Just Type It in Isabelle! AI Agents Drafting, Mechanizing, and Generalizing from Human Hints

Kevin Kappelmann[1], Maximilian Schäffeler[2], Lukas Stevens[3], Mohammad Abdulaziz[2], Andrei Popescu[1], and Dmitriy Traytel[3]

[1] Department of Computer Science, University of Sheffield, United Kingdom
{k.kappelmann,a.popescu}@sheffield.ac.uk
[2] Department of Informatics, King's College London, United Kingdom
{maximilian.schaffeler,mohammad.abdulaziz}@kcl.ac.uk
[3] Department of Computer Science, University of Copenhagen, Denmark
{lukas.stevens,traytel}@di.ku.dk

Abstract. Type annotations are essential when printing terms in a way that preserves their meaning under reparsing and type inference. We study the problem of complete and minimal type annotations for rank-one polymorphic λ-calculus terms, as used in Isabelle. Building on prior work by Smolka, Blanchette et al., we give a metatheoretical account of the problem, with a full formal specification and proofs, and formalize it in Isabelle/HOL. Our development is a series of experiments featuring human-driven and AI-driven formalization workflows: a human and an LLM-powered AI agent independently produce pen-and-paper proofs, and the AI agent autoformalizes both in Isabelle, with further human-hinted AI interventions refining and generalizing the development.

Keywords: Printing-parsing roundtrip · Minimal type annotation · AI · Large language model · Autoformalization · Isabelle · Lambda calculus

1 Introduction

Some 30 years ago, Larry Paulson taught the working ML[4] programmers how to pretty print [34]. His textbook algorithm is mainly concerned with the printing layout for improved legibility, and has inspired the pretty printers of types, terms, and theorems used in the Isabelle proof assistant to this day.

When printing Isabelle's terms, or more generally terms of the typed lambda calculus (§2), in addition to legibility, it is crucial to preserve the type information. Specifically, we are interested in the following round-trip property: starting from a fully typed term, we wish to print it in a way so that reparsing it and using type inference afterwards yields the original term. For example, printing the term (c_α, d_α)[5] with polymorphic constants c and d as (c, d) violates this property: invoking type inference yields the more general typing (c_α, d_β).

[4] Hereby, we refer to the programming language Standard ML, and its spiritual successors, implementations, and variants such as OCaml, Poly/ML and Isabelle/ML.
[5] We prefer the compact t_τ over Isabelle's $t :: \tau$ type annotation syntax.

Achieving the round-trip hence requires type annotations. As legibility remains a desideratum, annotations should be as sparse as possible while still constraining type inference to recover the original term. Smolka, Blanchette et al. [8, 39, 40] identified these requirements and formulated the type annotation problem in the context of Isar proof reconstruction for Sledgehammer [35]. They call the round-trip property *completeness*, and they are interested in a solution that is also *minimal*. We use completeness and minimality as the *correct printing problem*'s specification (§3). Smolka, Blanchette et al. [8, 39, 40] also devised a solution to the correct printing problem, indicating that full minimality (w.r.t. a function assigning costs to annotation positions) is NP-hard, they settle for a tractable greedy algorithm, producing a complete, *locally minimal* solution. Their implementation is part of the Isabelle distribution and has been reused in different contexts in which terms are printed: `sketch` and `explore` [20], `super_sketch` [41], AutoCorres2 [9], types-to-sets [32], and Apply2Isar [7].

Despite widespread usage, the algorithm's implementation contained a completeness-compromising issue in one optimization. In the Isabelle2025-2 release, (c_α, d_α) is printed as (c, d)—the type annotation is incorrectly dropped. In addition, it is difficult to convince Isabelle to actually print type annotations correctly when they are attached to constants and bound variables: $[]_{\mathsf{nat\ list}}$ gets printed as $[]$ and $\lambda x.\, x_{\mathsf{nat}}$ as the garbled `_type_constraint_`. We resolved these issues by correcting the optimization, forbidding annotations on bound variables in favor of annotating binders, and adding a custom print translation that preserves all annotations inserted by the algorithm. We also set up a testing environment for the algorithm using Mirabelle [17]. These implementation problems, however, also exposed a deeper issue: without a precise formal account of what the algorithm is supposed to guarantee, it is difficult to explain exactly why an optimization is unsound, to justify the repairs, or to rule out similar regressions.

Alas, this paper is not about our technical refinements of the implementation. Our main goal as working programming-language metatheorists is to understand the type annotation problem precisely, with formal statements and complete proofs. (Smolka, Blanchette, et al. [8, 39, 40] give only informal claims and no proofs.) At the same time, we also identify as working metatheory formalizers and thus want a proof assistant to validate our proofs. Being modern working metatheorists and formalizers, we would like to conduct both activities assisted by modern technology, particularly large language models (LLMs), thus corroborating numerous recent accounts of their usefulness in our field.

To this end, we conduct the following case studies contrasting human-driven and AI-driven activities, starting from Smolka, Blanchette, et al.'s informal description, the Isabelle/ML implementation, and an example algorithm execution:

1. A human expert formalizes the type annotation problem's metatheory on paper in §§ 2 to 4, an occasion to describe the problem in detail.
2. Independently, an LLM-powered AI agent formalizes the problem's metatheory on paper. We discuss the experimental setup and how the result differs from the human pen-and-paper formalization (§5). Drafts were iteratively

refined through human reviews. Some feedback also went in the other direction: AI's work has informed the human formalizer to prove local minimality.
3. A second AI agent (disconnected from the first) formalizes both pen-and-paper proofs in Isabelle/HOL. Both resulting autoformalizations use mostly the same setup; we discuss differences in the experience (§6).
4. A human expert intervenes with the AI's pen-and-paper formalization, observing that a core part of the algorithm could be generalized by reducing it to a standard problem, for which a rich theory as well as off-the-shelf algorithms exist. The AI agent analyses the human hint, shows the reduction to be correct, and updates the Isabelle formalization to do the same (§7).

To our own surprise, *all* case studies were successful after a small number of human reviews. We obtained three Isabelle formalizations of the desired results without writing a single line of Isabelle code ourselves. As our LLM, we used Claude Opus 4.6, a state-of-the-art general purpose model with strong reasoning capabilities. Claude proved a capable, although not logically infallible, companion—one that can substantially amplify the work of human experts.

In contrast to prior work in autoformalization (§8), which typically starts from pen-and-paper proofs and/or targets classic mathematics, we begin with an algorithm that lacks formal correctness specifications. Another contribution is autoformalization of programming language metatheory, which has been largely unexplored. Finally, we are among the first to report on a comprehensive autoformalization in Isabelle and provide a simple, reusable setup to the community. Our experimental data [26] and an extended version [25] are available online.

2 Syntactic Preliminaries

Next we introduce the necessary background: the syntax of types and terms, and the typing relation for rank-one polymorphic λ-calculus. Our presentation is tailored to the forthcoming discussion of the Smolka-Blanchette type annotation removal algorithm, in that our notion of terms is more general than usual, allowing annotations at any position in the term's abstract syntax tree. The material in §§ 2 to 4 was *not* used for the AI pen-and-paper proofs (§5).

2.1 Types

For the entire paper, we fix an infinite set TVar of *type variables* (*tyvars* for short), ranged over by α, β, and an infinite set Var of *term variables* (*vars* for short), ranged over by x, y, z. We also fix a *type structure*, i.e., a pair $(\mathsf{K}, \mathsf{arOf})$ where K, ranged over by κ, is the set of *type constructors*, and $\mathsf{arOf} : \mathsf{K} \to \mathbb{N}$, is a function associating *arities* to the type constructors.

The *types*, ranged over by σ, τ, forming the set Type, are given by the grammar $\sigma \ ::= \ \alpha \mid \sigma \Rightarrow \tau \mid (\sigma_1, \ldots, \sigma_{\mathsf{arOf}(\kappa)})\,\kappa$. Thus, a type is a tyvar, or the function type constructor $\Rightarrow$ applied to two types, or another type constructor κ postfix-applied to a number of types matching its arity.

Finally, we fix a *signature* for the type structure $(\mathsf{K}, \mathsf{arOf})$, i.e., a pair $\Sigma = (\mathsf{C}, \mathsf{ctpOf})$, where: C, ranged over by c, is a set of symbols called *constants*, and $\mathsf{ctpOf} : \mathsf{C} \to \mathsf{Type}$ is a function associating a type to every constant.

A *(type) substitution* is a function $\rho : \mathsf{TVar} \to \mathsf{Type}$ with finite support, in that $\rho(\alpha) \neq \alpha$ for all but a finite set of tyvars. For a substitution ρ and a type σ, $\sigma[\rho]$ denotes the application of ρ to σ. For a type σ, $\mathsf{TV}(\sigma)$ denotes the set of its (occurring) tyvars. We say that σ is an *instance* of τ via a substitution ρ, written $\sigma \leq_\rho \tau$, when $\sigma = \tau[\rho]$; and that σ is an *instance* of τ, or that τ is *more general* than σ, written $\sigma \leq \tau$, when there exists ρ such that $\sigma \leq_\rho \tau$. For two substitutions ρ and ρ', their *composition* $\rho \bullet \rho'$ is defined by $(\rho \bullet \rho')(\alpha) = \rho'(\alpha)[\rho]$. We write σ/α for the substitution that takes α to σ and any other tyvar to itself, and $\rho[\alpha \leftarrow \sigma]$ for the substitution obtained by updating ρ at α with σ.

To model (partial) type-annotations, we will work with elements of the set $\mathsf{Type}_\bot = \mathsf{Type} \cup \{\bot\}$ (where $\bot \notin \mathsf{Type}$), which we call *maybe-types*, and let ξ, ζ range over them. The operators TV and $\_[\_]$ are extended from Type to $\mathsf{Type}_\bot$ by defining $\mathsf{TV}(\bot) = \emptyset$ and $\bot[\rho] = \bot$.

2.2 Terms

The *partially typed terms* (which we simply call *terms*), ranged over by s, t, forming the set Term, are given by the grammar $t ::= x_\xi \mid c_\xi \mid (t_1\, t_2)_\xi \mid (\lambda x_\xi.\, t)_\zeta$. We call x_ξ a *(maybe-)typed variable* and c_ξ a *constant instance*. Thus, a term is either (1) a typed variable, (2) a constant instance, (3) an application decorated with a maybe-type, or (4) a λ-abstraction of a typed variable, decorated with a maybe-type. Our terms are "decorated" with maybe-types on any occurring variable or constant, as well as at any subterm. We think of an actual type decoration (when the maybe-type is a type) as a *type annotation*, and of a $\bot$ decoration as the absence of a type annotation. Therefore, in examples we will sometimes omit some $\bot$ decorations, thus writing, e.g., $\lambda x_\alpha.\, y_\beta$ instead of $(\lambda x_\alpha.\, y_\beta)_\bot$.

Our term decoration scheme captures the process of type inference—of which we think of as "completing" a term such as $\lambda x_{\alpha \Rightarrow \beta \Rightarrow \gamma}.\ x\ c$ to a fully annotated term such as $(\lambda x_{\alpha \Rightarrow \beta \Rightarrow \gamma}.\, (x_{\alpha \Rightarrow \beta \Rightarrow \gamma}\ c_\alpha)_{\beta \Rightarrow \gamma})_{(\alpha \Rightarrow \beta \Rightarrow \gamma) \Rightarrow \beta \Rightarrow \gamma}$, where the highlighted types have been inferred. Additionally, this scheme allows for a smooth presentation of the algorithm that we will study, which, starting from a fully annotated term, repeatedly removes redundant annotations.

We let $\mathsf{TV}(t)$ be the set of tyvars occurring in t, i.e., occurring in all the type annotations from t; and $t[\rho]$ be the application of the substitution ρ to t, which is defined by applying ρ to (all the types occurring in) t. We let $\mathsf{FTV}(t)$ be the set of free typed variables occurring in t; e.g., $\mathsf{FTV}(\lambda x_\sigma.\, y_{\sigma \Rightarrow \tau}\ x_\sigma) = \{y_{\sigma \Rightarrow \tau}\}$.

We extend the "instance of" relation from types to terms: $t \leq_\rho s$ is defined as $t = s[\rho]$, and $t \leq s$ is defined as the existence of ρ such that $t \leq_\rho s$; in this case, we say t is an *instance* of s, or that s is *more general* than t.

