What's Decidable About (Atomic) Polymorphism?

Paolo Pistone
University of Bologna, Italy

Luca Tranchini
Eberhard Karls Universität Tübingen, Germany

Abstract

Due to the undecidability of most type-related properties of System F like type inhabitation or type checking, restricted polymorphic systems have been widely investigated (the most well-known being ML-polymorphism). In this paper we investigate System Fat, or atomic System F, a very weak predicative fragment of System F whose typable terms coincide with the simply typable ones. We show that the type-checking problem for Fat is decidable and we propose an algorithm which sheds some new light on the source of undecidability in full System F. Moreover, we investigate free theorems and contextual equivalence in this fragment, and we show that the latter, unlike in the simply typed lambda-calculus, is undecidable.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Higher order logic

Keywords and phrases Atomic System F, Predicative Polymorphism, ML-Polymorphism, Type-Checking, Contextual Equivalence, Free Theorems

Digital Object Identifier 10.4230/LIPIcs.FSCD.2021.27

Related Version *Full Version*: `https://arxiv.org/abs/2105.00748`

Funding *Luca Tranchini*: DFG TR1112/4-1 "Falsity and Refutation. On the negative side of logic"

1 Introduction

Polymorphism has been a central topic in programming language theory since the late sixties. Today, most general purpose programming languages employ some kind of polymorphism. At the same time, under the Curry-Howard correspondence, quantification over types corresponds to quantification over propositions, that is, to second-order logic. In particular, System F, the archetypical type system for polymorphism, can be seen as a proof-system for (the $\Rightarrow, \forall$-fragment of) second-order intuitionistic logic.

In spite of the numerous applications of polymorphism, practically all interesting type-related properties of (Curry-style) System F (e.g. type checking, type inhabitation, etc.) are undecidable, making this language impractical for any reasonable implementation. This is one of the reasons why a wide literature has investigated more manageable subsystems of System F. Notably, ML-*polymorphism* [41, 42, 40] has found much success due to its decidable type-checking.

Another direction of research was that of investigating *predicative* subsystems of System F [32, 33, 34, 6]. In particular, the so-called *finitely stratified polymorphism* [33] yields a stratification of System F through a sequence of predicative systems $(\mathrm{F}_n)_{n\in\mathbb{N}}$ of growing expressive power (notably, F_0 is the simply typed λ-calculus STλC, and ML-polymorphism coincides with the rank-1 part of F_1). Yet, in spite of such limitations, type checking becomes undecidable already at level 1 of this hierarchy [18].

Could one tell exactly at which point, in the range from the simply typed λ-calculus and ML to full System F, the type-related properties of polymorphism become undecidable?

6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021).
Editor: Naoki Kobayashi; Article No. 27; pp. 27:1–27:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

	$F_0 = ST\lambda C$	F_{at}	ML	F_1	F
TI	decidable [59]	undecidable [52]	open	undecidable [57]	undecidable [37]
TC	decidable [25]	**decidable**	decidable [40]	undecidable [18]	undecidable [64]
T	decidable [25]	**decidable**	decidable [40]	undecidable [18]	undecidable [64]
CE (for numerical functions)	decidable [44]	**decidable**	**undecidable**	**undecidable**	undecidable*
CE (full)	decidable [44]	**undecidable**	**undecidable**	**undecidable**	undecidable**

Figure 1 Decidable and undecidable properties of System F and some predicative fragments (in bold the properties established in the present paper).
*: easy consequence of Rice's theorem and the typability of all primitive recursive functions in F (see also Remark 18).
**: consequence of the undecidability of (CE) for numerical functions.

Atomic Polymorphism. In more recent times Ferreira et al. have undertaken the investigation of what can be seen as the least expressive predicative fragment of F, System F_{at}, or *atomic* System F [12, 11, 13, 15, 16, 10, 9]. The predicative restriction of F_{at} is such that a universally quantified type $\forall X.A$ can be instantiated solely with an atomic type, i.e. a type variable. In this way F_{at} sits in between level 0 (i.e. STλC) and level 1 of the finitely stratified hierarchy. Actually, F_{at} can be seen as a *type refinement* system (in the sense of [39]) of STλC, since all terms typable in F_{at} are simply typable (cf. Lemma 7).

In spite of its very limited expressive power, Ferreira et al. have shown that, thanks to polymorphism, F_{at} enjoys some proof-theoretic properties that STλC lacks. In particular, they defined a predicative variant of the usual encoding of sum and product types inside F, yielding an embedding of intuitionistic propositional logic inside F_{at}. However, while propositional logic is decidable, provability in second-order propositional intuitionistic logic, even with the atomic restriction, is undecidable [56]. This argument (as recently observed in [52]) can be extended to show that the *type inhabitation* property, which is decidable for STλC, is undecidable for F_{at}.

Contributions

In this paper we investigate the following type-related properties of System F_{at}:

Type inhabitation (TI):	given A, is there t such that $\vdash t : A$?
Type-checking (TC):	given Γ, A, t, does $\Gamma \vdash t : A$?
Typability (T):	given Γ, t, is there A such that $\Gamma \vdash t : A$?
Contextual equivalence (CE):	given A, t, u such that $\vdash t, u : A$, do $\mathsf{C}[t]$ and $\mathsf{C}[u]$ reduce to the same Boolean, for all context $\mathsf{C}[\ ] : A \Rightarrow \mathsf{Bool}$?

In Fig. 1 we sum up what is already known and what is established in this paper (in bold) about such properties in predicative fragments of System F. Our main results are that in F_{at} (TC) and (T) are both decidable, and that (CE) is decidable if one restricts oneself to numerical functions, and undecidable in the general case.

Several decidability properties of F_{at} are tight, meaning that they all fail already for F_1. In these cases, our arguments can be used to shed some new insights on the broader question of understanding where the source of undecidability for such properties in full System F lies.

Plan of the paper

After recalling the syntax of F and its fragment in Curry-style and Church-style, we address the properties (TI), (TC), (T) and (CE).

Type Inhabitation. In Section 3 we shortly discuss the undecidability of (TI), by showing how the argument in [57] for System F applies to $\mathrm{F_{at}}$ too. This argument yields an encoding inside $\mathrm{F_{at}}$ of an undecidable fragment of *first-order* intuitionistic logic. We also observe that $\mathrm{F_{at}}$ is actually equivalent to a first-order system, namely to the $\Rightarrow,\forall$-fragment $1\mathrm{Mon}^{\Rightarrow,\forall}$ of first-order *monadic* intuitionistic logic in a language with a unique monadic predicate. To our knowledge, the undecidability of $1\mathrm{Mon}^{\Rightarrow,\forall}$ has not been previously observed (although some slightly more expressive fragments - e.g. including a primitive disjunction [19] or finitely many monadic predicates [54] - have been proven undecidable).

Type-Checking and Typability. In Section 4 we consider the type-checking problem. The undecidability of (TC) for System F was established by Wells in [64], and was later extended to all predicative systems F_n, for $n > 0$ [18]. In all these cases this result was obtained by reducing an undecidable variant of *second-order unification* (SOU) to the type-checking problem. On the other hand, the decidability of (TC) for ML (and $\mathrm{F}_0 = \mathrm{ST}\lambda\mathrm{C}$) is based on the famous Hindley-Milner algorithm [40], which reduces this problem to *first-order unification* (FOU), which is decidable.

The fundamental source of undecidability of SOU is the presence of *cyclic* dependences between second order variables, expressed in the simplest case by equations of the form $X(t) = f(v_1, \dots, v_{k-1}, X(u), v_{k+1}, \dots, v_n)$. In fact, *acyclic* SOU is decidable [36]. When type-checking polymorphic programs, such cyclic dependencies are generated by *self-applications*, i.e. terms of the form $\lambda\vec{x}.xt_1 \dots t_{k-1}xt_{k+1} \dots t_n$. In fact, in this case the type $\forall X.A$ assigned to the variable x must satisfy a cyclic equation of the form

$$A[X \mapsto C_1] = B_1 \Rightarrow \dots \Rightarrow B_{k-1} \Rightarrow A[X \mapsto C_2] \Rightarrow B_{k+1} \Rightarrow \dots \Rightarrow B_n$$

(where C_1, C_2 are suitable type instantiations of X). By constrast, no term containing a self-application can be typed in $\mathrm{ST}\lambda\mathrm{C}$, since cyclic equations cannot be solved by FOU.

Since the terms typable in $\mathrm{F_{at}}$ can also be typed in $\mathrm{ST}\lambda\mathrm{C}$ (cf. Lemma 7), it follows that self-applications cannot be typed in $\mathrm{F_{at}}$ either. Using this observation, we describe a type-checking algorithm for $\mathrm{F_{at}}$ which works in two phases: first, it checks (using FOU) the presence of cyclic dependencies, and returns `failure` if it detects one; then, if phase 1 succeeds, it applies (a suitable variant of) acyclic SOU to decide type-checking. From the decidability of (TC), we deduce the decidability of (T) by a standard argument (see [4]).

Contextual Equivalence. Studying the typable terms of $\mathrm{F_{at}}$ might not seem very interesting from a computational viewpoint, as these terms are already typable in $\mathrm{ST}\lambda\mathrm{C}$. However, due to the presence of some form of polymorphism, investigating programs in $\mathrm{F_{at}}$ can be interesting for *equational* reasoning, as we do in Sections 5 and 6. In standard type systems, beyond the standard notions of program equivalence arising from the operational semantics (i.e. $\beta\eta$-equivalence), there may exist several other congruences arising from either denotational models or from some notion of *contextual equivalence*. In $\mathrm{ST}\lambda\mathrm{C}$, it is well-known that $\simeq_{\beta\eta}$ coincides with the congruence induced by any infinite extensional model [58], as well as with several notions of contextual equivalence (see [5], [7]). In polymorphic type systems the picture is rather different, since $\beta\eta$-equivalence is usually weaker than the congruences arising from extensional models (see [3, 23]), and also weaker than standard notions of contextual equivalence. Moreover, while $\beta\eta$-equivalence is decidable, contextual equivalence is undecidable. Since in many practical situations (see [62, 1]) it is more convenient to reason up to notions of equivalence stronger than $\beta\eta$-equivalence, several techniques to compute (approximations of) contextual equivalence have been investigated, e.g. *free theorems* [63], *parametricity* [53], and *dinaturality* [3].

Our investigation of contextual equivalence starts in Section 5 with an exploration of equational reasoning in $\mathrm{F_{at}}$ using free theorems. We show that the predicative encodings of sum and product types of Ferreira et al. produce products and coproducts in $\mathrm{F_{at}}$ in the categorical sense, provided terms are considered up to (CE) (a fact which is known to hold in F for the usual, impredicative, encodings [23, 61]). We then investigate (CE) for typable numerical functions. Using the fact that the primitive recursive functions are *uniquely* defined in System F up to (CE), we show that (CE) for the representable numerical functions is decidable in $\mathrm{F_{at}}$, and undecidable in ML. Such results rely on the observation that (CE) becomes undecidable as soon as some super-polynomial function (like bounded multiplication) becomes representable. From this it can be deduced that (CE) is undecidable in all fragments F_n, for $n > 0$, of the finitely stratified hierarchy as well.

Finally, in Section 6 we establish that (CE) is undecidable also in $\mathrm{F_{at}}$, by showing that the type inhabitation problem for a suitable extension of $\mathrm{F_{at}}$ can be reduced to it. This result, together with the previous ones, shows that there is no hope to get a decidable contextual equivalence for polymorphic programs through a predicative restriction, and one has rather to look for other kinds of restrictions (see for instance [49]).

2 Predicative Polymorphism and System $\mathrm{F_{at}}$

The systems we consider in this paper are all restrictions of usual Church-style and Curry-style System F. The types are defined in both cases by the grammar

$$A, B ::= X \mid A \Rightarrow B \mid \forall X.A$$

starting from a countable set $\mathtt{Var}^2$ of type variables $X_1, X_2, \dots$. The terms of Church-style System F are defined by the grammar below:

$$t^A, u^A ::= x^A \mid (\lambda x^A.t^B)^{A\Rightarrow B} \mid t^{B\Rightarrow A}u^B \mid (\Lambda X.t^A)^{\forall X.A} \mid (t^{\forall X.A}C)^{A[C/X]}$$

For readability, we will often omit type annotations, when these can be guessed from the context. The terms of Curry-style System F are standard λ-terms, with typing rules defined as in Fig. 2, where Γ indicates a partial function from term variables to types with a finite support, and by $X \notin \mathrm{FV}(\Gamma)$ we indicate that X does not occur free in any type in $\mathrm{Im}(\Gamma)$. We call the type C occurring in $(t^{\forall X.A}C)^{A[C/X]}$ and in the rule $\forall$E in Fig. 2 the *witness* of the type instantiation.

We indicate *term contexts* (i.e. terms with a *hole* $[\ ]$) as $\mathtt{C}[\ ], \mathtt{D}[\ ]$. Moreover, we let $\mathtt{C}[\ ] : A \vdash B$ be a shorthand for $x \mapsto A \vdash \mathtt{C}[x] : B$.

