟨*styler*⟩ `{`⟨*text*⟩`} {`⟨*subscript*⟩`...} {'...}`

This lets ⟨*styler*⟩ have individual control over how each part is styled.

Usage	**Definition**
`\NTOVERLINE {`⟨*text*⟩`}` ⟨*capture-seq*⟩	`\NTCAPTURE\overline`
`\NTUNDERLINE {`⟨*text*⟩`}` ⟨*capture-seq*⟩	`\NTCAPTURE\underline`
`\NTTEXTCOLOR {`⟨*color*⟩`} {`⟨*text*⟩`}` ⟨*capture-seq*⟩	`\NTCAPTURE{\textcolor{`⟨*color*⟩`}}`

Convenience macros that serve as example uses of `\NTCAPTURE` for overlined, underlined, and colored nonterminals.

3 Implementation

We begin by loading several packages:

```
⟨*package⟩
\RequirePackage{mathpartir}
\RequirePackage{ifthen}
\RequirePackage{keyval}
\RequirePackage{listproc}
```

3.1 Loading & Options

There are five package options. The first three determine the line break behavior between inference rule premises. The fourth turns on alternate presentation of nonterminal symbols using the NT class system, and the fifth uses the supertabular package for grammars.

```
\DeclareOption{implicitPremiseBreaks}{
```

```
  \renewcommand\ottaltpremisesep{\\}
  \renewcommand\ottaltpremisebreak{\\}
}
\DeclareOption{lineBreakHack}{
  \renewcommand\ottaltpremisesep{\and}
  \renewcommand\ottaltpremisebreak{\\\\}
}
\DeclareOption{implicitLineBreakHack}{
  \renewcommand\ottaltpremisesep{\\}
  \renewcommand\ottaltpremisebreak{\\\\}
}
\DeclareOption{alternateNonterms}{
  \let\ifnotalternateNonterms\@secondoftwo
}
\DeclareOption{supertabular}{
  \ottalt@supertabulartrue
}
```

\ottaltpremisesep
\ottaltpremisebreak
\ifnotalternateNonterms

We set the default values for the options (automatic line breaks, ignoring the line break hack, and not in alternate NT mode), and then process the actual options.

```
\newcommand\ottaltpremisesep{\\}
\newcommand\ottaltpremisebreak{\\}
\let\ifnotalternateNonterms\@firstoftwo
\newif\ifottalt@supertabular
\ProcessOptions
```

Load the supertabular package if requested.

```
\ifottalt@supertabular
  \RequirePackage{supertabular}
\fi
```

3.1.1 Renewing Ott Commands

\inputott
\renewottcommands
\ottaltcurrentprefix

`\inputott` inputs the file named by its argument and then calls `\renewottcommands`, which redefines several commands that are originally defined in the code generated by Ott. `\ottaltcurrentprefix` is the current Ott prefix set by the most recent call to `\renewottcommands`.

```
\newcommand\inputott[2][ott]{
  \input{#2}
  \renewottcommands[#1]
}
\newcommand\ottaltcurrentprefix{ott}
\newcommand\renewottcommands[1][ott]{
  \renewcommand\ottaltcurrentprefix{#1}
```

\renewottcomm@nd

Helper macro for redefining Ott commands using the supplied prefix `#1`:

```
  \def\renewottcomm@nd##1{
    \expandafter\renewcommand\csname #1##1\endcsname
  }
```

\ottdrule
\ottalt@nextpremise

This command cooperates with the next one, \ottpremise, to print with the proper line breaking premises. It does two things to set up evaluating the premises: it clears a token register in which premises will accumulator, and defines \ottalt@nextpremise to do nothing. The latter will be added to the token register before the next premise, so it's used by each premise to communicate to the next one what to place between then to produce spacing or a line break.

