

# 1 Enhancing Interactive Theorem Prover 2 Error Messages with Hints

3 Maria Khakimova

4 Delft University of Technology, The Netherlands

5 Sára Juhošová   

6 Delft University of Technology, The Netherlands

7 Jaro Reinders   

8 Delft University of Technology, The Netherlands

9 Jesper Cockx   

10 Delft University of Technology, The Netherlands

## 11 — Abstract —

12 Interactive theorem provers (ITPs) are promising tools for ensuring program correctness, but users  
13 often complain about their poor usability and steep learning curve. A common complaint, especially  
14 among new users, are confusing error messages that expose details of the ITP’s underlying theory  
15 or implementation details. In this work, we investigate how adding hints to three types of scope  
16 and type checking error messages in the Agda ITP affects the new users’ debugging experience.  
17 We evaluate the effectiveness and perceived helpfulness of those error messages by conducting a  
18 between-subjects user study where we provide a series of Agda code snippets, each containing a  
19 single error that the participants have to fix based on the error message. We measure the success  
20 rate, time taken to fix the error, and perceived helpfulness for each code snippet with the original as  
21 well as the enhanced error message and determine the statistical significance of adding the hint. Our  
22 results show that *correct* hints can improve the success rate and time taken to fix the error, and that  
23 error messages with hints are rated significantly more helpful than those without. Additionally, we  
24 find that while error messages with *incorrect* hints are often rated as more misleading, they do not  
25 significantly impact the success rate or time taken to fix the error. These results show that adding  
26 hints to error messages is a viable step on the path towards making ITPs more widely accessible.

27 **2012 ACM Subject Classification** Human-centered computing → Empirical studies in interaction  
28 design

29 **Keywords and phrases** Agda, error messages, hints, new users

30 **Digital Object Identifier** 10.4230/LIPIcs.ITP.2016.

## 31 **1** Introduction

32 Designed as programming languages with type systems strong enough to allow static pro-  
33 gram verification, dependently-typed interactive theorem provers (ITPs) such as Agda [32],  
34 Coq/Rocq [38], and Lean [15] can produce software that is completely verified with respect  
35 to its specification. This capability should, theoretically, make ITPs *the* choice tool for pro-  
36 grammers implementing critical software components, but we are yet to see their spread into  
37 mainstream development processes. The most prominent obstacles that new users face are  
38 poor usability and steep learning curves, with previous research showing that the languages’  
39 ecosystems do not provide adequate support for their users [24]. New users in particular  
40 claim to struggle with confusing error messages which “require theoretical knowledge and  
41 experience to be helpful” [23, p. 166].

42 Encountering error messages is a major part of programming, with Java programmers  
43 spending 13%–25% of their time reading compiler error messages [5]. The feedback that  
44 error messages provide aids programmers in understanding the roots of their errors and



© Maria Khakimova, Sára Juhošová, Jaro Reinders, and Jesper Cockx;  
licensed under Creative Commons License CC-BY 4.0

Seventeenth Conference on Interactive Theorem Proving (ITP 2026)

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## XX:2 Enhancing Interactive Theorem Prover Error Messages with Hints

45 brings them closer to a working program, which is especially important for new users who  
46 have limited experience with the language. Confusing error messages can cause students  
47 to lose confidence [8], demotivate potential users from learning the language, or even lead  
48 programmers away from the correct solution [28]. Such error messages are known to cause  
49 significant frustration among new users, and are frequently considered to be a “barrier to  
50 progress” [7]. We therefore anticipate that improving ITP error message helpfulness could  
51 reduce perceived barriers for new users and make ITPs more accessible to a wider range of  
52 programmers.

53 In a literature review on the effectiveness of error messages and their enhancements,  
54 Becker et al. [8] provided ten ways to enhance error messages:

- |    |                           |    |                                      |
|----|---------------------------|----|--------------------------------------|
| 55 | 1. increase readability,  | 60 | 6. show solutions or hints,          |
| 56 | 2. reduce cognitive load, | 61 | 7. allow dynamic interaction,        |
| 57 | 3. provide context,       | 62 | 8. provide scaffolding,              |
| 58 | 4. use a positive tone,   | 63 | 9. use logical argumentation, and    |
| 59 | 5. show examples,         | 64 | 10. report errors at the right time. |

65 There are two main research gaps in existing literature on enhancing ITP error messages  
66 (discussed in detail in Section 6) that are relevant to our work:

- 67 1. there is limited research that isolates the effect of *one* enhancement on the usability of  
68 error messages in general, and
- 69 2. none of the research focusing on error messages in ITPs conducts a *user study* to evaluate  
70 the effects of the improved error messages.

71 To close these two gaps, we enhanced an ITP’s error messages with one recommendation  
72 from the guidelines above, and evaluated the obtained enhanced error messages (EEMs)  
73 against the original error messages (OEMs) with a user study.

74 Out of the potential enhancements, we chose to provide hints or solutions to the pro-  
75 grammer. Although there is a lot of empirical evidence that supports the positive impact of  
76 hints in error messages [41, 34, 20, 6], Marceau et al. warn against providing hints because  
77 they can lead users “down the wrong path” [28]. Furthermore, despite the fact that papers  
78 providing empirical evidence for showing solutions or hints were most common in Becker  
79 et al.’s comprehensive literature review [8], all those studies combined the addition of hints  
80 with other enhancements. Therefore, it is difficult to isolate the effect that enhancing error  
81 messages with hints has on the usability of the error messages.

82 In this paper, we answer the following main research question:

83 **RQ:** How do hints in ITP error messages affect their usability for new users?

84 Firstly, we want to understand the practical influence of hints on the new users’ ability to  
85 fix errors. However, we also want to investigate whether hints can change how users *perceive*  
86 such error messages — after all, we wish to eliminate confusing error messages from the  
87 *perceived* obstacles they face when learning to use an ITP. As such, we use the following  
88 sub-questions to answer our main research question:

89 **RQ1** How do hints in ITP error messages affect the ability of new users to fix errors?

90 **RQ1.1** How do hints affect the success rate of new users fixing errors?

91 **RQ1.2** How do hints affect the time new users spend fixing an error?

92 **RQ2** How do new users perceive the helpfulness of hints in ITP error messages?

To answer these research questions, we chose to enhance three error messages in Agda’s scope and type checker. Agda is often used in computer science education and provides a good opportunity for studying the effect of its error messages on users who are new to ITPs but not necessarily new to programming. We make the following contributions in this work:

- a design for adding hints to three error messages in Agda’s scope and type checker;
- a user study design for understanding the effect of those hints, with all resources provided in our data repository under an MIT license [4]; and
- a detailed analysis of how hints influence the usability of ITP error messages for new users.

Our results show that *correct* hints can improve the success rate and time taken to fix the error, and that error messages with hints are rated significantly more helpful than those without. Additionally, we find that while error messages with *incorrect* hints are often rated as more misleading, they do not significantly impact the success rate or time taken to fix the error. As a result, we conclude that enhancing ITP error messages with hints is a viable way to improve their usability and recommend adding a hint where deemed potentially useful.

## 2 Hint Design

To investigate whether hints in ITP error messages are useful for new users, we first designed hints for some of Agda’s existing error messages. To pick which error messages to enhance, we turned to those identified as learning obstacles [23] and chose ones which we believe are idiosyncratic to Agda or ITPs in general. We focused on designing three enhanced error messages (EEMs) on top of the original error messages (OEMs) from Agda v2.8.0:

- WHITESPACE: forgetting whitespace between variables or operators;
- CONFUSABLE: using the wrong, visually confusable Unicode character; and
- TOOFEWARGS: not passing enough arguments to a function.