A term will be called:

- *unambiguous*, if its binding variable occurrences are non-repetitive, in that there exists no variable x and (possibly equal) maybe-types ξ and ζ such that λx_ξ and λx_ζ occur at two different positions in a term;

– a *fully typed term* (*F-term*), when all its occurring maybe-types are types;
– a *Church-typed term* (*C-term*), when (1) all maybe-types annotating its application and abstraction subterms are $\bot$, and (2) all maybe-types annotating its constants and variables, as well as its binding variables, are types.

The above concepts have straightforward inductive definitions. (We focus on unambiguous terms in order to ensure correct behavior of type substitution.)

We let FTerm, and CTerm denote the subsets of Term consisting of the F-terms and C-terms, respectively. We let u, v range over F-terms.

A *position* is a list of numbers in $\{1, 2\}$, identifying the location of a subterm or a binding variable in a term via its unique path. $\mathsf{Poss}(t)$ denotes the set of positions of term t. If $p \in \mathsf{Poss}(t)$, we write $\mathsf{mtpOf}(t, p)$ for the maybe-type decoration at that position. (Details in App. A.) For example, if t is $(\lambda x_{\mathsf{nat}}.\ (f_\bot\ x_\bot)_{\mathsf{bool}})_\bot$, then $\mathsf{Poss}(t) = \{[], [1], [2], [2, 1], [2, 2]\}$; $\mathsf{mtpOf}(t, []) = \bot$, $\mathsf{mtpOf}(t, [1]) = \mathsf{nat}$, $\mathsf{mtpOf}(t, [2]) = \mathsf{bool}$, and $\mathsf{mtpOf}(t, [2, 1]) = \mathsf{mtpOf}(t, [2, 2]) = \bot$. We write $\mathsf{mtpOf}(t)$ instead $\mathsf{mtpOf}(t, [])$. For F-terms u we write $\mathsf{tpOf}(u, p)$ and $\mathsf{tpOf}(u)$ instead of $\mathsf{mtpOf}(u, p)$ and $\mathsf{mtpOf}(u)$ (since we know that this is a type).

On $\mathsf{Type}_\bot$, we define the relation $\sqsubseteq$ by $\xi \sqsubseteq \zeta$ iff $\xi \in \{\bot, \zeta\}$. This order is extended to Term, by defining the *annotation subsumption* relation $t \sqsubseteq s$ to mean that s is obtained from t by adding zero or more type annotations:

$$\frac{\xi \sqsubseteq \zeta}{x_\xi \sqsubseteq x_\zeta} \qquad \frac{\xi \sqsubseteq \zeta}{c_\xi \sqsubseteq c_\zeta} \qquad \frac{s \sqsubseteq s' \quad t \sqsubseteq t' \quad \xi \sqsubseteq \xi'}{(s\ t)_\xi \sqsubseteq (s'\ t')_{\xi'}} \qquad \frac{\xi \sqsubseteq \xi' \quad t \sqsubseteq t' \quad \zeta \sqsubseteq \zeta'}{(\lambda x_\zeta.\ t)_\xi \sqsubseteq (\lambda x_{\zeta'}.\ t')_{\xi'}}$$

For a term t and position $p \in \mathsf{Poss}(t)$, let $t[p := \bot]$ denote the term obtained from t by erasing the type annotation at p (i.e., turning its decoration into $\bot$). For a term t, the (completely unannotated) term $\mathsf{erase}(t)$ denotes the term obtained from t by erasing *all* its type annotations. Note that $\mathsf{erase}(t) \sqsubseteq t[p := \bot] \sqsubseteq t$.

The *well-typedness* predicate $\vdash$ is defined on FTerm by the following rules:

$$\frac{}{\vdash x_\sigma} \qquad \frac{\sigma \leq \mathsf{ctpOf}(c)}{\vdash c_\sigma}$$

$$\frac{\vdash u \quad \vdash v \quad \mathsf{tpOf}(u) = \mathsf{tpOf}(v) \Rightarrow \sigma}{\vdash (u\ v)_\sigma} \qquad \frac{\vdash u \quad \forall x_\tau \in \mathsf{FTV}(u).\ \tau = \sigma}{\vdash (\lambda x_\sigma.\ u)_{\sigma \Rightarrow \mathsf{tpOf}(u)}}$$

Note that the condition $\forall x_\tau \in \mathsf{FTV}(u).\ \tau = \sigma$ guarantees that terms with type-incompatible bindings are ill-typed, such as $\lambda x_\alpha.\ x_\beta$. Alternatively, this would also be achieved by the explicit consideration of typing contexts.

An F-term u is said to be a *well-typed completion* of a term t, provided $t \sqsubseteq u$ and $\vdash u$. We call a term t *typable* when it has a well-typed completion. Note that for F-terms, typability is equivalent to well-typedness.

In our formalism, the process of type inference for a (partially annotated) term t means producing a most general well-typed completion for t. We will take for granted the classic Damas-Hindley-Milner type inference result [12,13,21,33], which we only need in existential form:

Thm 1 Assume that t is a typable unambiguous term. Then there exists a most general well-typed completion of t (i.e., a $\leq$-maximal F-term among the F-terms u such that $t \sqsubseteq u$ and $\vdash u$).

We let $\mathsf{mgen}(t)$ denote a choice (unique up to tyvar renaming) of a most general well-typed completion of t.

3 The Problem of Correct Printing and the Smolka-Blanchette Algorithm

The terms used internally by Isabelle are captured by our notion of unambiguous C-terms. A crucial property of these terms is that they contain enough information in order for their inferred type to be actually unique:

Prop 2 Assume that t is a typable unambiguous C-term. Then there is only one well-typed completion of t (hence only one choice for $\mathsf{mgen}(t)$).

We can now define a correct printing of an Isabelle term to be another term (not necessarily a C-term) that has the same type-free content and allows for the same unique most general typing as the original:

Def 3 Given a typable unambiguous C-term t, a *correct printing* of t is a term s such that $\mathsf{mgen}(t)$ is the unique most general well-typed completion of s.

We are interested in a correct printing that is minimal w.r.t. the annotation subsumption relation, $\sqsubseteq$. This is also the problem that Smolka and Blanchette have considered, leading to their implementation which is currently part of Isabelle. They have not formulated the definition of correct printing rigorously, but focused on more practical, operational matters. Namely, they described a criterion for deeming an annotation redundant, provided an implementation that starts with a fully annotated term and repeatedly removes positions deemed redundant for as long as possible following a reverse greedy pattern, and informally claimed minimality. Next, we discuss a simplified version of their algorithm and a detailed informal proof that it indeed achieves a minimal correct printing.

The difference between what we will describe and what Smolka and Blanchette actually implemented is threefold: (1) we abstract away from the specific cost function that they employ when deciding the annotation at which of the eligible positions to remove, (2) we ignore a number of small optimizations, and (3) we ignore typing contexts altogether. Concerning (2), in future work we plan to extend our formal development (and our partnership with LLMs) to cover these optimizations as well, and possibly to even go beyond them together with a further sharpening of the implementation. Concerning (3), our motivation for avoiding contexts is that they are completely uninteresting for the correct printing problem—in that the tyvars (from $\mathsf{TV}(\sigma_i)$) and variables (x_i) from a typing context $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ would be treated exactly as if they were part of the signature Σ, namely as nullary type constructors and constants; so the problem can be safely reduced to printing closed terms in the empty context.

We express the Smolka-Blanchette algorithm (subject to the simplifications discussed above) by the function smobla which we define below, after defining the auxiliary functions coverageTest and decrease.

The ternary predicate $\mathsf{coverageTest}$ acting on F-terms $v \in \mathsf{FTerm}$, terms $s \in \mathsf{Term}$ and positions $p \in \{1,2\}^*$, is defined by

$$\begin{aligned}&\mathsf{coverageTest}(v, s, p) \;=\\ &\quad p \in \mathsf{Poss}(s) \cap \mathsf{Poss}(v) \wedge \mathsf{mtpOf}(s, p) \neq \bot \wedge\\ &\quad (\forall \alpha \in \mathsf{TV}(\mathsf{tpOf}(v, p)).\\ &\qquad\quad \exists q \in \mathsf{Poss}(s) \setminus \{p\}.\; \alpha \in \mathsf{TV}(\mathsf{tpOf}(v, q)) \wedge \mathsf{mtpOf}(s, q) \neq \bot)\end{aligned}$$

As will be seen from the upcoming definitions, $\mathsf{coverageTest}$ will be called with v being $\mathsf{mgen}(\mathsf{erase}(t))$, the most general well-typed completion of the erasure of the original term t, and s being the current term obtained by repeated removal of position annotations from $\mathsf{mgen}(t)$; so we will always have $s \sqsubseteq v$, which (as we discuss in §4.2) will ensure $\mathsf{Poss}(s) = \mathsf{Poss}(v)$ and $\mathsf{mtpOf}(s, q) \sqsubseteq \mathsf{tpOf}(v, q)$. The predicate checks whether a position p on the one hand has an annotation within s, and on the other hand has all the tyvars occurring in v at that position covered by another annotation in s, in the sense that there is another position q that has an annotation within s and α occurs in v at q.

We say that a function $pickPos : \mathsf{FTerm} \times \mathsf{Term} \to \{1,2\}^*$ is $\mathsf{coverageTest}$-compatible when, for all $v \in \mathsf{FTerm}$ and $s \in \mathsf{Term}$, if $\exists p.\ \mathsf{coverageTest}(v, s, p)$ then $\mathsf{coverageTest}(v, s, pickPos(v, s))$. Thus, compatibility means being a correct choice function for the position argument of $\mathsf{coverageTest}$. We let Compat denote the set of $\mathsf{coverageTest}$-compatible functions.

Now, the function $\mathsf{decrease} : \mathsf{Compat} \times \mathsf{FTerm} \times \mathsf{Term} \to \mathsf{Term}$ is defined by

$$\mathsf{decrease}(pickPos, v, s) \;=\; \begin{cases} \mathsf{decrease}(pickPos, v, s[(pickPos(v, s)) := \bot]), \\ \qquad \text{if } \exists p.\ \mathsf{coverageTest}(v, s, p) \\ s, \;\text{otherwise} \end{cases}$$

Thus, $\mathsf{decrease}$ keeps removing annotations from s at positions chosen using $pickPos$ provided they pass the coverage test. Finally, the function $\mathsf{smobla} : \mathsf{Compat} \times \mathsf{Term} \to \mathsf{Term}$ is defined by calling $\mathsf{decrease}$ on $v = \mathsf{mgen}(\mathsf{erase}(t))$ (the fixed provider of tyvars that must stay covered) and $s = \mathsf{mgen}(t)$ (the changing term from which annotations keep being removed, initially set to the fully annotated term $\mathsf{mgen}(t)$).

$$\mathsf{smobla}(pickPos, t) = \mathsf{decrease}(pickPos,\ \mathsf{mgen}(\mathsf{erase}(t)),\ \mathsf{mgen}(t))$$

In the above algorithm, we can distinguish two components. First, there is the general-purpose reverse greedy component: The algorithm starts with a fully annotated term $\mathsf{mgen}(t)$ and, via its $\mathsf{decrease}$ function, keeps removing annotations for positions p satisfying a given test, $\mathsf{coverageTest}$, for as long as such annotation carrying positions exist (and the choice of the exact position to remove is regulated by a $\mathsf{coverageTest}$-compatible parameter function $pickPos$).

Then, there is the ad hoc component, given by the definition of $\mathsf{coverageTest}$, which is the heart of the algorithm. Intuitively, we are interested in removing annotations at positions for as long as we do not lose information about the (unique) typing of the term. The algorithm does not pursue this (clear yet non-effective) intuition directly, but in a roundabout manner: It first erases all

type annotations from t and builds a most general well-typed completion for the erased term, $v = \mathsf{mgen}(\mathsf{erase}(t))$. Then it proceeds based on the following crucial observation: The most general well-typed completion of the original term, $s = \mathsf{mgen}(t)$, is less than or equally general as v, which means that the tyvars of v correspond position-wise to specific subterms of s. Protection against loss of typing information is offered through the test coverageTest, which makes sure that any $\alpha \in \mathsf{TV}(v)$ is still "covered" by an annotation in s at a corresponding position, in that there is a position p such that (1) α appears in the type annotation at that position in v, and (2) p has a type annotation in s as well. Then the fact that the algorithm achieves (i) correct printing that is also (ii) minimal (which is the subject of our proof development) essentially amounts to the fact that this test (i) guarantees the desired preservation of typing information and (ii) nothing beyond that.

