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 `braket.sty` package to typeset tuples and sets:

$$\{\, a, b, c \,\}$$
$$\{\, a \in A \mid a \notin B \,\}$$
$$\langle a, b, c \rangle$$

2 Object Language Bits

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

3 Typesetting BNF's

Here's the language of Boolean and Arithmetic Expressions:

$$\begin{array}{rcl} t \in \textsc{Term}, & v \in \textsc{Value}, & nv \in \textsc{Num} \\ t & ::= & \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t \\ & \mid & \mathsf{z} \mid \mathsf{succ}(t) \mid \mathsf{pred}(t) \mid \mathsf{zero?}(t) \\ v & ::= & \mathsf{true} \mid \mathsf{false} \mid nv \\ nv & ::= & \mathsf{z} \mid \mathsf{succ}(nv) \end{array}$$

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

```
\reservestyle{\oblang}{\tbsf}

\oblang{if-then-else,
  true,false,if[if\;],then[\;then\;],else[\;else\;]
}
```

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

$$nat(\mathsf{z}) = 0$$
$$nat(\mathsf{succ}(nv)) = 1 + nat(nv).$$

5 An Inductive Definition

$\Downarrow \subseteq \textsc{Term} \times \textsc{Value}$

$$\frac{}{\mathsf{true} \Downarrow \mathsf{true}}\ (\text{etrue}) \qquad \frac{}{\mathsf{false} \Downarrow \mathsf{false}}\ (\text{efalse}) \qquad \frac{t_1 \Downarrow \mathsf{true} \quad t_2 \Downarrow v_2}{\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \Downarrow v_2}\ (\text{eif-t})$$

$$\frac{t_1 \Downarrow \mathsf{false} \quad t_3 \Downarrow v_3}{\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \Downarrow v_3}\ (\text{eif-f}) \qquad \frac{}{\mathsf{z} \Downarrow \mathsf{z}}\ (\text{ez}) \qquad \frac{t \Downarrow nv}{\mathsf{succ}(t) \Downarrow \mathsf{succ}(nv)}\ (\text{esucc}) \qquad \frac{t \Downarrow \mathsf{succ}(nv)}{\mathsf{pred}(t) \Downarrow nv}\ (\text{epred})$$

$$\frac{t \Downarrow \mathsf{z}}{\mathsf{zero?}(t) \Downarrow \mathsf{true}}\ (\text{ezero?-z}) \qquad \frac{t \Downarrow \mathsf{succ}(nv)}{\mathsf{zero?}(t) \Downarrow \mathsf{false}}\ (\text{ezero?-s})$$

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 $\emptyset \vdash \mathsf{A} \vee \bot \supset \mathsf{A}$ **true**, meaning that it is a theorem of constructive propositional logic (it can be entailed with no assumptions):

$$\dfrac{\dfrac{\{\mathsf{A} \vee \bot\} \vdash \mathsf{A} \vee \bot\ \mathbf{true} \qquad \dfrac{}{\{\mathsf{A} \vee \bot, \mathsf{A}\} \vdash \mathsf{A}\ \mathbf{true}}\ (\text{hyp}) \qquad \dfrac{\dfrac{}{\{\mathsf{A} \vee \bot, \bot\} \vdash \bot\ \mathbf{true}}\ (\text{hyp})}{\{\mathsf{A} \vee \bot, \bot\} \vdash \mathsf{A}\ \mathbf{true}}\ (\bot\text{E})}{\{\mathsf{A} \vee \bot\} \vdash \mathsf{A}\ \mathbf{true}}\ (\vee\text{E})}{\emptyset \vdash \mathsf{A} \vee \bot \supset \mathsf{A}\ \mathbf{true}}\ (\Rightarrow\text{I})$$

7 Abstract Derivations and Properties of Derivations

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

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

1. $P\left(\dfrac{}{\mathsf{true} \in \textsc{Term}}\ (\text{r-true})\right)$ holds;

2. $P\left(\dfrac{}{\mathsf{false} \in \textsc{Term}}\ \text{(r-false)}\right)$ holds;
3. If $P\left(\begin{matrix}\mathcal{D}_1\\ r_1 \in \textsc{Term}\end{matrix}\right)$, $P\left(\begin{matrix}\mathcal{D}_2\\ r_2 \in \textsc{Term}\end{matrix}\right)$, and $P\left(\begin{matrix}\mathcal{D}_3\\ r_3 \in \textsc{Term}\end{matrix}\right)$ hold then

$$P\left(\dfrac{\begin{matrix}\mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3\\ r_1 \in \textsc{Term} & r_2 \in \textsc{Term} & r_3 \in \textsc{Term}\end{matrix}}{\mathsf{if}\ r_1\ \mathsf{then}\ r_2\ \mathsf{else}\ r_3 \in \textsc{Term}}\ \text{(r-if)}\right)$$

holds.

Note the use of `\deduce` instead of `infer` to omit the horizontal bar. The `\sarray` macro, profided by `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 `\sarray` form, which is defined in `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 $eval_{BA} : \textsc{Term} \to \{\mathsf{true}, \mathsf{false}\} \cup \mathbb{N}$ by

$$eval_{BA}(t) = \begin{cases} \mathsf{true} & \text{if } t \Downarrow \mathsf{true} \\ \mathsf{false} & \text{if } t \Downarrow \mathsf{false} \\ nat(nv) & \text{if } t \Downarrow nv \end{cases}$$

9 A Proposition

Proposition 1 (Inversion)**.**

1. *If* $\mathsf{true} \Downarrow v$ *then* $v = \mathsf{true}$.
2. *If* $\mathsf{false} \Downarrow v$ *then* $v = \mathsf{false}$.
3. *If* $\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \Downarrow v$ *then either*
   (a) $t_1 \Downarrow \mathsf{true}$ *and* $t_2 \Downarrow v$ *or*
   (b) $t_1 \Downarrow \mathsf{false}$ *and* $t_3 \Downarrow v$
4. *If* $\mathsf{z} \Downarrow v$ *then* $v = \mathsf{z}$.
5. *If* $\mathsf{succ}(t) \Downarrow v$ *then* $t \Downarrow v_1$, $v_1 \in \textsc{Num}$, *and* $v = \mathsf{succ}(v_1)$.
6. *If* $\mathsf{pred}(t) \Downarrow v$ *then* $t \Downarrow \mathsf{succ}(v)$ *and* $v \in \textsc{Num}$.
7. *If* $\mathsf{zero?}(t) \Downarrow v$ *then either*
   (a) $t \Downarrow \mathsf{z}$ *and* $v = \mathsf{true}$ *or*
   (b) $t \Downarrow \mathsf{succ}(nv)$ *and* $v = \mathsf{false}$.

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. $\forall \mathcal{D}.\mathcal{D} :: \mathsf{zero?}(t) \Downarrow v \Rightarrow ((\exists \mathcal{E}.\mathcal{E} :: t \Downarrow \mathsf{z}) \wedge v = \mathsf{true}) \vee ((\exists \mathcal{E}.\mathcal{E} :: t \Downarrow \mathsf{succ}(nv)) \wedge v = \mathsf{false})$

11 A Trace (Using Let)

$$\begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow$$

$$\begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \begin{array}{l}\text{let } x = 5\\ \text{in } x + x\end{array} \longrightarrow \cdots$$