The bsymb package*

Laurent Voisin
Systerel

September 16, 2009

Abstract

This package provides macros for typesetting the operators of the Event-B language. It was developed at the Swiss Federal Institute of Technology Zurich.

Contents

1 Introduction

This package was developed in order to ease the typesetting of Event-B formulas in LaTeX. Particular care has been taken to provide macros that fit well into TeX algorithm for typesetting mathematical formulas, so that the user seldom needs to correct the spacing computed by TeX (using thin spaces or the like).

*This document corresponds to bsymb v1.9, dated 2009/09/16.

A former package with the same objectives was provided with Atelier B (a support tool for the B language distributed by ClearSy), but that package is now quite obsolete (it was designed to work with LaTeX 2.09) and has never been upgraded since.

2 Usage

Just like any other package, you need to request this package with a `\usepackage` command in the preamble. This package doesn't take any option per se. However, it internally uses package `amssymb`. So if one needs to pass special options to that latter package, one has to either request that package before requesting `bsymb` or otherwise, and in a more simpler way, pass those options directly to the `bsymb` package, which will in turn pass them to `amssymb`.

So, in the simpler case, one just types

```
\usepackage{bsymb}
```

and when one wants to use the `psamsfonts` option of `amssymb`, one uses

```
\usepackage[psamsfonts]{bsymb}
```

The rest of this section presents the macros to use for typesetting B formulas. Those macros are either defined by LaTeX, by `amssymb` or by this package. The macros are sorted by context, so that it's easier to seek the name of a macro when knowing the corresponding operator usage.

Each table contains three columns. The first one displays the symbol obtained from the macro, the second one the macro name. Finally, the third column displays a usage example of the macro.

Table 1: Predicate operators

$\bot$	`\bfalse`	$\bot$
$\top$	`\btrue`	$\top$
$\neg$	`\lnot`	$\neg P$
$\land$	`\land`	$P \land Q$
$\lor$	`\lor`	$P \lor Q$
$\Rightarrow$	`\limp`	$P \Rightarrow Q$
$\Leftrightarrow$	`\leqv`	$P \Leftrightarrow Q$
$\forall$	`\forall`	$\forall x \cdot P \Rightarrow Q$
$\cdot$	`\qdot`	$\forall x \cdot P \Rightarrow Q$
$\exists$	`\exists`	$\exists x \cdot P$

Table 2: Set relations

$\in$	`\in`	$x \in y$
$\notin$	`\notin`	$x \notin y$
$\subseteq$	`\subseteq`	$x \subseteq y$
$\not\subseteq$	`\not\subseteq`	$x \not\subseteq y$
$\subset$	`\subset`	$x \subset y$
$\not\subset$	`\not\subset`	$x \not\subset y$
partition	`\partition`	$\mathrm{partition}(S, s_1, s_2)$

TABLE 3: Definition

$\widehat{=}$	`\defi`	$x \mathrel{\widehat{=}} y$

TABLE 4: Equality and arithmetic relations

$=$	`=`	$x = y$
$\neq$	`\ne`	$x \neq y$
$<$	`<`	$x < y$
$\leq$	`\le`	$x \leq y$
$>$	`>`	$x > y$
$\geq$	`\ge`	$x \geq y$

TABLE 5: Basic set constructs

$\mapsto$	`\mapsto`	$x \mapsto y$
$\times$	`\cprod`	$x \times y$
$\mathbb{P}$	`\pow`	$\mathbb{P}(x)$

TABLE 6: Derived set constructs

$\cup$	`\bunion`	$x \cup y$
$\cap$	`\binter`	$x \cap y$
$\setminus$	`\setminus`	$x \setminus y$
$\{$	`\{`	$\{x, y, z\}$
$\}$	`\}`	$\{x, y, z\}$
,	,	$\{x, y, z\}$
$\mid$	`\mid`	$\{x \cdot P \mid E\}$
$\varnothing$	`\emptyset`	$\varnothing$
$\mathbb{P}_1$	`\pown`	$\mathbb{P}_1(x)$
inter	`\inter`	$\mathrm{inter}(x)$
union	`\union`	$\mathrm{union}(x)$
$\bigcap$	`\Inter`	$\bigcap x \cdot P \mid E$
$\bigcup$	`\Union`	$\bigcup x \cdot P \mid E$

