Practical Foundations for Programming Languages (Second Edition)

Errors and Corrections*

Robert Harper
Carnegie Mellon University

January 11, 2023

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 $\mathtt{nat}$" in the conclusion should read $\mathtt{succ}(\,b\,)\ \mathtt{nat}$.

- 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 62, Rule 8.2: rename meta-variables to be consistent with Rules 8.1(a) and 8.1(b) for clarity.

- Page 71, Lemma 9.2, "$e = \mathtt{z}$ or" omitted by publisher in printed edition.

- Page 72, Rules 9.5b and 9.5c: the premise governing e_1 should be $\Gamma, x : \mathtt{nat}, y : \tau \vdash e_1 : \tau$ in each case.

- Page 75, Exercise 9.6: "then e is also hereditarily terminating ..." should be "then e' is also hereditarily terminating ...".

- Page 81. Concrete syntax for product types and elements corrected to include label mapping as in abstract syntax.

- Page 82, last line: the type of the bound variable n is given as $\mathtt{nat} \times \mathtt{nat}$, but should be $\mathtt{nat}$.

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

$$\mathtt{iter}[\,\tau\,](\,e\,;\langle \mathtt{z}, e_0\rangle\,;x'\,.\,\langle \mathtt{s}(\,x'\cdot \mathtt{l}\,), \{\mathbf{x'}\cdot \mathtt{l}, \mathbf{x'}\cdot \mathtt{r}/\mathbf{x},\mathbf{y}\}\mathbf{e_1}\rangle\,).$$

- Page 87: Concrete syntax of sum types and case expressions should have label mapping as in abstract syntax.

- Page 88, first paragraph: replace "induction" by "inductive".

- Page 90, "`which`" in the abstract syntax should be "`ifnull`", consistently with the concrete syntax.

- Page 100, rule 12.9: The two premises should be replaced by $\Gamma \vdash p : \phi_1 \wedge \phi_2$.

- Page 100, rule 12.10a: the second premise should be $\Gamma \vdash p_1 : \phi_1$.

- Page 107, $u_1, \ldots, u_n$ should be $u_1, \ldots, u_m$.

- Page 111, first sentence of second paragraph should read: "All proofs in classical logic proceed by contradiction."

- Page 126 *et sq*: replace "recursor" by "iterator" to avoid confusion with the terminology of Chapter 9.

- Page 128 *et seq.*: Rules (15.2d), (15.4d), (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 129, Figure 15.2.2: the definition of $\Delta \vdash t\,.\,\tau$ pos was omitted, and should be defined when Δ, t type $\vdash \tau$ type as illustrated by the following selected rules:

$$\frac{\Delta \vdash \tau \text{ type}}{\Delta \vdash t\,.\,\tau \text{ pos}}\;\text{(CLOSED)} \qquad \frac{\Delta \vdash \tau_1 \text{ type} \qquad \Delta \vdash t\,.\,\tau_2 \text{ pos}}{\Delta \vdash t\,.\,\tau_1 \to \tau_2 \text{ pos}}\;\text{(ARROW)} \qquad \frac{\Delta, r\,.\vdash \text{pos}t\,.\,\rho \text{ pos}}{\Delta \vdash t\,.\,\mu(\,r\,.\,\rho\,) \text{ pos}}\;\text{(IND)}$$

  Note that t is prohibited from occurring within the domain of a function type by the first premise.

- Page 140, Rule 16.4a: the type label on the λ should be τ_1, not τ.

- Page 147, line 13: "identification convention", not "identification covnention".'

- Page 149, type of e_r should be $\texttt{natlist} \to (\,\texttt{nat} \times \texttt{natlist}\,)\,\texttt{opt}$.

- Page 150, type of e_r should be

$$(\,\texttt{natlist} \times \texttt{natlist}\,) \to (\,\texttt{nat} \times (\,\texttt{natlist} \times \texttt{natlist}\,)\,)\,\texttt{opt}.$$

- Page 152, definition of relation R: the use of observational equivalence, $e_1 \cong_\tau e_2$, anticipates its definition in Chapter 46. See below for an improvement to the argument.

- Page 154, and Exercise 1: the type of emp should be $\forall\, t :: \texttt{Ty}\,.\,q[\,t\,]$.

- Page 154, last paragraph: "have to give" should be "has to give".