In the following section, we explain the general considerations that guided our design and then dive into the specific implementation of each enhanced error message.

One of the important principles in how *all* error messages are presented is *consistency* [40, p. 10]. Consistency serves to reduce cognitive load on the programmers, allowing them to more efficiently process the information provided by the compiler [8]. Work on enhancing error messages in Agda (in addition to all other languages) should thus strive to maintain as much consistency with existing error messages as possible.

To identify hints within Agda’s existing error messages, we needed to define what a “hint” is. For the purpose of this work, we used the following definition:

- a hint has to be an *addition* to the base error message, and
- a hint needs to provide a (potential) solution to the programmer.

Even with these guidelines, hints can come in many shapes and sizes: the order in which information is presented can vary; the hint can be phrased as a question, suggestion, or direction; and the framing phrases can vary (e.g., “possible fix: ...” or “you may want to ...”).

Though it was impossible to identify *all* the messages with hints in Agda’s codebase (consisting of over 185K lines of source code), looking at just the **Error** and **Warning** modules of the project already showed that there is a lot to be desired in terms of hint consistency within the current pool of error messages. We therefore looked at hints in *similar* error messages to the ones we were planning to implement. The only existing hints we found similar to the first two errors were phrased as questions, presented in parentheses, and framed using the “did you ...?” phrase (e.g., “(did you forget space around the ‘:’?)”). Similarly,

## XX:4 Enhancing Interactive Theorem Prover Error Messages with Hints

139 the error message for passing *too many* (as opposed to *too few*) arguments to a function had  
140 the following hint appended to the end: “(did you supply too many arguments?)”. As a  
141 result, we design all our enhanced error messages to use this parenthesised phrasing.

142 Ideally, each hint would be presented only in situations when applying the fix suggested  
143 by the hint would “fix” the code. This could be done by, for example, having Agda apply the  
144 proposed fix and rerunning the type checker before presenting the EEM. However, given the  
145 limitations of the Agda compiler, we were not able to do this. Currently, Agda’s compiler  
146 stops type checking upon encountering *an* error, meaning that it is only ever aware of one  
147 error at a time. As a result, if the rerun were to encounter another error, we would have no  
148 reliable way of determining whether it is caused by the applied “fix” or by some unrelated  
149 part of the code. Therefore, within this project, we chose to display the hints according to  
150 some heuristic (described in the following subsections). This means that some error messages  
151 might display irrelevant hints, leading programmers “down the wrong path” (as Marceau et  
152 al. warned [28]).

### 153 Whitespace

154 The first type of error we enhanced with a hint is the `[NotInScope]` error thrown when  
155 the programmer forgets to add spaces around mixfix operators [37] — an error identified  
156 to be a common obstacle for new Agda users [23]. This error arises because Agda allows  
157 programmers to define operators by adding underscores to function names. For example,  
158 when a function `_&&_ : Bool → Bool → Bool` is defined, it can be called in three ways:

- 159 ■ by applying the arguments in the standard way: `_&&_ True False`;
- 160 ■ by applying the arguments infix: `True && False`; or
- 161 ■ by only applying *some* of the arguments infix: `(_&& False) True`.

162 However, the infix arguments *must* be separated from the operator with a whitespace, since  
163 Agda will consider anything not separated by a whitespace as one name. For example, if we  
164 would write `True&& False`, Agda would think we are trying to apply the function named  
165 `True&&` to `False` (which will throw a `[NotInScope]` error, since it does not exist).

166 To help guide users, especially the ones that are not used to Agda’s syntax, we added  
167 a hint telling them they might have forgotten to add a whitespace *when the not-in-scope*  
168 *name can be broken down completely into names that are in scope*. Note, though, that not  
169 all the options provided would actually solve the problem — some only satisfy the scope  
170 checker but not necessarily the type checker (e.g., `pancake` might be split into `pan` and `cake`  
171 if both are in scope, but applying `pan` to `cake` might not result in a value of the type that  
172 was expected in that position). The following demonstrates an example of the `WHITESPACE`  
173 enhanced error message, with the assumption that variables `sn` and `d` are also in scope:

```
174 swap : A × B → B × A  
175 swap (fst , snd) = (snd, fst)  
176
```

```
178 /home/me/examples/swap.agda:2.21-25: error: [NotInScope]  
179 Not in scope:  
180 snd, at /home/me/examples/swap.agda:2.21-25  
181 + (did you forget whitespace in  
182 +   'snd ,' or  
183 +   'sn d ,'?)  
184 when scope checking snd,  
185  
186  
187
```

(lines with “+” contain the enhancement)

### 188 Confusable

189 Agda supports the use of Unicode characters in function and variable names, which allows  
190 Agda proofs to look very similar to paper proofs, but “raises the barrier of entry” for new

191 users and creates confusion during program comprehension [23, p. 165]. Since the set of  
 192 Unicode characters contains many “confusables” (characters that are visually similar or  
 193 identical to each other), it can often be difficult to understand an error telling you that “=”  
 194 is not the same as “=̂” (a problem that gets worse with certain fonts). This is a relatively  
 195 common complaint from the students learning Agda at our university, but also users on the  
 196 internet [21]. Similar complaints have been bought up in other languages that offer Unicode  
 197 support, such as in Julia [1].

198 Since Unicode characters are ubiquitously used in Agda and Agda’s libraries, it is no  
 199 surprise that efforts have already been made to provide support to users for this scenario. In  
 200 the `agda-mode` plugin for the Emacs editor, users can use a command to get information  
 201 about the character the cursor is positioned over [36]. However, the programmer needs to  
 202 both know that this command exists, and think to check for the two characters that have  
 203 been confused. If the wrong confusable character was used within a larger string, this can be  
 204 challenging to identify. Furthermore, this option is only available in Emacs and programmers  
 205 who use a different development environment (e.g., VS Code<sup>1</sup> or Vim) will not have easy  
 206 access to this information.

207 The best way to provide this information such that the code can be fixed easily is through  
 208 a hint in the error message that is caused by precisely such a mismatch. The addition of  
 209 such a hint has also been suggested several times by Agda users [17, 29], but is yet to be  
 210 implemented by the Agda developers. For the design of this hint, we took inspiration from  
 211 the way Rust [27] presents these types of errors, and prioritised the following information to  
 212 display to the user:

- 213 ■ the location of the confusable character(s) within the name,
- 214 ■ what the confusable characters are (with their identifiers), and
- 215 ■ how to input both the correct and incorrect characters with `agda-mode`.

