

ON THE MINTS HIERARCHY IN FIRST-ORDER INTUITIONISTIC LOGIC*

ALEKSY SCHUBERT^a, PAWEŁ URZYCZYN^b, AND KONRAD ZDANOWSKI^c

^{a,b} Institute of Informatics, University of Warsaw, ul. S. Banacha 2, 02–097 Warsaw, Poland
e-mail address: {alx, urzy}@mimuw.edu.pl

^c Cardinal Stefan Wyszyński University in Warsaw, ul. Dewajtis 5, 01-815 Warsaw, Poland
e-mail address: k.zdanowski@uksw.edu.pl

ABSTRACT. We stratify intuitionistic first-order logic over (\forall, \rightarrow) into fragments determined by the alternation of positive and negative occurrences of quantifiers (Mints hierarchy). We study the decidability and complexity of these fragments. We prove that even the Δ_2 level is undecidable and that Σ_1 is EXPSpace-complete. We also prove that the arity-bounded fragment of Σ_1 is complete for *co-NEXPTIME*.

1. INTRODUCTION

The leading proof assistants such as Coq [6], Agda [3] or Isabelle [17] are founded on constructive logics. Still, the complexity behind proof search in constructive reasoning systems is not well understood even for their basic and crucial fragments where the implication and universal quantification are used. This situation is caused partly by the difficulty of the field and partly by the lack of a systematic approach, especially in the case of quantifiers.

Quantifiers are present in logic at least from the time of Aristotle but a modern theory of quantification was probably initiated by Ch.S. Peirce and G. Frege [1]. The systematic approach to quantifiers through their grouping at the beginning of a logical formula was originated by Peirce and worked out by A. Church [5], who first used the term “prenex normal form”. Since then classifying formulas according to the quantifier prefix remains a standard stratification tool in modern logic, just to mention Ehrenfeucht-Fraïssé games [11, Chapter 6] or the arithmetical hierarchy of Kleene and Mostowski [9, Chapter 7].

Classes of prenex formulas in the full first-order language, beginning with \exists (resp. \forall), and with n alternating groups of quantifiers are denoted in this paper by the sans-serif symbol Σ_n (resp. Π_n). (The ordinary serifed symbols Σ and Π are reserved for classes of the Mints hierarchy.) It is known that classes Σ_n and Π_n form a strict hierarchy with

2012 ACM CCS: [Theory of computation]: Models of computation—Computability—Lambda calculus; Computational complexity and cryptography—Complexity theory and logic; Logic—Proof theory / Constructive mathematics.

Key words and phrases: Intuitionistic logic, Mints hierarchy, complexity, automata.

* Project supported through NCN grant DEC-2012/07/B/ST6/01532. This paper is a revised and expanded version of [24].

respect to their classical expressive power [20]. While the prenex normal form is useful for classification of formulas, which was demonstrated in full strength by Börger, Grädel, and Gurevich in their influential book [2], it is rarely used in practice. The structure of formulas arising from actual reasoning (in particular proof formalization) often involves quantification in arbitrary positions. For instance this happens when a quantified definition is expanded in a formula.

In addition, the prenex normal form theorem applies to *classical* logic only. Things become quite different for constructive logic (aka intuitionistic logic), because the prenex fragment of intuitionistic logic is decidable [18]. This contrasts with the undecidability of the general case (see e.g., [26]) and that makes this form of stratification unsuitable in the constructive context.

Can we replace the prenex classification by something adequate for intuitionistic logic? Yes, we can: as observed by Grigori Mints [14], the principal issue is the alternation of positive and negative occurrences of quantifiers in a formula, understood as in [28]. Roughly speaking, a quantifier occurrence is positive iff a classical reduction to a prenex form turns it into a universal quantifier. Dually, negative occurrences of quantifiers are those which become existential quantifiers after normalization. This yields the *Mints hierarchy* of formulas, consisting of the following classes (note the serified Σ and Π):

Π_1 – All quantifier occurrences are positive.

Σ_1 – All quantifiers occurrences are negative.

Π_2 – Up to one alternation: no positive quantifier in scope of a negative one.

Σ_2 – Up to one alternation: no negative quantifier in scope of a positive one.

And so on. Every formula can be classified as a Π_n or a Σ_n formula without actually reducing it to a prenex form. Therefore, Mints hierarchy makes perfect sense for intuitionistic logic.

In this paper we address the question of decidability and complexity of the intuitionistic provability problem for classes Π_n and Σ_n . This of course resembles the subject of [2], and it is natural to compare our results with those in the book. As it may be expected, the intuitionistic case is at least as hard as the classical case. (Remember though that complexity results about classical logic are usually stated in terms of satisfiability.)

As for the existing knowledge, Mints proved that the fragment Π_1 of the constructive logic with all connectives and quantifiers is decidable [14]. An alternative proof of Mints' result (for the calculus with \forall and \rightarrow only) was given by Dowek and Jiang [8]. A similar decidability result was also obtained by Rummelhoff [21] for the positive fragment of second-order propositional intuitionistic logic (system F). The *co-2-NEXPTIME* lower bound for Π_1 was proved by Schubert, Urzyczyn and Walukiewicz-Chrząszcz [23], but the problem is conjectured to be non-elementary [22]. The undecidability of Σ_2 with all connectives and quantifiers can be derived from the undecidability of the classical satisfiability problem for $\forall^*\exists^*$ using a result of Kreisel [12, Thm. 7]. This would not work for Π_2 because the classical satisfiability of the Ramsey class $\exists^*\forall^*$ is decidable. Undecidability for Π_2 (for the full language with one binary predicate) is implied by a result of Orevkov [15]. The conference version [24] of the present paper strengthened Orevkov result by showing the undecidability for the (\forall, \rightarrow) -fragment.

There are other forms of quantifier-oriented hierarchical stratifications of intuitionistic formulas. For instance, the classical prenex hierarchy can be embedded in a fragment of the intuitionistic logic: a negation of a prenex formula is classically provable if and only if it is provable intuitionistically [12]. A similar, but more general class of formulas in so

called pseudoprenex form, where quantifiers may be separated by double negation $\neg\neg$, was studied in depth by Orevkov who gave a full characterization of decidable cases [16]. Also a full characterization of decidable cases was given for prenex formulas with equality and function symbols [7]. Other hierarchies of intuitionistic formulas were proposed e.g., by Fleischmann [10] and Burr [4] (the latter for arithmetic). However, we are not aware of any complexity-oriented results for those hierarchies.

In this paper we expand the systematic study of the decision problem in Mints hierarchy initiated in [24]. Basically, we restrict attention to the fragment where only the implication and the universal quantifier may occur. Our main results are as follows:

- A. The hierarchy is strict with respect to the expressive power.
- B. The decision problem for the class $\Delta_2 = \Sigma_2 \cap \Pi_2$ is undecidable.
- C. The decision problem for the class Σ_1 is EXPSPACE-complete.
- D. The decision problem for arity-bounded Σ_1 formulas is *co*-NEXPTIME complete.
- E. The decision problem for Σ_1 restricted to any finite signature is in *co*-NEXPTIME.

These results are supplemented by the *co*-2-NEXPTIME lower bound for Π_1 obtained in [23] and a strong evidence towards the conjecture that Π_1 is actually non-elementary [22]. Observe that, because of conservativity, part B applies directly to the full intuitionistic logic, and the same holds for the lower bound in C. The upper bound in C also extends to the general case at the cost of some additional complication.

The undecidabilities in B are shown for the monadic fragment of minimal logic (i.e., the language with only unary predicate symbols). Our proof of A requires a binary predicate but we conjecture that the monadic hierarchy is also strict. It is slightly different with C versus D, where we have arrived at the open problem whether *co*-NEXPTIME equals EXPSPACE.

The paper is organized as follows. Section 2 contains the basic definitions, and proves strictness of the hierarchy. Section 3 introduces the undecidable tiling puzzles. Those are encoded in Section 4 into Δ_2 formulas. In Section 4.1 we use a syntactic translation to obtain undecidability for the monadic fragment of Δ_2 . In Section 5 we show EXPSPACE-completeness for Σ_1 using the decision problem for bus machines [27]. In the last Section 6 we study Σ_1 formulas with predicates of bounded arity.

2. PRELIMINARIES

We consider first-order intuitionistic logic without function symbols and without equality. That is, the only individual terms are *object variables*, written in lower case, e.g., x, y, \dots . In this paper we restrict attention to formulas built only from implication and the universal quantifier. A formula is therefore either an atom $P(x_1, \dots, x_n)$, where $n \geq 0$, or an implication $\varphi \rightarrow \psi$, or it has the form $\forall x \varphi$.

We use common parentheses-avoiding conventions, in particular we take the implication to be right-associative. That is, $\varphi \rightarrow \psi \rightarrow \vartheta$ stands for $\varphi \rightarrow (\psi \rightarrow \vartheta)$.

