The package `sdrt.sty`

Paul Isambert
zappathustra@free.fr
http://paulisambert.free.fr/

May 13, 2007

Abstract

This package is designed to help authors typesetting papers addressing SDRT (Segmented Discourse Representation Theory). Since SDRT is formal semantics, many of the macros in this package will be useful for logic in general (and DRT in particular, of course). Actually, I just wrote some simple macros to make life simpler, and gathered many useful symbols, that I rename for them to be easier to remember and to work both in math mode and in text.

Contents

0 Installation

This package must be installed and loaded in the usual way. It requires the `xyling.sty` package (already available in some LaTeX distribution, like MiKTeK 2.5) to be installed (but not loaded in your preamble), in order to draw trees. If, for some reason, you don't want to download it, just put % before `RequiresPackage{xyling}` at the beginning of `sdrt.sty`. You won't be able to draw trees anymore.

Apart from that, `xyling.sty` uses `xypic` with the `dvips` option to draw coloured branches. But then, when building directly to PDF, branches of the tree disappear, which is somewhat annoying. Thus, either you suppress the `dvips` option in line 57 of `xyling.sty`, keeping in mind that you won't be able to draw coloured branches anymore (and actually all branches will look ugly), or you create your PDF file via DVI PS, for instance (as I did for this documentation: to get everything as nice as possible, especially tables without bold lines, you should convert your .ps file via GSview, using *File>Convert*, with *Type: pdfwrite*, *Resolution: 300* - better resolution yields ugly tables; finally, don't forget to add the extension .pdf to the name of the output file, since it is not automatic). Anyway, the fine conversion to a PDF file is a problem in itself, to which I don't know any complete solution. For instance, horizontal lines in the boxes of this documentation are sometimes a bit too long in PDF (as can be seen in the first box above), though they are fine in PS, ending exactly at the vertical line.

1 Boxes

1.1 Renaming pi

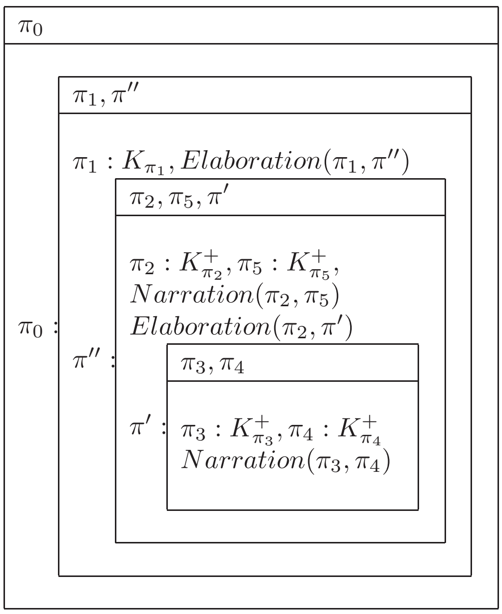
In SDRT, clauses are referred to with π and a subscript and/or a superscript. Thus, to print, for instance, π'_1, one has to write `$\pi^{\prime}_{1}$`, which is not impossible, but boring when typing it ten times a page. So I designed `\lab[]{}` (for *label*), which takes two arguments, to do the job. In the optional first argument (hence the brackets) you can place as many bars as you want, and the second one refers to the subscript. The subscript might be anything, and if you want none, leave this argument empty (but don't forget the braces).

Now, most labels have either a superscript, which is rarely more than four bars, or a subscript, which in general is a number from 0 to 9. So I wrote some commands to make life easier. Their names are easy to remember : `\labzero`, `labone`... `labnine` yield π_0, π_1... π_9, and `\labprime`, `\labsecond`, `\labthird`, `\labfourth` print π', π'', π''', π''''. This avoids excessive braces, and this will prove useful when building SDRSs. However, those commands eats subsequent space. When drawing a box or a tree, this won't be a problem, since in general they're followed by either a punctuation mark or nothing. Thus, no special care is required. On the other hand, in the course of a paragraph, gobbling of subsequent space is always annoying. If you want to use them anyway, a simple solution is to add a backslash at the end of the command. Thus, write `\labone\ is fine` to yield 'π_1 is fine'. Of course, don't use that backslash before a punctuation mark.

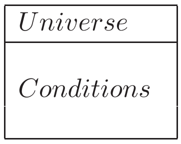
1.2 Building SDRSs

1.2.1 Boxes

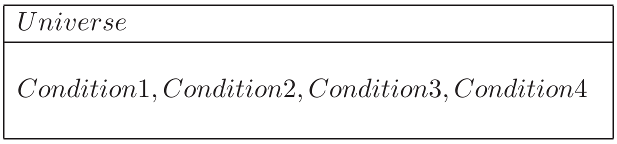
An SDRS look like this:



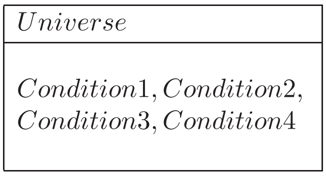
(This is the famous “Max’s great night” example.) We need the following command: `\SDRS`[1]. It takes two mandatory argument and an optional one. In the first mandatory argument, you put the so-called Universe of the (S)DRS (that is, the upper part of the box), and in the second one, the Conditions (the lower part of the box). Thus, `\SDRS{Universe} {Conditions}` will yield:



In the Universe and in the Conditions, you can put commas between the elements. However, although there is no big risk with the Universe, you might create ugly long lines in the Conditions, so you’d better break them with `\\`. So, instead of `\SDRS{Universe}{Condition1, Condition2,Condition3, Condition4}`, which gives:

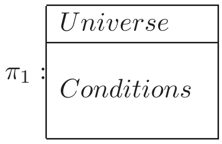


write `\SDRS{Universe}{Condition1, Condition2,\\Condition3, Condition4}` and you’ll get :

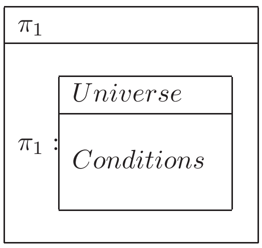


Moreover, Conditions in SDRT are on their own line in general, though you may put two on the same to save space. Whatever you decide, remember that legibility must be the rule, hence always write a condition containing another sub-box on a line alone, just like in the example above.