216 The hint is displayed whenever a name is not in scope and there is at least one name in  
 217 scope in which all the characters (in order) are either exactly the same or confusable. To  
 218 determine whether characters are confusable, we used the `text-icu` package<sup>2</sup> that uses the  
 219 `confusablesWholeScript.txt`<sup>3</sup> file for spoof-checking [13]. The `agda-mode` Unicode input  
 220 key bindings are drawn from a list compiled by Chonavel [14]. The following demonstrates  
 221 an example of the CONFUSABLE enhanced error message:

```

222 id : Set → Set
223 id baρ = baρ
224
225
226 /home/me/examples/alpha.agda:2.10-13: error: [NotInScope]
227 Not in scope:
228 baρ at /home/me/examples/alpha.agda:2.10-13
229 + (did you accidentally use a confusable character?)
230 + You typed:    baρ
231 + In scope:    baρ
232 + (diff)      -^^
233 +
234 +   Character      Character name      agda-mode input  Emacs input
235 + ----> 'a' (0x1d552)  MATH DOUBLE-STROKED SMALL A  \ba           C-x 8 RET 1d552
236 +   vs 'α' (0x3b1)    GREEK SMALL LETTER ALPHA     \Ga           C-x 8 RET 3b1
237 + ----> 'ρ' (0x3c1)    GREEK SMALL LETTER RHO       \Gr           C-x 8 RET 3c1
238 +   vs 'p' (0x70)     LATIN SMALL LETTER P         \p            C-x 8 RET 70
239 +
240 when scope checking baρ
  
```

<sup>1</sup> VS Code does have extensions that can identify a Unicode character [42, 22], but no references are made to them in the Agda documentation.

<sup>2</sup> <https://hackage.haskell.org/package/text-icu-0.8.0.5>

<sup>3</sup> <http://unicode.org/Public/security/latest/confusablesWholeScript.txt>

## XX:6 Enhancing Interactive Theorem Prover Error Messages with Hints

242

(lines with “+” contain the enhancement)

243 There are a few limitations to this solution:

- 244 ■ **The definition of which characters are confusable is not Agda-specific.** There  
245 are many characters that are important to Agda programmers that are not covered by  
246 `confusablesWholeScript.txt` (e.g., “<” and “{”)
- 247 ■ **Confusability is only checked against names in scope.** There is also a chance that  
248 the programmer used a confusable character instead of a *reserved name*, in which case  
249 our implementation does not add the hint.
- 250 ■ **Unicode characters with a non-standard length cause misalignment of the  
251 diff in the error message.** Some Unicode characters have a different size than the  
252 standard, even in monospace font, and can misalign the `diff` strings and arrows.

### 253 TooFewArgs

254 The final error message we enhanced was an `[UnequalTerms]` type checking error which  
255 arises when too few arguments are passed to a function. For simplicity, we only provide the  
256 name of the function in the hint. A more sophisticated hint could also include the location  
257 of the function definition and the types of the missing arguments.

258 Unfortunately, due to polymorphism and type inference it is not always clear when too  
259 few arguments are supplied to a function call. In our design, the EEM is shown when the  
260 error appears at a function call site and the inferred type of the function application is a  
261 function type. This does mean that, for example, when the programmer does not supply the  
262 function with *any* arguments, the hint will not appear, since the error is not at the “call site”  
263 of a function. The following demonstrates an example of the `TOOFEWARGS` enhanced error  
264 message:

```
265 addThree : Nat → Nat → Nat → Nat
266 addThree a b c = a + b + c
267 num : Nat
268 num = addThree 1 2
269
```

```
271 /home/me/examples/args.agda:4.7-19: error: [UnequalTerms]
272 Nat → Nat !=< Nat
273 when checking that the inferred type of an application
274   Nat → Nat
275 matches the expected type
276   Nat
277 + (did you supply too few arguments to addThree ?)
278
```

280

(lines with “+” contain the enhancement)

### 281 3 Study Setup

282 To answer our research questions, we conducted a between-subjects user study on a class  
283 of final-year bachelor students learning to use Agda for the first time. We designed code  
284 snippets with errors they had to find and fix, collecting the **timestamp and compilation  
285 status of each compilation attempt**, as well as their **opinion on the helpfulness of  
286 the error message** using a five-point Likert scale. We describe the participants, study  
287 design, and data processing in the following section. The study was conducted with the  
288 approval of the human research ethics committee of our university.

### 289 3.1 Context & Participants

290 The participants in this study were final-year computer science bachelor students taking an  
291 elective Functional Programming course at our university. The course held lectures over  
292 seven weeks, consisting of ten lectures on functional programming in Haskell and four Agda  
293 lectures with the goal of teaching students to

- 294 ■ interactively develop Agda programs,
- 295 ■ use the Curry-Howard correspondence [35, p. 108] to express logical properties as types,
- 296 ■ use indexed data types and dependent pattern matching to enforce invariants of their  
297 programs, and
- 298 ■ formally prove properties of purely functional programs by using the identity type and  
299 equational reasoning.

300 The students were introduced to these topics during live lectures and were given programming  
301 exercises to practice on. This was the first time the students came into contact with  
302 an ITP and, as such, could be labelled “new users”. They were encouraged to use the  
303 `agda-mode` extension in the Visual Studio (VS) Code editor. However, their work was  
304 submitted, compiled, and verified through submission to an online learning management  
305 system developed at our university (described in Section 3.2.3).

### 306 3.2 Study Design

307 To determine whether hints improve the developer experience of new users when fixing  
308 errors, we designed a between-subjects experiment with ten debugging assignments for each  
309 participant to attempt. Each assignment featured

- 310 ■ a code snippet that contained *exactly one* compilation error, simulating being in the  
311 middle of writing or debugging the code, and
- 312 ■ a follow-up question, asking participants to rate the helpfulness of the error message on a  
313 five-point Likert scale.

314 The participants’ task was to run the compiler on the code snippet, read the error message,  
315 and enter a cycle of attempting to fix the error and recompile the code until it succeeded.  
316 We then used the status and timestamp of each compilation attempt to answer **RQ1** and  
317 the Likert scale ratings to answer **RQ2**.

#### 318 3.2.1 Code Snippet

319 We designed the study such that each participant would receive one debugging assignment  
320 for each “correct” EEM (i.e., offering the hint that will actually fix the error). Because  
321 WHITESPACE and TOOFEWARGS can also appear in incorrect situations (i.e., they are hinting  
322 at something that will not fix the error), we added an assignment for both a *correct* and  
323 *incorrect* “error message intention” for these two hint types. The CONFUSABLE hint appears  
324 only in very specific situations, and we could not find a sufficiently realistic scenario to  
325 provide for an *incorrect* situation. We deemed it unnecessary to include such an assignment  
326 in the study.

327 Furthermore, we wanted participants to encounter each error message in both original  
328 and enhanced form, but did not want them to see the same code twice. That would give  
329 them a chance to use prior experience with the code to fix the error, thus skewing the  
330 results. Therefore, we designed two assignments for each possible hint type and error message  
331 intention and used a between-subjects design to distribute two variants of the survey. For  
332 example, assignment TOOFEWARGS#A used the following code snippet and EEM:

## XX:8 Enhancing Interactive Theorem Prover Error Messages with Hints

```
333
334 data Fin : Nat → Set where
335   zero : {n : Nat} → Fin (suc n)
336   suc  : {n : Nat} → Fin n → Fin (suc n)
337
338 data Vec (A : Set) : Nat → Set where
339   []      : Vec A zero
340   _::_   : {n : Nat} → A → Vec A n → Vec A (suc n)
341 infixr 5 _::_
342
343 updateElement : {A B : Set}{n : Nat}
344   → Vec A n → Fin n → B → (A → B → A) → Vec A n
345 updateElement (x :: xs) zero   b f = f x :: xs
346 updateElement (x :: xs) (suc i) b f = x :: updateElement xs i b f
347
348 /home/student/solution.agda:12.39-42: error: [UnequalTerms]
349 (B → A) !=< A
350 when checking that the inferred type of an application
351   B → A
352 matches the expected type
353   A
354 + (did you supply too few arguments to f ?)
355
356 (lines with “+” contain the enhancement)
```