System F is impredicative: *any* type can figure as a witness. In particular, one can construct "circular" instantiations, in which a term of type $\forall X.A$ is instantiated with the same type as witness. A *predicative* fragment of System F is one in which witnesses are restricted in such a way to avoid such circular instantiations.

We will focus on three predicative fragments of System F, both in Church- and Curry-style. The first is System $\mathrm{F_1}$, which is the fragment of F in which witnesses are *quantifier-free*. The second is System $\mathrm{F_{at}}$, which is the fragment of F in which witnesses are *atomic*, that is,

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A}\ \mathrm{Var}$$

$$\frac{\Gamma, x \mapsto A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}\ \mathrm{Abs} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}\ \mathrm{Appl}$$

$$\frac{\Gamma \vdash t : B \qquad X \notin \mathrm{FV}(\Gamma)}{\Gamma \vdash t : \forall X.A}\ \forall\mathrm{I} \qquad \frac{\Gamma \vdash t : \forall X.A}{\Gamma \vdash t : A[C/X]}\ \forall\mathrm{E}$$

Figure 2 Typing rules for Curry-style System F.

type variables. The third is system ML [41, 40], which essentially coincides with the *rank 1* fragment of F_1. For any type A, the rank $\mathsf{r}(A)$ is the maximum number of nesting between $\Rightarrow$ and $\forall$, and is defined inductively by $\mathsf{r}(X) = 0$, $\mathsf{r}(A \Rightarrow B) = \max\{\mathsf{r}(A) + 1, \mathsf{r}(B)\}$ and $\mathsf{r}(\forall X_1 \dots X_n.A) = \mathsf{r}(A) + 1$ (where $n > 0$ and A does not start with a quantifier). To define ML (since type-checking is decidable in ML, we limit ourselves to Curry-style) one first has to enrich the set of λ-terms with the let-constructor, and add a rule

$$\frac{\Gamma, x \mapsto A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{let}\ x\ \mathsf{be}\ u\ \mathsf{in}\ t : B}\ \mathrm{let}$$

ML is the fragment of the resulting system in which typing rules only contain judgements $\Gamma \vdash t : A$, where $\mathsf{r}(A) \leqslant 1$ and for all $B \in \mathrm{Im}(\Gamma)$, $\mathsf{r}(B) \leqslant 1$.

Observe that in F_1 one can encode $\mathsf{let}\ x\ \mathsf{be}\ u\ \mathsf{in}\ t$ by $(\lambda x.t)u$, so that the rule above becomes derivable. This is not possible in ML, due to the rank restriction.

Impredicative and Predicative Encodings. It is well-known that sum and product types can be encoded inside System F by letting

$$A \tilde{+} B = \forall X.(A \Rightarrow X) \Rightarrow (B \Rightarrow X) \Rightarrow X$$
$$A \tilde{\times} B = \forall X.(A \Rightarrow B \Rightarrow X) \Rightarrow X$$

where the type variable X is fresh. The encoding of term constructors $\iota_i(\cdot)$, $\langle \cdot, \cdot \rangle$ and term destructors $\mathrm{Case}_C(\cdot, x^A.\cdot, x^B.\cdot)$ and $\pi_i(\cdot)$ is given (in Church-style) by:

$$\iota_1(t) = \Lambda X.\lambda f^{A \Rightarrow X}.\lambda g^{B \Rightarrow X}.ft \qquad \mathrm{Case}_C(t, x^A.u, x^B.v) = tC(\lambda x^A.u)(\lambda x^B.v)$$
$$\iota_2(t) = \Lambda X.\lambda f^{A \Rightarrow X}.\lambda g^{B \Rightarrow X}.gt \qquad \pi_1(t) = tA\lambda x^A.\lambda y^B.x$$
$$\langle t, u \rangle = \Lambda X.\lambda f^{A \Rightarrow B \Rightarrow X}.ftu \qquad \pi_2(t) = tB\lambda x^A.\lambda y^B.y$$

At the level of provability, the encoding is *faithful*: a type is inhabited in the extension of System F with sum and product types iff the encoded type is inhabited in System F. Moreover, the encoding of $\tilde{+}$ satisfies the *disjunction property*: $A \tilde{+} B$ is inhabited iff either A or B are inhabited.

At the level of conversions, the encoding translates β-reduction step for sum and product types into (finite sequences of) β-reduction steps in F. On the other hand, the η-rules for sums and products are not translated by the β- and η- rules of System F. Yet, the equivalence generated by β- and η-rules is preserved by *contextual equivalence* in System F (more on this in Section 5).

The encoding of sum and product types is impredicative: the encoding of term destructors requires witnesses of arbitrary complexity. Notably, given a term t of type $A \tilde{+} B$, the term $\mathrm{Case}_{A \tilde{+} B}(t, x^A.\iota_1(x), x^B.\iota_2(x))$, of type $A \tilde{+} B$, has a circular instantiation of $A \tilde{+} B$.

In [12], and more recently in [9] some alternative, predicative, encodings were defined having System $\mathrm{F_{at}}$ as target. The fundamental observation is that the unrestricted $\forall$E rule is derivable from the restricted one for the types of the form $A \tilde{+} B$ and $A \tilde{\times} B$ (the authors call this phenomenon *instantiation overflow*). In fact, for any type C of System F one can define contexts $\mathsf{IO}^{+}_{C}[\,] : A \tilde{+} B \vdash (A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C$ and $\mathsf{IO}^{\times}_{C}[\,] : A \tilde{\times} B \vdash (A \Rightarrow B \Rightarrow C) \Rightarrow C$ by induction on C:

$$
\begin{aligned}
\mathsf{IO}^{+}_{X}[\,] = \mathsf{IO}^{\times}_{X}[\,] &= [\,]X \\
\mathsf{IO}^{+}_{C_1 \Rightarrow C_2}[\,] &= \lambda f^{A \Rightarrow C_1 \Rightarrow C_2}.\lambda g^{B \Rightarrow C_1 \Rightarrow C_2}.\lambda y^{C_1}.\mathsf{IO}^{+}_{C_2}[\,](\lambda z^{A}.fzy)(\lambda z^{B}.gzy) \\
\mathsf{IO}^{\times}_{C_1 \Rightarrow C_2}[\,] &= \lambda f^{A \Rightarrow B \Rightarrow C_1 \Rightarrow C_2}.\lambda y^{C_1}.\mathsf{IO}^{+}_{C_2}[\,](\lambda z^{A}.\lambda w^{B}.fzwy) \\
\mathsf{IO}^{+}_{\forall Y.C'}[\,] &= \lambda f^{A \Rightarrow \forall Y.C'}.\lambda g^{A \Rightarrow \forall Y.C'}.\Lambda Y.\mathsf{IO}^{+}_{C'}[\,](\lambda z^{A}.fzY)(\lambda z^{B}.gzY) \\
\mathsf{IO}^{\times}_{\forall Y.C'}[\,] &= \lambda f^{A \Rightarrow B \Rightarrow \forall Y.C'}.\Lambda Y.\mathsf{IO}^{+}_{C'}[\,](\lambda z^{A}.\lambda w^{B}.fzwY)
\end{aligned}
$$

One can thus encode the type destructors as for F, but replacing the type application xC in $\mathrm{Case}_C(t, x^A.u, x^B.v)$ with either $\mathsf{IO}^{+}_{C}[x]$ or $\mathsf{IO}^{\times}_{C}[x]$.

At the level of provability, this embedding is faithful when restricted to simple types, i.e. for the intuitionistic propositional calculus (see [13]): a simple type (possibly containing finite sums and products) is inhabited iff its encoding is inhabited in $\mathrm{F_{at}}$. However, faithfulness does not hold for the extension of $\mathrm{F_{at}}$ with sum and product types (see [47]). In particular, one can construct types C, D of F such that $C \tilde{+} D$ is inhabited in $\mathrm{F_{at}}$ while $C + D$ is *not* inhabited in the extension of $\mathrm{F_{at}}$ with sums and products. This also implies that the disjunction property fails for $C \tilde{+} D$ in $\mathrm{F_{at}}$, since neither C nor D are inhabited.

Interestingly, at the level of conversions, this encoding is stronger than the usual one: it translates not only β-reductions, but also the permutative conversions and a restricted form of η-conversion for sums, into sequences of β and η-reductions of $\mathrm{F_{at}}$ (see [11, 14, 9]).

3 Type Inhabitation

In this section we discuss type inhabitation in the systems $\mathrm{F_{at}}$ and $\mathrm{F_1}$. We briefly recall the undecidability argument for (TI) in System F from [57], and observe that this applies to $\mathrm{F_{at}}$ (a more detailed reconstruction can be found in [52]).

The argument in [57] (which was later simplified in [8]) is based on an embedding inside F of an undecidable fragment of first-order logic. We recall the argument in a few more details, so that it will be clear that the same argument shows the undecidability of type inhabitation in both $\mathrm{F_{at}}$ and $\mathrm{F_1}$.

Let $\mathsf{Dyad}_{\Rightarrow,\forall}$ indicate the $\Rightarrow, \forall$-fragment of intuitionistic first-order logic in a language with no function symbol and a finite number of at most *binary* relation symbols. We consider sequents of the form $\Gamma \vdash \bot$ where Γ consists of three type of assumptions:

i. atomic formulas different from $\bot$;
ii. closed formulas of the form $\forall \vec{\alpha}.(\varphi_1 \Rightarrow \dots \Rightarrow \varphi_n \Rightarrow \psi)$, where $\varphi_1, \dots, \varphi_n, \psi$ are atomic formulas and each variable in ψ occurs in some the φ_i;
iii. closed formulas of the form $\forall \alpha(\forall \beta(p(\alpha, \beta) \Rightarrow \bot) \Rightarrow \bot)$.

The problem of checking if a sequent $\Gamma \vdash \bot$ as above is deducible in $\mathsf{Dyad}_{\Rightarrow,\forall}$ is undecidable ([57], Theorem 8.8.2).

We fix a finite number of distinguished type variables:

- for each relation symbol p, three variables p_1, p_2, p_3;
- five more variables $\spadesuit, \bullet, \circ_1, \circ_2, \star$.

We let, for any type A, $A^\bullet := A \Rightarrow \bullet$, and we define, for all types A, B:

$$p_{AB} = (A^\bullet \Rightarrow p_1) \Rightarrow (B^\bullet \Rightarrow p_2) \Rightarrow p_3$$
$$p(A,B) = p_{AB} \Rightarrow \star$$

For any type A, we let $\mathcal{U}(A)$ be the set of all types $(A^\bullet \Rightarrow p_i) \Rightarrow \circ_1, A^\bullet \Rightarrow \circ_2$, where $i = 1, 2$. Given a finite list of types $A_1, \dots, A_n$, we let $\mathcal{U}(A_1, \dots, A_n) \Rightarrow B$ be a shorthand for $C_1 \Rightarrow \dots \Rightarrow C_k \Rightarrow B$, where $C_1, \dots, C_k$ are the types in $\bigcup_i \mathcal{U}(A_i)$.

Each formula φ of $\mathsf{Dyad}_{\Rightarrow,\forall}$ is translated into a type $\overline{\varphi}$ as follows:

$$\overline{p(\alpha_i, \alpha_j)} = p(X_i, X_j) \qquad \overline{\bot} = \spadesuit$$
$$\overline{\varphi \Rightarrow \psi} = \overline{\varphi} \Rightarrow \overline{\psi}$$
$$\overline{\forall \alpha_i.\varphi} = \forall \vec{X}_i.(\mathcal{U}(X_i) \Rightarrow \overline{\varphi})$$

One can easily check the following by induction:

▶ **Proposition 1.** *If $\varphi_1, \dots, \varphi_n \vdash \varphi$ is provable in $\mathsf{Dyad}_{\Rightarrow,\forall}$ and $\alpha_{i_1}, \dots, \alpha_{i_k}$ are the variables that occur in $\mathrm{FV}(\varphi)$ but not in $\mathrm{FV}(\varphi_1, \dots, \varphi_n)$, then $x_1 \mapsto \overline{\varphi}_1, \dots, x_n \mapsto \overline{\varphi}_n, \vec{y} \mapsto \mathcal{U}(X_{i_1}, \dots, X_{i_k}) \vdash t : \overline{\varphi}$ holds in F_{at} for some term t.*

The less trivial part is the following:

▶ **Theorem 2** ([57], Theorem 11.6.14)**.** *For all formulas $\varphi_1, \dots, \varphi_n$ satisfying i-iii, if $x_1 \mapsto \overline{\varphi}_1, \dots, x_n \mapsto \overline{\varphi}_n \vdash t : \spadesuit$ is deducible in System F, then $\varphi_1, \dots, \varphi_n \vdash \bot$ is provable in $\mathsf{Dyad}_{\Rightarrow,\forall}$.*

Since F_{at} and F_1 are both fragments of F, we can freely substitute them for System F in the statement of Theorem 2. Then, together with Proposition 1 we deduce:

▶ **Corollary 3.** *(TI) is undecidable in both F_{at} and F_1.*

▶ Remark 4. Although F_{at} and F_1 are both undecidable, they are not equivalent at the level of provability. For instance, the type $(\forall X.X \Rightarrow Y) \Rightarrow (Z \Rightarrow Z) \Rightarrow Y$ is inhabited in F_1 (by the term $\lambda x^{\forall X.X \Rightarrow Y}.\lambda y^{Z \Rightarrow Z}.x(Z \Rightarrow Z)y$), but not in F_{at} (as easily seen by a proof-search argument).