For brevity, our presentation of the algorithm did not formally separate the generic from the ad hoc component. However, for the sake of conceptual cleanness and reusability, a mechanization should ideally do that. In §7 we report on how an AI agent has performed this separation based on a minimal prompt.

4 Human-Authored Paper Proof Development

In this section we give a rigorous pen-and-paper description of our results about the type annotation problem, which we also call *minimal correct printing problem*, starting with the statements (§4.1) and then delving into the proofs (§4.2). We aim to provide enough detail for the reader to form an informed view of the statement and proof complexity involved in what we consider a fairly direct "no-nonsense" solution, and thereby to appreciate the extent of the creative effort an AI agent would require when starting from first principles.

4.1 The statements of the main results

Our first goal will be to prove that the Smolka-Blanchette algorithm returns a correct printing of the original term (something that Smolka and Blanchette refer to as "completeness"):

Thm 4 (Completeness) If t is a typable unambiguous C-term and $\mathit{pickPos} \in \mathsf{Compat}$, then $\mathsf{smobla}(\mathit{pickPos}, t)$ is a correct printing for t.

Then we will prove that what is being returned is a minimal such correct printing:

Thm 5 (Minimality) If t is a typable unambiguous C-term and $\mathit{pickPos} \in \mathsf{Compat}$, then $\mathsf{smobla}(\mathit{pickPos}, t)$ is $\sqsubseteq$-minimal among the correct printings for t.

One may object to the notion of correct printing, hence to the statement of completeness, as being too weak. Indeed, s being a correct printing of t means that $\mathsf{mgen}(t)$ is the unique *most general* well-typed completion of s—which makes sense because this is what the type inference algorithm produces. However, one

may argue, we want $\mathsf{mgen}(t)$ to be the unique well-typed completion of s (without the "most general" qualifier), because we do not want any alternative way to type the term sneaking in; let us call this (superficially) stronger notion a *strong correct printing*. We show that the two notions coincide, because the existence of two well-typed completions implies the existence of two most general ones.

Prop 6 Assume $s \in \mathsf{Term}$ is unambiguous, and $u, u' \in \mathsf{FTerm}$ are two distinct well-typed completions of s. Then there exist two distinct *most general* well-typed completions of s.

This immediately gives:

Corollary 7 Assume $s \in \mathsf{Term}$ is unambiguous. Then s is a correct printing of t if and only if s is a strongly correct printing of t.

4.2 Proof development

Next we give an overview of our proof development leading to the above results, namely Thm. 4, Thm. 5 and Prop. 6, as well as Prop. 2 which underlies the definition of correct printing. App. C gives full details. The development is supported by several types of background lemmas:

(1) Constructor-aware inversion lemmas associated to inductively defined predicates such as $\vdash$ and $\sqsubseteq$. These lemmas reconstitute "one inductive rule back" of history in situations where an inductive predicate holds with one of the arguments having a specific syntactic constructor at the top. For example, the λ-abstraction aware left inversion lemma for $\sqsubseteq$ states the following: If $(\lambda x_\zeta.\ s)_\xi \sqsubseteq t$ then there exist s', ζ' and ξ' such that $s \sqsubseteq s'$, $\zeta \sqsubseteq \zeta'$, $\xi \sqsubseteq \xi'$ and $t = (\lambda x_{\zeta'}.\ s')_{\xi'}$. The (immediate) proofs of these lemmas combine the freeness (injectiveness) of the syntactic constructors with the standard inversion (elimination) rules stemming from inductive definitions. (Incidentally, Isabelle automates the process of inferring these lemmas via the `inductive_cases` command.)

(2) Lemmas on syntax basics, describing properties of the occurring-variable and substitution application operators, TV and $\_[\_]$. These lemmas are mostly the typical ones proved when developing a theory of syntax, such as substitution compositionality, $t[\rho \bullet \rho'] = t[\rho'][\rho]$, distributivity of TV along substitution, $\mathsf{TV}(t[\rho]) = \bigcup_{\alpha \in \mathsf{TV}(t)} \mathsf{TV}(\rho(\alpha))$, and substitution extensionality (in the Isabelle ecosystem also known as substitution congruence), $\forall \alpha \in \mathsf{TV}(t).\ \rho(\alpha) = \rho'(\alpha)$ implies $t[\rho] = t[\rho']$. We will in fact need the converse of extensionality, which is slightly outside the usual repertoire in theories of syntax: $t[\rho] = t[\rho']$ implies $\forall \alpha \in \mathsf{TV}(t).\ \rho(\alpha) = \rho'(\alpha)$. There are both type and term variants of these lemmas, and their proofs go by straightforward structural induction on types or terms (with the proofs of the latter using the former).

(3) Annotation lemmas, by which we collectively mean simple lemmas about positions, type annotations at positions, and the annotation-removal operator and their interaction with substitution and occurring tyvars—stating, for example, that positions are not affected by substitutions, $\mathsf{Poss}(t[\rho]) = \mathsf{Poss}(t)$, that

substitution operates position-wise, $\mathsf{mtpOf}(t[\rho], p) = \mathsf{mtpOf}(t, p)[\rho]$, and that deleting an annotation yields a decrease in the annotation ordering, $t[p := \bot] \sqsubseteq t$. The proofs again proceed by straightforward structural inductions on terms.

(4) The type preservation lemma, stating that substitution preserves typing, i.e., $\vdash v$ implies $\vdash v[\rho]$, proved by induction on the definition of $\vdash$.

With these preparations, the proof justifying Prop. 2 is a low-hanging fruit.

Proof (Prop. 2.). Let u and v be such that $t \sqsubseteq u$, $t \sqsubseteq v$, $\vdash u$, $\vdash v$. Then $u = v$ follows by induction on t, using the constructor-aware left inversion rules for $\sqsubseteq$. The fact that t is a C-term is used in the variable, constant, and abstraction cases, ensuring that respective annotation is a type (not just a maybe-type). □

Moving towards the main results, we state lemmas about annotation subsumption and its interaction with the other operators. In particular, we need that $\sqsubseteq$ is a preorder on terms, does not affect the position sets, and interacts as expected with annotations at positions (in that $s \sqsubseteq t$ implies $\mathsf{mtpOf}(s, p) \sqsubseteq \mathsf{mtpOf}(s, p)$)—we will refer to these as *the* $\sqsubseteq$*-lemmas*. Moreover, we need that less annotated terms yield more general completions, in that $t \sqsubseteq s$ implies $\mathsf{mgen}(s) \leq \mathsf{mgen}(t)$. This fact, henceforth called *the* $(\sqsubseteq, \leq)$*-lemma*, supports the main intuition behind the Smolka-Blanchette algorithm—which, as discussed in §3, decides on removing annotations taking advantage of the position synchronization offered by $\mathsf{mgen}(t) \leq \mathsf{mgen}(\mathsf{erase}(t))$, and this in turn follows from $\mathsf{erase}(t) \sqsubseteq t$.

As a final preparation, we need some consequences of the (quasi-)generic reverse greedy component of the algorithm, namely that the result has fewer annotations than the original and that the coverage test no longer holds for the result.

Now, the technical core of the correct printing theorem (Thm. 4) is expressed by the following "sandwich" property: If an F-term u on the one hand subsumes the annotations of a term s, and on the other hand is less general than a term v, i.e., is $(\sqsubseteq, \leq)$-sandwiched between s and v, such that all tyvars of v are covered by an annotation of s, then u is uniquely determined by s and v.

Lemma 8 Assume $s \in \mathsf{Term}$ is unambiguous and $u, v \in \mathsf{FTerm}$ such that $s \sqsubseteq u \leq v$ and $\forall \alpha \in \mathsf{TV}(v). \exists p \in \mathsf{Poss}(v). \alpha \in \mathsf{TV}(\mathsf{tpOf}(v, p)) \wedge \mathsf{mtpOf}(s, p) \neq \bot$. Then u is the unique $u' \in \mathsf{FTerm}$ such that $s \sqsubseteq u' \leq v$.

Proof. From $u \leq v$, we obtain a substitution ρ such that $u \leq_\rho v$. Let $u' \in \mathsf{FTerm}$ be such that $s \sqsubseteq u' \leq v$, i.e., (1) $s \sqsubseteq u' \leq_{\rho'} v$ for some ρ'. We need to prove $u' = u$. By syntax basics, it suffices to fix $\alpha \in \mathsf{TV}(v)$ and to show that $\rho(\alpha) = \rho'(\alpha)$.

From $\alpha \in \mathsf{TV}(v)$ and our assumption, we obtain $p \in \mathsf{Poss}(v)$ such that (2) $\alpha \in \mathsf{TV}(\mathsf{tpOf}(v, p))$ and $\mathsf{mtpOf}(s, p) \neq \bot$. With the $\sqsubseteq$-lemmas and $s \sqsubseteq u, u'$, this implies (3) $\mathsf{tpOf}(u, p) = \mathsf{mtpOf}(s, p) = \mathsf{tpOf}(u', p)$. Moreover, by the annotation lemmas, we have (4) $\mathsf{tpOf}(u, p) = \mathsf{tpOf}(v[\rho], p) = \mathsf{tpOf}(v, p)[\rho]$, and similarly (5) $\mathsf{tpOf}(u', p) = \mathsf{tpOf}(v[\rho'], p) = \mathsf{tpOf}(v, p)[\rho']$. From (3), (4) and (5), we obtain $\mathsf{tpOf}(v, p)[\rho] = \mathsf{tpOf}(v, p)[\rho']$. Applying syntax basics (specifically the converse of substitution extensionality) to this and (2) yields $\rho(\alpha) = \rho'(\alpha)$, as desired. □

Proof (Thm. 4). Let $v = \mathsf{mgen}(\mathsf{erase}(t))$. By smobla's definition, we have $s = \mathsf{decrease}(pickPos, v, \mathsf{mgen}(t))$. We first prove $\varphi(s') \longrightarrow \varphi(\mathsf{decrease}(pickPos, v, s'))$ for terms s' by induction on $|\mathsf{Poss}_{\neq\bot}(s')|$, where

$$\varphi(s') = \mathsf{erase}(t) \sqsubseteq s' \sqsubseteq \mathsf{mgen}(t) \wedge \forall\alpha \in \mathsf{TV}(v).\, \exists p \in \mathsf{Poss}(v).\, \alpha \in \mathsf{TV}(\mathsf{tpOf}(v,p)) \wedge \mathsf{mtpOf}(s',p) \neq \bot,$$

$\mathsf{Poss}_{\neq\bot}(s') = \{p \in \mathsf{Poss}(s') \mid \mathsf{mtpOf}(s',p) \neq \bot\}$.

In the inductive proof, we distinguish two cases: (i) if $\exists p.\ \mathsf{coverageTest}(v, s', p)$, then we apply the induction hypothesis together with $\sqsubseteq$-lemmas, annotation lemmas, and some straightforward set-theoretic computation; (ii) otherwise, we have $\mathsf{decrease}(pickPos, v, s') = s'$, and the fact holds trivially.

Now, since $\varphi(\mathsf{mgen}(t))$ holds, we obtain $\varphi(s)$. By the $(\sqsubseteq, \leq)$-lemma, we have $\mathsf{mgen}(t) \leq v$. From this and $\varphi(s)$, by Lemma 8 we obtain that $\mathsf{mgen}(t)$ is the unique F-term u such that $s \sqsubseteq u \leq v$.

Since $\vdash \mathsf{mgen}(t)$ and $\forall u \in \mathsf{FTerm}.\ s \sqsubseteq u \wedge \vdash u \longrightarrow u \leq v$ (by the $(\sqsubseteq, \leq)$-lemma and the $\mathsf{erase}(t) \sqsubseteq s$ fact), we obtain that $\mathsf{mgen}(t)$ is the unique well-typed completion of s, i.e., the unique F-term u such that $\vdash u$ and $s \sqsubseteq u$. □

What we actually proved in Thm. 4 is the a priori stronger property that s is a strongly correct printing of t—as we did not assume u to be most general.

It remains to prove Prop. 6 and the minimality theorem Thm. 5. These require further preparations. First, a trivial lemma about observable difference in annotations when descending on the annotation subsumption relation:

Lemma 9 If s, s' are terms such that $s' \sqsubset s$, then there exists $p \in \mathsf{Poss}(s) = \mathsf{Poss}(s')$ such that $\mathsf{mtpOf}(s', p) = \bot \neq \mathsf{mtpOf}(s, p)$.