- Page 156, after displayed rules: replace "three" by "type-forming", namely the function, universal, and existential constructors.

- 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 166, after displayed equation: "given the code $\overline{\ulcorner e \urcorner}$" should be "given the code $\ulcorner e \urcorner$".

- Page 169, exercise 19.5: should be "specified *as*". The requirement is that the result is z id either is, regardless of whether the other diverges.

- Page 171: "fucntion" should be "function."

- Page 187, first equation: missing parentheses: $(\,(\,\lambda\,(\,x\,)\,u_1\,)(\,u_2\,)\,)^\dagger$.

- Page 187, second equation: missing parentheses: $(\,\lambda\,(\,x : D\,)\,u_1^\dagger\,)(\,u_2^\dagger\,)$.

- 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 $\mathtt{s}(n)$.
- Page 193, Section 22.2: "berepresented" should be "be represented".
- Page 195, first displayed term: should be

$$\mathtt{fun}(x.\mathtt{fix}(p.\mathtt{fun}(y.\mathtt{ifz}[x;y'.\mathtt{succ}(p(y'))](y)))).$$

- Pages 209, 211, 216: notation $\langle \tau_i \rangle_{i \in n}$ should be $\langle \tau_i \rangle_{i \in n}$, and $[\tau_i]_{i \in n}$ should be $[\tau_i]_{i \in n}$.
- Page 209, second paragraph: replace "the only operation" with "the only elimination" and delete mention of binding to a variable.
- Page 211, first paragraph: "a type constructor *is* covariant".
- Page 214, Rule 24.15b: premise should be $\Delta \vdash \tau$ type.
- Page 219, second paragraph: should read: "serve **as** behavioral specifications".
- Page 221, second paragraph: "entails any refinement all" should read "entails any refinement *at* all."
- Page 223, Rule 25.4d: Premise should read "$\Phi \vdash e \in_{\mathtt{dyn}} \mathtt{fun}\,!\,\phi$."
- Page 224, Rule 25.5d: Conclusion should read "$\Phi \vdash e \cdot \mathtt{r} \in_{\tau_2} \phi_2$."
- Page 228, Section 25.3: "$\Phi_\Gamma \vdash a \in_\tau \phi$" should be "$\Phi_\Gamma \vdash e \in_\tau \phi$."
- Page 228, Section 25.3: Should read "we may *prove* type preservation."
- Page 259: Replace first paragraph by the following case for by-name evaluation:

  > Suppose that $s = k;\mathtt{ap}(-;e_2) \triangleleft \lambda[\tau](x.e_1)$, and $s' = k \triangleright \{e_2/x\}e_1$. Let e' be such that $k;\mathtt{ap}(-;e_2) \bowtie \lambda[\tau](x.e_1) = e'$ and let e'' be such that $k \bowtie \{e_2/x\}e_1 = e''$. Noting that $k \bowtie \mathtt{ap}(\lambda[\tau](x.e_1);e_2) = e'$, the result follows from Lemma 28.6.

- Page 261, Rule 29.3a: Conclusion should be $\epsilon \triangleright e$ initial.
- Page 262, third paragraph, last sentence: To clarify, the stack machine dynamics should be augmented with rules to express a call-by-value dynamics.
- Page 263, Rule 29.6c: Premise should require e val.

- 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: $\tau\,\mathsf{cont}\,\mathsf{cont}$ should be $\tau'\,\mathsf{cont}\,\mathsf{cont}$.

- Page 267, remove spurious fragment "k' of type return."

- Page 268, rule 30.2: the premise should read "$k \div \tau$."

- Page 277, rule 30.5(a): two occurrences of "$\mathtt{cont}(\,k\,)$" should be "$\mathtt{cont}(\,k'\,)$."

- 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, Transition rules reformulated to make symbol declarations explicit, and revised discussion of substitution for names.

- Page 294, line 11: $\mathtt{isin}\langle a\rangle(\,e\,;x\,.\,e_1\,;e_2\,)$ should be $\mathtt{isin}\langle a\rangle(\,e\,;y\,.\,e_1\,;e_2\,)$ to align the variable name y with its definition to follow.

- Page 298, Excercises 33.3 and 33.4 depend on Exercises 29.4 and 29.5, which rely on the modal formulation of exceptions.

- Page 303, description of Rule (34.1e), the type of the cell is implicit, all are of type `nat`.