Our proof notation is an extended lambda-calculus of *proof terms* or simply *proofs* or *terms*. Formulas are treated as types assigned to proof terms. In addition to object variables, in proof terms there are also *proof variables*, written as upper-case letters, like X, Y, Z . An *environment* is a set of declarations $(X : \varphi)$, where X is a proof variable and φ is a formula. The type-assignment rules in Figure 1 infer judgments of the form $\Gamma \vdash M : \varphi$, where Γ is an environment, M is a proof term, and φ is a formula. In $(\forall I)$ we require $x \notin \text{FV}(\Gamma)$ and y in $(\forall E)$ is an arbitrary object variable.

$$\begin{array}{c}
\Gamma, X : \varphi \vdash X : \varphi \quad (Ax) \\
\frac{\Gamma, X : \varphi \vdash M : \psi}{\Gamma \vdash \lambda X : \varphi. M : \varphi \rightarrow \psi} \quad (\rightarrow I) \qquad \frac{\Gamma \vdash M : \varphi \rightarrow \psi \quad \Gamma \vdash N : \varphi}{\Gamma \vdash MN : \psi} \quad (\rightarrow E) \\
\frac{\Gamma \vdash M : \varphi}{\Gamma \vdash \lambda x M : \forall x \varphi} \quad (\forall I) \qquad \frac{\Gamma \vdash M : \forall x \varphi}{\Gamma \vdash My : \varphi[x := y]} \quad (\forall E)
\end{array}$$

Figure 1: Proof assignment rules

That is, we have two kinds of lambda-abstraction: the proof abstraction $\lambda X : \varphi. M$ and the object abstraction $\lambda x M$. There are also two forms of application: the proof application MN , where N is a proof term, and the object application My , where y is an object variable. We use the conventions common in lambda-calculus e.g., unnecessary parentheses are omitted and the application is left-associative: MNP means $((MN)P)$. Terms and formulas are taken up to alpha-conversion.

The formalism is used liberally. Terms are always assumed to be well-typed, even if type information is left out. For instance, we often say that “a term M has type φ ” leaving the environment implicit. Also we often identify environments with sets of formulas, as well as we write $\Gamma \vdash \varphi$ when $\Gamma \vdash M : \varphi$ and M is not relevant at the moment. Sometimes for convenience we drop φ from $\lambda X : \varphi. M$ when it can be deduced from the context.

Free (object) variables $\text{FV}(\varphi)$ in a formula φ are as usual. We also define free variables in proofs: $\text{FV}(X) = \emptyset$, $\text{FV}(\lambda X : \varphi. M) = \text{FV}(\varphi) \cup \text{FV}(M)$, $\text{FV}(MN) = \text{FV}(M) \cup \text{FV}(N)$, $\text{FV}(\lambda x M) = \text{FV}(M) - \{x\}$, $\text{FV}(My) = \text{FV}(M) \cup \{y\}$. The notation $M[\vec{x} := \vec{y}]$ stands for the simultaneous substitution of a vector of variables $\vec{y} = y_1 \dots y_n$ for free occurrences of (different) variables $\vec{x} = x_1 \dots x_n$. To make this precise, we take:

- $x_i[\vec{x} := \vec{y}] = y_i$, and $z[\vec{x} := \vec{y}] = z$, when z is not in \vec{x} ;
- $X[\vec{x} := \vec{y}] = X$;
- $(\lambda X : \varphi. M)[\vec{x} := \vec{y}] = \lambda X : \varphi[\vec{x} := \vec{y}]. M[\vec{x} := \vec{y}]$, where $\varphi[\vec{x} := \vec{y}]$ is as usual;
- $(MN)[\vec{x} := \vec{y}] = M[\vec{x} := \vec{y}]N[\vec{x} := \vec{y}]$;
- $(\lambda x M)[\vec{x} := \vec{y}] = \lambda x M[\vec{x} := \vec{y}]$, when x is not among \vec{x}, \vec{y} ;
- $(My)[\vec{x} := \vec{y}] = M[\vec{x} := \vec{y}]y[\vec{x} := \vec{y}]$.

Lemma 2.1. *If $\Gamma \vdash N : \varphi$ then $\Gamma[\vec{x} := \vec{y}] \vdash N[\vec{x} := \vec{y}] : \varphi[\vec{x} := \vec{y}]$.*

Proof. Easy induction. □

A term is in *normal form* when it contains no redex, i.e., no subterm of the form $(\lambda X : \varphi. M)N$ or of the form $(\lambda x M)y$. We also define the notion of a proof term in *long normal form*, abbreviated *lnf*.

- If N is an lnf of type φ then $\lambda x N$ is an lnf of type $\forall x \varphi$.
- If N is an lnf of type ψ then $\lambda X : \varphi. N$ is an lnf of type $\varphi \rightarrow \psi$.
- If N_1, \dots, N_n are lnf or object variables, and $XN_1 \dots N_n$ is of an atom type, then $XN_1 \dots N_n$ is an lnf.

The following lemma is shown in [22].

Lemma 2.2. *If φ is intuitionistically derivable from Γ then $\Gamma \vdash N : \varphi$, for some lnf N . □*

The *target* of a formula is the relation symbol at the end of it. Formally, $\text{target}(\text{P}(\vec{x})) = \text{P}$, for atomic formulas, $\text{target}(\varphi \rightarrow \psi) = \text{target}(\psi)$, and $\text{target}(\forall x \varphi) = \text{target}(\varphi)$. The following observation is essential in long normal proof search.

Lemma 2.3. *If $\Gamma \vdash N : \text{P}(\vec{x})$, where $\text{P}(\vec{x})$ is an atomic formula and N is an lnf, then $N = X\vec{D}$, where $(X : \psi) \in \Gamma$ with $\text{target}(\psi) = \text{P}$, and \vec{D} is a sequence that may contain proof terms and object variables.*

Proof. An easy consequence of the definition of an lnf. □

Miscellaneous: The set of all words over an alphabet \mathcal{A} is written as \mathcal{A}^* . By ε we denote the empty word. The relation $w \subseteq v$ holds when w is a prefix of v .

2.1. An example. To illustrate the computational flavour of intuitionistic proof search we consider the formula $\alpha_0 \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3 \rightarrow \beta \rightarrow \text{C}$, where C is a nullary atom and:

$$\begin{aligned} \alpha_0 &= \forall x(\text{F}(0, x) \rightarrow \text{S}(0, x) \rightarrow \text{T}(0, x) \rightarrow \text{loop}(x)) \rightarrow \text{C}; \\ \alpha_1 &= \forall x(\text{F}(0, x) \rightarrow \forall y(\text{F}(1, y) \rightarrow \forall z(\text{S}(z, x) \rightarrow \text{S}(z, y)) \rightarrow \\ &\quad \forall z(\text{T}(z, x) \rightarrow \text{T}(z, y)) \rightarrow \text{loop}(y)) \rightarrow \text{loop}(x)); \\ \alpha_2 &= \forall x(\text{F}(1, x) \rightarrow \text{S}(0, x) \rightarrow \forall y(\text{F}(0, y) \rightarrow \text{S}(1, y) \rightarrow \\ &\quad \forall z(\text{T}(z, x) \rightarrow \text{T}(z, y)) \rightarrow \text{loop}(y)) \rightarrow \text{loop}(x)); \\ \alpha_3 &= \forall x(\text{F}(1, x) \rightarrow \text{S}(1, x) \rightarrow \text{T}(0, x) \rightarrow \\ &\quad \forall y(\text{F}(0, y) \rightarrow \text{S}(0, y) \rightarrow \text{T}(1, y) \rightarrow \text{loop}(y)) \rightarrow \text{loop}(x)); \\ \beta &= \forall x(\text{F}(1, x) \rightarrow \text{S}(1, x) \rightarrow \text{T}(1, x) \rightarrow \text{loop}(x)). \end{aligned}$$

In the above, 0 and 1 are fixed free variables, playing the role of “bits”. The predicates F(irst), S(econd), and T(hird), are intended to always occur together to associate three bits to a variable. For instance, assumptions $\text{F}(1, x), \text{S}(0, x), \text{T}(1, x)$ associate the binary string 101 to the variable x . Our formula is constructed in such a way that every proof of it must “generate” variables associated with all binary strings of length three.

