The Ott LaTeX Layout Package `ottlayout.sty`

Rok Strniša   Matthew Parkinson

August 6, 2016

1 Introduction

The Ott LaTeX Layout Package, `ottlayout.sty`, provides a range of options to tune the typesetting of Ott-generated inductive definition rules and grammars, overriding the default typesetting of the Ott-generated LaTeX code.

This document illustrates the common-case usage of the package, using Lightweight Java (LJ) [1] as an example Ott project. It should be read in conjunction with the source for this document (`manual.tex`) and the `Makefile`.

2 Usage

To use the package, one first uses Ott to generate LaTeX code with a chosen `-tex_name_prefix`, by default `ott`, but here `lj`, as in the example `Makefile`:

```
lj_included.tex : $(lj)
        ott $(INC_ARGS) -tex_name_prefix lj -tex $@ \
            -merge true $(lj)
```

Then one builds a file such as `lj_override.tex`, e.g. as below.

```
%_override.tex: override.tex empty.ott
        ott $(INC_ARGS) -tex_name_prefix lj \
          -tex_filter override.tex $@ empty.ott
```

This file simply contains redefinitions of some default LaTeX commands generated by Ott (with the `lj` prefix) to use the `ottlayout.sty` commands, e.g.

```
  \renewcommand{\ljpremise}[1]{\premiseSTY{#1}}
  \renewcommand{\ljusedrule}[1]{\usedruleSTY{#1}}
  \renewcommand{\ljdrule}[4][]{\druleSTY[#1]{#2}{#3}{#4}}
  \renewenvironment{ljdefnblock}[3][]{%
    \defnblockSTY[#1]{#2}{#3}}{\enddefnblockSTY}
```

Finally, in the user LaTeX document (for example this `manual.tex`), one: (a) includes the generated LaTeX for the user language, e.g. with `\include{lj_included}`; (b) uses the `ottlayout.sty` package, e.g. with `\usepackage{ottlayout}`; and (c) uses the generated override file to link the generated LaTeX with `ottlayout.sty`, e.g. with `\include{lj_override}`.

3 Displaying grammar

To display all Ott-generated LaTeX for LJ, we would normally write the command `\ljall{}`. To output all the LJ's grammar, we would use the LaTeX command: `\ljgrammar{}`. To show only selected parts of the grammar, we would normally use the command `\ljgrammartabular{}`. For example, to display the grammar of LJ's statement (`s`) and class definition (`cld`), we would write [1]

```
\ljgrammartabular{\ljs\ljinterrule\ljcld\ljafterlastrule}
```

to produce:

s	$::=$		statement
	$\vert$	$\{\, \overline{s_k}^{\,k} \,\}$	block
	$\vert$	$var = x;$	variable assignment
	$\vert$	$var = x.f;$	field read
	$\vert$	$x.f = y;$	field write
	$\vert$	$\textbf{if}\ (x == y)\, s\ \textbf{else}\ s'$	conditional branch
	$\vert$	$var = x.meth(\overline{y});$	method call
	$\vert$	$var = \textbf{new}_{ctx}\, cl();$	object creation
cld	$::=$		class
	$\vert$	$\textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\, \overline{meth\_def}\}$	def.

Alternatively, we can use the `ottlayout` package's `\grammartabularSTY{}` to produce a slightly more compact output, usually more suitable for publications. To display the same grammars, we would write

```
\grammartabularSTY{\ljs\\\ljcld}
```

to produce:

s	$::=$		statement
	$\vert$	$\{\, \overline{s_k}^{\,k} \,\}$	block
	$\vert$	$var = x;$	variable assignment
	$\vert$	$var = x.f;$	field read
	$\vert$	$x.f = y;$	field write
	$\vert$	$\textbf{if}\ (x == y)\, s\ \textbf{else}\ s'$	conditional branch
	$\vert$	$var = x.meth(\overline{y});$	method call
	$\vert$	$var = \textbf{new}_{ctx}\, cl();$	object creation
cld	$::=$		class
	$\vert$	$\textbf{class}\ dcl\ \textbf{extends}\ cl\{\overline{fd}\, \overline{meth\_def}\}$	def.

[1] The automatically generated grammar tabular command, here `\ljgrammartabular{}`, uses the `supertabular` package. Therefore, if we use the default grammar tabular, we have to explicitly import this package by writing `\usepackage{supertabular}` in our LaTeX document's prelude.