358 Similarly, `TOOFEWARGS#B` was designed to have the same style of error, and was provided  
359 to a participant with the opposite type of error message to `TOOFEWARGS#A`:

```
360
361 data Vec (A : Set) : Nat → Set where
362   []      : Vec A zero
363   _::_   : {n : Nat} → A → Vec A n → Vec A (suc n)
364 infixr 5 _::_
365
366 _+_ : {A : Set}{n m : Nat} → Vec A m → Vec A n → Vec A (m + n)
367 []   ++ ys = ys
368 (x :: xs) ++ ys = x :: (xs ++ ys)
369 infixr 5 _+_
370
371 zipWith : {A B C : Set}{n : Nat}
372   → (A → B → C) → Vec A n → Vec B n → Vec C n
373 zipWith f [] [] = []
374 zipWith f (x :: xs) (y :: ys) = f x y :: zipWith f xs
375
376 /home/student/solution.agda:14.42-54: error: [UnequalTerms]
377 (Vec _B_24 _n_26 → Vec _C_25 _n_26) !=< Vec C n
378 when checking that the inferred type of an application
379   Vec _B_24 _n_26 → Vec _C_25 _n_26
380 matches the expected type
381   Vec C n
382 + (did you supply too few arguments to zipWith ?)
383
384 (lines with “+” contain the enhancement)
```

386 An overview of the assignments in each variant is available in Table 1.

387 In order to keep consistent with the experience level of our participants, the code snippets  
388 were inspired by existing exercises from the course. This meant that we did not use the  
389 Agda standard library, instead providing the needed functionality in student-visible “library”  
390 code. The syntax of this “library” code also uses fewer Unicode characters than the Agda  
391 standard library, since the online environment we used does not support Emacs-style Unicode  
392 insertion methods. Additionally, some exercises were adapted from the Agda standard  
393 library<sup>4</sup>, the Iowa Agda library<sup>5</sup>, and the *Programming Language Foundations in Agda*<sup>6</sup>

<sup>4</sup> <https://agda.github.io/agda-stdlib/>

<sup>5</sup> <https://github.com/cedille/ial>

<sup>6</sup> <https://plfa.github.io/>

intention	error message	tag	in variant		error line	fix
			A	B		
<i>correct</i>	WHITESPACE	#A	EEM	OEM	24/24	(x, y) → (x , y)
		#B	OEM	EEM	23/25	~b → ~ b
	CONFUSABLE	#A	EEM	OEM	20/22	cyrillic "c" → latin "c"
		#B	OEM	EEM	37/45	syllabics hyphen "=" → equals sign "="
	TOOFEWARGS	#A	EEM	OEM	13/14	f x → f x b
		#B	OEM	EEM	15/15	zipWith f xs → zipWith f xs ys
<i>incorrect</i>	WHITESPACE	#C	EEM	OEM	23/23	xs,i → xs i
		#D	OEM	EEM	44/54	+-id → +-identity
	TOOFEWARGS	#C	EEM	OEM	18/18	foldr x ys → intersperse x ys
		#D	OEM	EEM	24/24	f (f a) a → f b a

■ **Table 1** An overview of the debugging assignments used in the study, including the “expected” fix for each assignment and the line on which that fix had to be applied

394 book. All assignments are publicly available in our data repository [4], including the code,  
 395 the accompanying original and enhanced error messages, and the expected “corrected” code.  
 396 For quick reference, Table 1 contains an overview of the fix each assignment required.

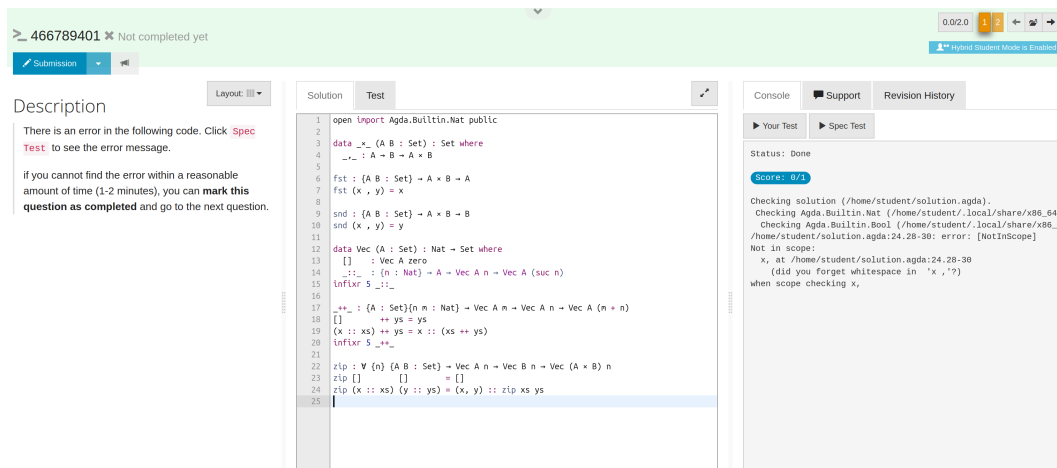
### 397 3.2.2 Likert Scale Ratings

398 Each debugging assignment was concluded by a multiple-choice question with a five-point  
 399 Likert scale, asking participants to rate the helpfulness of the provided error message on a  
 400 scale of “very helpful” to “very misleading”. We opted to include the neutral “neither helpful  
 401 nor misleading”. A sixth “(N/A): Did not read the error message” was added to differentiate  
 402 from the neutral option, and to help us eliminate responses which did not focus on the object  
 403 of study (the error message).

### 404 3.2.3 Programming Environment

405 We used an online system which allows courses to build exercises in a single environment.  
 406 Importantly, it allows for the creation of programming assignments, which students can edit,  
 407 compile, and test directly in the environment. By using this system, we were able to  
 408 ■ use two different versions of Agda (one with the original and one with the enhanced error  
 409 messages), without having to ask the participants to install and configure them,

## XX:10 Enhancing Interactive Theorem Prover Error Messages with Hints



■ **Figure 1** The online environment interface displaying the WHITESPACE#A assignment

410 ■ randomise which variant each participant got,  
411 ■ randomise the order in which each participant got the assignments, and  
412 ■ collect the timestamp and status of each compilation attempt in one system.  
413 Additionally, the students were familiar with the system from this and previous courses,  
414 minimising the influence that the programming environment had on our results. Figure 1  
415 shows the interface of a programming assignment in the system from a students' perspective.

### 416 3.3 Pilot & Distribution

417 Before distributing the survey, we conducted a pilot on four people (two for each variant)  
418 with varying levels of Agda experience. The purpose of this pilot was to identify any technical  
419 issues with the environment setup, check for spelling mistakes, ensure the difficulty was at  
420 the right level, and get an estimate of how long the full survey would take to complete. Three  
421 of the pilot participants had more experience with Agda than the target participants, and  
422 one was a new Agda user.

423 After looking at the pilot participants' statistics and conducting an exit interview with  
424 them, we judged the difficulty to be appropriate but the duration to be surprisingly long.  
425 On average, the more experienced participants took 16 minutes, while the new Agda user  
426 took 35 minutes. Most of the delay was caused by spending too much time on an error  
427 they were not able to fix. Based on these findings, we added a line to the description of  
428 each assignment urging participants to continue on to the next assignment if they could not  
429 complete it "within a reasonable amount of time (1-2 minutes)".

430 We distributed the survey during the last lecture of the course as well as via an an-  
431 nouncement posted on the main communication page of the course. The students were given  
432 ten minutes of lecture time to complete the survey, with the option to continue into the  
433 15-minute mid-lecture break. Alternatively, they could complete it on their own time in the  
434 eleven-day window during which it was available.