TABLE 7: Binary relation constructs (first series)

$\leftrightarrow$	`\rel`	$x \leftrightarrow y$
$\nleftrightarrow$	`\trel`	$x \nleftrightarrow y$
$\twoheadleftarrow$	`\srel`	$x \twoheadleftarrow y$
$\nleftrightarrow$	`\strel`	$x \nleftrightarrow y$
dom	`\dom`	$\mathrm{dom}(x)$
ran	`\ran`	$\mathrm{ran}(x)$
;	`\fcomp`	$x ; y$
$\circ$	`\bcomp`	$x \circ y$
id	`\id`	$\mathrm{id}(x)$
$\lhd$	`\domres`	$x \lhd y$
$\not\lhd$	`\domsub`	$x \not\lhd y$
$\rhd$	`\ranres`	$x \rhd y$
$\not\rhd$	`\ransub`	$x \not\rhd y$

TABLE 8: Binary relation constructs (second series)

[	`[`	$r[s]$
]	`]`	$r[s]$
$\leftarrow\!\!\!\!\!+$	`\ovl`	$x \leftarrow\!\!\!\!\!+ y$
$\otimes$	`\dprod`	$x \otimes y$
prj_1	`\prjone`	$\mathrm{prj}_1(x, y)$
prj_2	`\prjtwo`	$\mathrm{prj}_2(x, y)$
$\parallel$	`\pprod`	$x \parallel y$

TABLE 9: Function constructs (first series)

$\rightarrow\!\!\!\!\!\!\mid$	`\pfun`	$x \rightarrow\!\!\!\!\!\!\mid y$
$\rightarrow$	`\tfun`	$x \rightarrow y$
$\rightarrowtail\!\!\!\!\!\!\mid$	`\pinj`	$x \rightarrowtail\!\!\!\!\!\!\mid y$
$\rightarrowtail$	`\tinj`	$x \rightarrowtail y$
$\twoheadrightarrow\!\!\!\!\!\!\mid$	`\psur`	$x \twoheadrightarrow\!\!\!\!\!\!\mid y$
$\twoheadrightarrow$	`\tsur`	$x \twoheadrightarrow y$
$\rightarrowtail\!\!\!\!\!\!\twoheadrightarrow$	`\tbij`	$x \rightarrowtail\!\!\!\!\!\!\twoheadrightarrow y$

TABLE 10: Function constructs (second series)

λ	`\lambda`	$\lambda x \cdot P \mid E$
(	`(`	$f(x)$
)	`)`	$f(x)$

Table 11: Integer constructs

$\mathbb{N}$	`\nat`	$\mathbb{N}$
$\mathbb{N}_1$	`\natn`	$\mathbb{N}_1$
$\mathbb{Z}$	`\intg`	$\mathbb{Z}$
..	`\upto`	$x \mathinner{..} y$
finite	`\finite`	finite(x)
card	`\card`	card(x)

Table 12: Arithmetic operators

pred	`\upred`	pred(x)
succ	`\usucc`	succ(x)
$+$	`+`	$x + y$
$-$	`-`	$x - y$
$*$	`*`	$x * y$
$\div$	`\div`	$x \div y$
mod	`\bmod`	$x \bmod y$
$\mathbin{\widehat{}}$	`\expn`	$x \mathbin{\widehat{}} y$

Table 13: Boolean operators

BOOL	`\Bool`	BOOL
TRUE	`\True`	TRUE
FALSE	`\False`	FALSE
bool	`\bool`	bool(P)

Table 14: Substitutions

$:=$	`\bcmeq`	$x := y$
$:\in$	`\bcmin`	$x :\in y$
$:\mid$	`\bcmsuch`	$x :\mid y$

Table 15: Typing operator

$\mathbin{⦂}$	`\oftype`	$(\varnothing \mathbin{⦂} \mathbb{P}(S))$

3 Implementation

The implementation is quite straightforward. It's just a matter of loading the `amssymb` package and then defining new macros for each symbol. In some rare cases, we need to define a new name just to change the category of a mathematical symbol (for example, to transform a relation symbol into a binary symbol).