The optional argument (between brackets) is the label which is defined by the box. Thus, `\SDRS[\labone]{Universe}{Conditions}` for example prints the following:



Of course, you can put any structure into another one by writing it among the Conditions. So, for example, you can write `\SDRS{\labone}{\SDRS[\labone]{Universe}{Conditions}}` and yield:



Now, you have to be aware of the fact that everything in (S)DRSs is in math mode. And in math mode, everything is in italics and spaces between words is suppressed. It is exactly what we need when drawing usual (S)DRSs, but this might be problematic if we want something like this:

[1] `\drs` and `\sdrs` were part of the `covington.sty` package. I modified the code slightly and rewrote it for `sdrt.sty`, since I wanted better alignment in the boxes and generalized math mode. Moreover, `\sdrs` just printed a sentence above the box, and didn’t handle what `\SDRS` does. Finally, my command is written in capital letters, so it won’t conflict with `\sdrs` if you also use `covington.sty`.

π_1
π_1 : [John loves Mary]

If we just write `\SDRS{\labone}{\labone: [John loves Mary]}`, well, this will yield:

π_1
$\pi_1 : [John loves Mary]$

All we have to do is to add $'s around the sentence. Since math mode is defined by $...$ (automatically in this package), it is obvious that embedding another pair of $'s in the latter will produce two math modes with text mode in between. Thus, just write `\SDRS{\labone}{\labone: [$John loves Mary$]}` and everything will be fine. On the other hand, never write something like `$\alpha$` in a (S)DRS, since it would suppress the math mode greek letters need, for exactly the same reason. So just remember that (S)DRS are 'automatic math environment'.

1.2.2 Conditions

Now, we can build boxes as we want. But we must be able to write conditions of the form $\pi_2 : K_{\pi_2}$easily. The command for this is `\klab`, which works just like `\lab`, i.e. takes two arguments, one for the superscript (optional) and one for the subscript. Thus, $\pi_2 : K_{\pi_2}$ is typed out with `\klab{2}`. Just like `\lab`, `\klabzero`, `\klabone`... `\klabnine` will print $\pi_0 : K_{\pi_0}$, $\pi_1 : K_{\pi_1}$... $\pi_9 : K_{\pi_9}$. There is also the 'starred' version, when some underspecification is at stake : $\pi_3 : K^+_{\pi_3}$. So there's the code `\klabstar`, which works exactly like `\klab`, with the easy version too, that is `\klabstarzero`, `\klabstarone`, and so on[2].

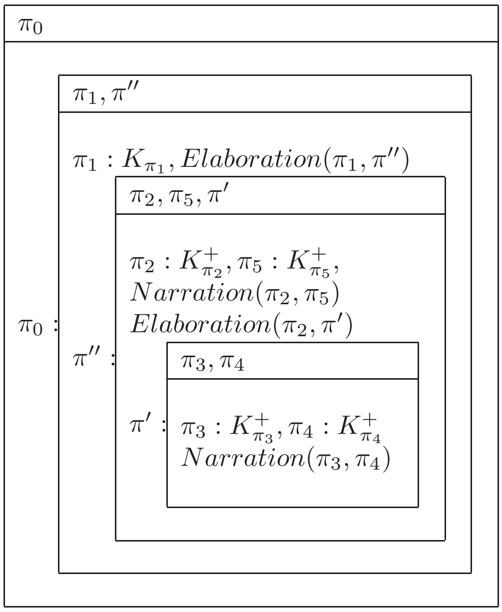
Finally, conditions of the form $Narration(\pi_2, \pi_5)$ are simply written with `Narration(\labtwo, \labfive)`. Since (S)DRSs are in math mode, you don't need to emphazise the name of the relation. This also means that in the course of your text, you have to add math mode, hence `$Narration(\labtwo, \labfive)$` to yield the same thing. If the arguments of you relation have only subscripts, there is a command, namely `\dr{Relation}{subscript1}{subscript2}`, which automatically produce the right form. Thus `\dr{Narration}{3}{5}` yields $Narration(\pi_3, \pi_5)$ in any environment.

1.2.3 Back to our example

With all these commands, we can build our example. Here is the code with the result :

```
\SDRS{\labzero}
      {\SDRS[\labzero]
         {\labone, \labsecond}{\klabone, Elaboration(\labone, \labsecond)\\
         \SDRS[\labsecond]
            {\labtwo, \labfive, \labprime}{\klabstartwo, \klabstarfive,\\
            Narration(\labtwo, \labfive\\
            Elaboration(\labtwo, \labprime)\\
            \SDRS[\labprime]
               {\labthree, \labfour}{\klabstarthree, \klabstarfour\\
               \dr{Narration}{3}{4}}}}}
```

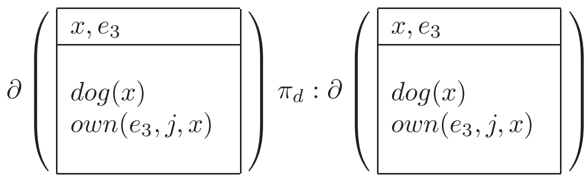
[2] I didn't designed `\klabprime` or `\klabstarprime` an so on like I did with `\labprime`, since barred labels in general refer to SDRSs and not to clauses. But they are easy to write with the `\klab` or `\klabstar` commands : `\klabstar[''']{}` for instance will print $\pi''' : K^+_{\pi'''}$.



This might seem complicated at first sight, but actually it's rather easy if you pay attention to braces. Of course you don't need to write the code with all these indents like I did here for visual convenience.

1.2.4 Some more stuff

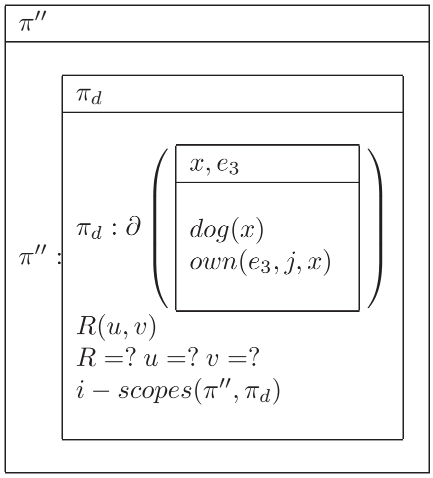
There is a 'presupposed' version of `SDRS` to produce boxes like the following:



`\PSDRS` is that command, and it works just like `\SDRS`, taking the same three arguments. If you want to use presupposition in text, type `\pres`, which takes one argument : for instance, `\pres{\varsub{K}{\lab{p}}}` yields $\partial(K_{\pi_p})$.

In this latter code there is an additional command `\varsub{}{}`. It is useful to type any kind of variable (or actually anything else) with a subscript. The first argument is the variable, the second is the subscript. Of course, it is recursive, so you can typeset $A_{B_{C_D}}$ with `\varsub{A}{\varsub{B}{\varsub{C}{D}}}`. Thus, e_3 in the boxes above is produced by `\varsub{e}{3}`.