To derive C from $\Gamma = \{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \beta\}$, one must use an assumption with target C , and α_0 is the only such assumption. So we need to prove

$$\Gamma \vdash \forall x(\text{F}(0, x) \rightarrow \text{S}(0, x) \rightarrow \text{T}(0, x) \rightarrow \text{loop}(x)),$$

and this amounts to proving $\Gamma \vdash \text{F}(0, x_1) \rightarrow \text{S}(0, x_1) \rightarrow \text{T}(0, x_1) \rightarrow \text{loop}(x_1)$, where x_1 is a fresh eigenvariable. That is, we now have the new proof goal $\text{loop}(x_1)$ to be derived using additional assumptions $\text{F}(0, x_1), \text{S}(0, x_1), \text{T}(0, x_1)$. (We interpret it as “ x_1 is associated to the string 000”.) Given this knowledge about x_1 we readily discover that α_1 is the only applicable assumption, as otherwise we would have to prove $\text{F}(1, x_1)$, which is clearly hopeless. So we instantiate α_1 with x_1 in place of x and we now need to derive the universal formula $\forall y(\text{F}(1, y) \rightarrow \forall z(\text{S}(z, x_1) \rightarrow \text{S}(z, y)) \rightarrow \forall z(\text{T}(z, x_1) \rightarrow \text{T}(z, y)) \rightarrow \text{loop}(y))$. This introduces a new eigenvariable x_2 . Our new goal is $\text{loop}(x_2)$, and our new assumptions are $\text{F}(1, x_2)$ and $\forall z(\text{S}(z, x_1) \rightarrow \text{S}(z, x_2)), \forall z(\text{T}(z, x_1) \rightarrow \text{T}(z, x_2))$. The latter two can be used (if needed) to derive $\text{S}(0, x_2)$ and $\text{T}(0, x_2)$. (This implicitly assigns the string 100 to x_2 .) With this knowledge at hand, we can now try to apply α_2 towards proving $\text{loop}(x_2)$. We leave it to the reader to check that our proof construction will lead us to introducing (at least) six other eigenvariables x_3, \dots, x_8 and that the proof will be completed with an application of β , when we reach the string 111, i.e., when the assumptions $\text{F}(1, x_8), \text{S}(1, x_8), \text{T}(1, x_8)$ become available. Note that various instances of α_1 occur in the proof four times, and α_2 is used twice.

If we use proof variables X_0, X_1, X_2, X_3, Y to denote assumptions $\alpha_0, \alpha_1, \alpha_2, \alpha_3, \beta$, respectively, then the proof can be written as the following lambda-term. The possibly confusing subterm $Z_3^4 0(Z_3^3 0(Z_3^2 0 Z_3^1))$ has type $T(0, x_4)$ and corresponds to a composition of assumptions $Z_3^{i+1} : \forall z(T(z, x_i) \rightarrow T(z, x_{i+1}))$, for $i = 1, 2, 3$, applied to the assumption $Z_3^1 : T(0, x_1)$.

$$\begin{aligned} & \lambda X_0 X_1 X_2 X_3 Y. X_0 (\lambda x_1 \lambda Z_1^1 Z_2^1 Z_3^1. \\ & \quad X_1 x_1 Z_1^1 (\lambda x_2 \lambda Z_1^2 Z_2^2 Z_3^2. \\ & \quad \quad X_2 x_2 Z_1^2 (Z_2^2 0 Z_2^1) (\lambda x_3 \lambda Z_1^3 Z_2^3 Z_3^3. \\ & \quad \quad \quad X_1 x_3 Z_1^3 (\lambda x_4 \lambda Z_1^4 Z_2^4 Z_3^4. \\ & \quad \quad \quad \quad X_3 x_4 Z_1^4 (Z_2^4 1 Z_2^3) (Z_3^4 0 (Z_3^3 0 (Z_3^2 0 Z_3^1))) (\lambda x_5 \lambda Z_1^5 Z_2^5 Z_3^5. \\ & \quad \quad \quad \quad \quad X_1 x_5 Z_1^5 (\lambda x_6 \lambda Z_1^6 Z_2^6 Z_3^6. \\ & \quad \quad \quad \quad \quad \quad X_2 x_6 Z_1^6 (Z_2^6 0 Z_2^5) (\lambda x_7 \lambda Z_1^7 Z_2^7 Z_3^7. \\ & \quad \quad \quad \quad \quad \quad \quad X_1 x_7 Z_1^7 (\lambda x_8 \lambda Z_1^8 Z_2^8 Z_3^8. \\ & \quad \quad \quad \quad \quad \quad \quad \quad Y x_8 Z_1^8 (Z_2^8 1 Z_2^7) (Z_3^8 1 (Z_3^7 1 (Z_3^6 1 Z_3^5)))))))))) \end{aligned}$$

The above proof is the shortest normal proof of our formula. Other proofs may “generate” additional variables associated to various strings. However, repeated strings are “redundant”, i.e., they do not help to complete the proof.

2.2. The Mints hierarchy. We define classes of formulas Σ_n and Π_n by induction, beginning with $\Sigma_0 = \Pi_0$ being the set of quantifier-free formulas. The induction step can be expressed by the following pseudo-grammar:

- $\Sigma_{n+1} ::= \mathbf{a} \mid \Pi_n \mid \Pi_{n+1} \rightarrow \Sigma_{n+1}$
- $\Pi_{n+1} ::= \mathbf{a} \mid \Sigma_n \mid \Sigma_{n+1} \rightarrow \Pi_{n+1} \mid \forall x \Pi_{n+1}$

where the metavariable \mathbf{a} stands for an atom. In addition, we take:

- $\Delta_n = \Sigma_n \cap \Pi_n$.

For example, the formula $(\forall x P(x) \rightarrow Q) \rightarrow Q$ is in Π_1 , the formula $\forall x (\forall y R(y) \rightarrow P(x)) \rightarrow Q$ is in Σ_2 , and $\forall x P(x) \rightarrow (\forall y R(y) \rightarrow Q) \rightarrow Q$ is in Δ_2 . By an easy induction one proves that every Σ_n formula is classically equivalent to a prenex formula of type Σ_n (recall that the sans-serif Σ and Π refer to the ordinary classical hierarchy of prenex forms), and similarly for Π_n versus Π_n . The converse is not true in the following sense. Consider the formula $\varphi = \forall x ((P(x) \rightarrow R(x)) \rightarrow P(x)) \rightarrow P(x)$. As a classical tautology, φ is classically equivalent to an arbitrary quantifier-free tautology, i.e., it is classically equivalent to a formula in Σ_0 . But in the intuitionistic logic φ is not equivalent to any open (Σ_0) formula.

To prove that the Mints hierarchy is strict, we use an analogous result about classical logic. The following theorem follows from [20].

Theorem 2.4 (Rosen). *For each n there is a Σ_n formula φ_n which is not classically equivalent to any Π_n formula and there is a Π_n formula ψ_n such that ψ_n is not classically equivalent to any Σ_n formula. Both formulas are in a language with one binary predicate. \square*

Since conjunction and disjunction are classically definable from \rightarrow and \perp , it follows that Theorem 2.4 holds for the language with \rightarrow and \perp as the only propositional connectives. If we replace all existential quantifiers in a Σ_n (resp. Π_n) formula by their classical definitions in terms of \forall , \rightarrow and \perp , we obtain a formula which is *almost* a Σ_n (resp. Π_n) formula in our sense. Since intuitionistic provability implies classical provability, the formula φ_n in Theorem 2.4 must not be intuitionistically equivalent to any Π_n formula. This immediately

implies a hierarchy theorem for intuitionistic logic with \forall , \rightarrow and \perp and one binary predicate. To get rid of \perp , we use the following obvious lemma, where \vdash_c refers to classical provability.

Lemma 2.5. *Let $\Gamma \vdash_c \varphi$, and let p be a nullary relation symbol. Then*

$$\Gamma[p := \perp] \vdash_c \varphi[p := \perp].$$

Proof. Routine induction. □

Corollary 2.6. *For each n there is a Σ_n formula φ_n which is not classically equivalent to any Π_n formula and there is a Π_n formula ψ_n which is not classically equivalent to any Σ_n formula. The formulas φ_n and ψ_n are in a language with one binary and one nullary predicate.*

Proof. Let φ'_n and ψ'_n be the formulas from Theorem 2.4, and let φ''_n and ψ''_n be obtained respectively from φ'_n and ψ'_n , by replacing each \exists by $\neg\forall\neg$ and then eliminating the connectives \vee , \wedge , and \neg in a standard way. Finally, we replace all occurrences of \perp in φ''_n and ψ''_n with a new nullary predicate symbol, say p , and we denote the results by φ_n and ψ_n . Let ϑ be a Π_n formula. If φ_n and ϑ are classically equivalent then by Lemma 2.5 so are the formulas φ''_n and $\vartheta[p := \perp]$. Hence φ'_n is classically equivalent to a Σ_n formula, contradicting Theorem 2.4. A similar argument applies to ψ_n . □

Now, as an easy consequence we can state the following.

Theorem 2.7. *The Mints hierarchy is strict, that is, for each n there is a Σ_n formula φ_n which is not intuitionistically equivalent to any Π_n formula and there is a Π_n formula ψ_n which is not intuitionistically equivalent to any Σ_n formula. The formulas φ_n and ψ_n are in a language with one binary and one nullary predicate.* □

3. MACHINES AND TILINGS

To give a concise account of our lower bound results, we disguise Turing Machines as tiling problems, cf. [2, Chapter 3.1.1]. While the masquerade is quite obvious to unveil, it is still useful: some formulas become simpler. In the following two subsections, we define two forms of slightly unusual tiling puzzles. Deterministic puzzles of Section 3.1 are later used for the undecidability of Δ_2 in Section 4. The branching puzzle defined in Section 3.2 is used later in Section 6 for the lower bound for monadic Σ_1 (Section 6).

