Practical Foundations for Programming Languages (Second Edition)

Errors and Corrections

Robert Harper
Carnegie Mellon University
September 29, 2017

Known Errors

The following errors in the second edition have been called to my attention.

- Page 82, Section 10.3: the definition of the recursor should be
  \[ \text{iter}\{\langle z, e_0 \rangle; x'.\langle s(x'.1), [x'.1, x'.r/x, y]e_1 \rangle\}(e). \]
- Page 88, first paragraph: replace “induction” by “inductive”.
- Page 90, “which” in the abstract syntax should be “ifnull”, consistently with the concrete syntax.
- Page 107, \( u_1, \ldots, u_n \) should be \( u_1, \ldots, u_m \).
- Page 140, Rule (16.4a): the type label on the \( \lambda \) should be \( \tau_1 \), not \( \tau \).
- Page 154, and Exercise 1: the type of \( \text{emp} \) should be \( \forall t :: T.q[t] \).
- Page 152, line -8: \( \langle bs, fs' \rangle \) should be \( \text{just}(\{ f, \langle bs, fs' \rangle \}) \).
- Page 162, line 3: “In other words \( e \) is defined to be …” should be “In other words \( f \) is defined to be …."
- Page 171: “fcntion” should be “function.”
- Page 191, dynamics of application: either a by-name or by-value interpretation is possible.
- Page 191, Rule (22.4h): 0 should be \( z \).
• Page 191, Rule (22.4i): \( n + 1 \) should be \( s(n) \).
• Page 193, Section 22.2: “berepresented” should be “be represented”.
• Page 219, paragraph 2: should read: “serve as behavioral specifications”.
• Page 228, Section 25.3: “\( \Phi_\Gamma \vdash a \in_\tau \phi \)” should be “\( \Phi_\Gamma \vdash e \in_\tau \phi \)”.
• Page 265, Exercises 29.4 and 29.5 are orphaned: the modal formulation of exceptions is no longer presented in the text.
• Page 267, Section 30.1: \( \tau \) cont cont should be \( \tau' \) cont cont.
• Page 268, rule 30.2: the premise should read “\( k \ll \tau \)”.
• Page 288, Section 32.5: “… the name of the fluid …”
• Page 291, Section 33.1: “A dynamic class is a symbol that is generated …”
• Page 292, last paragraph: Revised per online edition to clarify discussion of disequality of names.
• Page 298, Exercises 33.3 and 33.4 depend on Exercises 29.4 and 29.5, which rely on the modal formulation of exceptions.
• Page 314, first display equation: the printed edition inexplicably has
  \[ \tau \text{ cap} \triangleq \tau \text{ cmd} \times (\text{nat} \rightarrow \text{nat cmd}) \]
  which instead should be
  \[ \tau \text{ cap} \triangleq \tau \text{ cmd} \times (\tau \rightarrow \tau \text{ cmd}). \]
• Page 344, start of paragraph 2: should read: “finite mapping of the task names”.
• Pages 344-5: Rule 37.10(b) should have a premise \( \neg(e_2 \text{ val}) \) to prevent needless repetition. Rule 37.10(c) should send \( a_1 \) to \( e_1(e_2) \) in the result state, not to the senseless substitution instance.
• Page 366, Section 39.4, displayed example at end should be
• Page 381, Rules 4.10. There is no execution rule for \( \text{sync}(\text{never}) \), so that the intended, but unstated, progress theorem fails. A solution is to introduce a new action, say \#`, representing synchronization on the never-occurring event, and to add the execution rule
  \[ \text{sync}(\text{never}) \#_{\Sigma} \text{sync}(\text{never}). \]
The same correction applies to \( \text{DA} \), which also has a null event.
Improvements

The following changes improve the presentation in the text.

- Various places: ensure uniform use of “statics” and “dynamics” rather than “static semantics” and “dynamic semantics.”
- Chapter 11 and elsewhere: replace abort(e) by case e { } to avoid the implication that “abort” causes a run-time fault, which it does not (and can not!).
- Page 130, the dynamics is specified to be lazy, though it would have been easy to specify both eager and lazy semantics using optional premises and rules. The rules given in Figure 1 do so.
Figure 1: Eager-or-Lazy Dynamics for Inductive and Coinductive Types
\[
\begin{align*}
\Gamma \vdash e : \text{clsfd} & \quad e \text{ val}_\Sigma \\
\Gamma \vdash ! e \text{ proc} & \quad ! e \xrightarrow{e_1} 1 \\
\end{align*}
\]

\[
\begin{align*}
e \text{ val}_\Sigma & \quad \text{emit}(e) \xrightarrow{\sigma} \text{ret}() \otimes !e
\end{align*}
\]

Figure 2: Asynchronous Dynamics for Concurrent Algol

- Chapter 40 Concurrent Algol. The dynamics specifies synchronous communication, in which an emit of a message does not terminate until the message has been received. An asynchronous dynamics is easily specified by making the alterations given in Figure 2. These rules define an asynchronous send process, and re-define the dynamics of emit to create such a process.

- Section 33.2. It would be more suggestive to write \(\text{inref}(e_1; e_2)\) in place of \(\text{mk}(e_1; e_2)\) to stress its role as the dynamic analogue of \(\text{in}[a](e)\), much as \(\text{getref}(e)\) is the dynamic analogue of \(\text{get}[a]\) for assignables.

Acknowledgements

Thanks to Carlo Angiuli, Stephanie Balzer, Peter Brattveit Bock, Corwin de Boor, Jordan Brown, Karl Crary, Jan Hoffmann, Muhammed Hussein Nasrollahput, Selva Samuel, Stephanie Weirich, James R. Wilcox, Haoxuan (Aaron) Yue, Fangyi Zhou, Jake Zimmerman.