Finally, predicates are created like discourse relations, that is `own(\varsub{e}{3},j,x)` for instance (if you aren't in a (S)DRS, you must add math mode, of course, to get the italics, or add them yourself). Note that you don't have to add a space after the comma, since math mode handle it as needed. Now we can produce an SDRS like the following:



Here is the code:

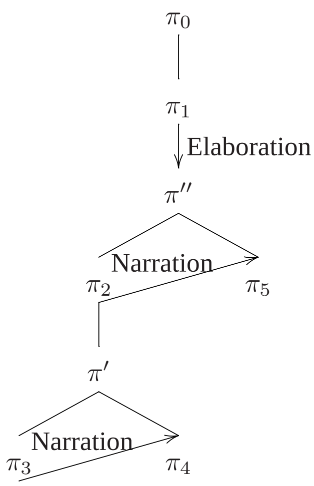
```
\SDRS{\labsecond}
     {\SDRS[\labsecond]
          {\lab{d}}{\PSDRS[\lab{d}]
               {x, \varsub{e}{3}}{dog(x)\\
               own(\varsub{e}{3},j,x)}\\
          R(u, v)\\
          R=?\ u=?\ v=?\\
          i-scopes(\labsecond, \lab{d})}}
```

Note that \ is necessary between $R =?$, $u =?$ and $v =?$, otherwise math mode will eat spaces between those conditions.

2 Trees

2.1 The commands

The most powerful package I know to draw trees is Ralf Vogel's `xyling.sty`. It is powerful but it needs some care. For instance, you can't produce an SDRT tree without adjusting the length of the branches and the alignment of the labels, otherwise you get something like this :



Obviously, that's not what we want to do. So I wrote some macros with the right adjustment. Before devising them, we need to know how exactly `xyling` works (for details, see the documentation of that package). A tree is made of nodes placed in a grid, which is like a tabular : & marks the passage to another column, while \\ begins another row. Here is an example to compare the output with the underlying grid:

A

```
        & A &   \\
B  &        & C
```

B C

In general, the code for the branches is written with the starting node (the mother or the leftmost sister), and the target node is specified as an argument. Now, here are the commands. `\sdrtree{}` is a kind of environment. The argument is the structure of the tree. `\LAB{}` denotes the node, whose name is the argument. Thus, for instance, with

```
\sdrtree{
                &\LAB{\labzero}  \\
                &\LAB{\labone}  \\
\LAB{\lab['']{p}}&              &\LAB{\labsecond}
}
```

we produce the following tree (I displayed the code with spaces for visual convenience, but of course you could write it on a single line with no space at all... although such a presentation avoids many errors with big trees):

π_0

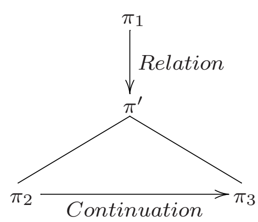
π_1

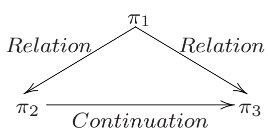
π''_p π''

Now we have to draw branches. `\cons` draws a vertical line from the mother (like π_0 in this example) to the sister (like π_1). `\consl` draws a line between a mother and a sister on the left (like between π_1 and π''_p) and `\consr` does the same with a sister on the right (like π'' if π_1 is the mother). `\srel{}`, `\srell{}` and `\srelr{}` work the same, except that they draw an arrow from the starting node to the target, and take an argument, which is the name of the (subordinating) discourse relation between the labels at the nodes[3]. Finally, `\crel{}` draws a horizontal arrow between two sisters with the name of the (coordinating) relation as the argument. Then, with the following code we have the following tree:

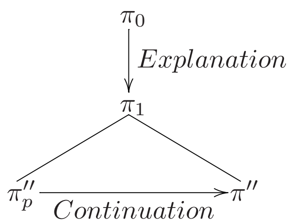
```
\sdrtree{
                                    &\LAB{\labzero}\srel{Explanation}  \\
                                    &\LAB{\labone}\consl\consr  \\
\LAB{\lab['']{p}}\crel{Continuation}&                              &\LAB{\labsecond}
}
```

[3]If there is a subordinating relation between, say π_1 and π_2, and the same relation between π_1 and π_3, usually in SDRT this relation holds between π_1 and an intermediate label like π', which in turn is made of π_2 and π_3 linked by at least a *Continuation* relation. So, in general, we have the first tree below but not the second one:

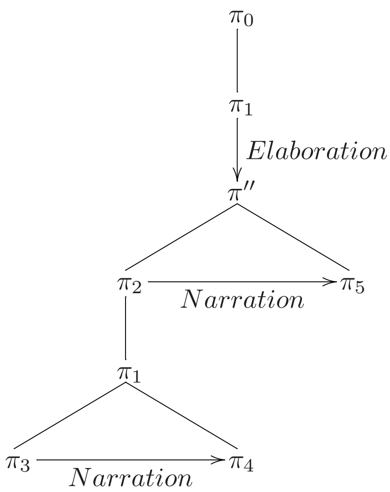




Thus, *srell* and *srelr* should be useless. But they aren't, since the analysis above might be discussed or at least might use trees like the second one to illustrate the demonstration.



And here is the tree drawn from our first big box:



And here is the code:

```
\sdrtree{                    &                                &\LAB{\labzero}\cons\\
                             &                                &\LAB{\labone}\srel{Elaboration}\\
                             &                                &\LAB{\labsecond}\consl\consr\\
                             &\LAB{\labtwo}\cons\crel{Narration}&                              &\LAB{\labfive}\\
                             &\LAB{\labone}\consl\consr\\
\LAB{\labthree}\crel[rr]{Narration}&                          &\LAB{\labfour}\\
}
```

2.2 The problem

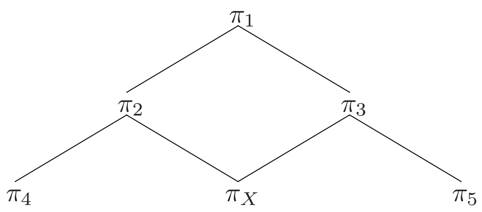
We can see that the code for a tree graphically simulates the structure of that tree: for instance, π_0 in the previous example is above π_1, which can be seen from the fact that they have the same number of &'s on the left. On the other hand, π_2 is a left sister of π'', and thus is one column left, i.e. π'' have one more & on its left. This is convenient, but it is also problematic. `xyling.sty`, and thus `sdrt.sty`, does not handle possible conflicts between nodes. To illustrate this, observe the following grid:

```
  &   & A &   &
  & B &   & C &
D &   & X &   & E
```