3.1. Deterministic tiling. Our (deterministic) *tiling puzzle* is defined as a quadruple

$$\mathcal{G} = \langle \mathcal{T}, \mathcal{R}, E, \text{OK} \rangle,$$

where \mathcal{T} is a finite set of *tiles*, $\mathcal{R} : \mathcal{T}^4 \rightarrow \mathcal{T}$ is a *tiling function*, and E, OK are different elements of \mathcal{T} . Such \mathcal{G} defines a unique *tiling* $\mathcal{G}^* : \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{T}$, as follows:

- $\mathcal{G}^*(m, n) = E$, when $n = 0$ or $m = 0$;
- $\mathcal{G}^*(m+1, n+1) = \mathcal{R}(K, L, M, N)$, where
 $K = \mathcal{G}^*(m, n+1)$, $L = \mathcal{G}^*(m, n)$, $M = \mathcal{G}^*(m+1, n)$, and $N = \mathcal{G}^*(m+2, n)$.

That is, the tile E is placed along the horizontal and vertical edges of the grid $\mathbb{N} \times \mathbb{N}$ and every other tile is determined by its neighbourhood consisting of four tiles: one tile to the left and three tiles below. This is illustrated by Fig. 2, where $T = \mathcal{R}(K, L, M, N)$. We say that \mathcal{G} is *solvable* when $\mathcal{G}^*(m, n) = \text{OK}$, for some numbers m, n . The following is unavoidable:

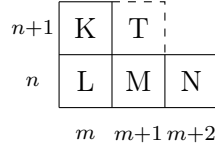


Figure 2: Result tile.

Lemma 3.1. *It is undecidable to determine if a given tiling puzzle is solvable.*

Proof. A routine reduction of the following problem:

Does a deterministic Turing Machine accept the empty input?

Row n in the tiling corresponds to the n -th step of a computation. □

Locations in tilings: Let $\mathcal{L}(m, n) = \{(k, l) \mid l \leq n \wedge k \leq m+n-l\}$. To place a tile at a location (m, n) , where $m, n > 0$, we must tile all locations in $\mathcal{L}(m, n)$, as illustrated in Figure 3, where the gray square is the location (m, n) . Define $(m, n) \preceq (k, l)$ when $\mathcal{L}(m, n) \subseteq \mathcal{L}(k, l)$.

Lemma 3.2. *The relation \preceq is a well-founded partial order.* □

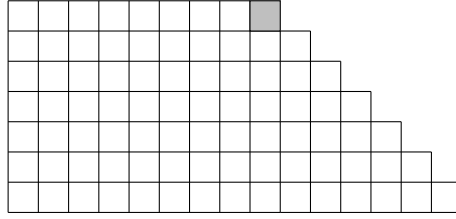


Figure 3: Dependency of locations.

3.2. Branching puzzle. We now generalize our definition of a tiling puzzle to account for branching (aka universal) computation, a phenomenon dual to nondeterminism. This will be needed in Section 6. A *branching Turing Machine* may divide its computation into multiple branches, each proceeding independently. The whole computation can thus be seen as a tree. We can imagine that every branch follows its own time line, so we deal with a tree-like, rather than linear, flow of computation.

The machine accepts when all these branches reach accepting states. For simplicity we assume that a branching machine divides the computation at *every* step (there are no ordinary deterministic states) and always into two: there is always a “left” and a “right” development. Therefore every computation branch can be identified by a sequence of binary choices. This resembles very much the behaviour of a deterministic tree automaton: the sequence of moves along any fixed branch is fully unique. In this respect, a branching machine is just a deterministic machine operating in a branching environment.

A *branching puzzle* is defined again as a tuple of the form

$$\mathcal{G} = \langle \mathcal{T}, \mathcal{R}, E, \text{OK} \rangle,$$

with the only difference that now the tiling function is $\mathcal{R} : \mathcal{T}^4 \rightarrow \mathcal{T}^2$. The *tiling* defined by \mathcal{G} is a function $\mathcal{G}^* : \mathbb{N} \times \{0, 1\}^* \rightarrow \mathcal{T}$, that is, the space to be tiled is $\mathbb{N} \times \{0, 1\}^*$. One can imagine a tiling of $\mathbb{N} \times \{0, 1\}^*$ as a full binary tree labeled by rows of tiles (the label of a node

$w \in \{0, 1\}^*$ is the sequence of tiles $\mathcal{G}^*(n, w)$, for all n). This tree represents a universally branching computation with all possible sequences of binary choices. The definition follows:

- $\mathcal{G}^*(n, w) = E$, when $n = 0$ or $w = \varepsilon$.
- $\mathcal{G}^*(m+1, wi) = \pi_i(\mathcal{R}(K_i, L, M, N))$, for $i = 0, 1$, where
 $K_i = \mathcal{G}^*(m, wi)$, $L = \mathcal{G}^*(m, w)$, $M = \mathcal{G}^*(m+1, w)$, and $N = \mathcal{G}^*(m+2, w)$;

A tiling \mathcal{G}^* determines, for every infinite path π in the tree $\{0, 1\}^*$, a tiling \mathcal{G}_π^* of $\mathbb{N} \times \mathbb{N}$, given by $\mathcal{G}_\pi^*(m, n) = \mathcal{G}^*(m, w)$, where $|w| = n$ and $w \subseteq \pi$. We call it a *local tiling* associated with π .

Let $s \in \mathbb{N}$. The puzzle \mathcal{G} is *s-solvable* iff, for every w with $|w| = s$, there is a prefix w' of w and a number $m \leq s$ such that $\mathcal{G}^*(m, w') = \text{OK}$. That is, an OK tile must be reached at every branch of the tree of length s and it must be at most the s -th tile in the row.

For technical reasons we also need a relativized notion of *s-solvability*. We say that \mathcal{G} is *s-solvable from v* when, for every w such that $v \subseteq w$ and $|w| = s$, there is a pair (m, w') with $m \leq s$, $w' \subseteq w$, and $\mathcal{G}^*(m, w') = \text{OK}$. (Then either $w' \subseteq v$ or $v \subseteq w' \subseteq w$.) We have:

Lemma 3.3. *A branching puzzle \mathcal{G} is s-solvable iff it is s-solvable from ε . It is s-solvable from a word v with $|v| < s$ if and only if it is s-solvable from both $v0$ and $v1$.* \square

Here is an analogue of Lemma 3.1:

Lemma 3.4. *The following problem is complete for universal exponential time (that is, co-NEXPTIME-complete):* Given a branching time puzzle \mathcal{G} and a number s (written in binary) determine if \mathcal{G} is *s-solvable*.

Proof. Fix a branching Turing Machine M working in time 2^{n^k} and an input word $a_1 a_2 \dots a_n$. Recall that all states of the machine are universal and the computation splits into two at each step. The encoding of the machine is quite natural: the number s is 2^{n^k} (this takes n^k space in binary) and the set of tiles is $\mathcal{T}_M = \{E, \text{OK}\} \cup \Sigma \cup (\Sigma \times Q)$, where Σ is the machine alphabet and Q is the set of states. Suppose for example that the machine divides the computation making these two moves when scanning a in state q :

- write b , move left, go to state p ;
- write c , move right, go to state r .

Fix some $* \in \Sigma$. We may now define $\mathcal{R}(x, x, y, (a, q)) = \langle (y, p), y \rangle$, $\mathcal{R}(b, (a, q), y, z) = \langle y, * \rangle$, $\mathcal{R}(\langle y, p \rangle, y, (a, q), z) = \langle b, * \rangle$, $\mathcal{R}(x, x, (a, q), z) = \langle *, c \rangle$, $\mathcal{R}(c, (a, q), y, z) = \langle *, (y, r) \rangle$, and $\mathcal{R}(\langle y, r \rangle, y, x, z) = \langle x, x \rangle$, and $\mathcal{R}(x, x, y, z) = \langle y, y \rangle$, for every $x, y, z \in \Sigma$ (assuming that $b \neq c$). Note that $*$ can be arbitrary — this value is irrelevant. The definition of \mathcal{R} must also ensure the proper positioning of tiles representing the input (in row number 1), etc. By induction with respect to (w, n) one proves that $\mathcal{G}^*(n, w)$ is the content of tape cell n at time $|w| - 1$ in one of the parallel computation branches. \square

4. UNDECIDABILITY FOR Δ_2

We encode a tiling puzzle $\mathcal{G} = \langle \mathcal{T}, \mathcal{R}, E, \text{OK} \rangle$ as a Δ_2 formula $\Phi_{\mathcal{G}}$ over the signature:

- nullary symbols: *start*, *loop*;
- unary relation symbol *add*;
- unary relation symbols T , for each tile $T \in \mathcal{T}$; including E ;
- unary relation symbols A, B , representing border positions;
- binary relation symbols H, V , representing horizontal and vertical neighbourhood.

