Supplement: LaTeX Cheat-Sheet (In-Progress)

CPSC 509: Programming Language Principles

Ronald Garcia*

1 February 2014

Here’s where I throw in a bunch of LaTeX that I have been using. For sanity’s sake I need your help to grow this organically. Ideally It will help you with typsetting your own materials. Let me know what pieces are missing and I will add them.

1 Explicit Sets, Set Comprehensions, and Tuples

I use the \texttt{braket.sty} package to typeset tuples and sets:

\begin{verbatim}
{a,b,c}
{a \in A \mid a \notin B}
\langle a, b, c \rangle
\end{verbatim}

2 Object Language Bits

Why so blue? Well, because I typeset object language items using the \texttt{\mbf{}} or \texttt{\tbsf{}} macros defined in \texttt{defs.tex}.

3 Typesetting BNF’s

Here’s the language of Boolean and Arithmetic Expressions:

\begin{verbatim}
t \in \textsc{term}, \quad v \in \textsc{value}, \quad nv \in \textsc{num}
t ::= \text{true} \mid \text{false} \mid \text{if } t \text{ then } t \text{ else } t \\
\mid z \mid \text{succ}(t) \mid \text{pred}(t) \mid \text{zero?}(t)
v ::= \text{true} \mid \text{false} \mid nv
nv ::= z \mid \text{succ}(nv)
\end{verbatim}

If you look at the file, you will see things like \texttt{<if> t <then> t <else> t} which come out like \texttt{if t then t else t}. I am using the keyword feature of the \texttt{semantic.sty} package to define special keywords that I can refer to in angle brackets. In this particular case, the definition in \texttt{defs.tex} that brings things to life is:

\begin{verbatim}
\reservestyle{\oblang}{\tbsf}
\oblang{if-then-else, 
   true,false,if[if:],then[;then:],else[;else;]}
\end{verbatim}
The `\reservestyle` macro creates a new “style” command (in this case `\oblang`) which I can use to create keywords (the thing before the square brackets) and what text they will render (the thing inside the square brackets) using the style function given in the `\reservestyle` keyword. For example, `<if>` actually translates to `\tbsf{if;}` where `;` is the latex command for “leave some space”.

4 A Function Definition

Let $nat : \text{Num} \to \mathbb{N}$ be defined by

$$
nat(z) = 0$$

$$
nat(succ(nv)) = 1 + nat(nv).$$

5 An Inductive Definition

$\Downarrow \subseteq \text{TERM} \times \text{VALUE}$

- $true \Downarrow true$ (etru)
- $false \Downarrow false$ (efalse)
- $t_1 \Downarrow true \quad t_2 \Downarrow v_2$ (eif-t)
- $t_1 \Downarrow false \quad t_2 \Downarrow v_3 \quad t_3 \Downarrow v_3$ (eif-f)
- $t \Downarrow n$ (ez)
- $succ(t) \Downarrow succ(nv)$ (esucc)
- $pred(t) \Downarrow nv$ (epred)
- $z \Downarrow z$ (ezero?-z)
- $succ(nv) \Downarrow succ(nv)$ (ezero?-s)

In the past I have tended to use `\inference` from `semantic.sty` to define inductive rules, and `\infer` from `proof.sty` to write derivations, because I think the `\inference` is prettier, but using it to write derivations is a big space hog. I’m moving toward consistently using `\infer` for everything just to keep it simple. Hence, the rules above are typeset using `\infer`.

Beware! `\inference` and `\infer` take their arguments in opposite order, so if you switch, then you have to switch the arguments too, otherwise you’ll end up with errors or upside-down derivations/rules.

6 A Concrete Derivation

As an example derivation of an entailment relation, here is a derivation of $\emptyset \vdash A \lor \bot \supset A \land true$, meaning that it is a theorem of constructive propositional logic (it can be entailed with no assumptions):

- $\{ A \lor \bot \} \vdash true$ (hyp)
- $\{ A \lor \bot, A \} \vdash A \land true$ (hyp)
- $\{ A \lor \bot, A \} \vdash A \land true$ (hyp)
- $\emptyset \vdash A \lor \bot \supset A \land true$ (⇒I)

7 Abstract Derivations and Properties of Derivations

This is useful for seeing how to typeset abstract derivations, where you don’t see everything.

