Practical Foundations for Programming Languages (Second Edition)

Errors and Corrections

Robert Harper
Carnegie Mellon University

November 7, 2019

Corrections

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

- Back cover: Andrew Pitts’s affiliation is printed as “University of Cambridge University Press” rather than “University of Cambridge.” (This is the only mistake that is not under my control!)

- Page 4: In the description of abt’s, operators with no children should also be deemed “leaves.”

- Page 16, Lemma 2.1: The proof could be simplified to a case analysis, requiring no appeal to the inductive hypothesis. Or else “b nat” in the conclusion should read succ(b) nat.

- Page 23, after displayed formula (3.6), “even from rules (2.2)” should read “even from rules (2.8).”

- Page 38, footnote: “do not [necessarily] validate weakening.”


- Page 82, Section 10.3: the definition of the recursor should be

$$\text{iter}\{\langle z, e_0 \rangle; x'.(s(x' \cdot 1), [x' \cdot 1, x' \cdot r/x, y|e_1])(e)\}.$$

- Page 75, Exercise 9.6: “then $e$ is also hereditarily terminating …” should be “then $e'$ is also hereditarily terminating …”.

- Page 82, last line: the type of the bound variable $n$ is given as nat $\times$ nat, but should be nat.
• 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 128 et seq.: Rules (15.2d), (15.4f), (15.6d), (15.9d), and (15.9h) lack premises to require that the principal argument be a value as a condition on taking a step.

• Page 128, Rule (15.2a): add an optional premise $e$ val, and an optional rule to evaluate $e$ if the introduction form is to be eager.

• Page 129, Rule (15.4a): add an optional premise $e$ val, and an optional rule to evaluate $e$ if the generator is to be eager.

• Page 129, Rule (15.6a): add an optional premise $e_2$ val and a corresponding rule to evaluate $e_2$ if the introduction form is to be eager.

• Page 140, Rule (16.4a): the type label on the $\lambda$ should be $\tau_1$, not $\tau$.

• Page 147, line 13: “identification convention”, not “identification convnention”.

• Page 154, and Exercise 1: the type of $\text{emp}$ should be $\forall t :: T. q[t]$.

• Page 152, line -8: $\langle bs, f s' \rangle$ should be $\text{just}(\langle f, \langle bs, f s' \rangle \rangle)$.

• Page 162, line 3: “In other words $e$ is defined to be . . .” should be “In other words $f$ is defined to be . . .”.

• Page 165, Section 9.3: “using recursive functions” should be “using fixed points.”

• Page 171: “function” 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: “be represented” should be “be represented”.

• Page 219, paragraph 2: should read: “serve as behavioral specifications”.

• Page 221, second paragraph: “entails any refinement all” should read “entails any refinement at all.”

• Page 228, Section 25.3: “$\Phi_\Gamma \vdash a \in_\tau \phi$” should be “$\Phi_\Gamma \vdash e \in_\tau \phi$.”
• Page 261, Rule (29.3a): Conclusion should be \( e \triangleright e \) initial.

• 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 “\( k \div \tau \).”

• Page 283, Exercise 31.4: There is no reason to specify the associated type of a symbol to be sexpr. For “pure” symbols the associated type would be unit, indicating no associated information. Alternatively, the associated type could be a product of attributes, including, for example, the “print name” (a string) associated with the symbol.

• 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 294, line 11: \( \text{isin}[a](e;x;e_1;e_2) \) should be \( \text{isin}[a](e;y;e_1;e_2) \) to align the variable name \( y \) with its definition to follow.

• 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 308, first displayed program: \( \text{cmd} \ (a := x) \) should be \( \{a := x\} \) (the bodies of procedures are commands, not expressions).

• Page 314, first display equation: the printed edition inexplicably has

\[
\tau \cap \triangleright = \tau \text{cmd} \times (\text{nat} \rightarrow \text{nat cmd})
\]

which instead should be

\[
\tau \cap \triangleright = \tau \text{cmd} \times (\tau \rightarrow \tau \text{cmd}).
\]

• 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 associate \( a_1 \) to \( e_1(e_2) \) in the result state. There should also be an analogous rule for the join point corresponding to the function position of the application.

• Page 366, Section 39.4, displayed example at end of section should be

\[
(v v.( V_0 \otimes V_1 )) \otimes U \overset{\Sigma}{\rightarrow} (v v. V_0 \otimes V_2 ) \otimes U \equiv v v.( V_0 \otimes V_0 \otimes U).
\]
• Page 380, displayed program has a missing use of cmd, should be

\[
\text{fix loop: } \tau \text{ cmd is cmd } \{ x \leftarrow \text{acc}; \text{match } x \text{ as } a \cdot y \leftrightarrow \text{ret } y \text{ ow } \leftrightarrow \text{emit}(x); \text{do loop}\}
\]

• 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

\[
\text{sync}(\text{never}) \xrightarrow{\Sigma} \text{sync}(\text{never}).
\]

The same correction applies to DA, which also has a null event.