- Page 307, the loop invariant maintains that `r` contains the stated quantity.

- Page 308, first displayed program: "$\mathtt{cmd}\,(\,a := x\,)$" should be "$\{a := x\}$."

- Page 310, Exercise 34.2: replace "$p : \tau$" by "$x : \tau$."

- Page 313, first paragraph: replace "determines" by "refers to". "may refer to the *same* assignable."

- Page 314, first display equation: should read

$$\tau\,\mathtt{cap} \triangleq \tau\,\mathtt{cmd} \times (\,\tau \rightharpoonup \tau\,\mathtt{cmd}\,).$$

- Page 318, Section 35.4: "... complicate the definition of the judgment $\nu\,\Sigma\,\{\,\mu \parallel m\,\}\ \mathsf{ok}$."

- Page 319: drop "the stronger form." It is rather a reformulation.

- Page 325: Rule 36.3 should have a premise stipulating that e is not itself an indirect reference.

- Page 326: "associates the expression a to a" should be "associates the expression $@\,a$ to a".

- Page 328: Rule (36.7b) should require that e_1 and e_2 not be addresses, otherwise (36.7a) would apply.

- Page 328, Theorem 36.2: the reference to rule (19.1a) should be to rule (36.4).

- Page 332, Exercise 36.4: the left-hand side of each equation should be "hatted," as in $\widehat{\texttt{unit}}$.

- Page 335, last sentence: should read "the *statics* of".

- Page 339: instances of "`let`" should be "`par`".

- 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 $\neg(e_2\ \mathsf{val})$ 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 345, first sentence: remove duplicated word "create."

- Page 352, last paragraph: "Future *and speculations* are only interesting. . . ".

- Page 356, displayed code: add "$\texttt{mapforce}(\,f\,)(\,s\,) \triangleq$" for clarity.

- Page 366, last paragraph: drop ", as always,".

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

$$(\,\nu\, v\,.\,(\,V_0 \otimes V_1\,)\,) \otimes U \underset{\Sigma}{\longmapsto} (\,\nu\, v\,.\,V_1 \otimes V_2\,) \otimes U \equiv \nu\, v\,.\,(\,V_1 \otimes V_2 \otimes U\,).$$

- Page 369, rule 39.21: both occurrences of "Σ" should be "$\Sigma, a \sim \tau$."

- Pages 370-371, rules 39.23b and 39.23c: omit premises (unnecessary in first case, spurious in second case).

- Page 380, displayed program has a missing use of `cmd`, should be

$$\texttt{fix}\, loop : \tau\ \texttt{cmd}\ \texttt{is}\ \texttt{cmd}\,\{x \leftarrow \texttt{acc}\,;\texttt{match}\, x\ \texttt{as}\ a \cdot y \hookrightarrow \texttt{ret}\, y\ \texttt{ow} \hookrightarrow \texttt{emit}(\,x\,)\,;\texttt{do}\, loop\}$$

- Page 381, Rules 40.10. There is no execution rule for $\mathtt{sync}(\,\mathtt{never}\,)$, 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

$$\mathtt{sync}(\,\mathtt{never}\,) \xRightarrow[\Sigma]{\#} \mathtt{sync}(\,\mathtt{never}\,).$$

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

- Page 382, Rule 40.15: `void` should be `unit`.

- Page 386, first paragraph: "when the *second* is".

- Page 387, Rule 41.2(b): Premise $\Gamma \vdash e_1 : \tau_1\,\mathtt{cmd} \mathbin{@} w$ should be $\Gamma \vdash e_1 : \mathtt{cmd}\langle w\rangle(\,\tau_1\,) \mathbin{@} w$.

- Page 389, Rule 41.6a: The process "$\mathtt{run}(\,m\,)$" should be "$\mathtt{run}(\,\mathtt{at}\langle w\rangle(\,m\,)\,)$" to record the site of the spawned process.

- Page 390, Theorems 41.2 and 41.3, the transition judgments should be $p \xmapsto[\Sigma]{\alpha \mathbin{@} w} p'$; the "$w$" should be above the transition arrow, not below it.

- Page 399: replace "$\Lambda(\,t\,)\,e[\,\tau\,]$" with "$(\,\Lambda(\,t\,)\,e\,)[\,\tau\,]$." for clarity.