The intuition is that object variables occurring in formulas may be interpreted as tile locations. Then $A(x)$ can be read as “ x belongs to the bottom row” and $B(x)$ as “ x belongs to the leftmost column”.¹ The intuitive meaning of $H(x, y)$ is “ x is to the left of y ” and $V(x, y)$ is understood as “ x is below y ”.

Below we show that \mathcal{G} is solvable if and only if $\Phi_{\mathcal{G}}$ has a proof. The reader has to be aware that the argument to follow is proof-theoretical rather than semantical. We are not concerned with the interpretation of our formulas in any model, but in their formal structure and in the mechanism of proof search. Every step in the construction of the tiling is encoded by an expansion of the proof environment: adding new tiles corresponds to adding more assumptions.

Let Θ be a set of formulas in the above signature. According to the intuition above, predicates H and V may determine the coordinates (m, n) of a variable x in the grid. In general, this is not always consistent, i.e., a variable x may have one or more pairs of coordinates in Θ . We define it more formally by induction with respect to (m, n) .

- If $A(x), B(x) \in \Theta$ then x has coordinates $(0, 0)$.
- If $H(x, y) \in \Theta$ and x has coordinates (m, n) then y has coordinates $(m + 1, n)$.
- If $V(x, y) \in \Theta$ and x has coordinates (m, n) then y has coordinates $(m, n + 1)$.

A finite set of formulas (i.e., an environment) Θ is *good* when all formulas in Θ are of the forms $A(x)$, $B(x)$, $H(x, y)$, $V(x, y)$, or $T(x)$, where $T \in \mathcal{T}$, and in addition:

- Each $x \in \text{FV}(\Theta)$ has exactly one pair of coordinates.
- For each $x \in \text{FV}(\Theta)$ with coordinates (m, n) , and every $T \in \mathcal{T}$,
 - $T(x) \in \Theta$ if and only if $\mathcal{G}^*(m, n) = T$;
 - $B(x) \in \Theta$ if and only if $m = 0$;
 - $A(x) \in \Theta$, if and only if $n = 0$.

The intuition is that a good environment consistently represents partial information about the tiling \mathcal{G}^* , with possible redundancy: several variables may have the same coordinates.

The formula $\Phi_{\mathcal{G}}$ to be constructed is of the form $\zeta_1 \rightarrow \dots \rightarrow \zeta_m \rightarrow \text{start}$, where some ζ_i are in Π_1 and others are in Σ_1 . Technically, it is convenient to define the environment $\Gamma_{\mathcal{G}} = \{\zeta_i \mid i = 1, \dots, m\}$ and consider the entailment problem $\Gamma_{\mathcal{G}} \vdash \text{start}$. For every “rule” of the form $\mathcal{R}(K, L, M, N) = T$, the set $\Gamma_{\mathcal{G}}$ contains the formula:

$$(0) \quad \forall xyzuv [K(y) \rightarrow L(z) \rightarrow M(u) \rightarrow N(v) \rightarrow V(z, y) \rightarrow H(z, u) \rightarrow H(u, v) \rightarrow \\ (\text{T}(x) \rightarrow H(y, x) \rightarrow V(u, x) \rightarrow \text{loop}) \rightarrow \text{add}(x)].$$

The intended meaning of the formula (0) is illustrated by Figure 4. Variables $xyzuv$ represent tile positions, and the assumptions $K(y), \dots, H(u, v)$ describe the situation in the tiling before placing tile T at x . Formula (0) provides a proof tactic which can be used towards a goal of the form $\text{add}(x)$ as follows. Find some $yzuv$ witnessing $K(y), \dots, H(u, v)$, and prove loop under the additional assumptions $T(x), H(y, x), V(u, x)$ which extend the proof environment to account for the new tile.

The other formulas in $\Gamma_{\mathcal{G}}$ are listed below. Observe that all quantifiers in formulas (0,2–4) are positive, while $\forall x$ in (1,5) are negative, and that in the formula $\Phi_{\mathcal{G}}$ all signs are reversed. Since there is no alternation of signs, we obtain that $\Phi_{\mathcal{G}}$ belongs to Δ_2 .

- (1) $\forall x (E(x) \rightarrow A(x) \rightarrow B(x) \rightarrow \text{loop}) \rightarrow \text{start}$;
- (2) $\forall x \forall y (E(y) \rightarrow A(y) \rightarrow (H(y, x) \rightarrow E(x) \rightarrow A(x) \rightarrow \text{loop}) \rightarrow \text{add}(x))$;
- (3) $\forall x \forall y (E(y) \rightarrow B(y) \rightarrow (V(y, x) \rightarrow E(x) \rightarrow B(x) \rightarrow \text{loop}) \rightarrow \text{add}(x))$;

¹We use “Asphalt” and “Barrier” as mnemonics.

y	x	
K	T	
L	M	N
z	u	v

Figure 4: Formula (0).

- (4) $\forall x (\text{OK}(x) \rightarrow \text{loop})$;
(5) $\forall x \text{add}(x) \rightarrow \text{loop}$.

The basic machinery here is as follows: to prove *loop* using (5) one needs to derive *add*(x), for a fresh x . This can be done using one of the proof tactics (0,2,3). Each of these tactics verifies some conditions, adds more assumptions, and brings back the proof goal *loop*. The iteration is started by an attempt to prove *start* using (1). An assumption of the form $\text{OK}(x)$ can be used to stop the iteration by applying (4). Before we state the next lemma, let us observe that good environments only consist of atoms, and targets of non-atomic formulas in $\Gamma_{\mathcal{G}}$ are *start*, *loop*, and *add*. Suppose that Θ is good and that α is a unary or binary atom other than *add*. It follows from Lemma 2.3 that $\Gamma_{\mathcal{G}}, \Theta \vdash \alpha$ is only possible when α actually belongs to Θ .

Lemma 4.1. *If $\Gamma_{\mathcal{G}}, \Theta \vdash P : \text{loop}$, for some good Θ , and some long normal proof P , then \mathcal{G} is solvable.*

Proof. We proceed by induction with respect to the length of P . Since *loop* is an atom, the long normal proof P must begin with a proof variable Y declared in $\Gamma_{\mathcal{G}}, \Theta$ so that its type ends with *loop* (cf. Lemma 2.3). If Y is of type (4) then $P = Yx'D$, where x' is an object variable and $\Gamma_{\mathcal{G}}, \Theta \vdash D : \text{OK}(x')$. Then $\text{OK}(x')$ must actually be in Θ . Hence $\mathcal{G}^*(m, n) = \text{OK}$, for some m, n .

Otherwise Y is of type (5) and $P = Y(\lambda x' F)$ with $\Gamma_{\mathcal{G}}, \Theta \vdash F : \text{add}(x')$ and x' not free in the environment $\Gamma_{\mathcal{G}}, \Theta$. Again, the term F must begin with a variable Z declared in $\Gamma_{\mathcal{G}}, \Theta$. If Z is of type (0) then $F = Zx'y'z'u'v'D_K D_L D_M D_N D_V D_H^1 D_H^2 (\lambda Z_1 Z_2 Z_3. D)$, where:

- Terms $D_K, D_L, D_M, D_N, D_V, D_H^1, D_H^2$ are respectively of types $\text{K}(y'), \text{L}(z'), \text{M}(u'), \text{N}(v'), \text{V}(z', y'), \text{H}(z', u'), \text{H}(u', v')$ in the environment $\Gamma_{\mathcal{G}}, \Theta$;
- $\Gamma_{\mathcal{G}}, \Theta, Z_1 : \text{T}(x'), Z_2 : \text{H}(y', x'), Z_3 : \text{V}(u', x') \vdash D : \text{loop}$;
- $\text{T} = \mathcal{R}(\text{K}, \text{L}, \text{M}, \text{N})$.

But if a long normal form has type $\text{K}(y')$ in $\Gamma_{\mathcal{G}}, \Theta$ then it must be a proof variable. The same holds for all the proofs mentioned in the first item above: these atoms must simply belong to Θ . Since Θ is good, we have $\mathcal{G}^*(m+1, n+1) = \text{T}$.

Let $\Theta' = \Theta, \text{T}(x'), \text{H}(y', x'), \text{V}(u', x')$. The environment Θ' is good, so the variables $y', z',$ and u' have only one pair of coordinates each. In addition, the presence of assumptions $\text{V}(z', y')$ and $\text{H}(z', u')$ forces that the coordinates of y', z', u' are of the form $(m, n+1)$, (m, n) , and $(m, n+1)$, respectively. Since $\text{H}(y', x'), \text{V}(u', x') \in \Theta'$, the added variable x' has coordinates $(m+1, n+1)$ in Θ' , and this is the only such pair. It follows that Θ' is a good environment, and we can apply induction to D because it is a proof of *loop* shorter than P .