Obviously, X is B's right daughter and C's left one at the same time. If we create a tree with that structure, i.e. if we type the following code:

```
\sdrtree{     &                          &\LAB{\labone}\consl\consr\\
              &\LAB{\labtwo}\consl\consr&                           &\LAB{\labthree}\consl\consr\\
\LAB{\labfour}&                          &\LAB{\lab{}{X}}            &                          &\LAB{\labfive}\\
}
```

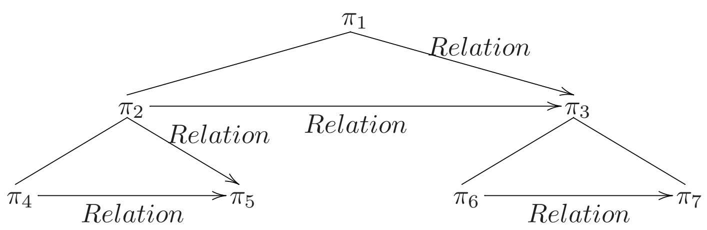
we produce the following tree:



It is a nice tree but ovbiously not of the kind that we need. So the question is: how can we draw a right daughter for π_2 and a left one for π_3 without merging them together? The answer is straightforward: add columns. That is, create the following grid:

```
  &   &   & A &   &   &
  & B &   &   &   & C &
D &   & X &   & Y &   & E
```

Up to now, this is ok. But branches have to be adjusted, otherwise they won't be able to reach their target. For instance, `\consl` starting from A won't reach B, but the position on the right of it (and an error message will be displayed, since there is no node here). Likewise, you won't be able to draw an arrow from B to C without modification. That is why `\cons`, `\srel` and `\crel` all have an optional argument between brackets. This argument is made of d's, l's and r's for 'down', 'left' and 'right' respectively: that's all we need to find the target. One d and you go down one row, two d's and you go down two rows, three r's and you go three columns right... In the grid above B is two columns left from A and one row below. So if you want a simple line from A to B, you type `\cons[dll]` next to A's node. If you want an arrow from B to C, you write `\crel[rrrr]{Relation}` next to B. Here is an example:



And here is the code:

```
\sdrtree{&&&\LAB{\labone}\cons[dll]\srel[drr]{Relation}\\
&\LAB{\labtwo}\consl\srelr{Relation}\crel[rrrr]{Relation}&&&&\LAB{\labthree}\consl\consr\\
\LAB{\labfour}\crel{Relation}&&\LAB{\labfive}&&\LAB{\labsix}\crel{Relation}&&\LAB{\labseven}\\
}
```

Of course, if π_5 had a right daughter and π_6 a left one, they would both be in the same column as π_1 and thus would merge together. In fact, you have to calculate the relative position of the nodes *before* you draw the tree, in order to know how many columns will be used. Fortunately, trees for discourse structures aren't syntactic trees and are in general far more simple, so drawing them is rather easy.

2.3 Definitions of the commands

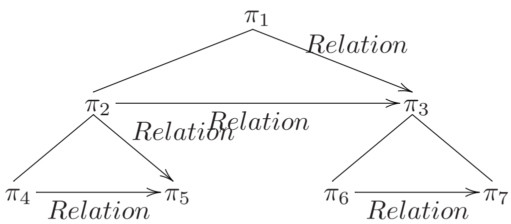
(This section might be skipped if you don't want to know how trees are defined in terms of the `xyling.sty` package and how to modify the adjustment.)

Here is the code for the commands above.

```
\newcommand{\sdrtree}[1]{\Treek[1]{2}{#1}}
\newcommand{\LAB}[1]{\K{ #1}}
\newcommand{\cons}[1][d]{\Bk{.5}{-2}{#1}}
\newcommand{\consl}{\Bk{1}{-2}{dl}}
\newcommand{\consr}{\Bk{1}{-2}{dr}}
\newcommand{\srel}[2][d]{\ARk{.5}{-2}{#1}^{$#2$}}
```

```
\newcommand{\srell}[1]{\ARk{1}{-2}{dl}_{$#1$}}
\newcommand{\srelr}[1]{\ARk{1}{-2}{dr}^{$#1$}}
\newcommand{\crel}[2][rr]{\GBkk{3,2.5}{-1.7,-3.5}{#1}{->}_{$#2$}}
```

I defined `\sdrtree` to have good-looking depth and width of the tree. If you want to modify them because they aren't satisfying to you, use `\Treek[width]{depth}{tree}` instead. For instance, here's the previous tree with a modified width:



I just replaced `\sdrtree` with `\Treek{2}`: since the width is an optional argument, not specifying it makes it 0. Thus, `\Treek{2}` is equivalent to `\Treek[0]{2}`. Note that negative values are allowed.

`\cons`, `\consl\consr` are made of `\Bk` which takes three arguments: vertical alignment of the starting node, vertical alignment of the target, and the direction as discussed above. `ARk` works the same. Finally, `\GBkk`, which is used to define `\crel`, has the following structure: the first argument specify the horizontal and vertical alignment of the starting node (seperated by a comma), the second argument specify the same thing for the target, the third argument is the direction, the fourth is the form of the arrow, and the last is the name of the relation. Finally, notice that `\LAB` has a space before its argument. In `xyling`, nodes are centered, but that centering don't look good with π when it has a superscript or a subscript. That extra space makes it look better, although it won't be nice with a bare π. In general, nodes in SDRT all have a sub- or superscript, so it's fine. However, if you don't want that space, juste use the original `\K` command, which is the usual one for nodes in `xyling`. Finally, note that the name of the relations are in math mode to get the right italics.

3 List of symbols used in SDRT

3.1 Notation index

I won't explain every symbol. Rather, I will reproduce the 'notation index' of Asher & Lascarides' *Logics of Conversation*, with the corresponding code. Comments in the left column are theirs.

1. Information Content: Object Language		
Variables denoting individuals	x, y,...	Use `$x$`, `$y$` and so on (math mode is useless in a (S)DRS, since it is automatically in math mode). If there is a subscript, use `\varsub{variable}{subscript}`.
Variables denoting eventualities	e_1, e_2,...	Use `\varsub{variable}{subscript}`.
Action terms	a_1, a_2,...	Use `\varsub{variable}{subscript}`.
Propositional terms	p, p_1,...	Use `\varsub{variable}{subscript}` or simply `$p$`.
The logical connectives and operators		I did not write any special macro for these, since they are very common. Moreover, a new command usually gobbles subsequent space and might conflict with other existing commands (since they're renamed in many packages). Don't forget math mode, or LaTeX will moan, except in a (S)DRS.