▶ Remark 5. The undecidability of the atomic fragment of (full) second-order intuitionistic logic has been known since (at least) [56]. However, from this one cannot deduce the undecidability of F_{at}, due to the fact that disjunction is not faithfully definable in F_{at} (see also [47]).

▶ Remark 6. It is not difficult to see that System F_{at} is equivalent to a first-order system, namely to the $\Rightarrow, \forall$-fragment $\mathsf{1Mon}_{\Rightarrow,\forall}$ of *monadic* first-order intuitionistic logic in the language with no function symbol and a *unique* monadic predicate. The equivalence is given by an obvious bijection between formulas and types given by $\widehat{p(\alpha_i)} = X_i$, $\widehat{\varphi \Rightarrow \psi} = \widehat{\varphi} \Rightarrow \widehat{\psi}$ and $\widehat{\forall \alpha_i.\varphi} = \forall X_i.\widehat{\varphi}$. Hence, a consequence of Corollary 3 is that provability in $\mathsf{1Mon}_{\Rightarrow,\forall}$ is undecidable. Provability in extensions of $\mathsf{1Mon}_{\Rightarrow,\forall}$ with either finitely many monadic predicates, or with disjunction, is known to be undecidable [19, 18]. To the best of our knowledge, the undecidability of $\mathsf{1Mon}_{\Rightarrow,\forall}$ has not been observed before.

4 Typability and Type-checking

In usual implementations of polymorphic type systems the Church-style type discipline is generally considered inconvenient, due to the heavy amount of type annotations. Instead, Curry-style languages, for which a compiler can (either completely or partially) reconstruct type annotations, are generally preferred (two standard examples are the languages ML and Haskell). This is the reason why type-checking algorithms for polymorphic type systems in Curry-style (or in some variants of Curry-style with *partial* type annotations [45]) have been extensively investigated [24, 26, 64, 18].

However, while ML admits a decidable type checking in Curry-style (a main reason for its success), type checking has been shown to be undecidable for System F and most of its variants (including the predicative system F_1 [18]), making the Curry-style version of such systems impractical for implementation.

For the simply typed λ-calculus (and crucially also for ML), the type-checking problem can be reduced to *first-order unification* (FOU), that is, to the problem of unifying first-order terms (in a language with a unique binary function symbol corresponding to $\Rightarrow$). Typically, an application $tu : b$ will produce a first-order equation of the form $a_t = a_u \Rightarrow b$, where a_t, a_u are variables indicating the type of t and the type of u, respectively. As FOU is decidable, this suffices to show that type-checking is decidable in this case.

In the case of full polymorphism FOU is not sufficient to solve type-checking. In fact, already in F_1 one can type terms, like e.g. $\lambda x.xx$, which contain *self-applications*. Using FOU, $\lambda x.xx$ yields the unsolvable equation $a_x = a_x \Rightarrow b$, so it is not typable in either STλC or ML. To type-check System F programs one can replace FOU with either *semi-unification* [24, 26] or *second order unification* (SOU) [45, 18]. Here we focus on the latter: in SOU one tries to unify equations involving terms constructed from first-order variables $a, b, c, \dots$ as well as second order variables $\mathsf{F}, \mathsf{G}, \dots$. For instance, the term $\lambda x.xx$ above yields the equations

$$\mathsf{F}a = (\mathsf{F}b) \Rightarrow \mathsf{G} \tag{1}$$

where $\forall X.\mathsf{F}X$ indicates the type of x, and the variables a, b encode the possible witnesses which permit to type xx (in Church-style one could indicate this with $\lambda x^{\forall X.\mathsf{F}X}.((xa)^{\mathsf{F}a}(xb)^{\mathsf{F}b})^{\mathsf{G}}$, so that Eq. (1) is precisely what is needed to make the typing correct). A (non-unique) solution to Eq. (1) is obtained by $\mathsf{F} \mapsto \lambda x.x$, $\mathsf{G} \mapsto Z$, $a \mapsto Y \Rightarrow Z$, $b \mapsto Y$.

Unfortunately, SOU is undecidable [22]. Moreover, one can encode restricted (but still undecidable) variants of SOU in the type checking problem for F_1 [18], showing that (TC) is undecidable for F_1. A fundamental ingredient of these undecidability arguments is the appeal to *variable cycles* (see the discussion in [36]) like the one in Eq. (1), that is, to unification problems from which one can deduce equations of the form $\mathsf{F}a_1 \dots a_n = u[\mathsf{F}]$, that is, equating a second-order variable F with some term containing F itself.

Conversely, *acyclic* SOU, that is, the problem of unifying SOU problems containing no variable cycles, is decidable [36]. These observations can be used to show that type-checking is actually decidable in F_{at}. In fact, a fundamental property of F_{at} (and a reason for its very limited expressive power) is that any term typable in F_{at} is already typable in the simply-typed λ-calculus. Indeed, the following is easily checked by induction:

▶ **Lemma 7.** *If $\Gamma \vdash t : A$ is derivable in the Curry-style F_{at}, then $|\Gamma| \vdash t : |A|$ is derivable in the simply typed λ-calculus, where $|A|$ is defined by $|X| = o$, $|A \Rightarrow B| = |A| \Rightarrow |B|$, $|\forall X.A| = |A|$, and $|\Gamma|(x) = |\Gamma(x)|$.*

An immediate consequence of Lemma 7 is that one cannot type $\lambda x.xx$ in F_{at} and, more generally, that any λ-term that would give rise to a variable cycle cannot be typed in F_{at}. Observe that the converse does not hold: from the fact that $|\Gamma| \vdash t : |A|$ holds, one cannot deduce $\Gamma \vdash t : A$ (take for instance $t = x$, $\Gamma(x) = X$ and $A = \forall X.X$).

However, these observations suggest that type checking for F_{at} can be decided by reasoning in two phases: to check if $\Gamma \vdash t : A$ is derivable in F_{at}, first check if $|\Gamma| \vdash t : |A|$ is derivable in STλC using FOU; if this first step fails, then the original problem must fail; if the first step succeeds, then the original type-checking problem for F_{at} yields an instance of (a suitable variant of) acyclic SOU, which must be decidable. By reasoning in this way, one can thus establish:

▶ **Theorem 8.** *(TC) for Curry-style* F_{at} *is decidable.*

In App. A (and more in detail in [50]) we describe the decision algorithm for type-checking in F_{at}, which is based on a variant of second-order unification, that we call F_{at}-*unification*. The fundamental idea is to consider SOU problems in a language with first-order *sequence variables* $a, b, \dots$ and two kinds of second-order variables: *projection variables* $\alpha, \beta, \dots$ and *second-order variables* $\mathsf{F}, \mathsf{G}, \dots$. The intuition is that a term of the form $\alpha a_1 \dots a_n$ describes a (skolemized) witness; since the witnesses in F_{at} are type variables, solving for α means associating it with either a constant function or a projection. Instead, a term of the form $\mathsf{F}\mathfrak{a}_1 \dots \mathfrak{a}_n$ stands for the application of suitable witnesses $\mathfrak{a}_1, \dots, \mathfrak{a}_n$ to some type F, hence solving for F means associating it with some function $\lambda X_1 \dots X_n.A(X_1, \dots, X_n)$, where $A(X_1, \dots, X_n)$ is some type expression parametric on the type variable $X_1, \dots, X_n$. Hence, for example, checking if $\Gamma \vdash xy : \forall Z.Z$ holds in F_{at}, where $\Gamma(x) = \forall X.X \Rightarrow X$ and $\Gamma(y) = \forall Y.Y$, yields the equations

$$\mathsf{F}X = X \Rightarrow X \qquad\qquad \mathsf{G}Y = Y$$
$$\mathsf{F}(\alpha Z) = \mathsf{G}(\beta Z) \Rightarrow \mathsf{H}Z \qquad\qquad \mathsf{H}Z = Z$$

which admit the solution $\mathsf{F} \mapsto \lambda X.X \Rightarrow X$, $\mathsf{G}, \mathsf{H} \mapsto \lambda X.X$ and $\alpha, \beta \mapsto \lambda X.X$. Instead, checking if $\Gamma \vdash xy : \forall Z.Z$, where now $\Gamma(x) = \forall X.X \Rightarrow X$ and $\Gamma(y) = Y$, yields the equations

$$\mathsf{F}X = X \Rightarrow X \qquad\qquad \mathsf{G} = Y$$
$$\mathsf{F}(\alpha Z) = \mathsf{G} \Rightarrow \mathsf{H}Z \qquad\qquad \mathsf{H}Z = Z$$

which have no solution (since one can deduce $Z = \mathsf{H}Z = Y$), showing that (TC) fails in this case (although $|\Gamma| \vdash xy : |\forall Z.Z|$ holds in the simply typed λ-calculus).

From the decidability of (TC) one can deduce the decidability of (T) by a standard argument: we can reduce (T) to (TC) by showing that a type A such that $\Gamma \vdash t : A$ holds exists iff $\Gamma \vdash (\lambda xy.y)t : \forall X.X \Rightarrow X$ holds. In fact, if $\Gamma \vdash t : A$ holds in F_{at}, then from $\Gamma \vdash \lambda xy.y : A \Rightarrow \forall X.(X \Rightarrow X)$ we deduce $\Gamma \vdash (\lambda xy.y)t : \forall X.X \Rightarrow X$. Conversely, from $\Gamma \vdash (\lambda xy.y)t : \forall X.X \Rightarrow X$, we deduce that there exists a type A such that $\Gamma \vdash \lambda xy.y : A \Rightarrow (X \Rightarrow X)$ and $\Gamma \vdash t : A$ holds.

▶ **Corollary 9.** *(T) for Curry-style* F_{at} *is decidable.*

5 Equational Reasoning in System F_{at}

As a consequence of Lemma 7 from the previous section, all terms which are typable in Curry-style F_{at} are simply typable. In other words, F_{at} can be seen as a *type refinement* system for STλC, in the sense of [39]. In particular, as we show below, the numerical functions which can be typed in F_{at} are precisely the simply typable ones (i.e. the so-called *extended polynomials* [55, 16]).

For this reason, investigating the typable terms of $\mathrm{F_{at}}$ might seem not very interesting from a computational viewpoint. However, in this section we show that studying such terms can be interesting for equational reasoning. In fact, similarly to System F, standard notions of contextual equivalence for $\mathrm{F_{at}}$ are stronger than $\beta\eta$-equivalence, and one can exploit well-known techniques, like the *free theorems* [63], to compute equivalences of $\mathrm{F_{at}}$-typable terms (which do not hold when viewing these terms as typed in STλC).

We first recall two standard notions of contextual equivalence:

▶ **Notation 10.** *We let* $\mathsf{Bool} = \forall X.X \Rightarrow X \Rightarrow X$ *and* $\mathsf{Nat} = \forall X.(X \Rightarrow X) \Rightarrow (X \Rightarrow X)$*. We let* $\mathbf{t} = \lambda xy.x$ *and* $\mathbf{f} = \lambda xy.y$ *indicate the two normal forms of type* Bool*, and for all* $n \in \mathbb{N}$*, we let* $\mathbf{n} = \lambda fx.(f)^n x$ *indicate the n-th Church numeral.*

▶ **Definition 11** (contextual equivalence). *Let* $F^* \in \{\mathrm{F_{at}}, \mathrm{ML}, \mathrm{F_1}, \mathrm{F}\}$*. For all closed terms* t, u *of type* A *in* F^**, we let*

- $t \simeq^{F^*}_{\mathsf{Bool}} u : A$ *iff for any context* $\mathtt{C}[\ ] : A \vdash \mathsf{Bool}$ *in* F^**,* $\mathtt{C}[t] \simeq_{\beta\eta} \mathtt{C}[u]$*;*
- $t \simeq^{F^*}_{\mathsf{Nat}} u : A$ *iff for any context* $\mathtt{C}[\ ] : A \vdash \mathsf{Nat}$ *in* F^**,* $\mathtt{C}[t] \simeq_{\beta\eta} \mathtt{C}[u]$*.*

It is easily seen that $\simeq^{F^*}_{\mathsf{Bool}}$ and $\simeq^{F^*}_{\mathsf{Nat}}$ are congruences of the terms of F^*. Moreover, in System F these two congruences coincide, due to the fact that the identity relation $\mathtt{id} : \mathsf{Nat} \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Bool}$ is typable. Since this function is also typable in ML, the same holds for ML and $\mathrm{F_1}$. On the other hand, since the identity relation is *not* simply typable, we can deduce (see Lemma 16 below) that it is not typable in $\mathrm{F_{at}}$. For this reason the congruences $\simeq^{\mathrm{F_{at}}}_{\mathsf{Bool}}$ and $\simeq^{\mathrm{F_{at}}}_{\mathsf{Nat}}$ must be treated separately in this case. In what follows we will mostly focus on the latter, since the former identifies distinct normal forms of type Nat, which is not convenient for obvious computational reasons.