Note that in both cases the comments on the right are aligned according to the longest production in the block. Therefore, if the length of productions varies a lot, it is sometimes suitable to split them up into separate grammar tabulars. We could split the above example by writing

```
\grammartabularSTY{\ljs}\\
\grammartabularSTY{\ljcld}
```

to produce:

$$\begin{array}{lll}
s & ::= & & \text{statement} \\
 & \mid & \{\ \overline{s_k}^{\,k}\ \} & \text{block} \\
 & \mid & \mathit{var} = x; & \text{variable assignment} \\
 & \mid & \mathit{var} = x.f; & \text{field read} \\
 & \mid & x.f = y; & \text{field write} \\
 & \mid & \textbf{if}\ (x == y)\, s\ \textbf{else}\ s' & \text{conditional branch} \\
 & \mid & \mathit{var} = x.\mathit{meth}(\overline{y}); & \text{method call} \\
 & \mid & \mathit{var} = \textbf{new}_{\mathit{ctx}}\, \mathit{cl}(); & \text{object creation}
\end{array}$$

$$\begin{array}{llll}
\mathit{cld} & ::= & & \text{class} \\
 & \mid & \textbf{class}\ \mathit{dcl}\ \textbf{extends}\ \mathit{cl}\{\overline{\mathit{fd}}\,\overline{\mathit{meth\_def}}\} & \text{def.}
\end{array}$$

4 Displaying rules

To display all the rules of LJ, we would normally use the Ott-generated LaTeX command `\ljdefnss{}`. To show all the rules of the LJ's reduction relation, we would use the command `\ljdefnrXXstmt{}` — if you are not sure what the name of the command you are looking for is, the easiest way to find out is to check the Ott-generated LaTeX file, which is in our case `lj_included.tex`.

The `ottlayout` package gives many different options for displaying a particular rule and groups of rules. The currently available display options are:

Setting name	**Possible values**	**Default value**
showruleschema	yes | no	yes
showcomment	yes | no	yes
rulelayout	oneperline | nobreaks	oneperline
premiselayout	oneperline | oneline | justify	justify
premisenamelayout	right | left | topright | none	right
numberpremises	yes | no	no
numbercolour	*any dvips colour name*	Gray

The default settings result in the same output as if the `ottlayout` package was not used.

We use LJ's (fairly complicated) reduction rule for methods to demonstrate a few of the available display settings for an example. To display the LJ rule `r_mcall` with default settings we write

```
\ljdrulerXXmcall{}
```

which produces:

$$\frac{\begin{array}{l}
L(x) = oid\\
H(oid) = \tau\\
find\_meth\_def(P, \tau, meth) = (ctx, cl\ meth(\overline{cl_k\ var_k}^{k})\{\overline{s'_j}^{j}\ \textbf{return}\ y;\})\\
\overline{var'_k}^{k} \perp \textbf{dom}\,(L)\\
\textbf{distinct}\,(\overline{var'_k}^{k})\\
x' \notin \textbf{dom}\,(L)\\
x' \notin \overline{var'_k}^{k}\\
\overline{L(y_k) = v_k}^{k}\\
L' = L[\overline{var'_k \mapsto v_k}^{k}][x' \mapsto oid]\\
\theta = [\overline{var_k \mapsto var'_k}^{k}][\textbf{this} \mapsto x']\\
\overline{\theta \vdash s'_j \leadsto s''_j}^{j}\\
\theta(y) = y'
\end{array}}{(P, L, H, var = x.meth(\overline{y_k}^{k});\ \overline{s_l}^{l}) \longrightarrow (P, L', H, \overline{s''_j}^{j}\ var = y';\ \overline{s_l}^{l})}\ \textsc{r\_mcall}$$

Note that math mode is entered automatically, which means that we can use any LaTeX text layout utilities to layout our rules as we wish.

We can change the default setting with command `\ottstyledefaults{}` by passing keys and values in the KeyVal style. For example, to make the premises display more compactly in all rules from now on, we write

```
\ottstyledefaults{premiselayout=justify}
```