	$\wedge$
	$\vee$
	$\Rightarrow$
	$>$
	
	$\neg$
	$\square$
	
	$\Diamond$
		`$\wedge$`
		`$\vee$`
		`$\Rightarrow$`
		`$>$` (if you don't use math mode it will produce ¿).
		`$\neg$`
		`$\square$` (you have to load the `amsfonts` package in your preamble)
		`$\Diamond$` (you have to load the `wasysym` package in your preamble)

The proposition expressed by the formula K This symbol is not in the 'notation index' but it is the counterpart of the previous one, so it might be useful in formal semantics in general	${}^{\wedge}K$ ${}^{\vee}K$	`\intens{K}` or anything you want in the argument. `\extens{K}` or anything you want as the argument.
(S)DRSs	K_1, K_2,...	Use `\varsub{variable}{subscript}`.
The universe of discourse referents of the DRS K	U_K	`\varsub{U}{K}`.
The set of conditions of the DRS K	C_K	`\varsub{C}{K}`. Of course, with this one or the previous one, you could type something like $C_{\pi_2'}$ by putting `\lab[]{}` in the second argument hole.
The action of bringing it about that K is true	δK	`\true{K}` or anything you want as the argument.
A formula, conveying: if a (or δK) is performed, the ϕ necessarily (or possibly for $\langle a\rangle\phi$) follows.	$[a]\phi$, $[\delta K]\phi$, $\langle a\rangle\phi$, $\langle\delta K\rangle\phi$	`\necess{a}{\phi}` and `\possib{a}{\phi}`
K is a DRS, γ is a DRS condition, and $K^\frown\gamma =_{def} \langle U_K, Con_K \cup \gamma\rangle$	$K^\frown\gamma$	`\append{K}{\gamma}`. $=_{def}$ is just `\varsub{=}{def}`, $\langle$ and $\rangle$ are `\langle` and `\rangle`, all of them in math mode.
A DRS which summarises the content in K and K'	$K \sqcap K'$	`\summary`
labels for DRSs and action terms	α, β, ..., π_1, π_2,...	Use greek letters (in math mode) or `\lab`
An SDRS: A is a set of labels, $\mathcal{F}$ is a function which assigns labels in A SDRS-formulae, ans $LAST \in A$	$\langle A, \mathcal{F}, LAST\rangle$	`\aflast`. A and $LAST$ are of course the same letters in math mode, while $\mathcal{F}$ is `$\mathcal{F}$`, and $\in$ is `$\in$`

About $\mathcal{F}$: An expression like $\mathcal{F}(\pi_2)$ may be useful. So we have `flab[]{}`, which works once again exactly like `\lab`, i.e. optional primes as the first argument and subscript as the second. Similarly, `\fklab['']{2}`, for instance, yields $\mathcal{F}(\pi_2') = K_{\pi_2'}$, just like `\klab[]{}`. Finally, there is also an 'easy' version for both of them, namely `\flabone`, `\flabtwo`... `\flabnine` and `\fklabone`, `\fklabtwo`... `\fklabnine`. They also eats subsequent space, so use \ (e.g. `\flabnine\`) when needed.

Now, let's get back to our notation index:

The formula $\mathcal{F}(\pi_\alpha)$, that's labelled by α	K_α	Use `\varsub`. No math mode needed for α, since `varsub` automatically launches it when needed.
The main eventuality that's introduced in K_α	e_α	Use `\varsub`
Rhetorical relations	$\Downarrow$, *Narration*, *Contrast*,...	$\Downarrow$ is produced by `\topic`, but it gobbles subsequent space. So add a \ when it might be a problem. Other relations are just text in math mode.

The disputed counterpart to the relation R	$Dis(R)$	Simply `Dis(R)` in math mode, i.e. `$Dis(R)$`.
Label ϕ labels formula K (i.e., $\mathcal{F}(\pi) = K$)	$\pi : K$	This 'bare' version is simply `$\lab{} : K$`. For more elaborated stuff (i.e. with sub- and/or superscript), use `\klab` and `\klabstar`.
The formula representing the 'extra content', over and above K_α and K_β, that must be true (or, more accurately, that must update the context) for $R(\alpha, \beta)$ to update the context	$\phi_R(\alpha, \beta)$	`\varsub{\phi}{R}(\alpha, \beta)` in math mode.
An individual term denoting the agent who conveyed/uttered the content that's labelled α	$S(\alpha)$	`S(\alpha)` in math mode
Agent A believes that K	$\mathcal{B}_A(K)$	`\believes[content]{agent}`. The content is optional since we will need $\mathcal{B}_A$ later. By the way, $\mathcal{B}$ is produced with `\mathcal{B}` in math mode.
Agent A intends the action a	$\mathcal{I}_A(a)$	`\intends[action]{agent}`. the action is optional for the same reason as above. $\mathcal{I}$ is produced with `\mathcal{I}` in math mode.
The speech act related goal of the utterance labelled α is the action $\delta^\vee$p	SARG(α, p)	`\sarg{\alpha}{p}`. This command won't work in math mode, because of small capitals. So, although you might never use it, here is the code: `\scshape sarg\upshape\ensuremath{(#1, #2)}`. When in math mode, just add a $ before `\scshape` and between `\upshape` and `\ensuremath`.
2. Information Content: Metalanguage		
Possible worlds (in the model)	w, w', w_1, w_2,...	Use `$w$`, `$w'$` or `\varsub`.
Variable assignment functions	$f, g, ...$	Use math mode.
The domaine of f	$dom(f)$	`dom(f)` in math mode.
g extends f. I.e., $dom(f) \subseteq dom(g)$ and $\forall x \in dom(f), f(x) = g(x)$	$f \subseteq g$	Write `f \extends g`. By the way, the code for $\forall x$ is `\forall x` and the one for $\exists x$ is `\exists x`, both in math mode.
The formula (or action term) K relates the input context (w, f) with the output context (w', g)	$(w, f)[\![K]\!]_M(w', g)$	Use `\ccp[optional world index] {input pair}{formula}{output pair}`. If you happen to need [[and]], I designed `\Lbracket` and `Rbracket`, so you won't have to load any package.
Γ monotonically entails ϕ (model theory)	$\Gamma \models \phi$ or $\Gamma \models_f \phi$	Use `\entm[]` whose optional argument is the subscript.
Γ monotonically entails ϕ (proof theory)	$\Gamma \vdash \phi$ or $\Gamma \vdash_f \phi$	Use `\entp[]` whose optional argument is the subscript.
3. Underspecified Information Content: The Language $\mathcal{L}_{ulf}$		
		First of all, $\mathcal{L}_{ulf}$ is typed with `\lulf`, which eats subsequent space, so use an extra `\`.