▶ Remark 12. The typability of the identity relation $\mathtt{id}$ implies that any extensional model of F must be *infinite*, since for all $n \in \mathbb{N}$, the interpretations of $\mathbf{n}$ and $\mathbf{n+1}$ cannot coincide. Instead, it is not difficult to construct an extensional model of $\mathrm{F_{at}}$ in which any type is interpreted by a *finite* set (to give an idea, let $\mathcal{C}_k$ be a collection of sets of cardinality bounded by a fixed $k \in \mathbb{N}$; one can let then $[\![X]\!] \in \mathcal{C}_k$, $A \Rightarrow B = [\![B]\!]^{[\![A]\!]}$ and $[\![\forall X.A]\!] = \prod_{S \in \mathcal{C}_k} [\![A]\!][X \mapsto S]$).

The so-called free theorems are a class of syntactic equations for typable terms which can be justified by relying on either relational parametricity [53] or dinaturality [3]. We let $t \approx u : A$ indicate that t, u have type A in System F, and that the equivalence $t \simeq u$ can be deduced using β-, η-rules, standard congruence rules (i.e. reflexivity, symmetry, transitivity and context closure), as well as instances of free theorems for System F.

Free theorems can be used to deduce contextual equivalence of $\mathrm{F_{at}}$-terms, thanks to the following:

▶ **Lemma 13** (free theorems in $\mathrm{F_{at}}$). *Let* t, u *be terms of type* A *in* $\mathrm{F_{at}}$*. If* $t \approx u : A$*, where* t, u *are seen as terms of System F, then* $t \simeq^{\mathrm{F_{at}}}_{\mathsf{Nat}} u : A$*.*

Proof. From $t \approx u : A$ it follows $t \simeq^{\mathrm{F}}_{\mathsf{Nat}} u : A$, since $\simeq^{\mathrm{F}}_{\mathsf{Nat}}$ is the coarsest congruence not equating normal forms of type Nat. From $t \simeq^{\mathrm{F}}_{\mathsf{Nat}} u : A$ we deduce $t \simeq^{\mathrm{F_{at}}}_{\mathsf{Nat}} u : A$, since any context in $\mathrm{F_{at}}$ is a context in F. ◀

We discuss below two applications of free theorems to study (CE) in $\mathrm{F_{at}}$.

Categorical Products and Coproducts. As mentioned in Section 2, the usual encoding of products and coproducts in System F preserves β-equivalence but not η-equivalence. For this reason, the encodings of $\times$ and $+$ do not form categorical products and coproducts in System F up to $\beta\eta$-equivalence (more precisely, in the syntactic category in which objects are the types of System F and arrows are the typable terms up to $\simeq_{\beta\eta}$). Instead, it is well-known [51, 23, 61] that η-equivalence of $\times$ and $+$ is preserved in System F up to free theorems: hence $\times$ and $+$ do form categorical products and coproducts in System F up to $\simeq^{\mathrm{F}}_{\mathsf{Nat}}$ (more precisely, in the syntactic category whose arrows are the typable terms up to $\simeq^{\mathrm{F}}_{\mathsf{Nat}}$).

In a similar way, the predicative encodings of $\times$ and $+$ in $\mathrm{F_{at}}$, although preserving some restricted case of η-equivalence, still do not form categorical products and coproducts in $\mathrm{F_{at}}$ up to $\simeq_{\beta\eta}$. We will show that they similarly do form categorical products and coproducts in $\mathrm{F_{at}}$ up to $\simeq^{\mathrm{F_{at}}}_{\mathsf{Nat}}$, as a consequence of the application of free theorems.

For simplicity, we here only consider the case of $+$. However, our argument scales straightforwardly to the encoding of all finite polynomial types, i.e. of all types of the form $\sum_{i=1}^{k}\prod_{j=1}^{k_i} A_{ij}$ (see the [47] for a more detailed discussion).

The fundamental step is showing that the impredicative and predicative encodings are equivalent up to free theorems:

▶ **Lemma 14.** *For all types A, B, C and terms $x \mapsto A \vdash u : C$ and $x \mapsto B \vdash v : C$, the equivalence $\mathsf{IO}^{+}_{C}[y](\lambda x.u)(\lambda x.v) \approx \mathrm{Case}_C(y, x.u, x.v) : C$ holds in System* F.

Proof. The free theorem associated with the type $A\tilde{+}B$ is the schematic equation

$$\mathrm{Case}_E(t_1, x.\mathsf{C}[t_2], x.\mathsf{C}[t_3]) \approx \mathsf{C}\Big[\mathrm{Case}_D(t_1, x.t_2, x.t_3)\Big] \tag{2}$$

where $\vdash t_1 : A\tilde{+}B$, $x \mapsto A \vdash t_2 : D$, $x \mapsto B \vdash t_2 : D$ and $\mathsf{C}[\,] : D \vdash E$. In fact, this equation is an instance of the *dinaturality* condition for the type $A\tilde{+}B$ (see [51, 23, 49]).

We argue by induction on C:

- if $C = Y$, then $\mathsf{IO}^{+}_{C}[y](\lambda x.u)(\lambda x.v) = yY(\lambda x.u)(\lambda x.v) = \mathrm{Case}_C(y, x.u, x.v)$;
- if $C = C_1 \Rightarrow C_2$, then

$$\begin{aligned}
\mathsf{IO}^{+}_{C}[y](\lambda x.u)(\lambda x.v) &= \Big(\lambda fgz.\mathsf{IO}^{+}_{C_2}[y](\lambda x.fxz)(\lambda x.gxz)\Big)(\lambda x.u)(\lambda x.v)\\
&\overset{[\text{I.H.}]}{\approx} \Big(\lambda fgz.\mathrm{Case}_{C_2}(y, x.fxz, x.gxz)\Big)(\lambda x.u)(\lambda x.v)\\
&\simeq_\beta \lambda z.\mathrm{Case}_{C_2}(y, x.uz, x.vz)\\
&\approx \lambda z.\Big(\mathrm{Case}_C(y, x.u, x.v)\Big)z\\
&\simeq_\eta \mathrm{Case}_C(y, x.u, x.v)
\end{aligned}$$

  where in the penultimate step we applied Eq. (2) with the context $\mathsf{C}[\,] = [\,]z : C \vdash C_2$.
- if $C = \forall Z.C'$, then

$$\begin{aligned}
\mathsf{IO}^{+}_{C}[y](\lambda x.u)(\lambda x.v) &= \Big(\lambda fg.\Lambda Z.\mathsf{IO}^{+}_{C'}[y](\lambda x.fxZ)(\lambda x.gxZ)\Big)(\lambda x.u)(\lambda x.v)\\
&\overset{[\text{I.H.}]}{\approx} \Big(\lambda fg.\Lambda Z.\mathrm{Case}_{C'}(y, x.fxZ, x.gxZ)\Big)(\lambda x.u)(\lambda x.v)\\
&\simeq_\beta \Lambda Z.\mathrm{Case}_{C'}(y, x.uZ, x.vZ)\\
&\approx \Lambda Z.\Big(\mathrm{Case}_C(y, x.u, x.v)\Big)Z\\
&\simeq_\eta \mathrm{Case}_C(y, x.u, x.v)
\end{aligned}$$

  where in the penultimate step we applied Eq. (2) with the context $\mathsf{C}[\,] = [\,]Z : C \vdash C'$. ◀

▶ **Proposition 15.** *$A\tilde{+}B$ is a categorical coproduct in F_{at} up to $\simeq^{\mathrm{F}_{\mathrm{at}}}_{\mathsf{Nat}}$.*

Proof. It suffices to check that the η-rule of the coproduct (see [29]) holds in F_{at}. By translating this rule in F one obtains the equation

$$y \approx \mathrm{Case}_{A\tilde{+}B}(y, x.\iota_1(x), x.\iota_2(x)) : A\tilde{+}B$$

which holds in F up to free theorems (see [51, 23, 61]). Using Lemma 14 we thus deduce that $y \approx \mathsf{IO}^+_{A\tilde{+}B}[y](\lambda x.\iota_1(x))(\lambda x.\iota_2(x)) : A\tilde{+}B$ holds in F, and by Lemma 13 we deduce $y \simeq^{\mathrm{F}_{\mathrm{at}}}_{\mathsf{Nat}} \mathsf{IO}^+_{A\tilde{+}B}[y](\lambda x.\iota_1(x))(\lambda x.\iota_2(x)) : A\tilde{+}B$. ◀

Numerical Functions. We now consider the representable numerical functions, that is, the closed typable terms of type $\mathsf{Nat} \Rightarrow \ldots \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat}$. In this case we can strengthen Lemma 7 as follows:

▶ **Lemma 16.** *For any β-normal λ-term t, $\vdash t : \mathsf{Nat} \Rightarrow \ldots \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat}$ holds in Curry-style F_{at} iff $\vdash t : |\mathsf{Nat}| \Rightarrow \ldots \Rightarrow |\mathsf{Nat}| \Rightarrow |\mathsf{Nat}|$ holds in STλC.*

Proof. One direction follows from Lemma 7. For the converse one, let t (which we can suppose w.l.o.g. to be of the form $\lambda x_1 \ldots x_k.u$) be such that $\vdash t : |\mathsf{Nat}| \Rightarrow \ldots \Rightarrow |\mathsf{Nat}| \Rightarrow |\mathsf{Nat}|$. By letting $\mathsf{Nat}[X] = (X \Rightarrow X) \Rightarrow (X \Rightarrow X)$ we deduce that $\{x_i \mapsto \mathsf{Nat}[X]\} \vdash u : \mathsf{Nat}[X]$ holds in F_{at}, and thus that $\{x_i \mapsto \mathsf{Nat}\} \vdash u : \mathsf{Nat}[X]$ holds too, from which we conclude $\vdash u : \mathsf{Nat} \Rightarrow \ldots \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat}$. ◀

A consequence of Lemma 16 is that the representable numerical functions in F_{at} are precisely the extended polynomials, i.e. the smallest class of functions arising from projections, constant functions, addition, multiplication and the `iszero` function. Instead, it is well-known that the predecessor function (which is not an extended polynomial) is typable in ML [17] and, more generally, the representable functions of ML are included in the class $\mathcal{E}_3$ of the Grzegorczyk hierarchy [33].

Still, in both STλC and F_{at} the same extended polynomial can be represented by different normal forms. For instance the two normal forms $\lambda xyfz.x(yf)z$ and $\lambda xyfz.y(xf)z$ (encoding the algorithms $n, m \mapsto \underbrace{m + \cdots + m}_{n \text{ times}}$ and $n, m \mapsto \underbrace{n + \cdots + n}_{m \text{ times}}$) both represent the multiplication function.

In System F, one can show that all primitive recursive functions are *uniquely defined* up to free theorems, that is, that for any two terms t, u representing the same primitive recursive function, one can prove $t \approx u$ (see [48], Section 7.5). Using Lemma 13 we deduce then:

▶ **Lemma 17.** *For all $t, u : \mathsf{Nat} \Rightarrow \ldots \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat}$ in $\mathrm{F}^* \in \{\mathrm{F}_{\mathrm{at}}, \mathrm{ML}, \mathrm{F}_1, \mathrm{F}\}$, if for all $p_1, \ldots, p_k \in \mathbb{N}$, $t\mathbf{p}_1 \ldots \mathbf{p}_k \simeq_{\beta\eta} u\mathbf{p}_1 \ldots \mathbf{p}_k : \mathsf{Nat}$, then $t \simeq^{\mathrm{F}^*}_{\mathsf{Nat}} u$.*

▶ Remark 18. From Lemma 17 and the fact that all primitive recursive functions are typable in F, one can deduce that $\simeq^{\mathrm{F}}_{\mathsf{Nat}}$ for numerical functions is undecidable in F as a consequence of Rice's theorem.

The problem $\mathrm{Eq}_{\mathcal{C}}$ of deciding $f = g$, where f, g belong to some subclass $\mathcal{C}$ of the primitive recursive functions, is well-investigated. In particular, it is known that:

- if $\mathcal{C}$ is the class of extended polynomials, then $\mathrm{Eq}_{\mathcal{C}}$ is decidable [38];
- if $\mathcal{C}$ contains projections, constants, $+$, $\times$ and bounded multiplication, then $\mathrm{Eq}_{\mathcal{C}}$ is undecidable [31].

From these facts, using Lemma 17, we deduce then:

▶ **Proposition 19.**
(i) *The problem of deciding* $\simeq^{\mathrm{F}_{\mathrm{at}}}_{\mathsf{Nat}}$ *over numerical functions in* F_{at} *is decidable.*
(ii) *The problem of deciding* $\simeq^{\mathrm{F}^*}_{\mathsf{Nat}}$ *over numerical functions in* $\mathrm{F}^* \in \{\mathrm{ML}, \mathrm{F}_1\}$ *is undecidable.*

Proof. (i) is immediate from Lemma 16 and Lemma 17. To prove (ii) it suffices to show that the representable functions in ML are closed under bounded multiplication. We show this fact in detail in [50], App. B. ◀

An immediate corollary is that (CE) is undecidable in both ML and F_1.