Then a crucial lemma allowing us to change an F-term's instance (by changing the instantiating substitution) without affecting annotation subsumption:

Lemma 10 Assume s is a term, v is an F-term and ρ, ρ' are substitutions such that $s \sqsubseteq v[\rho].$ and $\forall\alpha, p.\ p \in \mathsf{Poss}(v) \wedge \alpha \in \mathsf{TV}(v, p) \wedge \mathsf{mtpOf}(s, p) \neq \bot \longrightarrow \rho(\alpha) = \rho'(\alpha)$. Then $s \sqsubseteq v[\rho']$.

Finally, we need a lemma that allows us to construct a second well-typed most general completion from a given one with "loose" tyvars:

Lemma 11 Assume $s \in \mathsf{Term}$ is unambiguous, and v is a most general well-typed completion of s such as $\mathsf{TV}(v) \setminus \mathsf{TV}(s) \neq \emptyset$. Then there exists a most general well-typed completion of s different from v.

Proof (Prop. 6). Expanding the definitions, we have $u \neq u'$, $s \sqsubseteq u$, $\vdash u$, $s \sqsubseteq u'$ and $\vdash u'$. Let $v = \mathsf{mgen}(s)$. From the definition of mgen, we have $\vdash v$ and obtain the substitutions ρ, ρ' such that $u = v[\rho]$ and $u' = v[\rho']$. Since $u \neq u'$, by syntax basics we obtain $\alpha \in \mathsf{TV}(v)$ such that (a) $\rho(\alpha) \neq \rho'(\alpha)$.

We claim that $\alpha \notin \mathsf{TV}(s)$. Assuming $\alpha \in \mathsf{TV}(s)$, the annotation lemmas yield $p \in \mathsf{Poss}(v) = \mathsf{Poss}(u) = \mathsf{Poss}(u') = \mathsf{Poss}(s)$ such that (b) $\mathsf{mtpOf}(s, p) \neq \bot$ and (c) $\alpha \in \mathsf{TV}(\mathsf{mtpOf}(s, p))$. From these and $s \sqsubseteq v$, we have (d) $\alpha \in \mathsf{TV}(\mathsf{tpOf}(v, p))$.

From (b) and $s \sqsubseteq u, u'$, we have $\mathsf{tpOf}(u, p) = \mathsf{tpOf}(u', p) = \mathsf{mtpOf}(s, p)$. From this, by annotation lemmas we have $\mathsf{tpOf}(v, p)[\rho] = \mathsf{tpOf}(v, p)[\rho']$, which together with (a) and (d) yields a contradiction by syntax basics.

Hence, $\alpha \in \mathsf{TV}(v) \setminus \mathsf{TV}(s)$. From this and v's definition, by Lemma 11, we obtain v' such that $v' \neq v$ and v' is a most general well-typed completion of s. As desired, v and v' are distinct most general well-typed completions of s. □

Proof (Thm 5). Let s' be an unambiguous term such that (a) $s' \sqsubset s$. Let $v = \mathsf{mgen}(\mathsf{erase}(t))$. By Lemma 9, there exists $p \in \mathsf{Poss}(s) = \mathsf{Poss}(s')$ such that (b) $\mathsf{mtpOf}(s', p) = \bot \neq \mathsf{mtpOf}(s, p)$. Since by generic reverse greedy lemmas we have $\neg\, \mathsf{coverageTest}(v, s, p)$, we obtain $\alpha \in \mathsf{TV}(\mathsf{tpOf}(v, p))$ such that $\forall q \in \mathsf{Poss}(s) \setminus \{p\}.\ \alpha \in \mathsf{TV}(\mathsf{tpOf}(v, q)) \longrightarrow \mathsf{mtpOf}(s, q) = \bot$. With (a) and (b), this gives us (c) $\forall q \in \mathsf{Poss}(s').\ \alpha \in \mathsf{TV}(\mathsf{tpOf}(v, q)) \longrightarrow \mathsf{mtpOf}(s', q) = \bot$.

Let ρ be such that $\mathsf{mgen}(t) = v[\rho]$. From this, since $\mathsf{mgen}(s) = \mathsf{mgen}(t)$ by Thm. 4, we obtain $\mathsf{mgen}(s) = v[\rho]$, and therefore (1) $s' \sqsubseteq v[\rho]$.

We choose a fresh type τ such that $\tau \neq \rho(\alpha)$ (for example, a fresh tyvar), and let $\rho' = \rho[\alpha \to \tau]$. Using (c) and the definition of ρ', we obtain
(2) $\forall q \in \mathsf{Poss}_{\neq\bot}(s'), \beta \in \mathsf{TV}(v, q).\ \rho(\beta) = \rho'(\beta)$.
For (1) and (2), applying Lemma 10, we obtain (3) $s' \sqsubseteq v[\rho']$.

By the type substitution lemma, from $\vdash v$ we obtain (4) $\vdash v[\rho']$. Since $\alpha \in \mathsf{TV}(v)$ and $\rho(\alpha) \neq \rho'(\alpha)$, by syntax basics (again the converse of the substitution extensionality lemma), we obtain (5) $v[\rho] \neq v[\rho']$.

Thus, (1–5) tell us that $v[\rho]$ and $v[\rho']$ are two distinct well-typed completions of s. Finally, we apply Prop. 6 to obtain two distinct *most general* well-typed completions of s', as desired. □

Note that we used Prop. 6 (which forms the basis of the equivalence between correct printing and strong correct printing) in order to finalize our proof of minimality. This is natural, because in the proof it was easier to prove minimality in the extent of the (superficially) narrower concept of strong correct printing—which could then be lifted to minimality w.r.t. correct printing via Prop. 6.

Summary of the human development. Unsurprisingly for a metatheory of syntax, a sizable number of simple lemmas were needed, having straightforward inductive proofs. In addition, some creativity was required to pin down the intuition of why the coverage test correctly models the typing information preservation goal (via results such as the sandwich Lemma 8 and the instance changing Lemma 10). Apart from the proof development per se, modeling the concepts required nontrivial design choices, such as working with terms decorated with maybe-types and introducing annotation subsumption $\sqsubseteq$ inductively.

5 AI-Authored Paper Proof Development

In parallel to the human-authored development described in the previous section, we independently employed an AI agent to produce a pen-and-paper (LaTeX) specification and proofs for the Smolka-Blanchette algorithm. We first describe

our technical setup (§5.1) and the paper generation process (§5.2), and then provide an analysis and a comparison with the human proof development (§5.3).

5.1 Technical setup

Our setup is based on *OpenCode* [3], an open source AI coding environment. It provides AI agents with several tools, such as the ability to read and edit files, search the web, and usage of the command line and the Model Context Protocol (MCP). As our agent, we used Claude Opus 4.6-v1 [4], a state-of-the-art model optimized for coding and reasoning tasks.

In each experiment, we provided the agent some background material (e.g., links to related work) and an experiment-specific `AGENTS.md` file as input. The `AGENTS.md` files contain the detailed instructions for the agent. The files were created by the agent itself from short hand-crafted prompts containing the problem statement and some guidelines (take notes, make backups, etc.). The experiments mimicked an academic peer-review process: The agent was instructed to create a complete artifact (a LaTeX document), which was then reviewed by human experts. The feedback was passed to the agent, and the process was repeated.

5.2 Generating the paper proofs

Generating proofs for the Smolka-Blanchette algorithm poses challenges. First, their original presentation of the concepts [8, 39, 40] is extremely informal. Not only does it lack proofs, but the desirable properties are not rigorously defined. Concerning the notion of correct printing, they introduce it by saying: "types are inferred correctly when the generated HOL formulas are parsed again by Isabelle" [8]. As for minimality, they note that the *greedy algorithm*'s "goal is to compute a locally minimal set of sites that completely covers all type variables [in the substitution's domain]" [8]. Thus, the notion of completeness (complete coverage) with respect to which they formulate minimality appears not to be the "declarative" notion of not losing typing information (which in §3 we captured by the definition of correct printing) but simply the "operational" notion employed by the algorithm—namely, that of passing the coverage test (coverageTest in our §3 notation). Completeness and minimality w.r.t. this operational notion follow directly from the nature of the reverse greedy algorithm. However, the algorithm is undoubtedly aimed at achieving the declarative versions (of the kind we formulated in §3), but that is not made explicit in their text. Another smaller challenge comes from their implementation diverging from their paper presentation by including several additional optimizations.

Despite these challenges, the agent produced a specification, with some limitations on generality, and mostly correct proof sketches in its first attempt, as further elaborated below. Following the initial version, we ran three revision rounds in which the agent resolved the limitations while also adding proof details, resulting in a correct proof document amenable to autoformalization (cf. §6.2). We next describe the agent's input followed by a summary of the generation process. All artifacts are available in the supplementary material.

Input. We created a `notes.md` file, containing (1) links to previous works [8,39, 40], (2) the corrected Isabelle/ML implementation, and (3) the execution trace of the algorithm on an example. In the `AGENTS.md` file, we instructed the agent to create results publishable at a peer-reviewed venue and to take notes in files while iteratively working on its tasks. The notes serve as reference for the agent, e.g., to recover its progress upon memory loss due to context compaction.

In rounds 2–4, we moreover passed as input a review by a human expert and extra instructions to use an internal two-stage review process before finishing the paper: (1) *Self-review*: the agent creates a review for its paper and revises it. (2) *Simulated peer-review*: the agent performs simulated peer reviews for a scientific submission and finalizes the paper based on the feedback. We found that the agent identified several presentational and some logical issues using internal reviews, such as the wrong usage of an assumption, the need for extra lemmas, and missing definitions. However, some significant shortcomings were only identified by human review, as explicated below.

Round 1. In the initial round, we prompted the agent to create a precise, formal exposition for the unoptimized algorithm including proofs. The agent successfully produced a paper with a specification and proof sketches. Here are some notable highlights, impressing a non-empty subset of the authors: (1) While the statement of completeness is informal—"reparsing [the output] recovers exactly the types of t [the input]"— the agent showed the correct formal statement (cf. Thm. 4 and Def. 3) in its proof. (2) The agent introduced an almost correct statement of minimality. (3) The agent discovered correct proof approaches for completeness and minimality and introduced several required lemmas, such as converse substitution extensionality (cf. §4.2).

However, the human reviewer also identified several defects, most notably: (1) The completeness and minimality proofs were restricted to ground terms (for no good reason), and minimality also needed a small correction to exclude an existential quantifier instantiation that made the statement vacuously true. (2) There were some self-contradictory statements, e.g., "The output of the annotation algorithm is a constraint-free term augmented with type constraints". (3) Definitions for key concepts (e.g., annotated terms, well-typedness, position enumeration), and statements of key lemmas (e.g., preservation of well-typedness under substitution) were missing.

Some milder issues included the AI's affinity for the Isabelle/ML implementation, resulting in a very technical presentation. For example, it used Isabelle jargon and notations, such as dummyT, inference in "pattern" mode, and a carbon copy of Isabelle's term and type syntax (including a false claim: "for any type τ of sort s, the type τ list also has sort s"). Moreover, it delved into implementation-specific performance optimizations that are irrelevant for the algorithm.

Rounds 2–4. Each revision round was initiated by a short prompt: read the new input (notably, the human review), devise a plan, and revise. The human review summarized the shortcomings of the previous round. Admittedly, writing the reviews was not an overly pleasant task, but it seemed like a necessary one:

while the agent was able to convince itself and its role-played scientific peers that the paper was worthy of publication fairly quickly, the human controller thought otherwise. The agent was able to correct, adapt and generalize its statements and proofs and to significantly improve its exposition in each revision round, but it required substantial human work to spot and review logical flaws, defects in generality, and deficiencies in rigor. After four rounds, we were convinced that the document is ready for autoformalization (though not for "publication").

5.3 Discussion

The most remarkable aspect of this experiment's outcome was the agent's ability to make its way towards the correct concepts and results, and to produce proof sketches that were plausible (and later validated by autoformalization). Specifically, the agent discovered the notion of strongly correct printing—although it did not introduce it as a separate notion, the agent's statements of completeness and minimality are equivalent to variations of our Thms. 4 and 5 that have "strongly correct printing" instead of "correct printing". Our Corollary 7 shows that the two concepts are equivalent, though this is not something that the agent has attempted (nor was it prompted) to establish. Another remarkable aspect is the informative nature of the agent's output. The agent's human controller learned the problems's correct specification and the proof ideas by reading, and also by critically scrutinizing the generated document. Moreover, before being informed by the agent's work, the human expert producing our pen-and-paper development from §3 had not realized the necessity to even state minimality (essentially due to confusing the operational and the declarative versions, hence wrongly assuming minimality to be trivial).