The translation function form the ULFs to the unlabelled language	ν	`\trfunc`
Labels	l_1, l_2,...	Use `\varsub`.
Variables over labels	$?_1$, $?_2$,...	Use `\varsub`.
Higher order variables	$X, Y, R...$	`X, Y, R` in math mode or `\varsub` if there is a subscript.
The predicate corresponding to the constructor f from the base (unlabelled) language	R_f	Use `\varsub`.
A notational variant of $R_f(l_1,..., l_{n+1})$, where l_i labels x_i, $1 \leq i \leq n$; e.g., $l : \wedge(p,q)$ is shorthand for $R_\wedge(l_q, l_p, \text{l}) \wedge p(l_p) \wedge q(l_q)$	$l_{n+1} : f(x_1,..., x_n)$	All those notations are just an efficient use of `varsub`. Note that you can write anything as the second argument, so for instance `\varsub{R}{\wedge}` produce $R_\wedge$.
Gloss for $\exists Y(R_=(l_x, l_y, l) \wedge R_x(l_x) \wedge Y(l_y))$	$x = ?$	Simply `$x =?$`, and once again `varsub` for the notations in the left column.
Label l outscopes l'	$l \succ l'$	`\outscopes`.
The conditions in l are accessible to those in l'	$l \succ_a l'$	`\varsub{\outscopes}{a}`
4. Underspecified Information Content: Metalanguage		
The set of all labels in the model	U	Just `U` in math mode.
Successor relation on labels (corresponds to *immediately outscopes*).	$Succ$, $Succ_D$	Use `Succ` in math mode or `\varsub`.
The interpretation fonction	I	Just `I` in math mode.
The satisfaction relations of the labelled language (this is different from $\models_f$)	$\models_l$	`\entm[l]`.
5. Glue Logic: Object Language		
A ULF (which in the glue language forms a one-place predicate)	$\mathcal{K}$	`\ulf`
Individual variables	$x, y, ...$	Use math mode.
Labels	π_1, π_2, $\alpha, \beta...$	`\lab` and greek letters.
An example of a formula that's transferred via $\vdash$tr into the glue language from other more expressive languages (e.g., from the logic of information content)	$push(e, x, y, \pi_2)$	Use math mode and simply write your text.
The SDRSK_l (i.e., (λ)) includes as a conjunct some rhetorical relation connecting α and β	$?(\alpha, \beta, \lambda)$	Same as above: math mode!
in the SDRS$\langle A, \mathcal{F}\rangle$, where $l \in A$, $\mathcal{F}(\lambda)$ includes $R(\alpha, \beta)$ as one of its conjuncts.	$R(\alpha, \beta, \lambda)$	Once again: math mode!
As in the language of information content	$\wedge, \vee, \rightarrow, \neg, >$	As above, except that $\rightarrow$ is `\rightarrow` (i.e., without a capital letter).
The information about content that's transferred from $\mathcal{K}$ into the glue logic, where $\mathcal{K}$ is a set of formulae of the ULF-logic	$Info(\mathcal{K})$	`Info(\mathcal{K})` in math mode.

σ outscopes α and nothing outscopes σ	$Top(\sigma, \alpha)$	Simple text in math mode.
There is evidence in the discourse σ that α is a subtype of β; similarly for $cause_D(\sigma, \alpha, \beta)$	$subtype_D(\sigma, \alpha, \beta)$	Use `varsub`
A schema, which one can replace with the aktionsart of α and β, whatever their values	$Aspect(\alpha, \beta)$	Text in math mode.
The formula α' labels is just like that labelled by α, save that the former resolves some or all of the underspecifications that's present in the latter.	$\alpha \rightsquigarrow \alpha'$	This arrow is produced with `\resolves`.
A DRS which is the same as K, save that some of the underspecified conditions in K are resolved in K^+	K^+	Use `\kstar`, which can be an argument of `\varsub`, so you can write, for instance, $K^+_{\pi'_5}$ with `\varsub{\kstar}{\lab[']{5}}`.
At the part labelled λ_2 in the discourse structure, the content K_{λ_1} that λ_1 labels (and which in turn is outscoped by λ_2) is settled.	$settled(\lambda_1, \lambda_2)$	Use text in math mode and `\varsub`.
Type declarations, respectively: α labels an indicative, interrogative, imperative	$\alpha : \vert, \alpha :?, \alpha :!$	Simple math mode once again.
	6. Glue Logic: Metalanguage	
Γ monotonically entails ϕ (model theory)	$\Gamma \models \phi$ or $\Gamma \models_g \phi$	Use `\entm` with optional subscript (between brackets).
Γ monotonically entails ϕ (proof theory)	$\Gamma \vdash \phi$ or $\Gamma \vdash_g \phi$	Use `\entp` with optional subscript (between brackets).
Γ nonmonotonically entails ϕ (model theory)	$\Gamma\vert\!\approx\phi$ or $\Gamma\vert\!\approx_g\phi$	Use `\nmentm` with optional subscript (between brackets).
Γ nonmonotonically entails ϕ (proof theory)	$\Gamma\vert\!\sim\phi$ or $\Gamma\vert\!\sim_g\phi$	Use `\nmentp` with optional subscript (between brackets).
An extension of the theory T	$T^{\rightarrow}$	`\thext`, which of course can be argument of `\varsub` to produce things like $T^{\rightarrow}_{max}$ as usual.
$Ant(T) =_{def} \{C : T \vdash C > D\}$	$Ant(T)$	Here is how to write the formula in the left column: `Ant(T)\varsub{=}{def}\{C:T\entp C>D\}` The whole in math mode, of course. As you can see, the only thing you have to pay attention to is the braces, which are one of the special characters of LaTeX. To typeset them, you have to write `\{` and `\}`.