6 Contextual Equivalence is Undecidable

In this section we show that the congruences $\simeq^{\mathrm{F}_{\mathrm{at}}}_{\mathsf{Nat}}$ and $\simeq^{\mathrm{F}_{\mathrm{at}}}_{\mathsf{Bool}}$ are both undecidable. To do this, we will reduce the type inhabitation problem for a suitable extension of F_{at} to contextual equivalence. We discuss in some detail the undecidability argument for $\simeq^{\mathrm{F}_{\mathrm{at}}}_{\mathsf{Bool}}$, while the (very similar) argument for $\simeq^{\mathrm{F}_{\mathrm{at}}}_{\mathsf{Nat}}$ can be found in [50], App. C.

Let $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$ be the extension of F_{at} with new a type constant $\clubsuit$ and a new term constant $\star : \clubsuit$. It is not difficult to see that the undecidability argument for (TI) from Section 3 also applies to $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$.

Let $\tilde{\top} : \forall X.X \Rightarrow X$ and $\mathsf{Id} := \Lambda X.\lambda x.x$ be the unique closed β-normal term of type $\tilde{\top}$.

The fundamental idea will be to construct, for each type A of $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$, two terms t_A, u_A of type $(A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool}$ (where $A^* = Y \Rightarrow A[Y/\clubsuit]$, for some fresh Y), such that $t_A \simeq^{\mathrm{F}_{\mathrm{at}}}_{\mathsf{Bool}} u_A$ holds in F_{at} iff A is inhabited in $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$.

Let us fix a type A of $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$, a variable Y not occurring free in A, and let $A^* = Y \Rightarrow A[Y/\clubsuit]$. We let u_A, v_A be the terms below:

$$u_A = \lambda x.\mathbf{f} \qquad v_A = \lambda x.\mathsf{IO}^{+}_{\mathsf{Bool}}[\ ](\lambda x.\mathbf{t})(\lambda x.\mathbf{f})$$

First observe that if there exists some term t such that $\vdash t : A$ holds in $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$, then we can construct a context $\mathtt{K}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash \mathsf{Bool}$ separating u_A and v_A: let $t^* = \lambda y.t[y/\star]$, so that $\vdash t^* : A^*$ and let $\mathtt{K}[\ ] = [\ ](\iota_1(t^*))$. We then have $\mathtt{K}[u_A] \simeq_\beta \mathbf{f}$ and $\mathtt{K}[v_A] \simeq_\beta \mathsf{IO}^{+}_{\mathsf{Bool}}[\iota_1(t^*)](\lambda x.\mathbf{t})(\lambda x.\mathbf{f}) \simeq_{\beta\eta} (\lambda x.\mathbf{t})t^* \simeq_\beta \mathbf{t}$.

The difficult part is to show that if A is not provable in $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$, then no context $\mathtt{K}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash \mathsf{Bool}$ can separate u_A and v_A. We will establish this fact by analyzing all possible β-normal term contexts of type $(A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash \mathsf{Bool}$.

In the following, for a term context $\mathtt{K}[\ ]$, we let $\mathtt{K}[\ ] : A \vdash^{\Gamma} B$ be a shorthand for $\Gamma, x \mapsto A \vdash \mathtt{K}[\ ] : B$ (where we suppose that Γ is not defined on x).

We let $\mathbb{G}_1$-$\mathbb{G}_4$ be the families of term contexts defined by mutual recursion as shown in Fig. 3, and typed according to the contexts below

$$\begin{aligned} &\Gamma = \{x_1 \mapsto Z_1, x'_1 \mapsto Z_1, \dots, x_p \mapsto Z_p, x'_p \mapsto Z_p\} && \Theta = \{w_1 \mapsto W_1, \dots, w_q \mapsto W_q\} \\ &\Delta = \{y_1 \mapsto A^* \Rightarrow Y_1, \dots, y_r \mapsto A^* \Rightarrow Y_r\} && \Sigma = \{z_1 \mapsto \tilde{\top} \Rightarrow Y_1, \dots, z_r \mapsto \tilde{\top} \Rightarrow Y_r\} \end{aligned} \tag{3}$$

for some $p, q, r \in \mathbb{N}$ and variables $Z_1, \dots, Z_p, W_1, \dots, W_q, Y_1, \dots, Y_r$ pairwise distinct and disjoint from A.

It can be checked that none of these contexts can separate u_A and v_A (see [50]):

▶ **Lemma 20.**
1. *For all* $\mathtt{C}[\ ] \in \mathbb{G}_1$, $\mathtt{C}[u_A] \simeq_{\beta\eta} \mathtt{C}[v_A]$.
2. *If* $\mathtt{D}[\ ] \in \mathbb{G}_2$, *then* $\mathtt{D}[u_A] \simeq_{\beta\eta} \mathtt{D}[v_A] \simeq_{\beta\eta} z_i\mathsf{Id}$.
3. *If* $\mathtt{E}[\ ] \in \mathbb{G}_3$, *then* $\mathtt{E}[u_A] \simeq_{\beta\eta} \mathtt{E}[v_A]$.
4. *If* $\mathtt{F}[\ ] \in \mathbb{G}_4$, *then* $\mathtt{F}[u_A] \simeq_{\beta\eta} \mathtt{F}[v_A] \simeq_{\beta\eta} w_i$.

$\mathbb{G}_1$:	$\mathtt{C}[\ ] ::= x_i \mid x_i' \mid \mathtt{E}[\ ]Z_j\mathtt{C}[\ ]\mathtt{C}[\ ]$	$: (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} Z_j$
$\mathbb{G}_2$:	$\mathtt{D}[\ ] ::= z_i(\Lambda W.\lambda w.\mathtt{F}[\ ]) \mid \mathtt{E}[\ ]Y_i\mathtt{D}[\ ]\mathtt{D}[\ ]$	$: (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} Y_i$
$\mathbb{G}_3$:	$\mathtt{E}[\ ] ::= \mathbf{t} \mid \mathbf{f} \mid [\ ](\Lambda Y.\lambda y.\lambda z.\mathtt{D}[\ ])$	$: (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} \mathsf{Bool}$
$\mathbb{G}_4$:	$\mathtt{F}[\ ] ::= w \mid \mathtt{E}[\ ]W_i\mathtt{F}[\ ]\mathtt{F}[\ ]$	$: (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} W_i$

Figure 3 Contexts $\mathbb{G}_1$-$\mathbb{G}_4$.

The key ingredient is a lemma stating that, when A is not inhabited in $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$, the families of contexts $\mathbb{G}_1$-$\mathbb{G}_4$ can be used to generate all possible term contexts.

▶ **Lemma 21.** *Let* $\mathtt{K}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{x_1 \mapsto Z, x_1' \mapsto Z} Z$ *be a β-normal term context. If A is not inhabited in* $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$*, then* $\mathtt{K}[\ ] \in \mathbb{G}_1$*.*

Proof. We will prove the following claim: either there exists contexts $\Gamma, \Theta, \Delta, \Sigma$ as in Eq. (3), for some $p, q, r \in \mathbb{N}$ and variables $Z_1, \dots, Z_p, W_1, \dots, W_q, Y_1, \dots, Y_r$ pairwise distinct and disjoint from A, and a context $\mathtt{H}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} A^*$, or $\mathtt{K}[\ ] \in \mathbb{G}_1$. If the main claim is true we can deduce the statement of the lemma as follows: suppose $\mathtt{K}[\ ] \notin \mathbb{G}_1$; then let θ be the substitution sending all variables in $\Gamma, \Theta, \Delta, \Sigma$ plus Y onto $\clubsuit$ and being the identity on all other variables. Then $\mathtt{H}\theta[\ ] : ((\clubsuit \Rightarrow A) \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma\theta,\Theta\theta,\Delta\theta,\Sigma\theta} : \clubsuit \Rightarrow A$. Then we have $\Gamma\theta, \Theta\theta, \Delta\theta, \Sigma\theta \vdash t : A$, where $t = \mathtt{H}\theta[\lambda x.\mathbf{t}]\star$ and we can conclude that $\vdash t' : A$ holds, where t' is obtained from t by substituting the variables in Γ and Θ by $\star$ and those in Δ and Σ by $\lambda x.\star$.

Let us prove the main claim. Suppose by contradiction that for no $\Gamma, \Theta, \Delta, \Sigma$ there exists a context $\mathtt{H}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} A^*$. We will show by simultaneous induction the following claims:

1. for all $\Gamma, \Theta, \Delta, \Sigma$ as above, if $\mathtt{K}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} Z_i$, then $\mathtt{K}[\ ] \in \mathbb{G}_1$;
2. for all $\Gamma, \Theta, \Delta, \Sigma$ as above, if $\mathtt{K}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} Y_i$, then $\mathtt{K}[\ ] \in \mathbb{G}_2$;
3. for all $\Gamma, \Theta, \Delta, \Sigma$ as above, if $\mathtt{K}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} \mathsf{Bool}$ and $\mathtt{K}[\ ]$ is an elimination context, then $\mathtt{K}[\ ] \in \mathbb{G}_3$;
4. for all $\Gamma, \Theta, \Delta, \Sigma$ as above, if $\mathtt{K}[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} W_i$, then $\mathtt{K}[\ ] \in \mathbb{G}_4$.

The main claim then follows from 1. by taking $\Gamma = \{x \mapsto Z, x' \mapsto Z\}$ and $\Theta = \Delta = \Sigma = \varnothing$.

We argue for each case separately:

1. There exist two possibilities for $\mathtt{K}[\ ]$:
   a. $\mathtt{K}[\ ] = x_i$ (resp. $= x_i'$), hence $\mathtt{K}[\ ] \in \mathbb{G}_1$;
   b. $\mathtt{K}[\ ] = \mathtt{K}'[\ ]Z\mathtt{K}_1[\ ]\mathtt{K}_2[\ ]$, where $\mathtt{K}'[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} \mathsf{Bool}$ and $\mathtt{K}_i[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} Z$, and where $\mathtt{K}'[\ ]$ is an elimination context. By the induction hypothesis then $\mathtt{K}'[\ ] \in \mathbb{G}_3, \mathtt{K}_i[\ ] \in \mathbb{G}_1$, hence $\mathtt{K}[\ ] \in \mathbb{G}_1$.
2. There exist three possibilities for $\mathtt{D}[\ ]$:
   a. $\mathtt{K}[\ ] = y_i\mathtt{K}'[\ ]$, where $\mathtt{K}'[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} A^*$, but this case is excluded by the hypothesis;
   b. $\mathtt{K}[\ ] = z_i(\Lambda W.\lambda w.\mathtt{K}'[\ ])$, where $\mathtt{K}'[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta\cup\{w\mapsto W\},\Delta,\Sigma} W$ and where W does not occur in $\Gamma, \Theta, \Delta, \Sigma$. By the induction hypothesis then $\mathtt{K}'[\ ] \in \mathbb{G}_4$, hence $\mathtt{K}[\ ] \in \mathbb{G}_2$;
   c. $\mathtt{K}[\ ] = \mathtt{K}'[\ ]Y_i\mathtt{K}_1[\ ]\mathtt{K}_2[\ ]$, where $\mathtt{K}'[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} \mathsf{Bool}$, $\mathtt{K}_i[\ ] : (A^* \tilde{+} \tilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} Y_i$, and $\mathtt{K}'[\ ]$ is an elimination context. By the induction hypothesis this implies $\mathtt{K}'[\ ] \in \mathbb{G}_3$ and $\mathtt{K}_i \in \mathbb{G}_2$, so we can conclude $\mathtt{K}[\ ] \in \mathbb{G}_2$.