Each call to \ottpremise will add itself to the token register. Then \ottdrule passes the token register to \inferrule* to use as the premise(s) of the rule.

```
\renewottcomm@nd{drule}[4][]{
  \def\ottalt@nextpremise{}
  \ottalt@premisetoks={ }
  ##2
  \expandafter\ottalt@inferrule\expandafter
    {\the\ottalt@premisetoks}{##3}{##4}{##1}
}
```

\ottpremise

This is the command that Ott wraps each inference rule premise with, so we redefine it to add each premise to the token register. It checks whether its contents end with \, in which case it sets \ottalt@nextpremise to produce a line break before the next premise (if line-break-hack–mode is on).

```
\renewottcomm@nd{premise}[1]{%
  \ottalt@premisetoks=
      \expandafter\expandafter\expandafter
              {\expandafter\the\expandafter\ottalt@premisetoks
               \ottalt@nextpremise##1}
  \ottalt@iflinebreakhack##1\ottlinebreakhack\ottalt@iflinebreakhack{
    \let\ottalt@nextpremise\ottaltpremisebreak
  }{
    \let\ottalt@nextpremise\ottaltpremisesep
  }
}
```

\ottusedrule

We redefine \ottusedrule to automatically insert \and between rules. We assume that for the first rule in the group, \ifottalt@firstrule will be true, and we change it to false for subsequent rules. This works provided each sequence of rules is in the same TEX group, because we set locally.

```
\renewottcomm@nd{usedrule}[1]{%
  \ifottalt@firstrule
    \ottalt@firstrulefalse
  \else
    \and
  \fi
  \ensuremath{##1}
}
```

\ottdefnblock

Ott uses this command for enclosing all the rules in a judgment. We delegate to drulepar, which is defined in §3.3.

```
\renewenvironment{#1defnblock}[3][]
```

```
    {\begin{drulepar}{##2}{##3}}
    {\end{drulepar}}
```

`\ottdrulename` To print rule names, we replace underscore with hyphen and turn the rule name into a rule anchor:

```
  \renewottcomm@nd{drulename}[1]{%
    \ottalt@replace@cs\ranchor\_-{}##1\\
  }
```

`\ottprodline` `\ottprodnewline` For each production line in a grammar, we test for M and S rules and don't print them. Because we add newlines in the `\ottprodline`, we make `\ottprodnewline` do nothing.

```
  \renewottcomm@nd{prodline}[6]{
    \ifthenelse{\equal{##3}{}}{
      \\ & & $##1$ & $##2$ & & $##5$ & $##6$
    }{}
  }
  \renewottcomm@nd{prodnewline}{\relax}
```

`\ottgrammartabular` This is the command that Ott uses for grammars.

```
  \renewottcomm@nd{grammartabular}[1]{%
    \begin{ottaltgrammar}##1\end{ottaltgrammar}%
  }

}
```

`\drule@h@lper` `\nonterm@h@lper` These two macros are not Ott commands that we are redefining, but helper macros that depend on the most recent prefix passed to `\renewottcommands` as `#1`.

`\drule@h@lper` is used to print an Ott inference rule by name. It takes three arguments: an "optional" argument to pass to the defined rule macro (which likely does nothing), the full, printable name of the rule (for error messages, in case the rule isn't found), and the encoded name that should be used to actually look up the rule.

```
\newcommand*\drule@h@lper[3]{%
  \expandafter\ifx\csname\ottaltcurrentprefix drule#3\endcsname\relax
    \PackageWarning{ottalt}{Unknown ott rule: #3}%
    \mbox{\textbf{(#2?)}}%
  \else
    \csname\ottaltcurrentprefix usedrule\endcsname
      {\csname\ottaltcurrentprefix drule#3\endcsname{#1}}%
  \fi
}
```

The name of an Ott-defined nonterminal is just the prefix name concatenated with the nonterminal name:

```
\newcommand*\nonterm@h@lper[1]{\csname\ottaltcurrentprefix#1\endcsname}
```

3.2 Rule References

\rrefruletext
\Rrefruletext
\rrefrulestext
\Rrefrulestext
\rrefstyle
\ranchorstyle
\wraparoundrref

These commands are for formatting rule references. Redefine these to change how rule references appear.

```
\newcommand\rrefruletext{rule}
\newcommand\Rrefruletext{\expandafter\MakeUppercase\rrefruletext}
\newcommand\rrefrulestext{\rrefruletext s}
\newcommand\Rrefrulestext{\Rrefruletext s}
\newcommand\rrefstyle{\normalfont\scshape}
\newcommand\ranchorstyle{\rrefstyle}
\providecommand\wraparoundrref{\relax}
```