Now the the command `\ljdruleXXmcall{}` produces the following instead:

$$\frac{\begin{array}{l}
L(x) = oid \qquad\qquad H(oid) = \tau\\
find\_meth\_def(P, \tau, meth) = (ctx, cl\ meth(\overline{cl_k\ var_k}^{k})\{\overline{s'_j}^{j}\ \textbf{return}\ y;\})\\
\overline{var'_k}^{k} \perp \textbf{dom}\,(L) \qquad \textbf{distinct}\,(\overline{var'_k}^{k}) \qquad x' \notin \textbf{dom}\,(L)\\
x' \notin \overline{var'_k}^{k} \qquad\qquad \overline{L(y_k) = v_k}^{k}\\
L' = L[\overline{var'_k \mapsto v_k}^{k}][x' \mapsto oid] \quad \theta = [\overline{var_k \mapsto var'_k}^{k}][\textbf{this} \mapsto x']\\
\overline{\theta \vdash s'_j \leadsto s''_j}^{j} \qquad\qquad\qquad\qquad \theta(y) = y'
\end{array}}{(P, L, H, var = x.meth(\overline{y_k}^{k});\ \overline{s_l}^{l}) \longrightarrow (P, L', H, \overline{s''_j}^{j}\ var = y';\ \overline{s_l}^{l})}\ \textsc{r\_mcall}$$

To use the non-default settings for a particular rule, we can write the same KeyVal pairs inside as the parameter to the rule command. For example, to number the premises in the rule, to make the numbers yellow-orange, and to place the rule's name in top-right corner, we write

```
\ljdrulerXXmcall{numberpremises=yes,
                 numbercolour=YellowOrange,
                 premisenamelayout=topright}
```

which produces:

$$\frac{\begin{array}{l}1.\ L(x) = oid \qquad 2.\ H(oid) = \tau \\ 3.\ find\_meth\_def(P, \tau, meth) = (ctx, cl\ meth(\overline{cl_k\ var_k}^{k})\{\overline{s'_j}^{j}\ \mathbf{return}\ y;\}) \\ 4.\ \overline{var'_k}^{k} \perp \mathbf{dom}\,(L) \qquad 5.\ \mathbf{distinct}\,(\overline{var'_k}^{k}) \qquad 6.\ x' \notin \mathbf{dom}\,(L) \\ 7.\ x' \notin \overline{var'_k}^{k} \qquad 8.\ \overline{L(y_k) = v_k}^{k} \\ 9.\ L' = L[\overline{var'_k \mapsto v_k}^{k}][x' \mapsto oid] \\ 10.\ \theta = [\overline{var_k \mapsto var'_k}^{k}][\mathbf{this} \mapsto x'] \qquad 11.\ \overline{\theta \vdash s'_j \rightsquigarrow s''_j}^{j} \\ 12.\ \theta(y) = y'\end{array}}{(P, L, H, var = x.meth(\overline{y_k}^{k});\ \overline{s_l}^{l}) \longrightarrow (P, L', H, \overline{s''_j}^{j}\ var = y';\ \overline{s_l}^{l})}\ \text{R\_MCALL}$$

As you can see, the setting for compacting the premises was kept, because it was set globally for all rules following the `\ottstyledefaults{}` command.

As with commands for individual rules, we can pass in KeyVal pairs to the commands that display groups of rules, which will affect how the rules of that particular group of rules is displayed. If we wanted to display LJ's reduction rules for statements with current default display settings, we would write `\ljdefnrXXstmt{}`. To display the reduction rules with rule names on the left side, and their premises numbered, we write

```
\ljdefnrXXstmt{numberpremises=yes, premisenamelayout=left}
```

This produces[2]:

$\boxed{config \longrightarrow config'}$ one step reduction

$$\text{R\_FIELD\_READ\_NPE}\quad \frac{1.\ L(x) = \mathbf{null}}{(P, L, H, var = x.f;\ \overline{s_l}^{l}) \longrightarrow (P, L, H, \mathbf{NPE})}$$

$$\text{R\_FIELD\_WRITE\_NPE}\quad \frac{1.\ L(x) = \mathbf{null}}{(P, L, H, x.f = y;\ \overline{s_l}^{l}) \longrightarrow (P, L, H, \mathbf{NPE})}$$

$$\text{R\_VAR\_ASSIGN}\quad \frac{1.\ L(x) = v}{(P, L, H, var = x;\ \overline{s_l}^{l}) \longrightarrow (P, L[var \mapsto v], H, \overline{s_l}^{l})}$$

$$\text{R\_FIELD\_READ}\quad \frac{1.\ L(x) = oid \qquad 2.\ H(oid, f) = v}{(P, L, H, var = x.f;\ \overline{s_l}^{l}) \longrightarrow (P, L[var \mapsto v], H, \overline{s_l}^{l})}$$

$$\text{R\_FIELD\_WRITE}\quad \frac{1.\ L(x) = oid \qquad 2.\ L(y) = v}{(P, L, H, x.f = y;\ \overline{s_l}^{l}) \longrightarrow (P, L, H[(oid, f) \mapsto v], \overline{s_l}^{l})}$$

$$\text{R\_MCALL\_NPE}\quad \frac{1.\ L(x) = \mathbf{null}}{(P, L, H, var = x.meth(\overline{y_k}^{k});\ \overline{s_l}^{l}) \longrightarrow (P, L, H, \mathbf{NPE})}$$