3. If $\mathtt{K}[\,]$ is an elimination context, then it must be $\mathtt{K}[\,] = x\mathtt{K}'[\,]$, where $\mathtt{K}'[\,] : (A^*\widetilde{+}\widetilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma\cup\{x_1\mapsto Z', x_2\mapsto Z''\},\Theta,\Delta,\Sigma} A^*\widetilde{+}\widetilde{\top}$. Moreover, $\mathtt{K}'$ must be of the form $\Lambda Y.\lambda y.\lambda z.\mathtt{K}''[\,]$, where $\mathtt{K}''[\,] : (A^*\widetilde{+}\widetilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma\cup\{x_1\mapsto Z', x_2\mapsto Z''\},\Theta,\Delta\cup\{y\mapsto A^*\Rightarrow Y\},\Sigma\cup\{z\mapsto\widetilde{\top}\Rightarrow Y\}} Y$, and where Y is distinct from all variables in $\Gamma \cup \{x_1 \mapsto Z', x_2 \mapsto Z''\}, \Theta, \Delta, \Sigma$; then by the induction hypothesis we deduce $\mathtt{K}''[\,] \in \mathbb{G}_2$, and thus $\mathtt{K}[\,] \in \mathbb{G}_3$.
4. There are two possible cases:
   a. $\mathtt{K}[\,] = w_i$, hence $\mathtt{K}[\,] \in \mathbb{G}_4$;
   b. $\mathtt{K}[\,] = \mathtt{K}'[\,]W_i\mathtt{K}_1\mathtt{K}_2$, where $\mathtt{K}'[\,] : (A^*\widetilde{+}\widetilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} \mathsf{Bool}$, $\mathtt{K}_i[\,] : (A^*\widetilde{+}\widetilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{\Gamma,\Theta,\Delta,\Sigma} W_i$ and $\mathtt{K}'[\,]$ is an elimination context. By the induction hypothesis this implies $\mathtt{K}'[\,] \in \mathbb{G}_3$ and $\mathtt{K}_i \in \mathbb{G}_4$, whence $\mathtt{K}[\,] \in \mathbb{G}_4$. ◀

▶ **Proposition 22.** *$u_A \not\simeq^{\mathrm{F_{at}}}_{\mathsf{Bool}} v_A$ iff A is inhabited in $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$.*

Proof. We only need to show one side of the statement: suppose A is not inhabited in $\mathrm{F}^{\clubsuit}_{\mathrm{at}}$. Any context $\mathtt{K}[\,] : (A^*\widetilde{+}\widetilde{\top}) \Rightarrow \mathsf{Bool} \vdash \mathsf{Bool}$ can be written, up to η-equivalence, as $\mathtt{K}[\,] = \Lambda Z.\lambda x_1 x_2.\mathtt{K}'[\,]$, with $\mathtt{K}'[\,] : (A^*\widetilde{+}\widetilde{\top}) \Rightarrow \mathsf{Bool} \vdash^{x_1\mapsto Z, x_2\mapsto Z} \mathsf{Bool}$. As we can suppose $\mathtt{K}[\,]$ to be β-normal, by Lemma 21, it must be $\mathtt{K}'[\,] \in \mathbb{G}_1$. Hence, by Lemma 20 we deduce that $\mathtt{K}[u_A] \simeq_{\beta\eta} \mathtt{K}[v_A]$. ◀

▶ **Theorem 23.** *The congruences $\simeq^{\mathrm{F_{at}}}_{\mathsf{Bool}}$ and $\simeq^{\mathrm{F_{at}}}_{\mathsf{Nat}}$ are both undecidable.*

7 Conclusion

Related works. The literature on ML-polymorphism, both at theoretical and applicative level, is vast. Several extensions of ML to account for first-class polymorphism while retaining a decidable type-checking have been investigated, mostly following two directions: first, that of considering type systems with explicit type annotations (as the system PolyML [20]); second, that of encoding first-class polymorphism in a ML-style system by means of *coercions* (as in System Fc [60] or in ML^{F} [30]). In the last case, coherently with our discussion of FOU and SOU, the price to pay to remain decidable is that self-applications of λ-abstracted variables must come with explicit type annotations. This approach is currently followed in the design of the Haskell compiler, which supports first-class polymorphism.

Predicative restrictions of System F and their expressive power have been also largely investigated [32, 33, 6]. For example, the numerical functions representable in Leivant's finitely stratified polymorphism are precisely those at the third level of Grzegorczyk's hierarchy [33], and transfinitely stratified systems have been shown to represent all primitive recursive functions [6]. In [34] a system with expressive power comparable to System $\mathrm{F_{at}}$ is shown to characterize the polytime functions.

Research by Ferreira and her collaborators on System $\mathrm{F_{at}}$ has mostly focused on predicative translations of intuitionistic logic and their reduction properties [12, 11, 10]. As mentioned before, these translations rely on the observation that for certain types the unrestricted $\forall$E-rule is admissible in $\mathrm{F_{at}}$. The characterization of the class of types satisfying this property is an open problem (a partial characterization is described in [46]).

Another way to obtain interesting subsystems of System F is by restricting the class of types which can be universally quantified (instead of the admissible witnesses). For instance, the system in [2] forbids quantifier nestings, while the system in [35] only allows quantification $\forall X.A$ when X occurs at depth at most 2 in A (i.e. when X occurs at most twice to the left of an implication). Interestingly, both systems have the expressive power of Gödel's System T (which is not a first-order system).

Another kind of restrictions on the shape of types have been investigated by the authors in [49], motivated by ideas from the categorical semantics of polymorphism [3]. The two resulting fragments $\Lambda 2^{\kappa\leqslant 0}, \Lambda 2^{\kappa\leqslant 1}$ are equivalent, respectively, to the simply typed λ-calculus with finite sums and products, and to its extension with least and greatest fixpoints (in particular, (CE) is decidable in $\Lambda 2^{\kappa\leqslant 0}$).

Finally, polymorphism in *linear* type systems has been investigated too. Interestingly, (TI) [28, 27] and (CE) [43] remain undecidable even in this case.

Future work. The main interest we found in investigating $\mathrm{F_{at}}$ was to shed some new light on the source of undecidability of type-related properties for full System F. Yet, one might well ask whether the decidability of type-checking makes $\mathrm{F_{at}}$ a reasonable candidate for implementations. Admittedly, our decision algorithm, which was only oriented to prove decidability, is not very practical: checking failure is coNP with respect to the number of type symbols. Yet, it does not seems unlikely that more optimized algorithms can be developed.

By the way, given that the terms typable in $\mathrm{F_{at}}$ are simply typable, would an implementation of atomic polymorphism be interesting at all? In contrast with ML, type-checking atomically polymorphic programs is decidable at any rank. One could thus investigate extensions of ML with first class atomic polymorphism (realistically, in presence of other type constructors like e.g. some restricted version of dependent types, see [65]).

A more interesting direction, suggested by our decision algorithm, would be to investigate systems with full, impredicative, polymorphism, but obeying some condition ensuring *acyclicity*, so that TC (based on SOU) remains decidable. One would thus retain some advantages of first-class polymorphism (e.g. the modularity/genericity of programs) while admitting self-applications only in "ML-style" (or with explicit type annotations, as in $\mathrm{ML^F}$ [30]). For instance, a way to ensure acycliclity might be to require that a polymorphic λ-abstracted variable be used in an *affine* way, i.e. at most once.

References

1 Amal Ahmed, Dustin Jamner, Jeremy G. Siek, and Philip Wadler. Theorems for free for free: parametricity, with and without types. In *Proceedings of the ACM on Programming Languages*, volume 1 of *ICFP*, page Article No. 39, New York, 2017.

2 Thorsten Altenkirch and Thierry Coquand. A finitary subsystem of the polymorphic λ-calculus. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications*, pages 22–28, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

3 E.S. Bainbridge, Peter J. Freyd, Andre Scedrov, and Philip J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990.

4 Henk Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.

5 Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda calculus with types.* Perspectives in Logic. Cambridge University Press, 2013.

6 Norman Danner and Daniel Leivant. Stratified polymorphism and primitive recursion. *Mathematical Structures in Computer Science*, 9(4):507–522, 1999.

7 Kosta Došen and Zoran Petrić. The typed Böhm theorem. *Electronic Notes in Theoretical Computer Science*, 50(2):117–129, 2001.

8 Andrej Dudenhefner and Jakob Rehof. A simpler undecidability proof for system F inhabitation. In *TYPES 2018*, pages 2:1–2:11, 2018.

9 José Espírito Santo and Gilda Ferreira. A refined interpretation of intuitionistic logic by means of atomic polymorphism. *Studia Logica*, 108(3):477–507, 2020. doi:10.1007/s11225-019-09858-1.

10 José Espírito Santo and Gilda Ferreira. The Russell-Prawitz embedding and the atomization of universal instantiation. *Logic Journal of the IGPL*, July 2020. jzaa025.

11 Fernando Ferreira and Gilda Ferreira. Commuting conversions vs. the standard conversions of the "good" connectives. *Studia Logica*, 92(1):63–84, 2009.

12 Fernando Ferreira and Gilda Ferreira. Atomic polymorphism. *Journal of Symbolic Logic*, 78(1):260–274, 2013.

13 Fernando Ferreira and Gilda Ferreira. The faithfulness of Fat: a proof-theoretic proof. *Studia Logica*, 103(6):1303–1311, 2015.

14 Gilda Ferreira. η-conversions of IPC implemented in atomic F. *Logic Journal of the IGPL*, 25(2):115–130, June 2016.

15 Gilda Ferreira and Bruno Dinis. Instantiation overflow. *Reports on Mathematical Logic*, 51:15–33, 2016.

16 Gilda Ferreira and Vasco T Vasconcelos. The computational content of atomic polymorphism. *Logic Journal of the IGPL*, 27(5):625–638, December 2018.

17 Steven Fortune, Daniel Leivant, and Michael O'Donnell. The expressiveness of simple and second-order type structures. *Journal of the ACM*, 30(1):151–185, 1983.

18 Ken-etsu Fujita and Aleksy Schubert. The undecidability of type related problems in the type-free style system F with finitely stratified polymorphic types. *Information and Computation*, 218:69–87, 2012.

19 Dov M. Gabbay. *Semantical Investigations in Heyting's Intuitionistic Logic*, volume 148. Springer Science + Business, Dordrecht, 1981.

20 Jacques Garrigue and Didier Rémy. Extending ml with semi-explicit higher-order polymorphism. In Martín Abadi and Takayasu Ito, editors, *Theoretical Aspects of Computer Software*, pages 20–46, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

21 Paola Giannini and Simona Ronchi Della Rocca. Characterization of typings in polymorphic type discipline. In *Proceedings of the 3-th Annual IEEE Symposium on Logic in Computer Science*, pages 61–70, Edinburgh, 1988.

22 Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

23 Ryu Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science*, 4(1):71–109, 2009.

24 Fritz Henglein. *Polymorphic type inference and semi-unification.* PhD thesis, The State University of New Jersey, 1989.

25 Roger J. Hindley. The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1069.

26 Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. The undecidability of the semi-unification problem. *Information and Computation*, 102(1):83–101, 1993.

27 Yves Lafont. The undecidability of second order linear logic without exponentials. *The Journal of Symbolic Logic*, 61(02):541–548, 1996. `doi:10.2307/2275674`.

28 Yves Lafont and Andre Scedrov. The Undecidability of Second Order Multiplicative Linear Logic. *Information and Computation*, 125(1):46–51, 1996. `doi:10.1006/inco.1996.0019`.

29 Joachim Lambek and Philip J. Scott. *Introduction to higher order categorical logic.* Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1988.

30 Didier Le Botlan and Didier Rémy. Mlf: Raising ml to the power of system f. In *Proc. of the International Conference on Functional Programming (ICFP '03)*, pages 27–38, 2003.

31 R. D. Lee. *Decidable classes of recursive equations.* PhD thesis, University of Leicester, 1969.

32 Daniel Leivant. Stratified polymorphism. In *LICS '89. Proceedings of the 4th Annual Symposium on Logic in Computer Science*. IEEE, 1989.

33 Daniel Leivant. Finitely stratified polymorphism. *Information and Computation*, 93(1):93–113, 1991.

34 Daniel Leivant. A foundational delineation of Poly-time. *Information and Computation*, 110(2):391–420, 1994.

35 Daniel Leivant. Peano's lambda calculus: the functional abstraction implicit in arithmetic. In *Logic, meaning and computation, Essays in Memory of Alonzo Church*, volume 305 of *Synthese Library, Studies in Epistemology, Logic, Methodology and Philosophy of Science*, pages 313–329. Springer Netherlands, 2001.

36 Jordi Levy. Decidable and undecidable second-order unification problems. In Tobias Nipkow, editor, *Rewriting Techniques and Applications*, pages 47–60, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

37 M. H. Löb. Embedding first order predicate logic in fragments of intuitionistic logic. *Journal of Symbolic Logic*, 41:705–718, 1976.

38 Jan Małolepszy, Małgorzata Moczurad, and Marek Zaionc. Schwichtenberg-style lambda definability is undecidable. In Philippe de Groote and J. Roger Hindley, editors, *Typed Lambda Calculi and Applications*, pages 267–283, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

39 Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 3–16, New York, NY, USA, 2015. Association for Computing Machinery.

40 R. Milner and L. Damas. The principal type schemes for functional programs. In *Symposium on Principles of Programming Languages, ACM*, 1982.

41 Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Science*, 17(3):248–375, 1978.

42 Robin Milner. The standard ml core language. *Polymorphism*, 2(2), 1985.

43 Le Than Dung Nguyen, Paolo Pistone, Thomas Seiller, and Lorenzo Tortora de Falco. Finite semantics of polymorphism, complexity and the expressive power of type fixpoints, 2019. URL: `https://hal.archives-ouvertes.fr/hal-01979009`.

44 Vincent Padovani. *Filtrage d'ordre supérieur*. PhD thesis, Université Paris 7, 1996.

45 Frank Pfenning. On the undecidability of partial polymorphic type reconstruction. *Fundamenta Informaticae*, 19(1-2):185–199, 1993.