\rref
\rref*
\Rref
\Rref*

These dispatch between the star and starless variants:

```
\newcommand*\rref{%
  \@ifnextchar*
    {\rref@star}
    {\rref@with\rrefruletext\rrefrulestext}}
\newcommand*\Rref{%
  \@ifnextchar*
    {\rref@star}
    {\rref@with\Rrefruletext\Rrefrulestext}}
```

\rref@with
\rref@start
\@one@rref@nohyper
\@ranchor@nhyper

The first two of these produce rule references, using `\FormatList` (from package listproc), either passing an intro word (for the non-star variant) or no intro word (for the star variants). The last two commands format rule references and anchors without hyperlinks.

```
\newcommand*\rref@with[2]{\FormatList{#1~}{#2~}{\one@rref}}
\newcommand*\rref@star[1]{\FormatList{}{}{\one@rref}}
\newcommand*\@one@rref@nohyper[1]{\wraparoundrref{{\rrefstyle{#1}}}}
\newcommand*\@ranchor@nohyper[1]{{\ranchorstyle{#1}}}
```

\one@rref
\ranchor

We try to detect the hyperref package, and define the rule reference macros to create hyperlinks if it's loaded. Rather than do the detection now, we wait until the end of the preamble, so that it doesn't matter in what order the packages are loaded.

```
\AtBeginDocument{
  \ifcsname hypertarget\endcsname
    \newcommand*\one@rref[1]{%
      \hyperlink
        {ottalt:rule:\ottaltcurrentprefix:#1}
        {\@one@rref@nohyper{#1}}%
    }
    \newcommand*\ranchor[1]{%
      \hypertarget
        {ottalt:rule:\ottaltcurrentprefix:#1}
        {\@ranchor@nohyper{#1}}%
    }
  \else
```

```
    \newcommand\one@rref{\@one@rref@nohyper}
    \newcommand\ranchor{\@ranchor@nohyper}
  \fi
}
```

3.3 Including Inference Rules and Grammars

`\drules` Create a `drulepar` and iterate through the requested rules, adding each one.

```
\newcommand*{\drules}[4][\relax]{%
  \begin{drulepar}[#1]{#2}{#3}
    \@for\@ottalt@each:=#4\do{%
      \expandafter\drule\expandafter{\@ottalt@each}
    }
  \end{drulepar}%
}
```

`drulepar` `drulepar*` A rule paragraph is merely a math paragraph wrapped in a rule section:

```
\newenvironment{drulepar}[3][\relax]
  {\begin{rulesection}[#1]{#2}{#3}%
   \begin{mathparpagebreakable}}
  {\end{mathparpagebreakable}%
   \end{rulesection}}
\newenvironment{drulepar*}[3][\relax]
  {\begin{rulesection*}[#1]{#2}{#3}%
   \begin{mathparpagebreakable}}
  {\end{mathparpagebreakable}%
   \end{rulesection*}}
```

`rulesection` `rulesection*` `\ottalt@rulesection@prefix` `\drulesectionhead` `\drulesectionhead*` A rule section is set as a one-item `\trivlist`, with the heading set by `\drulesectionhead`. If the optional argument is given, then we save it in `\ottalt@rulesection@prefix`, so that `\drule` can add the prefix to each rule name. Initially, the rule section prefix is empty.

```
\newenvironment{rulesection}[3][\relax]
  {\trivlist\item
   \ifx#1\relax\else\def\ottalt@rulesection@prefix{#1-}\fi
   \drulesectionhead{#2}{#3}%
   \nopagebreak[4]%
   \noindent}
  {\endtrivlist}
\newenvironment{rulesection*}[3][\relax]
  {\trivlist\item
   \ifx#1\relax\else\def\ottalt@rulesection@prefix{#1-}\fi
   \drulesectionhead*{#2}{#3}%
   \nopagebreak[4]%
   \noindent}
  {\endtrivlist}
\newcommand\ottalt@rulesection@prefix{}
\newcommand*{\drulesectionhead}{%
```

```
  \@ifnextchar *{\drulesectionheadMany}{\drulesectionheadOne}%
}
\newcommand*{\drulesectionheadOne}[2]{%
  \FormatDruleSectionHead{#1}%
  \hfill\FormatDruleSectionHeadRight{#2}%
  \par
}
\newcommand*{\drulesectionheadMany}[3]{%
  {%
    \let\FormatListSepTwo\FormatDruleSepTwo
    \let\FormatListSepMore\FormatDruleSepMore
    \let\FormatListSepLast\FormatDruleSepLast
    \FormatList{}{}{\FormatDruleSectionHeads}{#2}%
  }%
  \hfill\FormatDruleSectionHeadRight{#3}%
  \par
}
```