### 435 3.4 Data Cleaning

436 Because some participants only opened the survey but did not actually participate, we had  
437 to differentiate their response from the valid ones. We removed a participant (and their

438 submissions) from our dataset when the data showed they attempted only a single assignment  
439 and their “submission” to that assignment contained only a single timestamp (i.e., the first  
440 compilation) and did not have a rating for the perceived helpfulness. We reasoned that  
441 though this data *could* be relevant to the success rate, we had no way of distinguishing that  
442 from participants who simply got distracted or lost interest when they saw the format of the  
443 study. This removed a total of 17 participants (and their 17 single submissions).

444 Additionally, since the user study was open for eleven days, and no strict order to the  
445 assignments was enforced, participants were free to stop debugging at any point and return  
446 to the assignment later. While this likely increased participation rates per assignment, it  
447 occasionally resulted in data saying that a participant took, e.g., 9 hours to debug a code  
448 snippet. This is (almost certainly) not true. We removed timing data for a submission from  
449 our dataset in three cases:

- 450 **1. If the participant did not successfully fix the error:** This removed 136 submissions  
451 (responses to an individual assignment) from our timing analysis.
- 452 **2. If the participant worked on another assignment before finishing the current  
453 submission:** For each submission  $S_1$ , if there was another submission  $S_2$  from the same  
454 participant that had a compilation timestamp between the start and end times of  $S_1$ ,  $S_1$   
455 was marked invalid. This removed an additional 9 submissions from our timing analysis.
- 456 **3. If the longest time interval between code compilations was unreasonable:** We  
457 estimated that the longest reasonable time a new user would spend between compilations  
458 is two minutes (this is also the time we instructed the participants to consider “reasonable”  
459 before moving on to the next assignment). As a sanity check, we ran the analysis both  
460 on data where all submissions with an interval larger than two minutes were cleaned,  
461 and on data where all submissions with an interval larger than five minutes were cleaned.  
462 There was no statistically significant difference between the two. As such, we left the  
463 two-minute cut-off and removed an additional 25 submissions from our timing analysis.

### 464 3.5 Data Analysis

465 To test for significance in the differences between the enhanced and original error messages,  
466 we used single-tailed Mann-Whitney U tests. This is a statistical test to “compare two  
467 independent groups that do not require large normally distributed samples” [30, p. 13], which  
468 made it an appropriate choice for this study. In a Mann-Whitney U test, there are two  
469 hypotheses:

- 470 ■ the null hypothesis  $H_0$ , which states that there is no difference in the distributions of the  
471 two groups, and
- 472 ■ the alternative hypothesis  $H_1$ , which is against  $H_0$ .

473 A single-tailed test has an alternative hypothesis that suggests the *direction* of the difference  
474 between the two group (either positive or negative). In our case, this would allow us to test  
475 if EEMs are better or worse than OEMs, depending on whether the error message offered a  
476 *correct* or *incorrect* hint. All null and alternative hypotheses used in our study can be seen  
477 in Table 2.

478 Based on these hypotheses, we calculated the  $U$  value for each result (i.e., how often  
479 results from one type of error message are larger than from the other). In Section 4, we  
480 specify the  $p$ -value for each result (i.e., the likeliness of observing a  $U$  value this extreme  
481 if there was no difference between the two distributions). To determine if the evidence for  
482 the alternate hypothesis was statistically significant, all analyses used a significance level of  
483  $p < 0.05$ , as was proposed by Fisher [16]. We report the actual  $p$ -values unless the  $p$  value

## XX:12 Enhancing Interactive Theorem Prover Error Messages with Hints

variable	intention	hypotheses
success rate	<i>correct</i>	$H_0$ : There is no difference between the success rates of participants with OEMs and EEMs. $H_1$ : Participants with the EEM have a <i>higher</i> success rate than participants with the OEM.
	<i>incorrect</i>	$H_1$ : Participants with the EEM have a <i>lower</i> success rate than participants with the OEM.
timing	<i>correct</i>	$H_0$ : There is no difference in the time taken to resolve the error between participants with OEMs and EEMs. $H_1$ : Participants with the EEM take <i>less</i> time to resolve the error than participants with the OEM.
	<i>incorrect</i>	$H_1$ : Participants with the EEM take <i>more</i> time to resolve the error than participants with the OEM.
perception	<i>correct</i>	$H_0$ : There is no difference between the perceived helpfulness of OEMs and EEMs. $H_1$ : EEMs are rated as <i>more helpful</i> than OEMs.
	<i>incorrect</i>	$H_1$ : EEMs are rated as <i>more misleading</i> than OEMs.

■ **Table 2** The hypotheses for all variables

484 was less than 0.001, in which case we reported it as  $p < 0.001$ . This is in accordance with  
485 the common guidance on  $p$ -value reporting [2].

## 486 **4 Results & Implications**

487 After cleaning our data, we recorded a total of 56 participants in our study (divided 32 / 24  
488 over the variants). Each participant completed an average of 7 assignments (with a median  
489 of 10), for a total of 412 submissions (divided 228 / 184 over the variants). This meant that  
490 each assignment received between 37 and 55 responses with a mean of 41 and a median of 38  
491 (original and enhanced combined). The data and the  $p$ -values from the Mann-Whitney U  
492 analysis (see Section 3.5) are summarised in Table 3. We use these to answer our research  
493 questions below. The response data as well as the assignments used in the study are available  
494 under an MIT Licence in our public data repository [4].

495 **RQ1:** How do hints in ITP error messages affect the ability of new users to fix errors?

496 Overall, there is strong evidence that EEMs with *correct* hints *can* improve both the  
497 likelihood that new users are able to resolve an error (**RQ1.1**) and the time taken to resolve  
498 it (**RQ1.2**). We see from Table 3 that the CONFUSABLE EEM in particular had significant  
499 influence over the participants' ability to fix the errors, and that for those that succeeded at  
500 the WHITESPACE assignment, the EEM significantly improved their time taken. Conversely,  
501 we see that there is no convincing evidence showing that *incorrect* hints lower the participants'  
502 success rate or that they significantly increase the time taken to find the error.

503 **RQ2:** How do new users perceive the helpfulness of hints in ITP error messages?

504 Our data provides strong evidence for  $H_1$  for the perception of both error message  
505 intentions: *correct* EEMs are rated as *more helpful* than OEMs and *incorrect* EEMs are  
506 rated as *more misleading* than OEMs. The only outlier seems to be the *incorrect* intention  
507 of TOOFEWARGS where, on average, participants actually rated the EEM *higher* than the  
508 OEM (though not significantly). We suspect that this is because by telling a participant