7. Discourse Update		
The transfer relation from (richer) sources of information to the glue language	$\vdash_{tr}$	`\entp[tr]`
The set of labels to which β is attached	$att-sites(\beta)$	Text in math mode.
The set of available attachment sites in the set of SDRSs σ	$avail-sites(\sigma)$	Text in math mode.
$\{\langle\alpha,l\rangle : \alpha\in avail-sites(\sigma)$ and $Succ_D(l,\alpha)\}$	$avail-pairs(\sigma)$	Text in math mode. The left column is written just like the definition of $Ant(A)$. Note that 'and' mustn't be in math mode, so you have to stop it before and start it again after.
The set of all possible sequences of all possible subsets of $avail-pairs(\sigma)$	$\mathcal{P}(avail-pairs(\sigma))$	$\mathcal{P}$ is `\mathcal{P}` in math mode, and you must have guessed how the rest was typed...
The SDRT update function from an old context and new information to a new context.	*update*$_{\text{SDRT}}$	Use `\update`. Note that this was designed thanks to the `subscript.sty` package. I rewrote that part of the code in `sdrt.sty` so you won't have to (down)load it. By the way, this won't work in math mode. To yield *Best-update*$_{\text{SDRT}}$, write `\bestupdate`.
A set of SDRSs	σ	Greek letter sigma.
The set of all ULF-formulae ϕ such that for all SDRSs in σ, $s\models_l\phi$	$Th(\sigma)$	Math mode
The simple update of σ with the (assumption about) attachment $?(\alpha,\beta,\lambda)$	$\sigma+?(\alpha,\beta,\lambda)$	Math mode.
The sequence of simple updates of σ with $?(\alpha,\beta,\lambda)$ for each $\langle\alpha,l\rangle\in X$	$\Sigma_X(\sigma,\mathcal{K}_\beta)$	This might seem complicated, but this is not. Here is the code: `\varsub{\Sigma}{X}` `(\sigma, \varsub{\mathcal{K}}{\beta})`
Downdating: the set of the biggest bits of σ that you can retain while ensuring that the result does not entail ϕ.	$\sigma\downarrow\phi$	Use `\downdate` to draw $\downarrow$.
σ with all $R(\gamma,\alpha,\lambda)$ where $\phi(R)$ retracted, and replaced with $Dis(R)(\gamma,\alpha,\lambda)$	$\sigma\Downarrow_\phi\alpha$	Use `\varsub{\topic}{\phi}` to yield $\Downarrow_\phi$.
Simple revision (which generalises update)	$\sigma\otimes?(\alpha,\beta,\lambda)$	`\revision` to produce $\otimes$.
8. Cognitive Modelling Language		
There is nothing new in that section. Everything is made of `\varsub` or math mode. You already know that $\mathcal{B}$ is `\mathcal{B}` in math mode.		
Propositional variables	p_1,p_2,q,q',...	
Action terms	a_1, a_2,...	
Labelled propositional variables	p_α, p_π	Of course, you could write something like $p_{\pi'_r}$ with `\varsub{p}{\lab[']{r}}`.

Labelled action terms	a_α, a_π	Same comment.
An action term, corresponding to the action of seeing to it that ϕ is true	$\delta\phi$	`\true{}` as above.
The speaker who conveyed the content associated with α; and the hearer of that content	$S(\alpha), H(\alpha)$	
Agent A believes that; Agent A intends that; A and B mutually believe that	$\mathcal{B}_A$, $\mathcal{I}_A$, $MB_{A,B}$	Use `believes{agent}` and `\intends{agent}` without the optional argument. $MB_{A,B}$ is simply `\varsub{MB}{A,B}`.
A's choice for fulfilling the action $\delta\psi$ is to carry out the action $\delta\phi$	$choice_A(\phi,\psi)$	`\varsub` and math mode.
The action of $S(\alpha)$ uttering α	$Say(\alpha)$	Use math mode.
The action a has been performed	$Done(a)$	Use math mode.
p is an answer to the question labelled by α	$Sanswer(\alpha,p)$	Use math mode.

3.2 Additional symbols

Wandering through *Logics of Conversation*, one can realize that the above notation index is not sufficient to typeset all formulae in SDRT . So here are some more useful symbols.

First of all, a 'superscript' variant of `\varsub{}{}` will be interesting. It is simply `\varsup{}{}`. So you can type, for instance, $\mathcal{K}^{sup}$ with `\varsup{\ulf}{sup}`. Note that `\varsub` and `\varsup` can be arguments of each other. So you can type complex stuff like $\mathcal{K}^{sup}_{sub}$ with `\varsub{\varsup{\ulf}{sup}}{sub}`. Note that `\varsup{\varsub{\ulf}{sub}}{sup}` will yield exactly the same thing.

Now, here are some more symbols, with the code:

$\cup$	`\cup` in math mode
$\circ$ (to define $[\![a_1; a_2]\!]$)	`\circ` in math mode
$K_1 \leq K_2$ (accessibility relation)	`\access`
$K := Definition$	Simply `:=`
ℓ (in models for $\mathcal{L}_{ulf}$)	`\ell` in math mode
$e \prec now$ (temporal precedence)	`\tempprec`
$\mathcal{X}$	`\mathcal{X}` in math mode
$\frac{R_f}{Y}$(in the interpretation of the labelled language)	`frac{above}{below}` in math mode
Negated versions of inference operators:	
$\not\models$	`\Nentm`
$\nvdash$	`\Nentp`
$\mid\!\not\approx$	`\Nnmentm`
$\mid\!\not\sim$	`\Nnmentp`
$\bigcup_{X\in\mathcal{S}_\sigma}$ (in SDRT Update)	`\union{limit}`
$\alpha \sqcup \beta$	`\merging`
$x \sqsubseteq y$	`\subtype`

Many relations can be negated with the prefix `\not` (which needs math mode). Thus `\not\extends` yields $\not\sqsubseteq$ and `\not\in` yields $\notin$. Finally, if you want to draw HPSG-like AVMs for lexical semantics, use Christopher Manning's `avm.sty` package.

4 Math mode or not?

I am aware of the fact that the many mentions of 'math mode' might be very confusing, and that in the end you might not know when to use it. Moreover, maybe you are a new LaTeX user and you ignore what math mode is and why so many $ are appearing here and there along these pages. So first of all, a definition: math mode is a pair of $ between which math formulae are nicely formatted. So it is good. However, there is another feature that I can't explain to me: some commands (those in the menu item named 'math' in TeXnic-Center) *need* math mode. Greek letters for instance. If you write `\alpha is a nice letter`, it will type 'α is a nice letter', but, since `\alpha` is not surrounded by $, LaTeX will moan '`Missing $ inserted`', and you'll have two errors. Fortunately, all the macros in this package 'control' their 'math-modality'[4].

The following commands don't need math mode, nor do their argument(s) need it. For instance, `varsub{}{}` don't need math mode and you don't need to write `\alpha` between $ if you want α as one of the arguments.