\FormatDruleSepTwo
\FormatDruleSepMore
\FormatDruleSepLast
\FormatDruleSectionHead
\FormatDruleSectionHeads
\FormatDruleSectionHeadRight

This macros allow customizing the formatting of `\drulesectionhead` and `\drulesectionhead*`.

```
\newcommand*\FormatDruleSepTwo{\,,~}
\newcommand*\FormatDruleSepMore{\FormatDruleSepTwo}
\newcommand*\FormatDruleSepLast{\FormatDruleSepTwo}
\newcommand*\FormatDruleSectionHead[1]{\fbox{#1}}
\newcommand*\FormatDruleSectionHeads[1]{\fbox{\strut#1}}
\newcommand*\FormatDruleSectionHeadRight[1]{\emph{(#1)}}
```

\drule
\drule@helper

To include a rule by name. Turns `-` into `XX`, since that's how the rule names become defined as macros. We call `\drule@h@elper`, which is defined by `\renewottcommands` to refer to the correct Ott prefix.

```
\newcommand*\drule[2][]{%
  \expandafter\drule@helper\expandafter{\ottalt@rulesection@prefix}{#1}{#2}%
}
\newcommand*\drule@helper[3]{%
  \ottalt@replace@cs{\drule@h@lper{#2}{#1#3}}-{XX}{}#1#3\\
}
```

\ottaltinferrule
\ottalt@inferrule

Here we define several commands that help with typesetting rules. We begin with the final macro for typesetting rules. Redefine this to set inference rules differently. It is called by `\ottalt@inferrule`, which reorders its arguments from a useful order from the perspective of `\ottdrule` to the natural order for `\ottaltinferrule`.

```
\newcommand\ottaltinferrule[4]{
  \inferrule*[narrower=0.3,lab=#1,#2]
    {#3}
    {#4}
}
\newcommand\ottalt@inferrule[4]{
  \ottaltinferrule{#3}{#4}{#1}{#2}
}
```

\ifottalt@firstrule These are used when dealing with rule premises to maintain rule spacing and eal with the line break hack. Several of these are defined here but used by \ottdrule, which is redefined by \renewottcommands.

```
\newif\ifottalt@firstrule \ottalt@firstruletrue
\newcommand{\ottalt@nextpremise}{\relax}
\newtoks\ottalt@premisetoks
\newcommand{\ottlinebreakhack}{\relax}
\def\ottalt@iflinebreakhack#1\ottlinebreakhack #2\ottalt@iflinebreakhack{%
  \ifthenelse{\equal{#2}{}}\@secondoftwo\@firstoftwo
}
```

\ottalt@replace@cs This helper macro is used to search for a control sequence in a list of tokens and replace it with a given list of tokens. The usage is:

\ottalt@replace@cs{⟨*next*⟩}{⟨*needle*⟩}{⟨*replacement*⟩}{}⟨*haystack*⟩\\

This replaces ⟨*needle*⟩ with ⟨*replacement*⟩ in ⟨*haystack*⟩ and passes the result to ⟨*next*⟩. It is important that ⟨*needle*⟩ be a single token, and that ⟨*haystack*⟩ *not* be wrapped in curly braces. The extra curly braces before ⟨*haystack*⟩ and the double backslash after are necessary as well.

```
\newcommand\ottalt@replace@cs[5]{%
  \ifx\\#5\relax
    \def\ottalt@replace@cs@kont{#1{#4}}%
  \else
    \ifx#2#5\relax
      \def\ottalt@replace@cs@kont{\ottalt@replace@cs{#1}{#2}{#3}{#4#3}}%
    \else
      \def\ottalt@replace@cs@kont{\ottalt@replace@cs{#1}{#2}{#3}{#4#5}}%
    \fi
  \fi
  \ottalt@replace@cs@kont
}
```