46 Paolo Pistone. Proof nets and the instantiation overflow property, 2018. `arXiv:1803.09297`.

47 Paolo Pistone and Luca Tranchini. The naturality of natural deduction II. some remarks on atomic polymorphism, 2020. `arXiv:1908.11353`.

48 Paolo Pistone and Luca Tranchini. The Yoneda Reduction of Polymorphic Types (extended version), 2020. `arXiv:1907.03481`.

49 Paolo Pistone and Luca Tranchini. The Yoneda Reduction of Polymorphic Types. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:22, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

50 Paolo Pistone and Luca Tranchini. What's decidable about (atomic) polymorphism?, 2021. Available at `arXiv:2105.00748`.

51 Gordon Plotkin and Martin Abadi. A logic for parametric polymorphism. In *TLCA '93, International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer Berlin Heidelberg, 1993.

52 M Clarence Protin. Type inhabitation of atomic polymorphism is undecidable. *Journal of Logic and Computation*, January 2021. exaa090.

53 John C. Reynolds. Types, abstraction and parametric polymorphism. In R.E.A. Mason, editor, *Information Processing '83*, pages 513–523. North-Holland, 1983.

54 Aleksy Schubert, Paweł Urzyczyn, and Konrad Zdanowski. On the Mints hierarchy in first-order intuitionistic logic. *Logical Methods in Computer Science*, 12(4:11):1–25, 2016.

55 Helmut Schwichtenberg. Definierbare funktionen im λ-kalkül mit typen. *Archiv für mathematische Logik und Grundlagenforschung*, 17(3):113–114, 1975.

56 S. K. Sobolev. The intuitionistic propositional calculus with quantifiers. *Mathematical Notes of the Academy of Sciences of the USSR*, 22(528-532), 1977.

57 Morten Heine Sorensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149 of *Studies in logic and the foundations of mathematics*. Elsevier Science, 2006.

58 R. Statman. Completeness, invariance and λ-definability. *The Journal of Symbolic Logic*, 47(1):17–26, 1982.

59 Richard Statman. intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9(1):67–72, 1979.

60 Marin Sulzmann, Manuel M. T. Chakravarty, Simon Peyton Jones, and Kevin Donnelly. System f with type equality coercions. In *TLDI '07 Proceedings of the 2007 ACM SIGPLAN Internatinal workshop in Types in languages design and implementation*, pages 53–66. ACM New York, 2007.

61 Luca Tranchini, Paolo Pistone, and Mattia Petrolo. The naturality of natural deduction. *Studia Logica*, `https://doi.org/10.1007/s11225-017-9772-6`, 2017.

62 J. Voigtländer. Proving correctness via free theorems: the caser of the destroy/build-rule. In *Proceedings of the ACM SIGPLAN symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 13–20, New York, 2008. ACM press.

63 Philip Wadler. Theorems for free! In *Proceedings of the fourth international conference on functinoal programming languages and computer architecture - FPCA '89*, 1989.

64 J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98:111–156, 1998.

65 Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 214–227, New York, NY, USA, 1999. Association for Computing Machinery.

A $\mathrm{F_{at}}$-unification

In this section we describe a decidable unification problem, that we call $\mathrm{F_{at}}$-unification, and we show that this problem captures type-checking for $\mathrm{F_{at}}$.

A decidable second-order unification problem. We consider a second-order language composed of three different sorts of variables: *sequence variables* $a, b, c, \dots$, *projection variables* $\alpha^n, \beta^n, \gamma^n, \dots$ and *second-order variables* $\mathsf{F}^n, \mathsf{G}^n, \dots$ (where in the last two cases n indicates the arity of the variable). The language includes expressions of three sorts, noted $\langle * \rangle$, $*$ and $T(*)$; the expressions of each type are defined by the grammars below:

$$\begin{aligned} \mathfrak{a}, \mathfrak{b}, \mathfrak{c} &::= \langle X_1 \dots X_n \rangle \mid a \mid \alpha^n a_1 \dots a_n && (\text{sort } \langle * \rangle) \\ \phi, \psi &::= X \mid \pi^l(\mathfrak{a}) \mid \mathsf{F}^n \mathfrak{a}_1 \dots \mathfrak{a}_n \mid \Phi \Rightarrow \Psi && (\text{sort } *) \\ \Phi, \Psi &::= \forall a.\phi && (\text{sort } T(*)) \end{aligned}$$

A $\mathrm{F_{at}}$*-unification problem* is a pair (U, E), where U is a set of equations of the form $\phi = \psi$ between expressions of type $*$, and E is a set of *constraints* of the form $(\alpha : a)$ or $(a : k)$, where $k \in \mathbb{N}$.

Given a $\mathrm{F_{at}}$-unification problem (U, E), for all projection variable α^n occurring in U, let $\deg(\alpha)$ indicate the maximum l such that $\pi^l(\alpha^n a_1 \dots a_n)$ occurs in U.

A *substitution* for a $\mathrm{F_{at}}$-unification problem (U, E) is given by the following data:

- for each sequence variable a, a natural number $k^S_a \in \mathbb{N}$;
- for each projection variable α^n, a pair $(k^S_\alpha, S(\alpha))$ made of a natural number $k^S_\alpha \geqslant \deg(\alpha)$ and a sequence $S(\alpha) = \langle S(\alpha)_1, \dots, S(\alpha)_{k^S_\alpha} \rangle$, where $S(\alpha)_i$ is either of the form $\lambda x_1. \dots .x_n.X$ or of the form $\lambda x_1. \dots .x_n.\pi^l(x_j)$, where l is such that, whenever $\alpha^n a_1 \dots a_n$ occurs in U, $l \leqslant k^S_{a_j}$;

- for each second-order variable F^n, a function $S(\mathsf{F})$ of the form $\lambda\rho_1.\dots.\rho_n.A(\rho_1,\dots,\rho_n)$, where $A(\rho_1,\dots,\rho_n)$ is given by the grammar

  $$A, B ::= X \mid \pi^l(\rho_i) \mid A \Rightarrow B \mid \forall X.A$$

  with $i \in \{1,\dots,n\}$ and l being such that, if $\mathsf{F}^n\mathfrak{a}_1\dots\mathfrak{a}_n$ occurs in U, then $l \leqslant k^S_{\mathfrak{a}_i}$ (where $k^S_{\mathfrak{a}}$ is k if $\mathfrak{a} = \langle X_1,\dots,X_k\rangle$, is k^S_a if $\mathfrak{a} = a$, and is k^S_α if $\mathfrak{a} = \alpha^r a_1 \dots a_r$).

Given a substitution S, we define (1) for any expression $\mathfrak{a}$ of sort $\langle *\rangle$, a sequence $S(\mathfrak{a})$ of type variables, (2) for any expression ϕ of sort $*$, a type $S(\phi)$, and (3) for any expression Φ of sort $T(*)$, a type $S(\Phi)$ as follows:

- if $\mathfrak{a} = a$, $S(a)$ is an arbitrary sequence of pairwise distinct variables $\langle S(a)_1,\dots,S(a)_{k_a}\rangle$ (chosen in such a way that if $a \neq b$, $S(a)$ and $S(b)$ are disjoint);
- if $\mathfrak{a} = \langle X_1,\dots,X_r\rangle$, then $S(\mathfrak{a}) = \langle X_1,\dots,X_r\rangle$;
- if $\mathfrak{a} = \alpha^n a_1 \dots a_n$, then $S(\mathfrak{a}) = \langle U_1,\dots,U_{k^S_\alpha}\rangle$ where for all $i \leqslant k^S_\alpha$:
  - if $S(\alpha)_i = \lambda\vec{x}.X$, then $U_i = X$;
  - if $S(\alpha)_i = \lambda\vec{x}.\pi^l(x_j)$, then $U_i = S(a_j)_l$;
- if $\phi = X$, then $S(\phi) = X$;
- if $\phi = \pi^l(\mathfrak{a})$, then $S(\phi) = S(\mathfrak{a})_l$;
- if $\phi = \mathsf{F}\mathfrak{a}_1\dots\mathfrak{a}_n$, and $S(\mathsf{F}) = \lambda\vec{\rho}.A$, then $S(\phi) = A[\pi^l(\rho_i) \mapsto S(\mathfrak{a}_i)_l]$;
- if $\phi = \Phi \Rightarrow \Psi$, then $S(\phi) = S(\Phi) \Rightarrow S(\Psi)$;
- if $\Phi = \forall a.\phi$, then $S(\Phi) = \forall S(a).S(\phi)$.

A substitution S for (U, E) is a *unifier* of (U, E) if the following hold:

1. for any equation $\phi = \psi \in U$, $S(\phi) = S(\psi)$ holds;
2. for any constraint of the form $\alpha : a \in E$, $k^S_a = k^S_\alpha$;
3. for any constraint of the form $a : k \in E$, $k^S_a = k$.

We let `Fat-unification` indicate the problem of finding a unifier for a $\mathrm{F_{at}}$-unification problem. The rest of this subsection is devoted to establish the following:

▶ **Theorem 24.** `Fat-unification` *is decidable.*

A $\mathrm{F_{at}}$-unification problem (U, E) is in *normal form* if if contains no equation of the form $\Phi_1 \Rightarrow \Psi_1 = \Phi_2 \Rightarrow \Psi_2$. Any unification problem can be put in normal form by repeatedly applying the following simplification rule:

$$\frac{U + \{(\forall a_1.\phi_1) \to (\forall b_1.\psi_1) = (\forall a_2.\phi_2) \to (\forall b_2.\psi_2)\}}{\big(U + \{\phi_1 = \phi_2, \psi_1 = \psi_2\}\big)\big[a_2 \mapsto a_1, b_2 \mapsto b_1\big]}$$

Given a $\mathrm{F_{at}}$-unification problem in normal form (U, E), we say that an equation $\phi = \psi$ *can be deduced from* U if $\phi = \psi$ can be deduced from a finite set of equations in U by applying standard first-order equality rules. We say that two second-order variables F, G are *equivalent* (noted $\mathsf{F} \simeq \mathsf{G}$) if an equation of the form $\mathsf{F}\mathfrak{a}_1\dots\mathfrak{a}_n = \mathsf{G}\mathfrak{b}_1\dots\mathfrak{b}_n$ can be deduced from U; we say that F is *connected with* G (noted $\mathsf{F} \rightsquigarrow \mathsf{G}$) if an equation of the form $\mathsf{F}\mathfrak{a}_1\dots\mathfrak{a}_n = \Phi \Rightarrow \Psi$, where U occurs in $\Phi \Rightarrow \Psi$, can be deduced from U. We say that (U, E) has a *variable cycle* if there exist variables $\mathsf{F}_1,\dots,\mathsf{F}_k$ such that $\mathsf{F}_1 \overset{\simeq}{\rightsquigarrow} \mathsf{F}_2 \overset{\simeq}{\rightsquigarrow} \dots \overset{\simeq}{\rightsquigarrow} \mathsf{F}_n \overset{\simeq}{\rightsquigarrow} \mathsf{F}_1$ (where $\mathsf{F} \overset{\simeq}{\rightsquigarrow} \mathsf{G}$ means that F is connected with some variable equivalent to G).

▶ **Lemma 25.** *Let (U, E) be a unification problem in normal form. If (U, E) has a variable cycle, then it has no solution.*

Proof. To prove the lemma we show that any unification problem (U, E) yields a *first-order* unification problem U^* and that any unifier of (U, E) yields a unifier of U^*. For the translation, we fix a constant c, and we associate any second-order variable F with a first-order variable x_{F}; any expression is translated into a first order expression by:

$$\begin{aligned} \mathfrak{a}^* &= c \\ \mathsf{F}^n \mathfrak{a}_1 \dots \mathfrak{a}_n &= x_{\mathsf{F}} \\ (\Phi \Rightarrow \Psi)^* &= \Phi^* \Rightarrow \Psi^* \\ (\forall a.\phi)^* &= \phi^* \end{aligned}$$

We finally let $U^* = \{\phi^* = \psi^* \mid \phi = \psi \in U\}$. Observe that if $\mathsf{F} \simeq \mathsf{G}$ in U, then $x_{\mathsf{F}} = x_{\mathsf{G}}$ in U^*, and if $\mathsf{F} \rightsquigarrow \mathsf{G}$ in U, then U^* contains an equation of the form $x_{\mathsf{F}} = t \Rightarrow u$, where x_{G} occurs in $t \Rightarrow u$. Hence a variable cycle in (U, E) induces a variable cycle in U^*.

For any substitution S for (U, E), we define a first-order substitution S^* as follows: given $\lambda\vec{\rho}.A$ we define A^* by $X^* = c$, $(\pi^l(\rho_i))^* = c$, $(A \Rightarrow B)^* = A^* \Rightarrow B^*$ and $(\forall X.A)^* = A^*$. We let then $S^*(x_{\mathsf{F}}) = S(\mathsf{F})^*$.