A persistent weakness of the agent was its lack of precision, including missing or slightly wrong definitions and lemmas, wrong assumptions, and a few logical errors. We employed simulated peer reviews by the agent to mitigate these issues, but many slipped through the review process, only being identified by the human expert. It is reported that LLMs perform worse detecting their own mistakes than those by other models [16], which we will consider for future work. All in all, the prevalence of such flaws urges the need for verification, which we consider in §6.

The agent's document, while not publication-ready, served as an effective self-study material that human experts can scrutinize and improve. The human document undoubtedly trumps the agent's in terms of clarity and conceptual cleanness. One qualitative difference stems from the agent staying close to the existing work and Isabelle/ML implementation, even when instructed to abstract from particularities; this complicates the reuse and invention of concepts (such as maybe-types and completion relations $\sqsubseteq$ from §4) that capture the problem's essence more cleanly. On the other hand, the AI development clearly wins cost-wise and time-wise: The agent created an autoformalizable document with a cost of $70, two hours of computation time and one day of human review effort, while the human-authored document took approximately five days of human work.

6 Autoformalization in Isabelle/HOL

Building on the results of the human and AI paper proofs, we conduct *autoformalization* experiments for both approaches in Isabelle/HOL. The generated Isabelle theories are available on Zenodo [26], and Isabelle snippets with the final theorem statements can be found in App. E. Our experiments are motivated by the following questions: (1) Can the correctness of algorithms in the domain of programming language theory be formalized automatically, with *reasonable costs and resources*? (2) What are the *limits* of state-of-the-art LLMs as Isabelle proof engineers? (3) How does the autoformalization effort differ between mechanizing human expert proofs versus AI-generated proofs? Relatedly, can autoformalization serve as an effective tool for *validating AI-generated results*?

6.1 Technical setup

Our setup extends our AI paper proof setup from §5.1. We connect OpenCode to Isabelle via *Isabelle/Q* [5], an MCP server for Isabelle/jEdit. Isabelle/Q lets the agent use Isabelle interactively, with tool access, such as modifying theory files, fact search, Sledgehammer, and retrieval of proof states. Inspired by prior autoformalization efforts [10], we provide an Isabelle guidance file to the agent. This guide contains both style guidelines, e.g., to create idiomatic Isar proofs, and proof engineering patterns to facilitate efficient interaction with Isabelle/Q.

6.2 AI-authored proof autoformalization

First, we tasked Claude Opus to mechanize the AI-authored proof. The agent receives all input and generated artifacts from the AI paper experiment (§5). Our `AGENTS.md` file instructs the agent to produce a formalization plan and implement it in Isabelle/HOL. We provide no problem-specific strategies for formalization.

From the initial prompt, we obtain a `sorry`-free proof in Isabelle/HOL that fills in details missing in the AI paper proof, while also improving its structure: (1) It uses an Isabelle *locale* [6] to introduce well-formedness assumptions on the inputs and to axiomatize the existence of most general types. (2) Another locale is used to abstract the proof of the main theorems, fixing a set of locally minimal annotations covering all inference variables. (3) To prove that each kept position is required for coverage, the agent identified a necessary, complex statement generalization, depicted in Fig. 1.

However, the first human review identified several instances where the theorem statements in Isabelle/HOL and the AI paper are not aligned: (1) Completeness and minimality are proved only within the abstraction locale, but are not instantiated to obtain statements about the algorithm's actual output. (2) The formalization claims to abstract from the order in which annotations are processed, but actually fixes a concrete post-order enumeration. This discrepancy still remains in the final version, despite human reviews repeatedly pointing to the problem. (3) The AI paper uses the informal notion of *consistency of a term with a set of annotations* in the algorithm's specifications. The formalization's

```
lemma rg_fold_witness_aux:
  assumes dist: "distinct (map fst cands)"
    and cnt_eq: "∀α. cnt α = length (filter (λ(_, K). α ∈ K) cands) + extra α"
    and mem: "(p, K) ∈ set cands"
    and kept: "p ∈ fst (rg_fold cands cnt)"
  shows "∃α ∈ K. extra α = 0 ∧
    (∀p' K'. (p', K') ∈ set cands ⟶ p' ∈ fst (rg_fold cands cnt) ⟶ p' ≠ p ⟶ α ∉ K')"
```

Fig. 1. AI-generalized statement for induction proof: The non-generalized statement expresses that each position kept by the reverse-greedy algorithm (`rg_fold`) must cover at least one type variable. Here, `cands` is a list of candidate annotations with their tyvars, `cnt` counts the occurrences of each tyvar in `cands`. The agent noticed that an induction proof requires generalizing the statement to also count for each tyvar in how many kept positions it occurs (via `extra`).

definition seems to not match the paper's intended meaning, though the authors also argued whether the paper version is ambiguous and thus the true culprit.

Based on the review, the agent correctly instantiates the abstraction locale. A second round of human reviews identified that the completeness statement refers to the set of annotations determined by the algorithm and not the output of the algorithm (as the paper version does). In a final run, Claude aligns the completeness statement with the paper. The agent also modifies the definition of consistency, which remains suboptimal but captures the right meaning in the statements of completeness and minimality.

6.3 Human-authored proof autoformalization

As a second experiment, we instructed the agent to mechanize the human proof, which was explicitly created with autoformalization in mind. The agent received as input a draft of §4 and the same prompt as in the first autoformalization experiment. In round one, Claude produces `sorry`-free proofs which faithfully capture the underlying material, except for a small but severe issue discovered in the review process: As part of a locale definition, the agent introduces assumptions on coverageTest before defining the constant. This allows the instantiation of these assumptions with arbitrary terms, that could have been exploited to derive a contradiction, but was not. Missing concrete instantiations of the minimality and completeness statements constitute further minor issues.

In round two, the agent makes minor adjustments that address all issues raised in the review. After the second run, we happened to discover an improvement in the paper definition of well-typedness from §2.2, namely the condition on the free typed variables from the typing of abstraction. When informed about the change, the AI agent correctly updates the definition of well-typedness in the formalization and adapts the proof of the type preservation lemma from §4.2.

6.4 Discussion

Both autoformalizations successfully produced results that faithfully capture a significant share of the underlying document, with minor problems identified

in human reviews. Even though the AI-authored proof was not optimized for autoformalization and its mechanization required more human feedback identifying alignment errors, the results are of comparable size and incur similar cost in terms of compute resources: The human-authored proof mechanization spans 1938 lines at an LLM usage cost of $139, vs. 2071 lines and $140 for the AI-authored proof (using Claude Opus 4.6 at a cost of $5 per 1M input and $25 per 1M output tokens). Resource usage in terms of human work is more difficult to quantify, but the AI paper proof mechanization demanded more human feedback in the review process, as well as more rounds of review. As the proofs themselves are checked for correctness by the proof assistant, the reviews were focused on aligning theorem statements and definitions with the underlying paper, where the more formal style of the human expert development proved advantageous.

Claude as a proof engineer. The proof documents are mostly idiomatic Isabelle developments: They use locales, inductive predicates, and recursive data types, as well as nested Isar proofs [46] of significant length and complexity. The proofs are generally of high quality and maintainable, yet at times overly verbose, e.g., several induction proofs with explicit case distinctions can easily be compressed into one-line proofs. Moreover, the agent showed a preference towards object-level quantifiers over structured Isar statements, making theorems harder to reuse (see Fig. 1 for an example). The resulting mechanizations may not meet the highest quality standards of expert proof engineers, but can serve as a good baseline for further improvements.

The agent routinely ignored the prescribed gradual workflow, instead producing large Isar proofs at once, then fixing the resulting errors. Still, human intervention into the proof engineering process was only required to remedy a misconception on non-terminating proof methods, where the agent misinterpreted Isabelle/Q's responses. Learning from our experience, we provide an improved Isabelle interaction guide for AI agents in the supplementary material.

Future directions. Our experiments are a playground for autoformalization with minimal dependencies and a simple background theory. In future work, we plan to fully stress the AI agent's capabilities, exploring navigability in large proof libraries, like the AFP [2] and complex proof developments in computer science. A key challenge is reducing resource cost, both in terms of compute and human feedback. This might involve improving utilization of symbolic tools, such as Sledgehammer [35], the Isabelle linter [31], and the development of specialized proof engineering LLMs.

7 Autogeneralization based on Human Hints

Our final experiment explores the AI agent's ability to separate the generic from the ad hoc component of the algorithm (§3). To this end, we handed a hint to the AI agent to consider the following independence system (IS), which abstracts the type annotation problem: (S, F) where $F = \{F' \mid F' \subseteq S \wedge S \setminus F' \text{ covers } V\}$ and

$S \setminus F$ corresponds to the to-be-annotated positions and V to the input term's tyvars. We then prompted the agent to reprove the results from the AI-authored paper proof (§5) by taking advantage of the generic setting for the standard best-in-greedy algorithm for ISs [28, Chapter 13]. The AI agent succeeded in reproving the local minimality result—that was the part benefitting most from the general results on the best-in-greedy algorithm. The proof was 6.25 pages, computed at a price of $11 (while having full access to the original proof). Next, we prompted the agent to formalize this pen-and-paper proof in Isabelle/HOL, building on an existing Isabelle formalization of ISs [1]. The agent was able to autoformalize the arguments of its own pen-and-paper proofs but, instead of importing the existing formalization, which we asked it to use, proceeded by copying definitions and lemmas. The formalization (of both local minimality and completeness) spans 1800 lines, in addition to 200 lines reused from the pre-existing formalization of ISs. The cost of the entire experiment was $93.

This experiment was performed by an author with no prior familiarity with the correct printing problem who wanted to understand the problem abstractly by formulating it in terms of ISs. The AI agent demonstrated that it can drastically accelerate the process of understanding the problem by pinning down all the details of the connection to ISs in less than three hours. Thus, our experiences, as well as those of others, hint to a role of LLMs as "propped up" automated proof methods or counter-example finders, benefiting the economics of formal theorem proving and even the practice of mathematics. We also hypothesize that user-provided hints make (formal) proving using LLMs cheaper.

8 Related Work and Conclusion

Type annotations are typically considered as part of *type inference*: the process of translating a partially typed term from an external language to a fully typed term in an internal language [36]. In this work, we consider the reverse direction: the translation of a fully typed term to a partially typed one. Since type inference for expressive languages, such as System F, is undecidable [45], type inference algorithms for these languages are only partially complete: for any well-typed t of type τ, there is t' with $t \sqsubseteq t'$ such that running type inference on t' yields τ. Completeness in this context is also called *annotatability* [18]. Note that annotatability does not tell us *where* type annotations are needed. Xue and Oliveira [51] hence call this *weak annotatability*, differentiating it from *strong annotatability* which also describes where annotations are needed. Existing work on strong annotatability [11, 19, 22, 29, 50, 51] covers the completeness part of the correct printing problem but, to our knowledge, does not address minimality. Unlike in our case, the languages used in many of these works do not admit most general well-typed completions. As a result, annotation criteria are typically tied to the respective type inference algorithm with little control to specify favorable annotation positions, whereas the approach studied here is independent of the inference algorithm and controllable via the *pickPos* function.

A related concern is the construction of correct pretty printers [14, 37]. The focus there is to create infrastructure for correct-by-construction parser and printer combinators, whereas we are concerned with the correctness of a particular pretty printer that minimizes type annotations.

For pen-and-paper mathematics, LLMs have shown significant and accelerating progress (see Ju and Dong's survey [24]). Applications include problems ranging from finding conjectures in research mathematics [15], to solving mathematics olympiad problems [42], all the way to proving open problems in theoretical computer science [27]. Our work here demonstrates the AI's abilities in a new domain of pen-and-paper mathematics: we show that off-the-shelf LLMs can produce new pen-and-paper theorem statements and proofs in programming language metatheory while based only on informal hints as input, e.g. an algorithm lacking formal correctness specifications (as in §5) or a pointer to a pen-and-paper description of a general concept (as in §7). Our use case showed that this can be done within reasonable time and financial cost, advocating for LLMs to be used in an interactive loop with human researchers, i.e., as a day-to-day proof automation tool, substantially accelerating the human's progress.