- Page 399: "Singletons *solve not only the type definition problem but . . .* ".

- Page 400, middle: replace "$\Lambda(\,t :: \mathtt{S}(\,\tau\,)\,)\,e[\,\tau\,]$" by "$(\,\Lambda(\,t :: \mathtt{S}(\,\tau\,)\,)\,e\,)[\,\tau\,]$."

- Page 427, rule 45.1a: "x projectible" should be "X projectible."

- 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 follow**ing** rule".

- Page 448, first sentence: The appeal to Corollary 47.17 yields m' such that

$$\mathcal{A}'\{\mathtt{fix}^{m'}\,x : \tau_0\,\mathtt{is}\,e'\} \simeq \mathcal{A}'\{\mathtt{fix}\,x : \tau_0\,\mathtt{is}\,e'\}.$$

But then the appeal to transitivity is incorrect. Please see the online edition for a corrected proof.

- Page 448, Section 47.3, should be ". . . it is enough to show that $\hat{\gamma}(e_1) \sim_\tau \widehat{\gamma'}(\mathbf{e_1'})$, ..."

- Page 450, middle: "(in fact, $m = n + 1$ suffices)."
- Page 456, Section 48.2, statement and proof of Lemma 48.3, Δ, t should be Δ, t type.
- Page 459, Section 48.3, proof of Lemma 48.8: second ρ should be ρ':

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

- Page 462, line 8: the second occurrence of ρ should be ρ'.
- Page 471, first sentence: replace $\mathcal{E}_0$ by $\mathcal{E}$ in conclusion.
- Page 471, middle: Q' in definition of Q'' should be Q'''.
- Page 471, bottom: "*zero* or more silent actions."

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 $e : \texttt{num}$ and e val, then $e = n$ for some number n.*
  2. *If $e : \texttt{str}$ and e val, then $e = "s"$ for some string s.*

  Proof. Each case is proved by induction on typing, making use of the definition of e val. □

- Chapter 11 and elsewhere: replace $\texttt{absurd}(e)$ by $\texttt{case}\, e\, \{\,\}$ 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 17, Section 4: The representation independence example makes use of observational equivalence, which has not yet been defined. Assuming an eager dynamics, the following conditions may be used instead to define the bisimulation relation R:

$$\frac{[e\ \mathsf{val}]}{\texttt{fold}[\,t\,.\,\tau\,](\,e\,)\ \mathsf{val}}$$

$$\left[\frac{e \longmapsto e'}{\texttt{fold}[\,t\,.\,\tau\,](\,e\,) \longmapsto \texttt{fold}[\,t\,.\,\tau\,](\,e'\,)}\right]$$

$$\frac{e_2 \longmapsto e_2'}{\texttt{rec}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;e_2\,) \longmapsto \texttt{rec}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;e_2'\,)}$$

$$\frac{}{\begin{array}{c}\texttt{rec}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;\texttt{fold}[\,t\,.\,\tau\,](\,e_2\,)\,)\\ \longmapsto \\ \{\texttt{map}^{+}[\,t\,.\,\tau\,](\,y\,.\,\texttt{rec}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;y\,)\,)(\,e_2\,)/x\}e_1\end{array}}$$

$$\frac{[e_2\ \mathsf{val}]}{\texttt{gen}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;e_2\,)\ \mathsf{val}}$$

$$\left[\frac{e_2 \longmapsto e_2'}{\texttt{gen}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;e_2\,) \longmapsto \texttt{gen}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;e_2'\,)}\right]$$

$$\frac{e \longmapsto e'}{\texttt{unfold}[\,t\,.\,\tau\,](\,e\,) \longmapsto \texttt{unfold}[\,t\,.\,\tau\,](\,e'\,)}$$

$$\frac{}{\begin{array}{c}\texttt{unfold}[\,t\,.\,\tau\,](\,\texttt{gen}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;e_2\,)\,)\\ \longmapsto \\ \texttt{map}^{+}[\,t\,.\,\tau\,](\,y\,.\,\texttt{gen}[\,t\,.\,\tau\,](\,x\,.\,e_1\,;y\,)\,)(\,\{e_2/x\}e_1\,)\end{array}}$$