Now suppose that $\Gamma_{\mathcal{G}}, \Theta \vdash F : \text{add}(x')$, where the long normal proof F begins with a variable Z of type (2). Then $F = Zx'y'D_E D_A (\lambda Z_1 Z_2 Z_3. D)$, where D_E and D_A are, respectively, of type $\text{E}(y')$ and $\text{A}(y')$, and

$$\Gamma_{\mathcal{G}}, \Theta, Z_1 : H(y', x'), Z_2 : E(x'), Z_3 : A(x') \vdash D : \text{loop}.$$

As in the previous case, the atoms $E(x')$ and $A(x')$ must occur in Θ . To apply induction it suffices to prove that the environment

$$\Theta' = \Theta, Z_1 : H(y', x'), Z_2 : E(x'), Z_3 : A(x')$$

is good. Since Θ is good, the variable y' has exactly one pair of coordinates $(m, 0)$. The new variable x' has the coordinates $(m + 1, 0)$ and this is its only pair of coordinates. We conclude that Θ' is good.

A long normal proof of $\text{add}(x')$ may also begin with a variable of type (3). Then the argument is similar as in case (2). \square

From Lemma 4.1 we immediately obtain:

Lemma 4.2. *If $\Gamma_{\mathcal{G}} \vdash \text{start}$ then \mathcal{G} is solvable.*

Proof. A long normal proof of start must be of the form $D = Z(\lambda x \lambda X Y V. D')$, for some variable Z of type (1) and some D' with

$$\Gamma_{\mathcal{G}}, X : E(x), Y : A(x), V : B(x) \vdash D' : \text{loop}.$$

The set $\Theta = \{E(x), A(x), B(x)\}$ is good and we apply Lemma 4.1. \square

Our next aim is to show the converse to Lemma 4.2. For the rest of this section we assume that \mathcal{G} is solvable with $\mathcal{G}^*(m_0, n_0) = \text{OK}$. For a good set Θ define

$$S_{\Theta} = \{(m, n) \mid \text{some } x \in \text{FV}(\Theta) \text{ has coordinates } (m, n)\}.$$

We say that a set Θ of formulas is *very good* when Θ is good and:

- The set S_{Θ} is a subset of $\mathcal{L}(m_0, n_0)$;
- For every $(m, n) \in S_{\Theta}$, exactly one $x \in \text{FV}(\Theta)$ has coordinates (m, n) ;
- If $x \in \text{FV}(\Theta)$ has coordinates $(m + 1, n)$ then some $H(y, x)$ is in Θ ;
- If $x \in \text{FV}(\Theta)$ has coordinates $(m, n + 1)$ then some $V(y, x)$ is in Θ .

A very good set represents the tile assignment without redundancy, and every non-zero location is “justified” by its neighbours occurring below and to the left of it.

Lemma 4.3. *If $\Theta \neq \emptyset$ is very good then $\Gamma_{\mathcal{G}}, \Theta \vdash \text{loop}$.*

Proof. The proof is by induction with respect to the cardinality of the set $\mathcal{L}(m_0, n_0) - S_{\Theta}$. In the base case we have $(m_0, n_0) \in S_{\Theta}$, whence $\text{OK}(x) \in \Theta$, for some x . We use the assumption (4) to derive loop .

For the induction step, let $(m', n') \in \mathcal{L}(m_0, n_0) - S_{\Theta}$ be minimal with respect to \preceq (cf. Lemma 3.2). Assume first that $m' = m + 1$ and $n' = 0$. By the minimality of (m', n') , there is a unique variable $y \in \text{FV}(\Theta)$ with coordinates $(m, 0)$ and with $E(y), A(y) \in \Theta$. Take a fresh variable x . Then $\Theta' = \Theta \cup \{H(y, x), E(x), A(x)\}$ is very good, whence $\Theta' \vdash \text{loop}$. That is, we have $\Gamma_{\mathcal{G}}, \Theta \vdash H(y, x) \rightarrow E(x) \rightarrow A(x) \rightarrow \text{loop}$. Using the assumption (2) we derive $\Gamma_{\mathcal{G}}, \Theta \vdash \text{add}(x)$. Since $x \notin \text{FV}(\Theta)$, we can generalize over x and obtain $\Gamma_{\mathcal{G}}, \Theta \vdash \forall x \text{add}(x)$. Now we use the assumption (5) to obtain $\Gamma_{\mathcal{G}}, \Theta \vdash \text{loop}$.

The case $m' = 0$ and $n' = n + 1$ is similar but we use assumption (3). Assume therefore that $m' = m + 1$ and $n' = n + 1$, for some m, n . By the minimality of (m', n') , there are variables $y, z, u, v \in \text{FV}(\Theta)$ with coordinates $(m, n + 1), (m, n), (m + 1, n), (m + 2, n)$. These variables are unique because Θ is very good. Also we have $K(y), L(z), M(u), N(v) \in \Theta$, for some unique choice of K, L, M, N . In addition, since Θ is very good, we must also have in Θ the assumptions $V(z, y), H(z, u), H(u, v)$. Let $\Theta' = \Theta \cup \{T(x), H(y, x), V(u, x)\}$, where x is a fresh variable, and $T = \mathcal{R}(K, L, M, N)$. Then $\mathcal{G}^*(m', n') = T$. The environment Θ' is very

good, because x is the unique variable with coordinates $(m + 1, n + 1)$. By the induction hypothesis, $\Gamma_{\mathcal{G}}, \Theta' \vdash \text{loop}$, whence $\Gamma_{\mathcal{G}}, \Theta \vdash \text{T}(x) \rightarrow \text{H}(y, x) \rightarrow \text{V}(u, x) \rightarrow \text{loop}$. Using the assumption (0) we can now derive $\Gamma_{\mathcal{G}}, \Theta \vdash \text{add}(x)$. But we actually have $\Gamma_{\mathcal{G}}, \Theta \vdash \forall x \text{add}(x)$, because x is not free in $\Gamma_{\mathcal{G}}, \Theta$. Hence $\Gamma_{\mathcal{G}}, \Theta \vdash \text{loop}$ by an application of (5). \square

Lemma 4.4. *If \mathcal{G} is solvable then $\Gamma_{\mathcal{G}} \vdash \text{start}$.*

Proof. The set $\Theta = \{\text{E}(x), \text{A}(x), \text{B}(x)\}$ is very good, so $\Gamma_{\mathcal{G}}, \text{E}(x), \text{A}(x), \text{B}(x) \vdash \text{loop}$ holds by Lemma 4.3. Hence $\Gamma_{\mathcal{G}} \vdash \text{E}(x) \rightarrow \text{A}(x) \rightarrow \text{B}(x) \rightarrow \text{loop}$. Using (1) one derives $\Gamma_{\mathcal{G}} \vdash \text{start}$. \square

Theorem 4.5. *Provability in Δ_2 is undecidable.*

Proof. By Lemma 3.1, solvability of tiling puzzles is undecidable. Lemmas 4.2 and 4.4 give an effective reduction from the tiling puzzle problem to provability. \square

A finite signature. Observe that our proof of Theorem 4.5 uses as many predicate symbols as there are tiles, i.e., it applies to an infinite signature. We now briefly explain how it can be adjusted to work for a finite language. First, redefine the tiling puzzle so that $\mathcal{G} = \langle \mathcal{T}, \mathcal{R}, \text{E}, n, \text{T}_1, \dots, \text{T}_n, \text{OK} \rangle$, where $n \in \mathbb{N}$ and $\text{T}_1, \dots, \text{T}_n \in \mathcal{T}$ (possibly with repetitions). This is to account for a non-empty input word. Require that the tiling satisfies $\mathcal{G}^*(i, 0) = \text{T}_i$, for all $i = 1, \dots, n$. Using a universal Turing Machine (which has a fixed number of states and uses a fixed alphabet) prove that for some $M \in \mathbb{N}$ the problem of solvability is undecidable for the modified puzzles with at most M tiles. This reduces the number of necessary predicates to a finite level. The remaining construction is essentially the same, but one has to adjust formula (1) as follows:

- $\forall x_0 \dots x_n (\text{E}(x_0) \rightarrow \text{B}(x_0) \rightarrow \text{A}(x_0) \rightarrow \dots \rightarrow \text{A}(x_n) \rightarrow \text{T}_1(x_1) \rightarrow \dots \rightarrow \text{T}_n(x_n) \rightarrow \text{H}(x_0, x_1) \rightarrow \dots \rightarrow \text{H}(x_{n-1}, x_n) \rightarrow \text{loop}) \rightarrow \text{start}$.