The area of autoformalization is rapidly growing. Several fine-tuned models [30, 47, 48] and sophisticated frameworks [23, 44] have been developed. We demonstrated that even a simple setup—using an off-the-shelf AI coding environment, LLM, and Isabelle MCP server—can be sufficient for non-trivial autoformalization tasks. Our setup was inspired by Urban's remarkable experiment [43], who used an off-the-shelf LLM to autoformalize textbook results in topology without human intervention. Notable experiments in the area of programming languages are by Xi and Odersky [49] and Ilya Sergey [38]. Both present an iterative development where humans wrote the definitions while agents handled the proof development. This is unlike our work, which hands all steps to the LLM, only providing human review once a complete formalization is obtained.

Conclusion We provide a formal account of the problem of sufficient and minimal type annotations for correct printing, in particular the Smolka-Blanchette algorithm, with mechanized statements and correctness proofs. A main focus was on using LLMs as research assistants in the context of programming language research. Our case study reveals that state-of-the-art models and human insights can be complementary and therefore mutually beneficial.

Our work suggests that LLMs can already be used for day-to-day proof assistance, in the same way as existing Isabelle tools like Sledgehammer, Nitpick, and auto, but with a substantially broader scope and power. We also point to a promising future in which a deep integration of AI agents and proof assistants could catalyze both AI's and humans' capabilities for proving theorems.

Acknowledgments. This research is conducted in the Copilots for Isabelle project under the AI for Math Fund, an initiative by Renaissance Philanthropy with support from XTX Markets. Our interest in adding type annotations during printing arose while implementing a copilot to translate apply-style proofs to readable Isar proofs.

Disclosure of Interests. The authors have no competing interests to declare.

References

1. Abdulaziz, M., Ammer, T., Meenakshisundaram, S., Rimpapa, A.: A Formal Analysis of Algorithms for Matroids and Greedoids. In: The 16th International Conference on Interactive Theorem Proving (ITP) (2025). https://doi.org/10.48550/arXiv.2505.19816
2. Archive of Formal Proofs (AFP), https://www.isa-afp.org, proof library collection for Isabelle, accessed Apr 03, 2026
3. Anomaly: Opencode (2026), https://github.com/anomalyco/opencode, open-source AI coding environment. GitHub repository; version 1.2.24
4. Anthropic: Introducing Claude Opus 4.6 (Feb 2026), https://www.anthropic.com/news/claude-opus-4-6, accessed Apr 03, 2026
5. AWS Labs: Isabelle/Q (2026), https://github.com/awslabs/AutoCorrode/tree/72160e202330fc050732f40648ec16b7628bee15/iq, MCP server for Isabelle/jEdit. GitHub repository; commit 72160e2
6. Ballarin, C.: Locales and locale expressions in isabelle/isar. In: Berardi, S., Coppo, M., Damiani, F. (eds.) Types for Proofs and Programs. pp. 34–50. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
7. Binder, S., Lachnitt, H., Kosaian, K.: Apply2Isar: Automatically converting Isabelle/HOL apply-style proofs to structured Isar (2026), https://arxiv.org/abs/2603.07771
8. Blanchette, J.C., Böhme, S., Fleury, M., Smolka, S.J., Steckermeier, A.: Semi-intelligible Isar proofs from machine-generated proofs. Journal of Automated Reasoning **56**(2), 155–200 (2016). https://doi.org/10.1007/S10817-015-9335-3
9. Brecknell, M., Greenaway, D., Hölzl, J., Immler, F., Klein, G., Kolanski, R., Lim, J., Norrish, M., Schirmer, N., Sickert, S., Sewell, T., Tuch, H., Wimmer, S.: Autocorres2. Arch. Formal Proofs **2024** (2024), https://www.isa-afp.org/entries/AutoCorres2.html
10. Bryant, D., Huerta y Munive, J.J., Kaliszyk, C., Urban, J.: Munkres' general topology autoformalized in isabelle/hol (2026), https://arxiv.org/abs/2604.07455
11. Chen, L.T., Ko, H.S.: A Formal Treatment of Bidirectional Typing, pp. 115–142. Springer Nature Switzerland (2024). https://doi.org/10.1007/978-3-031-57262-3_5
12. Damas, L.: Type Assignment in Programming Languages. Ph.D. thesis, University of Edinburgh (1985), technical Report CST-33-85
13. Damas, L., Milner, R.: Principal type-schemes for functional programs. In: Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL). pp. 207–212. ACM (1982)
14. Danielsson, N.A.: Correct-by-construction pretty-printing. In: Proceedings of the 2013 ACM SIGPLAN workshop on Dependently-typed programming. pp. 1–12. ICFP'13, ACM (Sep 2013). https://doi.org/10.1145/2502409.2502410
15. Davies, A., Veličković, P., Buesing, L., Blackwell, S., Zheng, D., Tomašev, N., Tanburn, R., Battaglia, P., Blundell, C., Juhász, A., Lackenby, M., Williamson, G., Hassabis, D., Kohli, P.: Advancing mathematics by guiding human intuition with AI. Nature (2021). https://doi.org/10.1038/s41586-021-04086-x
16. Dekoninck, J., Petrov, I., Minchev, K., Marinov, M., Drencheva, M., Konova, L., Shumanov, M.M., Tsvetkov, K., Drenchev, N., Todorov, L.D., Nikolova, K., Georgiev, N., Kalinkova, V., Ismoldayev, M., Balunovic, M., Vechev, M.: The open proof corpus: A large-scale study of LLM-generated mathematical proofs. In: The Fourteenth International Conference on Learning Representations (2026), https://openreview.net/forum?id=a2XmC7rHIU

17. Desharnais, M., Vukmirovic, P., Blanchette, J., Wenzel, M.: Seventeen provers under the hammer. In: Andronick, J., de Moura, L. (eds.) ITP 2022. pp. 8:1–8:18. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPICS.ITP.2022.8
18. Dunfield, J., Krishnaswami, N.: Bidirectional typing. ACM Computing Surveys **54**(5), 1–38 (May 2021). https://doi.org/10.1145/3450952
19. Dunfield, J., Krishnaswami, N.R.: Complete and easy bidirectional typechecking for higher-rank polymorphism. In: Morrisett, G., Uustalu, T. (eds.) ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013. pp. 429–442. ACM (2013). https://doi.org/10.1145/2500365.2500582, https://doi.org/10.1145/2500365.2500582
20. Haftmann, F.: Sketch and Explore. accessed Mar 30, 2026 (2019), https://isabelle.in.tum.de/dist-Isabelle2025-2/library/HOL/HOL-ex/Sketch_and_Explore.html
21. Hindley, J.R.: The principal type-scheme of an object in combinatory logic. Transactions of the American Mathematical Society **146**, 29–60 (1969). https://doi.org/10.2307/1995158
22. Jenkins, C., Stump, A.: Spine-local type inference. In: Proceedings of the 30th Symposium on Implementation and Application of Functional Languages. p. 37–48. IFL '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3310232.3310233
23. Jiang, A.Q., Welleck, S., Zhou, J.P., Lacroix, T., Liu, J., Li, W., Jamnik, M., Lample, G., Wu, Y.: Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=SMa9EAovKMC
24. Ju, H., Dong, B.: Ai for mathematics: Progress, challenges, and prospects (2026). https://doi.org/10.48550/ARXIV.2601.13209
25. Kappelmann, K., Schäffeler, M., Stevens, L., Abdulaziz, M., Popescu, A., Traytel, D.: Just type it in Isabelle! AI agents drafting, mechanizing, and generalizing from human hints (2026), https://arxiv.org/abs/2604.15713
26. Kappelmann, K., Schäffeler, M., Stevens, L., Abdulaziz, M., Popescu, A., Traytel, D.: Supplementary material associated to this submission. (Apr 2026). https://doi.org/10.5281/zenodo.19406624
27. Knuth, D.: Claude's cycles (2026), https://www-cs-faculty.stanford.edu/~knuth/papers/claude-cycles.pdf
28. Korte, B., Vygen, J.: Combinatorial Optimization. Springer (2012). https://doi.org/10.1007/978-3-642-24488-9
29. Le Botlan, D., Rémy, D.: Mlf: raising ml to the power of system f. In: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming. p. 27–38. ICFP '03, Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/944705.944709
30. Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J.H., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., Wu, J., Gesi, J., Lu, X., Acuna, D., Yang, K., Lin, H., Choi, Y., Chen, D., Arora, S., Jin, C.: Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In: The Fourteenth International Conference on Learning Representations (2026), https://openreview.net/forum?id=j4C0nALrgK
31. Megdiche, Y., Huch, F., Stevens, L.: A linter for isabelle: Implementation and evaluation (2022). https://doi.org/10.48550/ARXIV.2207.10424

32. Milehins, M.: An extension of the framework types-to-sets for Isabelle/HOL. In: Popescu, A., Zdancewic, S. (eds.) CPP 2022. pp. 180–196. ACM (2022). https://doi.org/10.1145/3497775.3503674
33. Milner, R.: A theory of type polymorphism in programming. Journal of Computer and System Sciences **17**(3), 348–375 (1978). https://doi.org/10.1016/0022-0000(78)90014-4
34. Paulson, L.C.: ML for the working programmer (2. ed.). Cambridge University Press (1996)
35. Paulson, L.C., Blanchette, J.C.: Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In: IWIL@LPAR. pp. 1–11. EPiC Series in Computing, EasyChair (2010)
36. Pierce, B.C., Turner, D.N.: Local type inference. ACM Transactions on Programming Languages and Systems **22**(1), 1–44 (Jan 2000). https://doi.org/10.1145/345099.345100
37. Rendel, T., Ostermann, K.: Invertible syntax descriptions: unifying parsing and pretty printing. In: Proceedings of the third ACM Haskell symposium on Haskell. pp. 1–12. ICFP '10, ACM (Sep 2010). https://doi.org/10.1145/1863523.1863525
38. Sergey, I.: Verifying Move borrow checker in Lean: an experiment in AI-assisted PL metatheory (Mar 2026), https://proofsandintuitions.net/2026/03/18/move-borrow-checker-lean/, archive snapshot: https://web.archive.org/web/20260330164922/https://proofsandintuitions.net/2026/03/18/move-borrow-checker-lean/
39. Smolka, S.J.: Synthesis of Robust, Semi-Intelligble Isar Proofs from ATP Proofs. Bachelor's thesis, Technical University of Munich (2013), https://smolka.st/papers/bsc-thesis.pdf
40. Smolka, S.J., Blanchette, J.C.: Robust, semi-intelligible Isabelle proofs from ATP proofs. In: Blanchette, J.C., Urban, J. (eds.) PxTP 2013. Third International Workshop on Proof Exchange for Theorem Proving. EPiC Series in Computing, vol. 14, pp. 117–132 (2013). https://doi.org/10.29007/zbdb
41. Tan, C., Donaldson, A.F., Huerta y Munive, J.J., Wickerson, J.: The burden of proof: Automated tooling for rapid iteration on large mechanised proofs. In: FormaliSE 2025. pp. 34–45. IEEE (2025). https://doi.org/10.1109/FORMALISE66629.2025.00010
42. Trinh, T.H., Wu, Y., Le, Q.V., He, H., Luong, T.: Solving olympiad geometry without human demonstrations. Nature (2024). https://doi.org/10.1038/s41586-023-06747-5
43. Urban, J.: 130k lines of formal topology in two weeks: Simple and cheap autoformalization for everyone? CoRR **abs/2601.03298** (2026). https://doi.org/10.48550/ARXIV.2601.03298
44. Wang, H., Xin, H., Zheng, C., Liu, Z., Cao, Q., Huang, Y., Xiong, J., Shi, H., Xie, E., Yin, J., Li, Z., Liang, X.: LEGO-prover: Neural theorem proving with growing libraries. In: The Twelfth International Conference on Learning Representations (2024), https://openreview.net/forum?id=3f5PALef5B
45. Wells, J.: Typability and type checking in system f are equivalent and undecidable. Annals of Pure and Applied Logic **98**(1), 111–156 (1999). https://doi.org/https://doi.org/10.1016/S0168-0072(98)00047-5
46. Wenzel, M.: Isar - A generic interpretative approach to readable formal proof documents. In: TPHOLs. pp. 167–184. Lecture Notes in Computer Science, Springer (1999)

47. Wu, Y., Huang, D., Wan, R., Peng, Y., Shang, S., Cao, C., Qi, L., Zhang, R., Zhang, X., Du, Z., Yan, J., Hu, X.: Stepfun-formalizer: Unlocking the autoformalization potential of llms through knowledge-reasoning fusion. Proceedings of the AAAI Conference on Artificial Intelligence **40**(40), 33980–33988 (Mar 2026). https://doi.org/10.1609/aaai.v40i40.40691
48. Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., Ruan, C., Li, W., Liang, X.: Advancing theorem proving in LLMs through large-scale synthetic data. In: The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24 (2024), https://openreview.net/forum?id=TPtXLihkny
49. Xu, Y., Odersky, M.: Agentic proof automation: A case study (2026). https://doi.org/10.48550/ARXIV.2601.03768
50. Xue, X., Cui, C., Jiang, S., Oliveira, B.C.d.S.: Local contextual type inference. Proceedings of the ACM on Programming Languages **10**(POPL) (Jan 2026). https://doi.org/10.1145/3776653
51. Xue, X., Oliveira, B.C.d.S.: Contextual typing. Proceedings of the ACM on Programming Languages **8**(ICFP) (Aug 2024). https://doi.org/10.1145/3674655

APPENDIX

This appendix contains further details on the concepts and statements from the main paper. In case of acceptance, the appendix will be replaced by a technical report cited from the paper and made available online.