	success rate		#	timing		p-value	perception					p-value		
	rate	$H_1: SO(u) < SE(u)$		mean (s)	median (s)		VM	M	N	H	VH		$\epsilon$	
<i>correct</i>														
		$H_1: SO(u) < SE(u)$			$H_1: SO(u) > SE(u)$		$H_1: SO(u) < SE(u)$							
WHITESPACE	OEM	31 / 39	79%	0.071	29	42 $\wedge$	26 $\wedge$	5	9	8	10	5	1	<0.001
	EEM	33 / 36	92%	$\times$	30	29 $\vee$	22 $\vee$	0	0	2	3	30	1	$\checkmark$
CONFUSABLE	OEM	15 / 42	36%	<0.001	11	98 $\wedge$	107 $\wedge$	6	8	9	6	5	8	<0.001
	EEM	32 / 38	84%	$\checkmark$	31	60 $\vee$	54 $\vee$	3	1	2	5	25	2	$\checkmark$
TOOFEWARGS	OEM	30 / 44	68%	0.369	21	77 $\vee$	75 $\vee$	2	6	16	17	1	2	<0.001
	EEM	35 / 49	71%	$\times$	31	102 $\wedge$	77 $\wedge$	1	4	3	26	12	3	$\checkmark$
<i>overall</i>														
OEM	76 / 125	61%	<0.001	61	64 $\vee$	50 $\vee$	50 $\vee$	13	23	33	33	11	11	<0.001
EEM	100 / 123	81%	$\checkmark$	92	64 $\vee$	51 $\wedge$	51 $\wedge$	4	5	7	34	67	6	$\checkmark$
<i>incorrect</i>														
		$H_1: SO(u) > SE(u)$			$H_1: SO(u) < SE(u)$		$H_1: SO(u) > SE(u)$							
WHITESPACE	OEM	26 / 41	63%	0.535	24	63 $\vee$	46 $\vee$	3	3	15	12	5	3	0.002
	EEM	27 / 42	64%	$\times$	27	76 $\wedge$	68 $\wedge$	8	15	6	8	3	2	$\checkmark$
TOOFEWARGS	OEM	24 / 40	60%	0.364	21	104 $\vee$	95 $\vee$	2	7	16	11	2	1	0.647
	EEM	23 / 41	56%	$\times$	17	131 $\wedge$	109 $\wedge$	0	9	13	11	2	6	$\times$
<i>overall</i>														
OEM	50 / 81	62%	0.424	45	82 $\vee$	74 $\vee$	74 $\vee$	5	10	31	23	7	4	0.024
EEM	50 / 83	60%	$\times$	44	97 $\wedge$	77 $\wedge$	77 $\wedge$	8	24	19	19	5	8	$\checkmark$
		rate	p-value	#	mean (s)	median (s)	timing	VM	M	N	H	VH	$\epsilon$	p-value
		success rate												

**Table 3** A summary of our dataset as well as the results of the Mann-Whitney U test ( $\checkmark$  = statistically significant;  $\times$  = not statistically significant; # = number of submissions considered; VM = very misleading; M = misleading; N = neutral; H = helpful; VH = very helpful;  $\epsilon$  = no rating given;  $\wedge$  = higher than other message type;  $\vee$  = lower than other message type)

## XX:14 Enhancing Interactive Theorem Prover Error Messages with Hints

509 that the number of arguments provided does not match what a function requires, they were  
510 able to at least pinpoint the problem area, even though they did not necessarily know how  
511 to fix it. For example, consider the (*incorrect*) TOOFEWARGS#C assignment:

```
512 intersperse x (y :: ys) = y :: x :: foldr x ys
513
515 solution.agda:18.37-47: error: [UnequalTerms]
516   (List _A_20 →_B_21) !=< List A
517   when checking that the inferred type of an application
518     List _A_20 →_B_21
519     matches the expected type
520     List A
521   + (did you supply too few arguments to foldr ?)
```

(lines with “+” contain the enhancement)

525 The solution was to replace `foldr` with `intersperse`:

```
526 intersperse x (y :: ys) = y :: x :: intersperse x ys
527
```

529 By pointing to `foldr`, participants knew they had to fix *something* about that particular  
530 function call, even though they might not have known what. On the other hand, the success  
531 rate for this particular debugging assignment actually *decreased* with the EEM, suggesting  
532 that the participants might have been misled by the error message due to unjustified trust in  
533 the hint being correct.

**RQ:** How do hints in ITP error messages affect their usability for new users?

534

535 From the results, we can conclude that adding hints to ITP error messages does improve  
536 their usability for new users. The actual influence they can have on the users’ ability to  
537 fix an error seems to depend on the type of error that is being enhanced, meaning that it  
538 might be worthwhile to pinpoint which types of errors would use hints effectively. However,  
539 it seems that there is no practical danger in adding hints even if they might not always be  
540 displayed at the correct moment. This leads us to believe that hints in ITP error messages  
541 are a safe and viable step on the path towards making ITPs more widely accessible.

542 What we found very interesting was just how important the hints seemed to be for the  
543 perceived helpfulness of the error messages. Despite *incorrect* hints having no significant  
544 influence on the success rate and timing, we saw that the participants recognised how  
545 misleading the error messages were. Even more strongly, our results show that *correct* hints  
546 clearly increased the perceived helpfulness of *all* error messages we studied. We feel that this  
547 is an important indicator that even if a small usability change might not have a big impact  
548 on how people use an ITP, it might still greatly influence their developer experience with it.  
549 Hopefully, hints in the correct error messages could help improve the “bad usability” image  
550 ITPs currently seem to be suffering from.

### 551 **5** Limitations & Threats to Validity

552 In order to contextualise the findings, we consider the limitations and threats that could  
553 have impacted the representativeness of the results:

- 554 ■ **Recruitment:** Since the participants were recruited during a lecture, this limits the  
555 sample population to a very specific group of people (i.e., bachelor-level, English-speaking,  
556 mostly male, computer science students with good attendance). This may not be  
557 representative of the general population of new Agda users, or of the users Agda might  
558 wish to attract.

- 559 ■ **Multiple Lecture Topics:** The code snippets in the study were based on different  
 560 exercises covered in the course. However, the distribution of the topics over the error  
 561 messages was not even, as they were randomly assigned. This could have made comparisons  
 562 between different groups of error messages less reliable. For example, the CONFUSABLE  
 563 snippets covered much more recent material than the TOOFEWARGS ones.
- 564 ■ **Delayed Exercise Shuffling:** To decrease fatigue effects and minimise collaboration, we  
 565 shuffled the code snippets for each participant. However, during the study, the shuffling  
 566 only activated after the participant opened the first assignment (TOOFEWARGS#A),  
 567 meaning that every participant received this assignment, skewing response rates.
- 568 ■ **User Interface:** The user interface of the code editor was not representative of the  
 569 standard Agda user interface. This was in part due to it using Haskell syntax highlighting,  
 570 which gave different information to the user than a local Agda editor would<sup>7</sup>. Further,  
 571 the environment did not provide functionality for entering non-ASCII Unicode, or display  
 572 column information for where the cursor was in the code. All of these discrepancies could  
 573 have skewed the results, as user interfaces are known to have a large impact on how  
 574 programmers write code [18].
- 575 ■ **The Code Snippets:** The participants were asked to fix errors in code they did not  
 576 write, without knowing the intent or thought process behind the code. This is likely not  
 577 a standard situation for an Agda user to be in.
- 578 ■ **Abandon Rate:** With the software used to distribute the user study, we had no way of  
 579 differentiating people who opened an exercise without intending to solve it, and those  
 580 who attempted to resolve the error without saving their changes to the code. Thus, the  
 581 real success rate might be higher than identified, despite the data cleaning we applied.
- 582 ■ **EEM Frequency:** Our study only looked at the effects of EEMs appearing *too frequently*,  
 583 but not at the effects of them not appearing frequently *enough* (i.e., in situations where  
 584 the EEM might be expected but does not appear).

## 585 6 Related Work

586 Even though interactive theorem provers are known to have usability issues, research focused  
 587 on addressing them is currently still scarce. Nevertheless, we highlight the most relevant  
 588 work on ITP usability in this section, and discuss research on enhancing error messages for  
 589 programming languages in general.