• Page 387, Rule 41.2(b): Premise \(\Gamma \vdash e_1 : \tau_1 \text{ cmd } w\) should be \(\Gamma \vdash e_1 : \text{cmd}[w](\tau_1) @ w\).

• Page 389, Rule 41.6a: The process “run(\(m\))” should be “run(at[w](\(m\)))” to record the site of the spawned process.

• Page 390, Theorems 41.2 and 41.3, the transition judgments should be \(p \xrightarrow{\Sigma} p'\); the “\(w\)” should be above the transition arrow, not below it.

• Page 437, last paragraph: strike “obviously reflexive.” Reflexivity is exactly Theorem 46.13.

• Page 438, Section 46.1, definition of a congruence: require only that the relations be partial equivalence relations (symmetric and transitive, not necessarily reflexive).

• Page 442, proof of Lemma 46.16: Add “By Definition 46.8 and Lemma 46.11 . . . .” for clarity.

• Page 447, Section 47.3, “the following rule”.

• Page 448, Section 47.3, should be “. . . it is enough to show that \(\hat{\gamma}(e_1) \sim_\tau \hat{\gamma}(e'_1), . . . .\)”

• Page 456, Section 48.2, statement and proof of Lemma 48.3, \(\Delta, t\) should be \(\Delta, t\) type.

• Page 459, Section 48.3, proof of Lemma 48.8: second \(\rho\) should be \(\rho'\):

\[
d'[\rho'] \cong_{[\rho'/t]} \hat{\rho}(\tau_2) e'[\rho'].
\]

• Page 462, line 8: the second occurrence of \(\rho\) should be \(\rho'\).
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.”

- Change rules (5.10a) and (5.10i) to have typing premises so that it is possible to prove the newly inserted Exercise 5.4 by induction on Rules (5.10).

- Chapter 6, Section 6.2: The statement and proof of the canonical forms lemma could be improved as follows:

  **Lemma 1 (Canonical Forms).**
  1. If $e : \text{num}$ and $e \text{ val}$, then $e = n$ for some number $n$.
  2. If $e : \text{str}$ and $e \text{ val}$, then $e = "s"$ for some string $s$.

  *Proof.* Each case is proved by induction on typing, making use of the definition of $e \text{ val}$. 

- Chapter 11 and elsewhere: replace $\text{abort}(e)$ by $\text{case } e \{ \}$ 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 is possible to specify both eager and lazy semantics using optional premises and rules. The rules given in Figure 1 do so.

- Chapter 33, Section 2: It would be preferable, and consistent with later chapters, to rename the operator $\text{isof}$ to $\text{isinref}$.

- 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 may be 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, Shawn Rashid, Nicholas Roberts, Selva Samuel, Kartik Singhal, Anthony Su, Stephanie Weirich, James R. Wilcox, Yongwei Yuan, Haoxuan (Aaron) Yue, Fangyi Zhou, Jake Zimmerman.
\[
\begin{align*}
\text{val} & \Rightarrow e' \\
\text{fold}\{t.\tau\}(e) & \Rightarrow \text{fold}\{t.\tau\}(e') \\
\text{e} & \Rightarrow e' \\
\text{rec}\{t.\tau\}(x.e_1; e_2) & \Rightarrow \text{rec}\{t.\tau\}(x.e_1'; e_2') \\
\text{e} & \Rightarrow e' \\
\text{gen}\{t.\tau\}(x.e_1; e_2) & \Rightarrow \text{gen}\{t.\tau\}(x.e_1'; e_2') \\
\text{u} & \Rightarrow u' \\
\text{unfold}\{t.\tau\}(e) & \Rightarrow \text{unfold}\{t.\tau\}(e') \\
\text{e} & \Rightarrow e' \\
\text{emit}(e) & \Rightarrow \text{ret}() \otimes e
\end{align*}
\]

Figure 1: Eager-or-Lazy Dynamics for Inductive and Coinductive Types

\[
\begin{align*}
\vdash \Sigma \ e : \text{clsfd} & \quad \text{val}_\Sigma \ e \\
\vdash \Sigma ! e \ \text{proc} & \quad e \ \text{val}_\Sigma \\
\vdash \Sigma ! e \ e_1 & \quad e_1 ! e_2 \ \text{val}_\Sigma \\
\vdash \Sigma \ e_1 & \quad e_1 e_2 \ \text{ret}() \otimes e
\end{align*}
\]

Figure 2: Asynchronous Dynamics for Concurrent Algol