`\lab[]{}`, and all its variants: `\labone`, `\klab`, `\flab`, etc.

```
\SDRS \PSDRS
\varsub{}{} \varsup{}{}
\intens{} \extens{} \true{}
\necess{}{} \possib{}{}
\append{}{}
\summary
\aflast
\topic
\believes[]{} \intends[]{}
\sarg{}{}
\extends
\ccp[]{}{}{}
\entm[] \entp[] \nmentm[] \nmentp[] \Nentm[] \Nentp[] \Nnmentm[] \Nnmentp[]
\lulf
\trfunc
\outscopes
\ulf
\resolves
\kstar
\thext
\downdate \revision \access \tempprec
\union{}
\merging
\subtype
```

As we have seen above with (S)DRSs, math mode has side-effects that you might want to avoid. For instance, normal text will be in italics and without space between words. So you have to interrupt math mode when needed (though normally you won't need it much in SDRT), with additional $. Thus, for instance, $\mathcal{B}_A(\text{my sentence})$ is typeset with `\believes[$my sentence$]{A}`.

On the other hand, greek letters, logical connectors, various calligraphic letters (i.e. produced with `\mathcal{}`) and the symbols $\in$ (`\in`), $\cup$ (`\cup`), $\circ$ (`\circ`), ℓ (`\ell`) $\frac{above}{below}$ (`\frac{above}{below}`) need math mode. That is, either they're written between $ or they're arguments of one of the commands above. Thus you'll write `\intens{\alpha}` and *never* `\intens{$\alpha$}`, or `$\alpha \outscopes \beta$` (although `\outscopes` doesn't need it, it won't cause any trouble).

The advantage of automatic math mode is that those commands are launched in the same way in math environment or in text: `\outscopes` produces $\succ$ in the last example and in a phrase like 'The $\succ$ relation'. Just note that in text, those commands that don't take arguments will eat subsequent space, so actually you have to write '`the \outscopes\ relation`' when space is needed. Finally, variables without `\varsub` or `\varsup`, as well as predicates, need math mode (or any of the commands above) to be typed properly, i.e. if you write just `own(x, y)`, you will get 'own(x, j)' and not '$own(x, j)$'.

[4] Thanks to the `\ensuremath` command.

5 Bugs and enhancements

5.1 Problems

I made the symbols for non-monotonic entailment out of two other symbols: $|$ and $\approx$ for $|\!\approx$ and $|$ and $\sim$ for $|\!\sim$. I looked for them everywhere, but I wasn't able to find them, that's why I designed them that way (since I don't know how to draw glyphs). They seem to work well, but they might sometimes mess up when LaTeX adjusts the filling of a line, especially in tables, so you might have to work out some adjustment yourself. Note that it sometimes moves from PS to PDF. That's the reason why I did not designed a nicer $\vdash$ whose branches would be of the same lenghth as those of $\models$ (notice by the way that in every SDRT papers that I read, $\vdash$ never matched the lenghth of $\models$; but *Logics of Conversation*, at least, was explicitely done with LaTeX). The same holds for $[\![$ and $]\!]$. Although they exist in some packages, they didn't look good to me, and anyway I wanted to avoid requiring many packages.

There is another problem, but this one seems to pervade through TeX in general, namely the 'double subscript' problem. If you want to print a complex stuff like $\models{}_{l}^{g^{\frac{\ell}{v}}}$ (which is needed in the interpretation of the labelled language), you can type `\varsup{\varsub{\entm}{l}}{\varsup{g}{\frac{\ell}{v}}}`, but you will have one error ('double subscript'). Moreover, the sub- and the superscripts are not next to the entailment symbol. If you 'recreate' $\models$ out of $|$ and $=$ (as I did for $|\!\sim$ and $|\!\approx$), however, you will have no problem. `\varsup{\varsub{|\hspace{-5pt}=}{l}}{\,\varsup{g}{\frac{\ell}{v}}}` will print $\models_{l}^{g^{\frac{\ell}{v}}}$.

5.2 Things that could be improved

A 'generalized' math mode could be interesting. I didn't renamed the logical operators nor the greek letters, since you might use many packages, and it could conflict with them. But here is a simple way to use math symbols in both math and text modes. Imagine you want α to work so, for instance. Then create a new command, namely `\newcommand{\Alpha}{\ensuremath{\alpha}}`. Of course, you could name it whatever you want, and '`\Alpha`' is just an example. With that command, you won't have to bother with math mode anymore, it will be automatic when needed. Notice that a command of the form `\newcommand{\Alpha}{$\alpha$}` would not do: in math mode, it would create an inner pair of $ that would interrupt it, and thus the greek letter would be in text mode. On the contrary, `\ensuremath{}` does not launch math mode when already in it. The only problem is that commands of that kind (without argument) eat subsequent space (and thus may require a suffixed \). You could use the `xspace.sty` package, that controls when subsequent space is needed or not. I didn't use it because it yielded bad results with predicates (the right parenthesis was preceded by a blank).

Apart from that, you might have noticed that the arrowheads in trees don't resemble the ones in SDRT. There is no such arrowheads in `xypic`, and I'm not able to draw them. This would be nice however if it could be done, but it would require another drawing package, and hence rewriting another code for the trees.

Finally, I did not attempt at drawing the diamond-shaped box that one encounters in DRT to handle donkey sentences, because I was not able to draw them properly. Note however that the `xytree.sty` package, which requires `xypic` too, has a command `\drsdiabox` to draw them. There are two problems with `xytree.sty`: first, its `\drsdiabox` command yields a shivering box. I think the reason is that this package requires `xypic` *without* the dvips option. Thus, there is no problem with PDFTeX, but all diagonal lines are ugly. Moreover, the diamond box is not stuck to the other boxes, as it should be. I think however that it is easy to fix. The second problem is the following. Compare those two boxes:

x, j
dog(x) *own(j,x)*

x, j
$dog(x)$ $own(j, x)$

The boxes themselves are not at stake. But if you take a look at the shape of the text, you can observe that there are two kinds of italics. The ones in the left box (made with `xytree`) are produced with the `\itshape` command, while the italics in the right box (made with `sdrt`) are the result of math mode. The fact is that all italics in SDRT papers, either in a box or in text, are produced with math mode, and not with `\itshape` or `\emph{}`. See the difference:

`$background$`	$background$
`\emph{background}`, `\itshape background` and `\textit{background}`	*background*
`\slshape background`	*background*

Math mode also prevents parentheses from being in italics, as usual with math formulae. Thus, `xytree` is not adequate to draw proper boxes.