So, let's start by requesting the `amssymb` package.

```
\RequirePackageWithOptions{amssymb}
```

Then, we define some little helper macros to factor repetitive parts. The first macro allows us to define a new named operator, like succ or card. The second one is used to defined an ordinary symbol such as $\mathbb{N}$ or $\bot$.

```
\newcommand\bsymb@defop[2]{
  \newcommand{#1}{\mathop{#2}\nolimits}
}
\newcommand\bsymb@deford[2]{
  \newcommand{#1}{\mathord{#2}}
}
```

`\bfalse` `\btrue` Now, let's start with the predicate operators. Those are really easy, most of them are already defined by LaTeX. We just need to complement.

```
\bsymb@deford{\bfalse}{\bot}
\bsymb@deford{\btrue}{\top}
```

`\limp` `\leqv` An important point here is that the `\implies` and `\iff` commands of LaTeX are inappropriate. The associated glyph is too large and, moreover, those symbols are defined as relations and we need binary operators here. Hence, we define those two macros to replace them.

```
\newcommand{\limp}{\mathbin\Rightarrow}
\newcommand{\leqv}{\mathbin\Leftrightarrow}
```

`\qdot` Another fine point is that the `\cdot` command of LaTeX produces a binary operator which is not appropriate for typesetting quantified formula (there is too much space around the dot). So, we define a new command for typesetting those dots, tuning finely the space to put around it.

```
\bsymb@deford{\qdot}{\mkern1mu\cdot\mkern1mu}
```

`\partition` The partition operator is straightforward.

```
\bsymb@defop{\partition}{\mathrm{partition}}
```

`\defi` The definition operator is entered as a relation in order to have correct spacing around it.

```
\newcommand\defi{\mathrel{\widehat=}}
```

For equality, set and arithmetic relations, things are even easier, all macros are already defined by LaTeX. No need to add anything.

`\pow` `\pown` `\cprod` `\bunion` `\binter` `\union` `\inter` `\Union` `\Inter` Let's proceed now to set constructs. There is just one little trick here: I added a `\hbox{}` to the `\pow` macro so that the operator doesn't appear to TeX as a single letter. Without that empty box, the powerset operator is drawn a little below the baseline, which is quite ugly.

```
\bsymb@defop{\pow}{\mathbb P\hbox{}}
```

```
\bsymb@defop{\pown}{\mathbb P_1}
\newcommand{\cprod}{\mathbin\times}
\newcommand{\bunion}{\mathbin{\mkern1mu\cup\mkern1mu}}
\newcommand{\binter}{\mathbin{\mkern1mu\cap\mkern1mu}}
\bsymb@defop{\union}{\mathrm{union}}
\bsymb@defop{\inter}{\mathrm{inter}}
\newcommand{\Union}{\bigcup\nolimits}
\newcommand{\Inter}{\bigcap\nolimits}
```

`\emptyset` There is one symbol which case is special: the empty set symbol. It's already defined in LaTeX, but the glyph used looks ugly to most people. So, we just redefine the LaTeX command associating an alternative glyph to the symbol.

```
\renewcommand{\emptyset}{\mathord\varnothing}
```

`\rel` We now proceed to binary relation constructs. The first one is easy

```
\newcommand{\rel}{\mathbin\leftrightarrow}
```

`\trel` `\srel` `\strel` Then, for the next ones, we need to do some trickery to build new symbols. The idea is quite simple, just draw two symbols that overlap in order to build the new symbol. The overlapping is done using some negative glue. There are nevertheless some important points to consider, otherwise things will turn bad. First, dimensions must be expressed using the `mu` unit, otherwise they won't scale correctly when one changes the font size. The second point is to take care of the apparent edges of our new symbol. As we use negative glue, we can easily drive TeX into messing things up. Just try to invert the two symbols in the definitions below and look at the result.

```
\newcommand{\trel}{\mathbin{\leftarrow\mkern-14mu\leftrightarrow}}
\newcommand{\srel}{\mathbin{\leftrightarrow\mkern-14mu\rightarrow}}
\newcommand{\strel}{\mathbin{\leftrightarrow\mkern-14mu\leftrightarrow}}
```