\nonterms To print a grammar, we use the ottaltgrammar environment defined right after this, and iterate through the requested nonterminal symbols, calling \nt with each.

```
\newcommand*\nonterms[2][8pt]{
  \begin{ottaltgrammar}[#1]
    \@for\@ottalt@each:=#2\do{%
      \expandafter\nt\expandafter{\@ottalt@each}
    }
  \end{ottaltgrammar}
}
```

ottaltgrammar \nt \ottaltintertext The ottaltgrammar environment sets up the table for printing grammars. It defines the \nt macro, which adds the grammar for a named nonterminal, for the extent of the environment.

```
\newenvironment{ottaltgrammar}[1][8pt]{%
  \begingroup
  \trivlist\item
```

We use a trick here to make `\nt` automatically insert a newline before all but the first nonterminal in a grammar. We define `\OTTALTNEWLINE` to insert a newline, but then right before the body of the environment starts, inside the `supertabular` environment, we redefine `\OTTALTNEWLINE` to do nothing. Because `supertabular` places each table cell in a group, the no-op redefinition prevents the first but not subsequent nonterminals from inserting a line break.

```
\def\OTTALTNEWLINE{\\[#1]}%
\def\nt##1{\OTTALTNEWLINE\relax\nonterm@h@lper{##1}\ignorespaces}%
\newcommand\ottaltintertext[2]{%
  \multicolumn{8}{l}{%
    \begin{minipage}{##1}%
      ##2%
    \end{minipage}%
  }%
}%
\ifottalt@supertabular
  \begin{supertabular}{llcllllll}
\else
  \begin{tabular}{llcllllll}
\fi
\let\OTTALTNEWLINE\relax
\ignorespaces
}
{%
\@ifundefined{ottafterlastrule}{\\}{\ottafterlastrule}%
\ifottalt@supertabular
  \end{supertabular}
\else
  \end{tabular}
\fi
\endtrivlist
\endgroup
\ignorespaces
}
```

3.4 Nonterminal Styling

`\newNTclass` This command defines three new commands with names based on its second argument. The `\new#2s` command uses the first argument to decide how to print nonterminals that are defined with it.

```
\newcommand\newNTclass[2][\ifnotalternateNonterms]{
  \expandafter\newcommand\csname new#2s\endcsname[4][]{
    #1{
      \expandafter\newcommand\csname ottalt@NT@#2@##2\endcsname{##1{##3}}
    }{
      \expandafter\newcommand\csname ottalt@NT@#2@##2\endcsname{##4}
    }
  }
```

```
  \expandafter\newcommand\csname new#2\endcsname[3][]{
    \csname new#2s\endcsname[##1]{##2}{##3}{##3}
  }
  \expandafter\newcommand\csname #2\endcsname[1]{%
    \csname ottalt@NT@#2@##1\endcsname
  }
}
```

`\@ifToif` `\ifTo@if` Commands for converting between two styles of conditionals.

```
\providecommand\@ifToif[1]{%
  #1\iftrue\iffalse
}
\providecommand\ifTo@if[1]{%
  #1%
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
```

`\NTOVERLINE` `\NTUNDERLINE` `\NTTEXTCOLOR` Some sample non-terminal stylers:

```
\newcommand\NTOVERLINE{\NTCAPTURE\overline}
\newcommand\NTUNDERLINE{\NTCAPTURE\underline}
\newcommand\NTTEXTCOLOR[1]{\NTCAPTURE{\textcolor{#1}}}
```

`\NTCAPTURE` `\NTCAPTURELOW` `\NTCAPTURE@FINISH` These are the commands for capturing subscripts and primes following a non-terminal. We define `\NTCAPTURE` by passing the helper `\NTCAPTURE@FINISH` to `\NTCAPTURELOW`, and it composes the nonterminal, subscript, and primes in the usual way. Then `\NTCAPTURELOW` uses `\NT@CAPTURE@LOOP` to do the work of gathering subscripts and primes.

```
\newcommand\NTCAPTURE[1]{\NTCAPTURELOW{\NTCAPTURE@FINISH{#1}}}
\newcommand\NTCAPTURE@FINISH[4]{#1{#2_{#3}#4}}
\newcommand\NTCAPTURELOW[2]{\NT@CAPTURE@LOOP{#1}{#2}\relax\relax}
```