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

1. The empty queues are related: $R(e_{\mathsf{m}}, e'_{\mathsf{m}})$.
2. Inserting the same element on each of two related queues yields related queues: if $d : \tau$ and $R(q, q')$, then $R(e_{\mathsf{i}}(\, d\,)(\, q\,), e'_{\mathsf{i}}(\, d\,)(\, q'\,))$.
3. If two queues are related, then either they are both empty, or their front elements are the same and their back elements are related: if $R(q, q')$, then either
   (a) $e_{\mathsf{r}}(\, q\,) \longmapsto^* \texttt{null}$ and $e'_{\mathsf{r}}(\, q'\,) \longmapsto^* \texttt{null}$, or
   (b) $e_{\mathsf{r}}(\, q\,) \longmapsto^* \texttt{just}(\,\langle d, r\rangle\,)$ and $e'_{\mathsf{r}}(\, q'\,) \longmapsto^* \texttt{just}(\,\langle d, r'\rangle\,)$ and $R(r, r')$.

One proviso is that the requirement in the third condition that the dequeued element, d, be the same in both cases is rather strong for elements of function type, or when a lazy dynamics is used. To be more accommodating requires the use of logical equivalence, as defined in Chapter 46.

- Chapter 19: the name **PCF** was introduced by Gordon Plotkin, meaning "Programming language for Computable Functions," referring to Dana Scott's "Logic for Computable Functions." The term "universal function" is synonymous with "evaluator" or "interpreter" that is used in computability theory. The Blum Size Theorem can be found in the textbook "Introduction to the General Theory of Algorithms" by Michael Machtey and Paul Young (Elsevier, 1978).
- Chapter 20: the name **FPC** is taken from Carl Gunter's textbook "Semantics of Programming Languages."
- Chapters 29, 31, and 33 refer to **MPCF**, which is never defined in the printed edition. See the supplemental notes on the book home page for the intended modal formulation of **PCF**.
- Chapter 33, Section 2: It would be preferable, and consistent with later chapters, to rename the operator `isof` to `isinref`.
- Chapter 33: It is preferable to avoid checking disequality of names in favor of making explicit the symbol declarations, relying on there being no repeated declarations in any signature.
- Chapter 34: Replace `if` by `ifz` to avoid confusion with other languages.
- Chapter 36: Clarify that the term "by-name" is being used for the semantic concept, and "by-need" for the implementation strategy.
- Chapter 38: The producer-consumer example could be altered to permit more parallelism by making the head elements future natural numbers, and eliminating the use of futures on the tails.
- Chapter 40 Concurrent Algol. The dynamics specifies *synchronous* communication, in which an emit of a message does not terminate until the

$$\frac{\vdash_\Sigma e : \texttt{clsfd} \quad e\ \mathsf{val}_\Sigma}{\vdash_\Sigma !\, e\ \mathsf{proc}} \qquad \frac{e\ \mathsf{val}_\Sigma}{!\, e \xrightarrow[\Sigma]{e\,!} \mathbf{1}}$$

$$\frac{e\ \mathsf{val}_\Sigma}{\texttt{emit}(\, e\,) \overset{\varepsilon}{\underset{\Sigma}{\Longrightarrow}} \texttt{ret}\,\langle\rangle \otimes !\, e}$$

Figure 2: Asynchronous Dynamics for Concurrent Algol

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 $\texttt{inref}(\, e_1\, ; e_2\,)$ in place of $\texttt{mk}(\, e_1\, ; e_2\,)$ to stress its role as the dynamic analogue of $\texttt{in}\langle a\rangle(\, e\,)$, much as $\texttt{getref}(\, e\,)$ is the dynamic analogue of $\texttt{get}\langle a\rangle$ for assignables.

Acknowledgements

Thanks to Carlo Angiuli, Stephanie Balzer, Peter Brattveit Bock, Corwin de Boor, Jordan Brown, Nathan Collins, Karl Crary, Harrison Grodin, Jan Hoffmann, Kuen-Bang Hou (Favonia), Yuan Ji, Alex Kavvos, Daniel R. Licata, Muhammed Hussein Nasroliahput, Max Orhai, Kevin Quick, Shawn Rashid, Giselle Reis, Nicholas Roberts, Selva Samuel, Kartik Singhal, Jon Sterling, Anthony Su, Stephanie Weirich, James R. Wilcox, Yongwei Yuan, Haoxuan (Aaron) Yue, Yu Zhang, Fangyi Zhou, Jiaqi Zuo, Jake Zimmerman.