`\dom` `\ran` `\fcomp` `\bcomp` `\id` Then, a batch of some quite easy symbols.

```
\bsymb@defop{\dom}{\mathrm{dom}}
\bsymb@defop{\ran}{\mathrm{ran}}
\newcommand{\fcomp}{\mathbin;}
\newcommand{\bcomp}{\circ}
\bsymb@defop{\id}{\mathrm{id}}
```

`\domres` `\domsub` `\ranres` `\ransub` Here, we also use the same trick to build the subtraction symbols.

```
\newcommand{\domres}{\mathbin\lhd}
\newcommand{\domsub}{\mathbin{\lhd\mkern-14mu-}}
\newcommand{\ranres}{\mathbin\rhd}
\newcommand{\ransub}{\mathbin{\rhd\mkern-14mu-}}
```

`\ovl` `\dprod` `\prjone` `\prjtwo` `\pprod` And we continue with some more binary relation constructs.

```
\newcommand{\ovl}{\mathbin{\lhd\mkern-9mu-}}
\newcommand{\dprod}{\mathbin\otimes}
\bsymb@defop{\prjone}{\mathrm{prj}_1}
\bsymb@defop{\prjtwo}{\mathrm{prj}_2}
\newcommand{\pprod}{\mathbin\|}
```

We now proceed with the symbols used to define function sets. We still use a similar trick to build the symbols that are not native in the fonts from `amssymb`. To ease thing, we first define a helper macro to typeset sets of partial function.

```
\newcommand{\bsymb@partial}[2]{
  \mathbin{\mkern#2mu\mapstochar\mkern-#2mu#1}
}
```

\pfun
\tfun
\pinj
\tinj
\psur
\tsur
\tbij

Then we can proceed with the symbols

```
\newcommand{\pfun}{\bsymb@partial\rightarrow6}
\newcommand{\tfun}{\mathbin\rightarrow}
\newcommand{\pinj}{\bsymb@partial\rightarrowtail9}
\newcommand{\tinj}{\mathbin\rightarrowtail}
\newcommand{\psur}{\bsymb@partial\twoheadrightarrow6}
\newcommand{\tsur}{\mathbin\twoheadrightarrow}
\newcommand{\tbij}{\mathbin{
      \rightarrowtail
      \mkern-18mu\twoheadrightarrow}
}
```

\nat
\natn
\intg
\upto
\finite
\card

We're almost finished, some more new operators and integer sets.

```
\bsymb@deford{\nat}{\mathbb N}
\bsymb@deford{\natn}{\mathbb N_1}
\bsymb@deford{\intg}{\mathbb Z}
\newcommand{\upto}{\mathbin{.\mkern1mu.}}
\bsymb@defop{\finite}{\mathrm{finite}}
\bsymb@defop{\card}{\mathrm{card}}
```

\upred
\usucc

Now, the pred and succ operators. Here, the name `\pred` and `\succ` are already defined, with a different meaning in LaTeX, so we add a `u` prefix (meaning unary).

```
\bsymb@defop{\upred}{\mathrm{pred}}
\bsymb@defop{\usucc}{\mathrm{succ}}
```

\expn

The exponentiation operator is a simple binary operator. The fine point is to find the adequate width of the hat and overall width of the operator.

```
\newcommand\expn{\mathbin{\widehat{\enskip}}}
```

\Bool
\True
\False
\bool

Then, the boolean operators.

```
\bsymb@deford{\Bool}{\mathrm{BOOL}}
\bsymb@deford{\True}{\mathrm{TRUE}}
\bsymb@deford{\False}{\mathrm{FALSE}}
\bsymb@defop{\bool}{\mathrm{bool}}
```

\bcmeq
\bcmin
\bcmsuch

And here we are. We just need to define the generalised substitutions.

```
\newcommand{\bcmeq}{\mathrel{:\mkern1mu=}}
\newcommand{\bcmin}{\mathrel{:\mkern1mu\in}}
\newcommand{\bcmsuch}{\mathrel{:\mkern1mu\mid}}
```

\oftype

Finally, the `\oftype` operator is defined by stacking on top of each other two instances of the `\circ` operator in the smallest available font.

```
\newcommand{\oftype}{\mathrel{%
  \ooalign{$\scriptstyle\circ$\cr%
           $\scriptscriptstyle\circ$\cr}}}
```

Change History