Let $P(D)$ be a predicate on (or property of) derivations $D$. Then $P$ holds for all derivations $D$ if:

1. $P\left( true \in \text{TERM} \quad (r-true) \right)$ holds;
2. \( P\left( \text{false} \in \text{TERM} \right) \) holds;

3. If \( P\left( r_1 \in \text{TERM} \right) \), \( P\left( r_2 \in \text{TERM} \right) \), and \( P\left( r_3 \in \text{TERM} \right) \) hold then

\[
P\left( \begin{array}{c} r_1 \in \text{TERM} \\ r_2 \in \text{TERM} \\ r_3 \in \text{TERM} \end{array} \right) \quad (r-if)
\]

holds.

Note the use of \texttt{\deduce} instead of \texttt{\infer} to omit the horizontal bar. The \texttt{sarray} macro, profided by \texttt{defs.tex} is a trick to get the large parentheses to size correctly (try removing it and see what happens).

Notice that when writing properties of derivations, I wrap the derivation in an \texttt{sarray} form, which is defined in \texttt{defs.tex}. That keeps there from being extra annoying blank space at the bottom of the derivation. I don’t have a good explanation for why this is necessary I’m afraid.

8 A Definition by Cases

This definition uses the case environment to lay out the three possible cases.

Define \texttt{eval\_BA : TERM \rightarrow \{ true, false \} \cup \mathbb{N}} by

\[
eval_{\text{BA}}(t) = \begin{cases} 
\text{true} & \text{if } t \downarrow \text{true} \\
\text{false} & \text{if } t \downarrow \text{false} \\
\text{nat}(nv) & \text{if } t \downarrow nv
\end{cases}
\]

9 A Proposition

Proposition 1 (Inversion).

1. If \( \text{true} \Downarrow v \) then \( v = \text{true} \).
2. If \( \text{false} \Downarrow v \) then \( v = \text{false} \).
3. If \( t_1 \text{ then } t_2 \text{ else } t_3 \Downarrow v \) then either
   \( \begin{array}{c} 
   (a) \ t_1 \Downarrow \text{true} \text{ and } t_2 \Downarrow v \text{ or} \\
   (b) \ t_1 \Downarrow \text{false} \text{ and } t_3 \Downarrow v
   \end{array} \)
4. If \( z \Downarrow v \) then \( v = z \).
5. If \( \text{succ}(t) \Downarrow v \) then \( t \Downarrow v_1, v_1 \in \text{NUM}, \text{and } v = \text{succ}(v_1) \).
6. If \( \text{pred}(t) \Downarrow v \) then \( t \Downarrow \text{succ}(v) \text{ and } v \in \text{NUM} \).
7. If \( \text{zero?}(t) \Downarrow v \) then either
   \( \begin{array}{c} 
   (a) \ t \Downarrow z \text{ and } v = \text{true} \text{ or} \\
   (b) \ t \Downarrow \text{succ}(nv) \text{ and } v = \text{false}
   \end{array} \)

10 A Logical Statement (about derivations)

To prove these propositions, we first expand them to be formal statements about derivations. For example, item 7 expands to the following:

Proposition 2. \( \forall D. D :: \text{zero?}(t) \Downarrow v \Rightarrow ((\exists \mathcal{E} : t \Downarrow z) \land v = \text{true}) \lor ((\exists \mathcal{E} : t \Downarrow \text{succ}(nv)) \land v = \text{false}) \)
11 A Trace (Using Let)

\[
\text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \\
\text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x \\
\text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \quad \rightarrow \quad \text{let } x = 5 \\
\text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x \quad \rightarrow \quad \text{in } x + x
\]