4.1. **Monadic Δ_2 .** Our proof of Theorem 4.5 used binary relation symbols. We now show how to eliminate them by a syntactic translation. This is possible, because we only used formulas of a very simple shape. We say that a formula φ is *easy* when it is an atom, or when *target*(φ) is unary or nullary and one of the following holds:

- $\varphi = \forall x \psi$, where ψ is easy;
- $\varphi = \psi \rightarrow \vartheta$, where ψ and ϑ are easy.

Observe that the set $\Gamma_{\mathcal{G}}$ in Section 4 consists of easy formulas.

Let **1** and **2** be fresh unary relation symbols (i.e., not occurring in the source language). With every binary relation symbol P we associate another fresh nullary symbol \mathbf{p} . We define $\overline{\text{P}(x, y)} = \mathbf{1}(x) \rightarrow \mathbf{2}(y) \rightarrow \mathbf{p}$, for binary P , and $\overline{\text{P}(x)} = \text{P}(x)$, $\overline{\text{P}} = \text{P}$, when P is unary or nullary. Then, by induction, define $\overline{\forall x \varphi} = \forall x \overline{\varphi}$, and $\overline{\varphi \rightarrow \psi} = \overline{\varphi} \rightarrow \overline{\psi}$.

Lemma 4.6. *The translation $\psi \mapsto \overline{\psi}$ has the following properties:*

- $\text{FV}(\overline{\psi}) = \text{FV}(\psi)$;
- $\overline{\psi[y/x]} = \overline{\psi}[y/x]$;
- If $\overline{\varphi} = \overline{\psi}$ then $\varphi = \psi$;
- If ψ is easy then so is $\overline{\psi}$.

Proof. Routine induction. \square

Lemma 4.7. *Let Σ consist of binary atoms and let targets of all formulas in Γ be nullary or unary. Then $\overline{\Gamma}, \overline{\Sigma} \vdash \overline{P(x, y)}$ implies $P(x, y) \in \Sigma$.*

Proof. We have $\overline{\Gamma}, \overline{\Sigma}, \mathbf{1}(x), \mathbf{2}(y) \vdash \mathbf{p}$. No formula in $\overline{\Gamma}$ may end with \mathbf{p} , thus a long normal proof of \mathbf{p} must begin with an element of $\overline{\Sigma}$: a variable of type $\overline{Q(u, v)} = \mathbf{1}(u) \rightarrow \mathbf{2}(v) \rightarrow \mathbf{q}$. Then $\mathbf{q} = \mathbf{p}$, i.e., $P = Q$, and we have $\overline{\Gamma}, \overline{\Sigma}, \mathbf{1}(x), \mathbf{2}(y) \vdash \mathbf{1}(u)$ and $\overline{\Gamma}, \overline{\Sigma}, \mathbf{1}(x), \mathbf{2}(y) \vdash \mathbf{2}(v)$. There is no other way to prove $\mathbf{1}(u)$ but to use the assumption $\mathbf{1}(x)$. Hence, $x = u$, and similarly we also obtain $y = v$. Thus, $P(x, y) = Q(u, v) \in \Sigma$. \square

Lemma 4.8. *If $\overline{\Gamma} \vdash \overline{\varphi}$, where φ and all formulas in Γ are easy, then $\Gamma \vdash \varphi$.*

Proof. A quasi-long eliminator is a term of the form $XE_1 \dots E_m$, where X is a proof variable and every E_i is either an lnf or an object variable. Observe that if $\overline{\Gamma} \vdash M : \tau$, where M is a quasi-long eliminator, then either $\tau = \overline{\varphi}$, for some φ , or $\tau = \mathbf{2}(y) \rightarrow \mathbf{p}$, or $\tau = \mathbf{p}$, for some \mathbf{p} and y . In the last two cases, we have $M = M'N_1$ or $M = M'N_1N_2$, with $M' : \overline{P(x, y)}$, and $N_1 : \mathbf{1}(x)$, and $N_2 : \mathbf{2}(y)$, for some x and y .

Let now $\overline{\Gamma} \vdash M : \overline{\varphi}$, where M is an lnf or a quasi-long eliminator. We prove that $\Gamma \vdash \varphi$, by induction with respect to M . The case of a variable is obvious.

Let $M = \lambda Z. N$. Without loss of generality we can assume that $\varphi = \psi \rightarrow \vartheta$, because the case of $\varphi = P(x, y)$ follows from Lemma 4.7. Then $\overline{\Gamma}, Z : \overline{\psi} \vdash N : \overline{\vartheta}$. By the induction hypothesis for N we have $\Gamma, \psi \vdash \vartheta$, whence $\Gamma \vdash \varphi$.

If $M = \lambda y. N$ (where we can assume y is fresh) then $\overline{\varphi} = \forall y \tau$, which means that $\varphi = \forall y \psi$ with $\overline{\psi} = \tau$. We have $\overline{\Gamma} \vdash N : \overline{\psi}$, so $\Gamma \vdash \psi$ and thus $\Gamma \vdash \varphi$ by generalization.

If $\overline{\Gamma} \vdash X\overline{E}N : \overline{\varphi}$ then the type of $X\overline{E}$ must be of the form $\overline{\psi} \rightarrow \overline{\varphi}$, because $\overline{\varphi}$ is neither of the form $\mathbf{2}(y) \rightarrow \mathbf{p}$ nor \mathbf{p} . By the induction hypothesis, both $\psi \rightarrow \varphi$ and ψ are provable, and so must be φ .

If $\overline{\Gamma} \vdash X\overline{E}y : \overline{\varphi}$, where y is an object variable, then $\overline{\Gamma} \vdash X\overline{E} : \forall x \tau$, for some τ with $\overline{\varphi} = \tau[y/x]$. Since $X\overline{E}$ is a quasi-long eliminator, we must have $\forall x \tau = \overline{\forall x \psi} = \forall x \overline{\psi}$, and $\overline{\varphi} = \overline{\psi}[y/x] = \overline{\psi}[y/x]$. Hence $\varphi = \psi[y/x]$. We apply induction to $X\overline{E}$. \square

The converse to Lemma 4.8 is obvious. Since all formulas used in our coding are easy, we can restate Lemmas 4.2 and 4.4 using $\overline{\Gamma}_{\mathcal{G}}$ instead of $\Gamma_{\mathcal{G}}$. We conclude with:

Theorem 4.9. *It is undecidable whether a Δ_2 formula with unary predicates is provable.* \square

Generalization: The translation $\varphi \mapsto \overline{\varphi}$ can be easily generalized to predicates of any fixed arity $n \geq 2$, by introducing n auxiliary symbols $\mathbf{1}, \mathbf{2}, \dots, \mathbf{n}$ and setting

$$\overline{P(x_1, x_2, \dots, x_n)} = \mathbf{1}(x_1) \rightarrow \mathbf{2}(x_2) \rightarrow \dots \rightarrow \mathbf{n}(x_n) \rightarrow \mathbf{p}.$$

It is convenient to assume without loss of generality that all many-argument predicates are of the same arity n . Then the proof of the following is virtually the same as in the binary case.

Proposition 4.10. *Let φ and all formulas in Γ be easy. Then $\Gamma \vdash \varphi$ iff $\overline{\Gamma} \vdash \overline{\varphi}$.*

5. EXPSPACE-COMPLETENESS FOR Σ_1

The lower bound is obtained by encoding the halting problem for bus machines [27] into the entailment problem for Σ_1 . A bus machine is an alternating computing device operating on a finite word (bus) of a fixed length. At every step the whole content of the bus is updated according to one of the instructions of the machine. In addition new instructions may be created each time and those can be used in later steps. A precise definition is as follows.

A *simple switch* over a finite alphabet \mathcal{A} is a pair of elements of \mathcal{A} , written $a \triangleright b$. A *labeled switch* is a quadruple, written $a \triangleright b(c \triangleright d)$, where the simple switch $c \triangleright d$ is the *label*. Finally, a *branching switch* is a triple, written $a \triangleright b \times c$.

A *bus machine* is a tuple $\mathcal{M} = \langle \mathcal{A}, m, w_0, w_1, \mathcal{I} \rangle$, where \mathcal{A} is a finite alphabet, $m > 0$ is the *bus length* of \mathcal{M} (the length of the words processed), w_0 and w_1 are words of length m over \mathcal{A} , called the *initial* and *final word*, respectively, and \mathcal{I} is a set of *global instructions*.

Every global instruction is an m -tuple $\mathbb{I} = \langle I_1, \dots, I_m \rangle$ of sets of switches. Switches in I_i are meant to act on the i -th symbol of the bus. It is required that all switches in a given instruction \mathbb{I} are of the same kind: either all are simple, or all are labeled, or all are branching. Therefore we classify instructions as simple, labeled, and branching. A *local instruction* is a special case of a simple instruction with singleton sets at all coordinates.

A *configuration* of \mathcal{M} is a pair $\langle w, \mathcal{J} \rangle$, where w is a word over \mathcal{A} of length m , and \mathcal{J} is a set of local instructions. The *initial* configuration is $\langle w_0, \emptyset \rangle$, and any configuration of the form $\langle w_1, \mathcal{J} \rangle$ is called *final*.