One can easily check that if S is a unifier for (U, E), then S^* is a unifier of U^*. As a consequence, if (U, E) has a variable cycle, so does U^*, and by well-known facts about first-order unification, U^* has no unifier, and so neither (U, E) does. ◀

Let us call a unification problem (U, E) *simple* if it contains no expression of the form $\Phi \Rightarrow \Psi$. If (U, E) has no variable cycle, then it can be reduced to a simple unification problem by applying the following rules:

$$\frac{U + \{X = \Phi \Rightarrow \Psi\}}{\{X = Y\}} \qquad\qquad \frac{U + \{\pi^l(\mathfrak{a}) = \Phi \Rightarrow \Psi\}}{\{X = Y\}}$$

$$\frac{U + \{\mathsf{F}^n \mathfrak{a}_1^1 \dots \mathfrak{a}_n^1 = (\forall c_1.\phi_1) \Rightarrow (\forall d_1.\psi_1), \dots, \mathsf{F}^n \mathfrak{a}_1^r \dots \mathfrak{a}_n^r = (\forall c_r.\phi_r) \Rightarrow (\forall d_r.\psi_r)\}}{U\Big[\mathsf{F}^n \vec{\mathfrak{a}} \mapsto (\mathsf{F}_1^{n+1} \vec{\mathfrak{a}} c \Rightarrow \mathsf{F}_2^{n+1} \vec{\mathfrak{a}} d)\Big] + \begin{Bmatrix} \mathsf{F}_1^{n+1} \mathfrak{a}_1^1 \dots \mathfrak{a}_n^1 c_1 = \phi_1, \dots, \mathsf{F}_1^{n+1} \mathfrak{a}_1^r \dots \mathfrak{a}_n^r c_r = \phi_r \\ \mathsf{F}_2^{n+1} \mathfrak{a}_1^1 \dots \mathfrak{a}_n^1 d_1 = \psi_1, \dots, \mathsf{F}_2^{n+1} \mathfrak{a}_1^r \dots \mathfrak{a}_n^r d_r = \psi_r \end{Bmatrix}}$$

Where in the first two rules Y is any type variable distinct from X, and in the last rule we suppose that U contains no equation of the form $\mathsf{F}^n \mathfrak{a}_1 \dots \mathfrak{a}_n = \Phi \Rightarrow \Psi$. Observe that, by acycliclity, F cannot occur in either ϕ_i or ψ_i. One can argue by induction on the well-founded preorder $\overset{\simeq}{\rightsquigarrow}$ that all terms of the form $\Phi \Rightarrow \Psi$ can be eliminated by applying a finite number of instances of the rules above.

The last step to ensure decidability is showing (1) that all solutions to a $\mathrm{F_{at}}$-unification problem (U, E) can be generated algorithmically and (2) that one can suppose that, if a solution exists at all, this can be found within a *finite* search-space (that is, one in which only projections $\pi^l(\mathfrak{a})$, with l less than some fixed value K depending on the size of (U, E), occur). Step (2) ensures that, if a solution is not found after a finite search, one can conclude that no solution exists at all. These are the two ingredients of the proof of the proposition below, which is shown in detail in [50].

▶ **Proposition 26.** *There is an algorithm that generates all unifiers of a simple unification problem, if there exists any, and returns* `failure` *otherwise.*

Type-checking $\mathrm{F_{at}}$ by second-order unification. A *type-checking problem* is a triple (Γ, t, A) where Γ is a term context, t is a λ-term with $FV(t) \subseteq \Gamma$ and A is a type. A $\mathrm{F_{at}}$*-solution* of a type-checking problem is a type derivation in $\mathrm{F_{at}}$ of $\Gamma \vdash t : A$. We wish to prove the following:

$$\frac{\Gamma(x) = A \qquad A \preceq B}{\Gamma \vdash x : \forall \vec{X} : B}\ \vec{X} \notin FV(\Gamma) \qquad \frac{\Gamma, x \mapsto A \vdash t : B}{\Gamma \vdash \lambda x.t : \forall \vec{X}.A \Rightarrow B}\ \vec{X} \notin FV(\Gamma)$$

$$\frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A \qquad B \preceq C}{\Gamma \vdash tu : \forall \vec{X}.C}\ \vec{X} \notin FV(\Gamma)$$

Figure 4 Synthetic typing rules for Curry-style $\mathrm{F_{at}}$.

▶ **Theorem 27.** *For any type-checking problem (Γ, t, A), there exists a $\mathrm{F_{at}}$-unification problem $\mathbf{V}(\Gamma, t, A)$ such that (Γ, t, A) has a solution in $\mathrm{F_{at}}$ iff $\mathbf{V}(\Gamma, t, A)$ has a unifier.*

The first step is to associate with each term t finite sets of sequence variables, projection variables and second-order variables as follows (we suppose that no variable occurs both free and bound in t, and that any bound variable is bound exactly once):

- with each variable x in t, we associate two sequence variables a_x, b_x, a projection variable α^1_x, and two second-order variables $\mathsf{F}^1_x, \mathsf{G}^1_x$;
- with each subterm of t of the form uv, we similarly associate two sequence variables a_{uv}, b_{uv}, a projection variable α^1_{uv} and two second-order variables $\mathsf{F}^2_{uv}, \mathsf{G}^1_{uv}$;
- with each subterm of t of the form $\lambda x.u$, we associate a sequence variable $b_{\lambda x.u}$, and a second order variable $\mathsf{G}^1_{\lambda x.t}$.

Given a set of equations U and a sequence variable a not occurring in U, we let Ua be the set of equations obtained by replacing all terms $\alpha^n a_1 \dots a_n$ by $\alpha^{n+1} a_1 \dots a_n a$ and all term $\mathsf{F}^n \mathfrak{a}_1 \dots \mathfrak{a}_n$ by $\mathsf{F}^{n+1} \mathfrak{a}_1 \dots \mathfrak{a}_n a$.

We define a set of equations $\mathbf{U}(t)$, by induction on t as follows:

- $\mathbf{U}(x)$ is formed by the equation

$$\mathsf{F}_x(\alpha_x b_x) = \mathsf{G}_x b_x$$

- $\mathbf{U}(\lambda x.t)$ is formed by $\mathbf{U}(t)b_{\lambda x.t}$ plus the equations

$$\mathsf{G}_{\lambda x.t} b_{\lambda x.t} = (\forall a_x.\mathsf{F}_x a_x \vec{b} b_{\lambda x.t}) \Rightarrow \forall b_t.\mathsf{G}_t b_t b_{\lambda x.t}$$

- $\mathbf{U}(tu)$ is formed by $\mathbf{U}(t)b_{tu}, \mathbf{U}(u)b_{tu}$ plus the equations:

$$\mathsf{G}_t b_t b_{tu} = (\forall b_u.\mathsf{G}_u b_u b_{tu}) \Rightarrow (\forall a_{tu}.\mathsf{F}_{tu} a_{tu} b_{tu})$$
$$\mathsf{F}_{tu}(\alpha_{tu} b_{tu}) b_{tu} = \mathsf{G}_{tu} b_{tu}$$

We let $\mathbf{V}(\Gamma, t, A) = (\mathbf{U}(\Gamma, t, A), \mathbf{E}(\Gamma, t, A))$, where $\mathbf{U}(\Gamma, t, A)$ is the union of $\mathbf{U}(t)$ and all equations $\forall a_x.\mathsf{F}_x a_x = \Gamma(x)$ and $\forall b_t.\mathsf{G}_t b_t = A$. $\mathbf{E}(\Gamma, t, A)$ is formed by all constraints of the form $(\alpha_x : a_x)$ and $(\alpha_{tu} : b_t)$, as well as all constraints of the form $(a_x : k)$, where $\Gamma(x) = \forall X_1 \dots X_k.C$, all constraints of the form $(b_u : 0)$ where t contains a subterm of the form uv, and the constraint (b_t, h), where $A = \forall X_1 \dots X_h.A'$.

To show that solving $\mathbf{V}(\Gamma, t, A)$ is equivalent to checking if $\Gamma \vdash t : A$, as in [21], we first define synthetic typing rules for Curry-style $\mathrm{F_{at}}$ as shown in Fig. 4, where $A \preceq B$ holds when $A = \forall X_1 \dots X_n.A$ and $B = A[X_1 \mapsto Y_1, \dots, X_n \mapsto Y_n]$.

One can check by induction on t that a synthetic type derivation of $\Gamma \vdash t : A$ yields a unifier of $\mathbf{V}(\Gamma, t, A)$. Conversely, we show that from a unifier S for $\mathbf{V}(\Gamma, t, A)$ we can construct a synthetic typing derivation of $\Gamma \vdash t : A$. We argue by induction on t:

- if $t = x$, then we have $\Gamma(x) = \forall X_1 \dots X_N.S(\mathsf{F}_x)\vec{X}$, where $N = k^S_{a_x}$, $A = \forall Y_1 \dots Y_P.S(\mathsf{G}_x)\vec{Y}$, where $P = k^S_{b_x}$, and moreover, $S(\mathsf{F}_x)(S(\alpha_x)_1\vec{Y}) \dots (S(\alpha_x)_N\vec{Y}) = S(\mathsf{G}_x)\vec{Y}$ (using the fact that $k^S_{\alpha_x} = k^S_{a_x} = N$). Observe that $(S(\alpha_x)_j\vec{Y})$ is a variable, and we deduce then that $\Gamma(x) \preceq S(\mathsf{G}_x)\vec{Y}$; since we can suppose that $\vec{Y}$ does not occur in Γ, we deduce then that

$$\dfrac{\Gamma(x) = \forall \vec{X}.S(\mathsf{F}_x)\vec{X} \qquad \forall \vec{X}.S(\mathsf{F}_x)\vec{X} \preceq S(\mathsf{G}_x)\vec{Y}}{\Gamma \vdash x : A}\ \vec{Y} \notin FV(\Gamma)$$

- if $t = \lambda x.u$, then we have that $A = \forall X_1 \dots X_N.A_1 \Rightarrow A_2$, where $A_1 = \forall Y_1 \dots Y_P.S(\mathsf{F}_x)\vec{Y}\vec{X}$ and $A_2 = \forall Z_1 \dots Z_Q.S(\mathsf{G}_u)\vec{Z}\vec{X}$, $N = k^S_{b_{\lambda x.u}}$, $P = k^S_{a_x}$, $Q = k^S_{b_u}$ and where we can suppose that the X_i do not occur free in Γ; since $\mathsf{U}(t) = \mathsf{U}(u)b_{\lambda x.t}$ we deduce that S unifies $\mathbf{V}(\Gamma \cup \{x : A_1\}, u, A_2))$. By I.H. we deduce then the existence of a type derivation of $\Gamma, x : A_1 \vdash u : A_2$, and since the X_i do not occur in Γ we finally have

$$\dfrac{\overset{[\text{I.H.}]}{\Gamma, x : A_1 \vdash u : A_2}}{\Gamma \vdash t : A}\ \vec{X} \notin FV(\Gamma)$$

- if $t = uv$, then we have that $A = \forall X_1 \dots X_N.S(\mathsf{G}_{uv})\vec{X}$, $S(\mathsf{G}_u)\vec{X} = (\forall Y_1 \dots Y_P.S(\mathsf{G}_v)\vec{Y}\vec{X}) \Rightarrow (\forall Z_1 \dots Z_Q.S(\mathsf{F}_{uv})\vec{Z}\vec{X})$ and that $S(\mathsf{F}_{uv})(S(\alpha_{uv})_1\vec{X}) \dots (S(\alpha_{uv})_N\vec{X})\vec{X} = S(\mathsf{G}_{uv})\vec{X}$, where $N = k^S_{b_{uv}}$, $P = k^S_{b_u}$ and $Q = k^S_{a_{uv}}$, and where we use the fact that $k^S_{b_u} = 0$. Moreover, for any choice of the variables $\vec{X}$, we have that S unifies $\mathbf{V}(\Gamma, u, (\forall Y_1 \dots Y_P.S(\mathsf{G}_v)\vec{Y}\vec{X}) \Rightarrow \forall Z_1 \dots Z_Q.S(\mathsf{F}_{uv})\vec{Z}\vec{X})$ and $\mathbf{V}(\Gamma, v, \forall Y_1 \dots Y_P.S(\mathsf{G}_u)\vec{Y}\vec{X})$; by choosing the $\vec{X}$ so that they do not occur free in Γ, using the I.H. and the fact that $k^S_{\alpha_{uv}} = k^S_{a_{uv}} = Q$, we deduce then

$$\dfrac{\overset{[\text{I.H.}]}{\Gamma \vdash u : (\forall Y_1 \dots Y_P.S(\mathsf{G})_v\vec{Y}\vec{X}) \Rightarrow (\forall Z_1 \dots Z_Q.S(\mathsf{F}_{uv})\vec{Z}\vec{X})} \qquad \overset{[\text{I.H.}]}{\Gamma \vdash v : \forall Y_1 \dots Y_P.S(\mathsf{G}_v)\vec{Y}\vec{X}} \qquad \forall Z_1 \dots Z_Q.S(\mathsf{F}_{uv})\vec{Z}\vec{X} \preceq S(\mathsf{G}_{uv})\vec{X}}{\Gamma \vdash t : A}\ \vec{X} \notin FV(\Gamma)$$