### 590 Usability of Interactive Theorem Provers

591 A pioneer in this field, Kadoda studied the usability of a variety of theorem provers and  
 592 developed a novel editor for formalising specifications [25]. In a later work, she collaborated  
 593 with Stone and Diaper on using the “Cognitive Dimensions” framework [19] to evaluate  
 594 educational proof editors. They found that having meaningful error messages has a high  
 595 effect on the learnability of such systems [26].

596 A more recent line of work by Beckert and Grebing studied the usability of the KeY  
 597 System, “a platform of software analysis tools for sequential Java” [3]. They found proof  
 598 guidance to be important for usability, including good feedback when a proof attempt  
 599 fails [11]. A later study with focus groups by Beckert et al. compared the KeY System to

<sup>7</sup> For example, in a normal Agda editor, syntax highlighting is not shown if type checking fails, which is a known complaint among users [23].

## XX:16 Enhancing Interactive Theorem Prover Error Messages with Hints

600 Isabelle [31] and found that unintuitive errors resulting from unexpected type inference are a  
601 drawback of Isabelle [12].

602 Most recently, Juhošová et al. investigated the obstacles that new users face when learning  
603 to use Agda specifically [23]. They found that confusing error messages are one of the most  
604 significant barriers for new users, among a variety of ecosystem obstacles.

### 605 Enhancing Error Messages

606 In 2019, Becker et al. conveniently conducted a literature review on enhancing error  
607 messages for programming languages in general [8]. It is important to note that these studies  
608 focus on programming novices, whereas we studied newcomers to Agda who already have  
609 programming experience in other languages.

610 Among the studies discussed in the literature review, the results are somewhat mixed,  
611 suggesting small or hard-to-measure benefits. Becker found that enhancing Java’s error  
612 messages reduces the number of errors made by novices and improves user experience [6, 9]  
613 and later confirmed these results with a control study [7]. Pettit et al., on the other hand,  
614 found no significant effect when testing enhanced error messages for C++ [33]. In a subsequent  
615 quiz-based experiment similar to our own, Becker et al. found enhanced error messages to  
616 be effective, but the signal may be weak, which could explain the conflicting results [10].

617 Similar to our findings, previous work reported on students often requesting and ap-  
618 preciating enhanced error messages with hints or proposed solutions, but did not study  
619 their practical influence on, e.g., the success rate. Marceau et al. even suggested that error  
620 messages should not propose solutions, as they might lead novices in the wrong directions [28].  
621 The main empirical study on enhanced error messages with hints was by Thieselton and  
622 Treude, who developed a tool for enhancing Python error messages by proposing solutions  
623 found on Stack Overflow [39]. Most participants in their study found the proposed solutions  
624 helpful, which correlates with our own observations.

## 625 **7** Conclusion

626 Based on our between-subject user study, we recommend adding a hint to ITP error messages  
627 where deemed to potentially improve usability. Hints can improve the helpfulness of the error  
628 message as perceived by the new user and provide practical help in terms of successfully fixing  
629 the error and time taken to do so. Additionally, they do not seem to have a practical influence  
630 on these variables when hinting at an incorrect solution. In the future, we recommend research  
631 into the following directions:

- 632 ■ identifying which ITP error messages are most likely to benefit from hints,
- 633 ■ studying the influence of hints on more experience users, and
- 634 ■ experimenting with other types of ITP error message enhancements.

### 635 Generative AI Statement

636 The only use of generative AI / LLMs in this work was via an activated GitHub Copilot  
637 plugin in the Pycharm editor, used when analysing the data.

### 638 — References —

- 639 1 Warning against Unicode Confusables - Internals & Design, January 2024. URL: <https://discourse.julialang.org/t/warning-against-unicode-confusables/108734>.

- 641 2 Herman Aguinis, Matt Vassar, and Cole Wayant. On Reporting and Interpreting Statistical  
642 Significance and p Values in Medical Research. *BMJ Evidence-Based Medicine*, 26(2):39–42,  
643 April 2021. doi:10.1136/bmjebm-2019-111264.
- 644 3 Wolfgang Ahrendt, Bernhard Beckert, Daniel Bruns, Richard Bubel, Christoph Gladisch,  
645 Sarah Grebing, Reiner Hähnle, Martin Hentschel, Mihai Herda, Vladimir Klebanov, Wojciech  
646 Mostowski, Christoph Scheben, Peter H. Schmitt, and Mattias Ulbrich. The KeY platform  
647 for verification and analysis of Java programs. In *Verified Software: Theories, Tools and  
648 Experiments*, pages 55–71. Springer, 2014. doi:10.1007/978-3-319-12154-3\_4.
- 649 4 Anonymous Authors. Dataset for: "Enhancing Interactive Theorem Prover Er-  
650 ror Messages with Hints", 2025. URL: [https://data.4tu.nl/private\\_datasets/  
651 d8fYf7cbn18qWThoAMoLMiRfi3LH\\_H\\_5PTiHouGkZSA](https://data.4tu.nl/private_datasets/d8fYf7cbn18qWThoAMoLMiRfi3LH_H_5PTiHouGkZSA).
- 652 5 Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-  
653 Hill, and Chris Parnin. Do Developers Read Compiler Error Messages? In *International  
654 Conference on Software Engineering (ICSE)*, pages 575–585. IEEE, 2017. ISSN: 1558-1225.  
655 doi:10.1109/ICSE.2017.59.
- 656 6 Brett Becker. An Exploration Of The Effects Of Enhanced Compiler Error Messages For  
657 Computer Programming Novices. Master's thesis, Technological University Dublin, 2015.  
658 URL: <https://arrow.tudublin.ie/ltdis/35>.
- 659 7 Brett A. Becker. An Effective Approach to Enhancing Compiler Error Messages. In *Technical  
660 Symposium on Computing Science Education*, pages 126–131. ACM, 2016. doi:10.1145/  
661 2839509.2844584.
- 662 8 Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian  
663 Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L.  
664 Pearce, and James Prather. Compiler Error Messages Considered Unhelpful: The Landscape of  
665 Text-Based Programming Error Message Research. In *Working Group Reports on Innovation  
666 and Technology in Computer Science Education*, ITiCSE-WGR '19, pages 177–210. Association  
667 for Computing Machinery, 2019. doi:10.1145/3344429.3372508.
- 668 9 Brett A. Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin,  
669 and Catherine Mooney. Effective compiler error message enhancement for novice pro-  
670 gramming students. *Computer Science Education*, 26(2-3):148–175, 2016. eprint: <https://doi.org/10.1080/08993408.2016.1225464>. doi:10.1080/08993408.2016.1225464.
- 671 10 Brett A. Becker, Kyle Goslin, and Graham Glanville. The Effects of Enhanced Compiler  
672 Error Messages on a Syntax Error Debugging Test. In *Technical Symposium on Computer  
673 Science Education*, SIGCSE '18, pages 640–645. Association for Computing Machinery, 2018.  
674 doi:10.1145/3159450.3159461.
- 675 11 Bernhard Beckert and S. Grebing. Evaluating the usability of interactive verification systems.  
676 *CEUR Workshop*, 2012.
- 677 12 Bernhard Beckert, Sarah Grebing, and Florian Böhl. A Usability Evaluation of Interactive  
678 Theorem Provers Using Focus Groups. September 2014. doi:10.1007/978-3-319-15201-1\_1.
- 679 13 Ben Hamilton. Data.Text.ICU.Spoof, 2015. URL: [https://hackage.haskell.org/package/  
680 text-icu-0.8.0.5/docs/Data-Text-ICU-Spoof.html](https://hackage.haskell.org/package/text-icu-0.8.0.5/docs/Data-Text-ICU-Spoof.html).
- 681 14 Lawrence Chonavel. Pull Request #7094: Move Unicode Keybindings into JSON File by  
682 Lawcho, 2024. URL: <https://github.com/agda/agda/pull/7094>.
- 683 15 Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer.  
684 The Lean Theorem Prover (System Description). In Amy P. Felty and Aart Middeldorp,  
685 editors, *Automated Deduction (CADE-25)*, pages 378–388. Springer International Publishing,  
686 2015. doi:10.1007/978-3-319-21401-6\_26.
- 687 16 R. A. Fisher. Statistical Methods for Research Workers. In Samuel Kotz and Norman L.  
688 Johnson, editors, *Breakthroughs in Statistics: Methodology and Distribution*, pages 66–70.  
689 Springer, New York, NY, 1992. doi:10.1007/978-1-4612-4380-9\_6.
- 690 17 Frederik Hanghøj Iversen. Issue #2898: Suggest Replacements for Various Unicode Characters,  
691 2018. URL: <https://github.com/agda/agda/issues/2898>.
- 692