A More Details on Syntax

The definition of substitution application $\_[\rho]$ on types consists of the following recursive equations:

- $\alpha[\rho] = \rho(\alpha)$;
- $(\sigma \Rightarrow \tau)[\rho] = \sigma[\rho] \Rightarrow \tau[\rho]$;
- $((\sigma_1, \ldots, \sigma_n)\, \kappa)[\rho] = (\sigma_1[\rho], \ldots, \sigma_n[\rho])\, \kappa$.

And similarly for the one on terms:

- $x_\xi[\rho] = x_{\xi[\rho]}$
- $c_\xi[\rho] = c_{\xi[\rho]}$
- $(t_1\, t_2)_\xi[\rho] = (t_1[\rho]\, t_2[\rho])_{\xi[\rho]}$
- $(\lambda x_\xi.\, t)_\zeta[\rho] = (\lambda x_{\xi[\rho]}.\, t[\rho])_{\zeta[\rho]}$

The definition of TV (the occurring tyvars operator) on types consists of the following recursive equations:

- $\mathsf{TV}(\alpha) = \{\alpha\}$;
- $\mathsf{TV}(\sigma \Rightarrow \tau) = \mathsf{TV}(\sigma) \cup \mathsf{TV}(\tau)$;
- $\mathsf{TV}((\sigma_1, \ldots, \sigma_n)\, \kappa) = \bigcup_{i=1}^{n} \mathsf{TV}(\sigma_i)$.

And similarly for the one on terms:

- $\mathsf{TV}(x_\xi) = \mathsf{TV}(\xi)$
- $\mathsf{TV}(c_\xi) = \mathsf{TV}(\xi)$
- $\mathsf{TV}((t_1\, t_2)_\xi) = \mathsf{TV}(t_1) \cup \mathsf{TV}(t_2) \cup \mathsf{TV}(\xi)$
- $\mathsf{TV}((\lambda x_\xi.\, t)_\zeta) = \mathsf{TV}(\xi) \cup \mathsf{TV}(t) \cup \mathsf{TV}(\zeta)$

We write $[n_1, \ldots, n_k]$ for the list consisting of $n_1, \ldots, n_k$ (in particular, [] for the empty list) and $n \cdot l$ for consing an element n to a list l. The set of positions of a term, $\mathsf{Poss}(t)$, is defined by:

$$\mathsf{Poss}(x_\xi) = \mathsf{Poss}(c_\xi) = \{[]\} \qquad \mathsf{Poss}((t\, s)_\xi) = \{[], 1 \cdot \mathsf{Poss}(t), 2 \cdot \mathsf{Poss}(s)\}$$
$$\mathsf{Poss}((\lambda x_\xi.\, t)_\zeta) = \{[], [1], 2 \cdot \mathsf{Poss}(t)\}$$

Each position of a term uniquely identifies a location of a subterm or a binding variable. If $p \in \mathsf{Poss}(t)$, the *maybe-type of (the subterm at) position p in t*, $\mathsf{mtpOf}(t, p)$, is defined recursively:

$$\mathsf{mtpOf}(x_\xi, p) = \xi \qquad \mathsf{mtpOf}((t_1\, t_2)_\xi, p) = \begin{cases} \mathsf{mtpOf}(t_1, p'), & \text{if } p = 1 \cdot p' \\ \mathsf{mtpOf}(t_2, p'), & \text{if } p = 2 \cdot p' \\ \xi, & \text{otherwise} \end{cases}$$

$$\mathsf{mtpOf}(c_\xi, p) = \xi \qquad \mathsf{mtpOf}((\lambda x_\xi.\, t)_\zeta, p) = \begin{cases} \xi, & \text{if } p = [1] \\ \mathsf{mtpOf}(t, p'), & \text{if } p = 2 \cdot p' \\ \zeta, & \text{otherwise} \end{cases}$$

The annotation deletion operator $t[p := \bot]$ has the following recursive definition:

$$x_\xi[p := \bot] = x_\bot \qquad c_\xi[p := \bot] = c_\bot$$

$$(t_1\ t_2)_\xi[p := \bot] = \begin{cases} (t_1[p' := \bot]\ t_2)_\xi, & \text{if } p \text{ has the form } 1 \cdot p' \\ (t_1\ (t_2[p' := \bot]))_\xi, & \text{if } p \text{ has the form } 2 \cdot p' \\ (t_1\ t_2)_\bot, & \text{otherwise} \end{cases}$$

$$(\lambda x_\xi.\ t)_\zeta[p := \bot] = \begin{cases} (\lambda x_\bot.\ t)_\zeta, & \text{if } p = 1 \\ (\lambda x_\xi.\ (t[p' := \bot]))_\zeta, & \text{if } p \text{ has the form } 2 \cdot p' \\ (\lambda x_\xi.\ t)_\bot, & \text{otherwise} \end{cases}$$

B More Details on the Smolka-Blanchette Algorithm

The definition of $\mathsf{decrease}$ is correct (i.e., its recursion terminates) because, in the recursive call, the number of positions p in s such that $\mathsf{mtpOf}(s, p) \neq \bot$, i.e., $|\mathsf{Poss}_{\neq\bot}(s)|$ where $\mathsf{Poss}_{\neq\bot}(s) = \{p \in \mathsf{Poss}(s) \mid \mathsf{mtpOf}(s, p) \neq \bot\}$, decreases by 1.

C More Details on the Human-Authored Paper Proof Development

C.1 Constructor-aware inversion lemmas

Lemma 12 (term-constructor-guided inversion rules for $\vdash$) The following hold:
(1) If $\vdash c_\sigma$ then $\sigma \leq \mathsf{ctpOf}(c)$
(2) If $\vdash (t_1\ t_2)_\sigma$ then there exists τ such that $\vdash t_1, \vdash t_2$ and $\mathsf{tpOf}(t_1) = \mathsf{tpOf}(t_2) \Rightarrow \sigma$.
(3) If $\vdash (\lambda x_\sigma.\ t)_\tau$ then $\vdash t$ and $\tau = \sigma \Rightarrow \mathsf{tpOf}(t)$.

Proof. Follows from the standard inversion rule associated to the inductive definition of $\vdash$ and the injectiveness and distinctness of the syntactic term constructors. □

Lemma 13 (term-constructor-guided left inversion rules for $\sqsubseteq$) The following hold:
(1) If $x_\xi \sqsubseteq t$ then there exists ξ' such that $\xi \sqsubseteq \xi'$ and $t = x_{\xi'}$.
(2) If $c_\xi \sqsubseteq t$ then there exists ξ' such that $\xi \sqsubseteq \xi'$ and $t = c_{\xi'}$.
(3) If $(s_1\ s_2)_\xi \sqsubseteq t$ then there exist s_1', s_2' and ξ' such that $s_1 \sqsubseteq s_1'$, $s_2 \sqsubseteq s_2'$, $\xi \sqsubseteq \xi'$ and $t = (s_1'\ s_2')_{\xi'}$.
(4) If $(\lambda x_\zeta.\ s)_\xi \sqsubseteq t$ then there exist s', ζ' and ξ' such that $s \sqsubseteq s'$, $\zeta \sqsubseteq \zeta'$, $\xi \sqsubseteq \xi'$ and $t = (\lambda x_{\zeta'}.\ s')_{\xi'}$.

Proof. Follows from the standard inversion rule associated to the inductive definition of $\sqsubseteq$ and the injectiveness and distinctness of the syntactic term constructors. □

Lemma 14 (term-constructor-guided right inversion rules for $\sqsubseteq$) The following hold:
(1) If $t \sqsubseteq x_\xi$ then there exists ξ' such that $\xi' \sqsubseteq \xi$ and $t = x_{\xi'}$.
(2) If $t \sqsubseteq c_\xi$ then there exists ξ' such that $\xi' \sqsubseteq \xi$ and $t = c_{\xi'}$.
(3) If $t \sqsubseteq (s_1\ s_2)_\xi$ then there exist s_1', s_2' and ξ' such that $s_1' \sqsubseteq s_1$, $s_2' \sqsubseteq s_2$, $\xi' \sqsubseteq \xi$ and $t = (s_1'\ s_2')_{\xi'}$.
(4) If $t \sqsubseteq (\lambda x_\zeta.\ s)_\xi$ then there exist s', ζ' and ξ' such that $s' \sqsubseteq s$, $\zeta' \sqsubseteq \zeta$, $\xi' \sqsubseteq \xi$ and $t = (\lambda x_{\zeta'}.\ s')_{\xi'}$.

Proof. Follows from the standard inversion rule associated to the inductive definition of $\sqsubseteq$ and the injectiveness and distinctness of the syntactic term constructors. □

C.2 Lemmas on syntax basics

Lemma 15 The following hold:
(1) $\tau[\rho \bullet \rho'] = \tau[\rho'][\rho]$.
(2) If $\beta \notin \mathsf{TV}(\tau)$, then $\tau[\beta/\alpha][\alpha/\beta] = \tau$.
(3) $\tau[\rho] = \tau[\rho']$ iff $\forall \alpha \in \mathsf{TV}(\tau).\ \rho(\alpha) = \rho'(\alpha)$.
(4) $\mathsf{TV}(\tau[\rho]) = \bigcup_{\alpha \in \mathsf{TV}(\tau)} \mathsf{TV}(\rho(\alpha))$.
(5) $\mathsf{TV}(\tau)$ is finite.
(6) $\leq$ is a preorder on types.
(7) Assume that $\forall \alpha \in \mathsf{TV}(\tau).\ \rho(\alpha) \leq \rho'(\alpha)$. Then $\tau[\rho] \leq \tau[\rho']$.

Proof. (1)–(5) and (7) follow by straightforward structural induction over τ, using the definitions of TV and $\_[\_]$ on types. (3) also needs the injectiveness and distinctness properties of the syntactic constructors for types.
(6) follows immediately from (1). □

Lemma 16 The following hold for all unambiguous terms t:
(1) $t[1_{\mathsf{TVar}}] = t$ and $t[\rho \bullet \rho'] = t[\rho'][\rho]$.
(2) If $\beta \notin \mathsf{TV}(t)$, then $t[\beta/\alpha][\alpha/\beta] = t$.
(3) $t[\rho] = t[\rho']$ iff $\forall \alpha \in \mathsf{TV}(t).\ \rho(\alpha) = \rho'(\alpha)$.[6] In particular, $t[\rho] = t$ iff $\forall \alpha \in \mathsf{TV}(t).\ \rho(\alpha) = \alpha$.
(4) $\mathsf{TV}(t[\rho]) = \bigcup_{\alpha \in \mathsf{TV}(t)} \mathsf{TV}(\rho(\alpha))$.
(5) $\mathsf{TV}(t)$ is finite.
(6) $\leq$ is a preorder.

Proof. (1)–(5) follow by straightforward structural induction over t, using the definitions of TV and $\_[\_]$ on terms, and the corresponding fact for types (from Lemma 15). Again, (3) needs injectiveness and distinctness of the syntactic constructors for terms. The second part of (3) follows from the first part of (3) and the first part of point (1).
(6) again follows easily from (1). □

[6] This "iff" version covers both substitution extensionality and its converse.

C.3 Annotation lemmas

Lemma 17 Assume $t \in \mathsf{Term}$ and $p \in \mathsf{Poss}(t)$. Then the following hold:
(1) $t[p := \bot] \sqsubseteq t$.
(2) $\mathsf{mtpOf}(t[p := \bot], p) = \bot$.
(3) For all $q \in \mathsf{Poss}(t) \setminus \{p\}$, $\mathsf{mtpOf}(t[p := \bot], q) = \mathsf{mtpOf}(t, q)$.
(4) If t is unambiguous, then $t[p := \bot]$ is unambiguous.