[2]We set the font size to `small` so that the rules are not too wide.

$$\text{R\_MCALL\_CNFE}\quad \frac{\begin{array}{l}1.\ L(x) = oid \qquad 2.\ H(oid) = \tau\\ 3.\ find\_meth\_def(P, \tau, meth) = \mathbf{null}\end{array}}{(P, L, H, var = x.meth(\overline{y_k}^{k});\ \overline{s_l}^{l}) \longrightarrow (P, L, H, \mathbf{CNFE})}$$

$$\text{R\_IF\_TRUE}\quad \frac{1.\ L(x) = v \qquad 2.\ L(y) = w \qquad 3.\ v == w}{(P, L, H, \mathbf{if}\ (x == y) s_1\ \mathbf{else}\ s_2\ \overline{s'_l}^{l}) \longrightarrow (P, L, H, s_1\ \overline{s'_l}^{l})}$$

$$\text{R\_IF\_FALSE}\quad \frac{1.\ L(x) = v \qquad 2.\ L(y) = w \qquad 3.\ v \neq w}{(P, L, H, \mathbf{if}\ (x == y) s_1\ \mathbf{else}\ s_2\ \overline{s'_l}^{l}) \longrightarrow (P, L, H, s_2\ \overline{s'_l}^{l})}$$

$$\text{R\_BLOCK}\quad \frac{}{(P, L, H, \{\ \overline{s_k}^{k}\ \}\ \overline{s'_l}^{l}) \longrightarrow (P, L, H,\ \overline{s_k}^{k}\ \overline{s'_l}^{l})}$$

$$\text{R\_NEW}\quad \frac{\begin{array}{l}1.\ find\_type(P, ctx, cl) = \tau \quad 2.\ \mathbf{fields}\ (P, \tau) = \overline{f_k}^{k}\\ 3.\ oid \notin \mathbf{dom}\ (H)\\ 4.\ H' = H[oid \mapsto (\tau, \overline{f_k \mapsto \mathbf{null}}^{k})]\end{array}}{(P, L, H, var = \mathbf{new}_{ctx}\, cl();\ \overline{s_l}^{l}) \longrightarrow (P, L[var \mapsto oid], H', \overline{s_l}^{l})}$$

$$\text{R\_MCALL}\quad \frac{\begin{array}{lll}1.\ L(x) = oid & 2.\ H(oid) = \tau & \\ \multicolumn{3}{l}{3.\ find\_meth\_def(P, \tau, meth) = (ctx, cl\ meth(\overline{cl_k\ var_k}^{k})\{\overline{s'_j}^{j}\ \mathbf{return}\ y;\})}\\ 4.\ \overline{var'_k}^{k} \perp \mathbf{dom}\ (L) & 5.\ \mathbf{distinct}\ (\overline{var'_k}^{k}) & 6.\ x' \notin \mathbf{dom}\ (L)\\ 7.\ x' \notin \overline{var'_k}^{k} & 8.\ \overline{L(y_k) = v_k}^{k} & \\ \multicolumn{3}{l}{9.\ L' = L[\overline{var'_k \mapsto v_k}^{k}][x' \mapsto oid]}\\ \multicolumn{2}{l}{10.\ \theta = [\overline{var_k \mapsto var'_k}^{k}][\mathbf{this} \mapsto x']} & 11.\ \overline{\theta \vdash s'_j \rightsquigarrow s''_j}^{j}\\ 12.\ \theta(y) = y' & & \end{array}}{(P, L, H, var = x.meth(\overline{y_k}^{k});\ \overline{s_l}^{l}) \longrightarrow (P, L', H, \overline{s''_j}^{j}\ var = y';\ \overline{s_l}^{l})}$$

Therefore, Ott-generated LaTeX commands for rules of a specific **defn** in the Ott source file can take KeyVal arguments. However, note that currently the Ott-generated LaTeX output does not allow for this package to allow the same arguments being passed to commands for a group of **defn**s, i.e. **defns**.

References

[1] Strniša, R., and Parkinson, M. Lightweight Java. `http://www.cl.cam.ac.uk/~rs456/lj`, Sept. 2006.