- 693 **18** Gavin Gray, Will Crichton, and Shriram Krishnamurthi. An Interactive Debugger for Rust Trait  
694 Errors. *Artifact for An Interactive Debugger for Rust Trait Errors*, 9(PLDI):199:1293–199:1314,  
695 2025. doi:10.1145/3729302.
- 696 **19** T. R. G. Green and M. Petre. Usability Analysis of Visual Programming Environments: A  
697 ‘Cognitive Dimensions’ Framework. *Journal of Visual Languages & Computing*, 7(2):131–174,  
698 1996. doi:10.1006/jvlc.1996.0009.
- 699 **20** Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. What Would  
700 Other Programmers Do: Suggesting Solutions to Error Messages. In *SIGCHI Conference*  
701 *on Human Factors in Computing Systems*, CHI ’10, pages 1019–1028. ACM, 2010. doi:  
702 10.1145/1753326.1753478.
- 703 **21** Heman Gandhi. How Not to Learn Agda. URL: [https://hemangandhi.github.io/  
704 blog-posts/agda.html](https://hemangandhi.github.io/blog-posts/agda.html).
- 705 **22** Josip Medved. Unicode Code Point, 2025. URL: [https://marketplace.visualstudio.com/  
706 items?itemName=medo64.code-point](https://marketplace.visualstudio.com/items?itemName=medo64.code-point).
- 707 **23** Sára Juhošová, Andy Zaidman, and Jesper Cockx. Pinpointing the Learning Obstacles of an  
708 Interactive Theorem Prover. In *International Conference on Program Comprehension (ICPC)*,  
709 pages 159–170. IEEE, 2025. doi:10.1109/ICPC66645.2025.00024.
- 710 **24** Sára Juhošová, Andy Zaidman, and Jesper Cockx. The Way of Types: A Report on De-  
711 veloper Experience with Type-Driven Development. In *International Conference on Program*  
712 *Comprehension (ICPC)*. ACM, 2026. doi:10.1145/3794763.3794812.
- 713 **25** Gada F. Kadoda. *Formal Software Development Tools: An Investigation into Usability*. PhD  
714 Thesis, Loughborough University, 1997. URL: <https://hdl.handle.net/2134/31907>.
- 715 **26** Gada F. Kadoda, Roger G. Stone, and Dan Diaper. Desirable features of educational  
716 theorem provers - a cognitive dimensions viewpoint. In *Annual Workshop of the Psychology of*  
717 *Programming Interest Group*, 1999.
- 718 **27** Steve Klabnik and Carol Nichols. *The Rust Programming Language*. No Starch Press, San  
719 Francisco, CA, USA, second edition, 2022.
- 720 **28** Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. Mind Your Language: On  
721 Novices’ Interactions with Error Messages. In *Symposium on New Ideas, New Paradigms,*  
722 *and Reflections on Programming and Software*, Onward! 2011, pages 3–18. ACM, 2011.  
723 doi:10.1145/2048237.2048241.
- 724 **29** Michael Nahas. Issue #5629: Unicode Identifier Visually Similar to ASCII One (Source of  
725 Confusion/Obfuscation), 2021. URL: <https://github.com/agda/agda/issues/5629>.
- 726 **30** Nadim Nachar. The Mann-Whitney U: A Test for Assessing Whether Two Independent  
727 Samples Come from the Same Distribution. *Tutorials in Quantitative Methods for Psychology*,  
728 4(1), 2008. doi:10.20982/tqmp.04.1.p013.
- 729 **31** Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: a proof assistant*  
730 *for higher-order logic*. Springer-Verlag, Berlin, Heidelberg, 2002.
- 731 **32** Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD  
732 Thesis, Chalmers University of Technology, Göteborg, Sweden, 2007.
- 733 **33** Raymond S. Pettit, John Homer, and Roger Gee. Do Enhanced Compiler Error Messages Help  
734 Students? Results Inconclusive. In *Technical Symposium on Computer Science Education*,  
735 SIGCSE ’17, pages 465–470. ACM, 2017. doi:10.1145/3017680.3017768.
- 736 **34** Phitchaya Mangpo Phothilimthana and Sumukh Sridhara. High-Coverage Hint Generation  
737 for Massive Courses: Do Automated Hints Help CS1 Students? In *Conference on Innovation*  
738 *and Technology in Computer Science Education*, ITiCSE ’17, pages 182–187. ACM, June 2017.  
739 doi:10.1145/3059009.3059058.
- 740 **35** Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, Cambridge,  
741 Massachusetts, 2002.
- 742 **36** The Agda Development Team. Emacs Mode, 2025. URL: [https://agda.readthedocs.io/  
743 en/v2.8.0/tools/emacs-mode.html](https://agda.readthedocs.io/en/v2.8.0/tools/emacs-mode.html).

- 744 37 The Agda Development Team. Mixfix Operators, 2025. URL: <https://agda.readthedocs.io/en/v2.8.0/language/mixfix-operators.html>.
- 745
- 746 38 The Coq Development Team. The Coq Proof Assistant, September 2024. URL: <https://zenodo.org/records/14542673>.
- 747
- 748 39 Emilie Thiselton and Christoph Treude. Enhancing Python Compiler Error Messages via Stack. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, 2019. ISSN: 1949-3789. doi:10.1109/ESEM.2019.8870155.
- 749
- 750
- 751 40 V. Javier Traver. On Compiler Error Messages: What They Say and What They Mean. *Advances in Human-Computer Interaction*, 2010(1):602570, 2010. doi:10.1155/2010/602570.
- 752
- 753 41 Christopher Watson, Frederick W. B. Li, and Jamie L. Godwin. BlueFix: Using Crowd-Sourced Feedback to Support Programming Students in Error Diagnosis and Repair. In Elvira Popescu, Qing Li, Ralf Klamma, Howard Leung, and Marcus Specht, editors, *Advances in Web-Based Learning*, pages 228–239. Springer, 2012. doi:10.1007/978-3-642-33642-3\_25.
- 754
- 755
- 756
- 757 42 zeithaste. Unicode Code Point of Current Character, 2024. URL: <https://marketplace.visualstudio.com/items?itemName=zeithaste.cursorCharCode>.
- 758