Proof. By straightforward induction on t, using the definitions of $\_[\_ := \_]$ and mtpOf, and that of unambiguity. □

C.4 The type preservation lemma

Lemma 18 Assume v is an unambiguous F-term such that $\vdash v$ and ρ a substitution. Then $\vdash v[\rho]$.

Proof. By structural induction on v, using the definitions of TV and $\_[\_]$. □

C.5 The$\sqsubseteq$- and $(\sqsubseteq, \leq)$-lemmas

Lemma 19 The following hold:
(1) $\sqsubseteq$ is a partial order (and therefore $\sqsubset$ is a strict partial order) on Term having the U-terms as the minimal elements and the F-terms as the maximal elements.
(2) If $t \sqsubseteq s$, then $\mathsf{mtpOf}(t) \sqsubseteq \mathsf{mtpOf}(s)$.
(3) $\mathsf{erase}(t) \sqsubseteq t$.
(4) If $t \sqsubseteq s$ then t is unambiguous iff s is unambiguous.

Proof. (1), (2) and (4): By induction on the definition of $\sqsubseteq$. For transitivity, we also need the right-inversion rules for $\sqsubseteq$ (Lemma 14).
(3): By structural induction on t, using the definition of erase.

The next lemma essentially says that any two terms connected by $\sqsubseteq$ or $\leq$ have the "same shape" hence the same positions, and their "at position" type annotations are related correspondingly.

Lemma 20 (1) If $t \sqsubseteq s$ then $\mathsf{Poss}(t) = \mathsf{Poss}(s)$.
(2) If $t \sqsubseteq s$ and $p \in \mathsf{Poss}(t)$ then $\mathsf{mtpOf}(t, p) \sqsubseteq \mathsf{mtpOf}(s, p)$.
(3) If $t \leq s$ then $\mathsf{Poss}(t) = \mathsf{Poss}(s)$.
(4) If $t \leq s$ and $p \in \mathsf{Poss}(t)$ then $\mathsf{mtpOf}(t, p) \leq \mathsf{mtpOf}(s, p)$.

Proof. (1) and (2): By induction on the inductive definition of $\sqsubseteq$, using the definitions of Poss and $\mathsf{mtpOf}(t, p)$.

(3) and (4): We obtain ρ such that $t = s[\rho]$. Then the proof goes by structural induction on t. □

C.6 The generic reverse greedy lemmas

Strictly speaking, only Lemma 21 is entirely generic, whereas Lemma 22 also uses some specific properties of $\sqsubseteq$, including transitivity.

Lemma 21 Assume t is an unambiguous term and v is an F-term and $pickPos \in \mathsf{Compat}$. Let $s = \mathsf{decrease}(pickPos, v, t)$. Then $\neg\, \exists p.\ \mathsf{coverageTest}(v, s, p)$.

Proof. Immediately by induction on $|\mathsf{Poss}_{\neq\bot}(s)|$ (alternatively, by the computation induction principle stemming from the definition of $\mathsf{decrease}$). □

Lemma 22 Assume t is an unambiguous term and v is an F-term and $pickPos \in \mathsf{Compat}$. Let $s = \mathsf{decrease}(pickPos, v, t)$. Then $s \sqsubseteq \mathsf{mgen}(t)$, in particular s is unambiguous.

Proof. By induction on $|\mathsf{Poss}_{\neq\bot}(s)|$ (alternatively, by the computation induction principle stemming from the definition of $\mathsf{decrease}$), using Lemma 19(1,4). □

C.7 Some omitted proofs and proof sketches

Proof of Lemma 9. By induction on the definition of $\sqsubseteq$, using the definition of mtpOf. □

Proof of Lemma 10. By structural induction on v, using the definitions of $\sqsubseteq$ and $\_[\_]$. □

Proof of Lemma 11. From the definitions, we have $s \sqsubseteq v$ and $\vdash v$, and we obtain α such that $\alpha \in \mathsf{TV}(v) \setminus \mathsf{TV}(s)$. From this and $s \sqsubseteq v$, using the annotation lemmas and the $\sqsubseteq$-lemmas, we have (a) $\forall p \in \mathsf{Poss}(v) = \mathsf{Poss}(s).\ \alpha \in \mathsf{TV}(v, p) \longrightarrow \mathsf{mtpOf}(s, p) = \bot$.

We choose β such that (b) $\beta \notin \mathsf{TV}(v)$ (in particular, $\beta \neq \alpha$), and let $v' = v[\beta/\alpha]$. By syntax basics and $\alpha \in \mathsf{TV}(v)$, we have $v \neq v'$. Moreover, again by syntax basics, from (b) we obtain (c) $v = v[\beta/\alpha][\alpha/\beta] = v'[\alpha/\beta]$.

By the type preservation lemma from $\vdash v$ we obtain $\vdash v'$. Moreover, by Lemma 10 used with the identity and β/α as substitutions, from $s \sqsubseteq v$ and (a) we obtain $s \sqsubseteq v'$. Thus, v' is distinct from v, and is a well-typed completion of s.

It remains to show that v' is a most general well-typed completions of s. Let w be a well-typed completion of s. Since v is a most general completion, we obtain ρ such that $w = v[\rho]$. Moreover, by syntax basics, from (c) we obtain $v[\rho] = v'[\alpha/\beta][\rho] = v'[\rho \bullet (\alpha/\beta)]$. So $w \leq v'$, as desired. □

D More Details on the AI Paper Development

The human and AI agent pen-and-paper developments have been developed mostly independently, with *the only influence occurring from the AI agent to the human*—who, as we report in the main paper, obtained the idea of stating and proving minimality from the AI agent. Unavoidably, independent developments lead to different design decisions, and this was the case here:

(1) The human expert considered the problem starting from an arbitrary typable C-term t, (which corresponds to an Isabelle term, i.e., features the usual Church-style type annotations), then proved in Prop 2 that such a term has a unique well-typed completion, namely the F-term $\mathsf{mgen}(t)$, and applied the algorithm starting with $\mathsf{mgen}(t)$. By contrast, the AI ignored the original term t and worked directly with an F-term (corresponding to what the human called $\mathsf{mgen}(t)$).[7] In this respect, the result proved by the AI is less complete, but consistent because $\mathsf{erase}(\mathsf{mgen}(t))$ is the same as $\mathsf{erase}(t)$.
(2) The human expert did not consider typing contexts, after making an argument that these are "not interesting" in that the problem can be reduced to an empty-context problem (by considering the typing context to be part of the signature). By contrast, the AI considered typing contexts, and formulated the results with the additional requirements for the proper treatment of tyvars from the context by the substitutions. In this respect, the result proved by the AI is more complete, as it avoids any informal (meta-)argument in order to cover the general case.
(3) In the human account, the algorithm operates with an evolving (partially annotated) term of the same shape as the original term, which iteratively loses annotations. In the AI account, the algorithm, and the corresponding theorems, operate instead on sets of positions, and therefore use inclusions between sets of positions as opposed to the annotation subsumption relation $\sqsubseteq$. This makes the AI account more direct, although arguably less intuitive.[8]
(4) The AI fixed a lot of items and made several assumptions globally, which were shared by many lemmas and proposition statements, including the main results (in the style of Isabelle locales), whereas the human preferred to distribute the assumptions as needed in each statement. These two approaches trade succinctness on the one hand with readability and precision on the other hand.[9]

E More Details on the Autoformalization Experiments

We show the complete statements of the formalization in App. E.1 and App. E.2, for the AI-authored and human-authored proofs respectively.

E.1 AI-authored proof

The main results of the formalization are stated within the context of the locale `annotation_problem` (cf. Fig. 2). It introduces

[7] Incidentally, since the AI did not prove a result analogous to Prop 2, it did not need syntactic-constructor-aware inversion rules or anything equivalent to that.
[8] Incidentally, the AI, following the implementation, represents positions differently from (but equivalently to) the human, namely as numbers rather than lists.
[9] Incidentally, the AI paper introduced some scoping issues, such as a definition inlined inside a theorem and referred to from another theorem, which were nevertheless sorted at autoformalization time. It also introduced some useless/tautological but harmless assumptions.

```
locale annotation_problem =
  fixes const_type :: "string ⇒ ty"
    and Γ :: ctx
    and a :: aterm
    and a_star :: aterm
    and σ :: "string ⇒ ty"
  assumes a_wt: "well_typed const_type Γ a"
    and a_star_wt: "well_typed const_type Γ a_star"
    and same_erasure: "erase a_star = erase a"
    and matching: "subst_aterm σ a_star = a"
    and freshness: "⋀α. α ∈ tvars_ctx Γ ⟹ σ α = TVar α"
```

Fig. 2. AI-based annotation problem locale: well-formedness assumptions on the input `a` as well as type inference in terms of `a_star`.

```
theorem completeness_t_out:
  assumes "well_typed const_type Γ a'"
    and "erase a' = strip t_out"
    and "∃σ'. subst_aterm σ' a_star = a' ∧ (∀α ∈ tvars_ctx Γ. σ' α = TVar α)"
    and "consistent_with a' (reverse_greedy candidates)"
  shows "a' = a"
```

Fig. 3. AI-based completeness statement: `a` is the unique term `a'` that is *consistent with* `a` at the positions determined by the `reverse_greedy` algorithm. The locale context only assumes type inference properties for `a`, so this assumption is repeated here for `a'`.

```
theorem local_minimality_t_out:
  assumes "p ∈ reverse_greedy candidates"
  shows "∃a'. a' ≠ a ∧ well_typed const_type Γ a'
    ∧ erase a' = erase a
    ∧ consistent_with a' (reverse_greedy candidates - {p})"
```

Fig. 4. AI-based minimality statement: removing any position the algorithm deems necessary allows us to obtain a well-typed term different from `a` that agrees with `a` on the remaining positions.

- the fully annotated input term `a`, corresponding to t in the definition of smobla;
- a most general completion `a_star` (of `a` stripped from type annotations), corresponding to mgen(erase(t));
- a type substitution relating `a` and `a_star`;
- a context `Γ` and a constant signature `const_type`;
- well-typedness assumptions on `a` and `a_star`;
- a freshness assumption on the type substitution, modeling that type inference uses fresh type variables.

Both completeness (cf. Fig. 3) and minimality (cf. Fig. 4) are stated within the `annotation_problem` locale. A limitation inherited from the AI-authored paper is that both statements refer to the set of annotation positions, not only the output term `t_out`.

```
locale signature_with_mgen = signature ctpOf
  for ctpOf :: "'c ⇒ ('tv,'k) ty" +
  fixes dummy_v :: "'v itself"
  assumes mgen_exists:
    "⟦ typeable (t::('tv,'k,'v,'c) trm); unambiguous t ⟧ ⟹ ∃v. is_mgen t v"
```

Fig. 5. Human-based signature locale: corresponds to Thm. 1

```
locale algorithm = signature_with_mgen ctpOf "TYPE('v)"
  for ctpOf :: "'c ⇒ ('tv,'k) ty" +
  fixes pickPos :: "('tv,'k,'v,'c) trm ⇒ ('tv,'k,'v,'c) trm ⇒ nat list"
  assumes compat: "test v s p ⟹ test v s (pickPos v s)"
```

Fig. 6. Human-based algorithm locale

```
theorem completeness:
  assumes ty: "typeable t" and ua: "unambiguous t" and ct: "cterm t"
  shows compl_part: "is_wt_completion (smobla t) (mgen t)"
    and unique_part: "∀u. is_wt_completion (smobla t) u ⟶ u = mgen t"
```

Fig. 7. Human-based completeness statement: corresponds to Thm. 4

```
corollary minimality_distinct_from_mgen_t:
  assumes "typeable t" "unambiguous t" "cterm t"
    "unambiguous (s'::('tv,'k,'v,'c) trm)" "s' ⊑ smobla t" "s' ≠ smobla t"
    "infinite (UNIV :: 'tv set)"
  shows "∃v'. v' ≠ mgen t ∧ is_mgen s' v'"
```

Fig. 8. Human-based minimality statement: corresponds to Thm. 5

E.2 Human-authored proof

The formalization defines the locale `signature_with_mgen` (cf. Fig. 5), it fixes a type signature and also follows the paper and assumes Thm. 1. The `algorithm` locale (cf. Fig. 6) abstracts over specific *pickPos* functions, where the `compat` assumption corresponds to Compat. Both completeness (cf. Fig. 7) and minimality (cf. Fig. 8) are stated within this `algorithm` locale.