`\NT@CAPTURE@LOOP` `\NT@CAPTURE@SUB` `\NT@CAPTURE@PRIME` This is the main loop for the nonterminal capture macros. The arguments are:

`#1` The operator to apply to the `#2`–`#4` when done

`#2` Hold the main nonterminal throughout the loop

`#3` Accumulates subscripts

`#4` Accumulates primes

The main loop checks whether the next character is an underscore or apostrophe, and if so, dispatches to either `\NT@CAPTURE@SUB` or `\NT@CAPTURE@PRIME` to grab it and loop again. If the next character is neither, then it finally calls `#1` with the nonterminal and the accumulated arguments.

```
\newcommand\NT@CAPTURE@LOOP[4]{%
```

```
  \@ifnextchar _{%
    \NT@CAPTURE@SUB{#1}{#2}{#3}{#4}%
  }{\@ifnextchar '{%
    \NT@CAPTURE@PRIME{#1}{#2}{#3}{#4}%
  }{%
    {#1{#2}{#3}{#4}}%
  }}%
}
\def\NT@CAPTURE@SUB#1#2#3#4_#5{\NT@CAPTURE@LOOP{#1}{#2}{#3#5}{#4}}
\def\NT@CAPTURE@PRIME#1#2#3#4'{\NT@CAPTURE@LOOP{#1}{#2}{#3}{#4'}}
⟨/package⟩
```

A Sample Ott `Makefile`

This `Makefile` has rules suitable for a workflow integrating Ott and LaTeX. It should be available in the file `Makefile.sample`.

From an Ott file ⟨*stem*⟩`.ott`, `make` ⟨*stem*⟩`.ott.tex` will generate an includable LaTeX file contain definitions of macros for typesetting the grammars and rules in ⟨*stem*⟩`.ott`. Given some other LaTeX file ⟨*stem*⟩`.tex` that wants to use Ott for preprocessing, `make` ⟨*stem*⟩`.mng.tex` will filter the file through Ott.

By default, Ott does not stop on all parse errors in filter mode, but emits LaTeX code that may result in errors later when compiling the LaTeX file. The rule for running Ott as a filter attempts to detect such errors, issue an error message, and halt (unless the environment variable `$DONTSTOP` is set).

```
⟨*makefile⟩
OTT             = ott $(OTTFLAGS)
OTTFLAGS        = -signal_parse_errors true \
                  -tex_wrap false -tex_show_meta false

%.ottdump: %.ott
        $(OTT) -picky_multiple_parses true -i $< -writesys $@

%.ott.tex: %.ottdump
        $(OTT) -readsys $< -o $@

%.mng.tex: %.tex %.ottdump
        $(OTT) -readsys $*.ottdump -tex_filter $< $@
        @if grep '<< no parses (' $@ >/dev/null 2>&1 && \
              [ -z "$(DONTSTOP)" ]; then \
                echo; \
                echo "***** OTT PARSE ERROR(S) *****"; \
                grep -n '<< no parses (' $@; \
                $(RM) $@; \
                exit 1; \
        fi >&2
⟨/makefile⟩
```

B Additional Ott Code

This is additional Ott code that is used to produce the Ott examples in §1.

```
⟨*ott⟩
grammar
metavar x {{ tex \nonterm x }} ::= {{ com variables }}
metavar a {{ tex \nonterm a }} ::= {{ com type variables }}

grammar
formula :: formula_ ::= {{ com formulas from meta-language }}
  | judgement    :: :: judgement
  | formula \\\\ :: :: lbh {{ tex [[formula]] \ottlinebreakhack }}
  | x : t in G   :: :: xinenv
  | a in G       :: :: ainenv
  | G |- t ok    :: :: tokay

terminals :: terminals_ ::=
  | ->           :: :: arr       {{ tex \to }}
  | all          :: :: all       {{ tex \forall }}
  | \            :: :: lam       {{ tex \lambda }}
  | /\           :: :: Lam       {{ tex \Lambda }}
  | in           :: :: in        {{ tex \in }}
  | |-           :: :: proves    {{ tex \vdash }}
  | empty        :: :: empty     {{ tex \bullet }}
⟨/ott⟩
```

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

R

S

T

U

V

W