Suppose that $\mathbb{I} = \langle I_1, \dots, I_m \rangle$, and let $w = a_1 \dots a_m$ and $w' = b_1 \dots b_m$, $w'' = c_1 \dots c_m$. Transitions of \mathcal{M} according to \mathbb{I} are defined as follows:

- If \mathbb{I} is a simple instruction, and for every $i \leq m$ the switch $a_i \triangleright b_i$ belongs to I_i , then $\langle w, \mathcal{J} \rangle \Rightarrow_{\mathbb{I}, \mathcal{M}}^{\mathbb{I}} \langle w', \mathcal{J} \rangle$;
- If \mathbb{I} is a labeled instruction and $a_i \triangleright b_i(c_i \triangleright d_i)$ belongs to I_i , for every $i \leq m$, then $\langle w, \mathcal{J} \rangle \Rightarrow_{\mathbb{I}, \mathcal{M}}^{\mathbb{I}} \langle w', \mathcal{J}' \rangle$, where $\mathcal{J}' = \mathcal{J} \cup \{ \langle \{c_1 \triangleright d_1\}, \dots, \{c_m \triangleright d_m\} \rangle \}$;
- If \mathbb{I} is a branching instruction, and the switch $a_i \triangleright b_i \times c_i$ is in I_i , for every $i \leq m$, then $\langle w, \mathcal{J} \rangle \Rightarrow_{\mathbb{I}, \mathcal{M}}^{\mathbb{I}} (\langle w', \mathcal{J} \rangle, \langle w'', \mathcal{J} \rangle)$. (Now the relation $\Rightarrow_{\mathbb{I}, \mathcal{M}}^{\mathbb{I}}$ has three arguments.)

The notion of an accepting configuration of a bus machine is defined recursively. We say that a configuration $\langle w, \mathcal{J} \rangle$ is *eventually accepting* if it is either a final configuration, or

- There is a non-branching instruction $\mathbb{I} \in \mathcal{I} \cup \mathcal{J}$, with $\langle w, \mathcal{J} \rangle \Rightarrow_{\mathbb{I}, \mathcal{M}}^{\mathbb{I}} \langle w', \mathcal{J}' \rangle$, where $\langle w', \mathcal{J}' \rangle$ is eventually accepting, or
- There is a branching instruction $\mathbb{I} \in \mathcal{I} \cup \mathcal{J}$ such that $\langle w, \mathcal{J} \rangle \Rightarrow_{\mathbb{I}, \mathcal{M}}^{\mathbb{I}} (\langle w', \mathcal{J} \rangle, \langle w'', \mathcal{J} \rangle)$, where both $\langle w', \mathcal{J} \rangle$ and $\langle w'', \mathcal{J} \rangle$ are eventually accepting.

The machine \mathcal{M} *accepts* iff the initial configuration is eventually accepting. As usual with alternating machines, an accepting computation of a bus machine should be imagined as a tree with final configurations at all leaves and branching transitions at branching nodes.

Example 5.1. This example, inspired by [13], is from [19]. Let $\mathcal{A} = \{a, b, c, d\}$, and let

$$\begin{aligned} I^+ &= \{a \triangleright b(c \triangleright d)\}, I^- = \{b \triangleright a(d \triangleright c)\}, \\ I &= \{a \triangleright a(c \triangleright c), b \triangleright b(d \triangleright d)\}, I^* = \{b \triangleright c\} \end{aligned}$$

Consider $\mathcal{M} = \langle \mathcal{A}, 4, aaaa, dddd, \mathcal{I} \rangle$, where \mathcal{I} consists of the following tuples:

$$\langle I, I, I, I^+ \rangle, \langle I, I, I^+, I^- \rangle, \langle I, I^+, I^-, I^- \rangle, \langle I^+, I^-, I^-, I^- \rangle, \langle I^*, I^*, I^*, I^* \rangle.$$

The machine \mathcal{M} behaves in a deterministic way, for example the only instruction applicable in the initial configuration $\langle aaaa, \emptyset \rangle$ is $\langle I, I, I, I^+ \rangle$. Executing it yields $\langle aaab, \{I_0\} \rangle$,

where I_0 is the local instruction $\langle \{c \triangleright c\}, \{c \triangleright c\}, \{c \triangleright c\}, \{c \triangleright d\} \rangle$. The latter can be used later to change a configuration of the form $\langle cccc, \mathcal{J} \rangle$ into $\langle cccd, \mathcal{J} \rangle$. But now the machine must execute $\langle I, I, I^+, I^- \rangle$ and enter $\langle aaba, \{I_0, I_1\} \rangle$, where $I_1 = \langle \{c \triangleright c\}, \{c \triangleright c\}, \{c \triangleright d\}, \{d \triangleright c\} \rangle$.

In the first phase of computation only global instructions are executed and all words over $\{a, b\}$ appear on the bus in the lexicographic order. Every application of a global instruction creates a new unique local instruction. After arriving at $bbbb$, the machine rewrites the bus to $cccc$ using $\langle I^*, I^*, I^*, I^* \rangle$ and then executes one by one all the local instructions, eventually reaching the final $dddd$. The total number of steps is $2 \cdot 2^4 - 1$; also the number of local instructions is exponential and so is the (implicit) space needed to store them.

Theorem 5.2 ([27]). *The halting problem for bus machines (“Does a given machine accept?”) is EXPSPACE-complete.* \square

Given a bus machine $\mathcal{M} = \langle \mathcal{A}, m, w_0, w_1, \mathcal{I} \rangle$, we construct (in LOGSPACE) a set of universal formulas $\Gamma_{\mathcal{M}}$ and an open formula $\alpha_{\mathcal{M}}$ such that $\Gamma_{\mathcal{M}} \vdash \alpha_{\mathcal{M}}$ if and only if \mathcal{M} halts. The free variables in $\Gamma_{\mathcal{M}}$ and $\alpha_{\mathcal{M}}$ are identified with the symbols in \mathcal{A} and the number, as well as arity, of relation symbols in our formulas also depend on \mathcal{M} . The main relation symbol Bus is m -ary and it is intended to represent the content of the bus. The obvious convention is to write $\text{Bus}(w)$ for $\text{Bus}(a_1, \dots, a_m)$, when $w = a_1 \dots a_m$ and \vec{a} for $a_1 a_2 \dots a_m$.

The formula $\alpha_{\mathcal{M}}$ is $\text{Bus}(w_0)$, and $\text{Bus}(w_1)$ is a member of $\Gamma_{\mathcal{M}}$. The idea is that a proof of $\text{Bus}(w_0)$ succeeds when every branch of a computation can terminate by calling the axiom $\text{Bus}(w_1)$.

We associate binary (resp. ternary, quaternary) predicate symbols I with sets I of simple (resp. branching, labeled) switches occurring in the instructions of \mathcal{M} . Then for every simple switch $a \triangleright b$ in I , the atomic formula $I(a, b)$ is placed in $\Gamma_{\mathcal{M}}$, and similarly for branching and labeled switches. For example, the set I in Example 5.1 yields two assumptions $I(a, a, c, c)$ and $I(b, b, d, d)$.

In $\Gamma_{\mathcal{M}}$ there are also formulas $\psi_{\mathbb{I}}$ for all global instructions \mathbb{I} in \mathcal{I} . In case of a simple instruction $\mathbb{I} = \langle I_1, \dots, I_m \rangle$, the formula takes the form:

$$(1) \quad \psi_{\mathbb{I}} = \forall \vec{x} \vec{y} (I_1(x_1, y_1) \rightarrow \dots \rightarrow I_m(x_m, y_m) \rightarrow \text{Bus}(\vec{y}) \rightarrow \text{Bus}(\vec{x})).$$

If $\mathbb{I} = \langle I_1, \dots, I_m \rangle$ is a labeled instruction, then:

$$(2) \quad \psi_{\mathbb{I}} = \forall \vec{x} \vec{y} \vec{z} \vec{u} (I_1(x_1, y_1, z_1, u_1) \rightarrow \dots \rightarrow I_m(x_m, y_m, z_m, u_m) \rightarrow ((\text{Bus}(\vec{u}) \rightarrow \text{Bus}(\vec{z})) \rightarrow \text{Bus}(\vec{y})) \rightarrow \text{Bus}(\vec{x})).$$

Finally, for a branching instruction $\mathbb{I} = \langle I_1, \dots, I_m \rangle$, we take:

$$(3) \quad \psi_{\mathbb{I}} = \forall \vec{x} \vec{y} \vec{z} (I_1(x_1, y_1, z_1) \rightarrow \dots \rightarrow I_m(x_m, y_m, z_m) \rightarrow \text{Bus}(\vec{z}) \rightarrow \text{Bus}(\vec{y}) \rightarrow \text{Bus}(\vec{x})).$$

A local instruction J may be identified with a rewrite rule of the form $w \Rightarrow v$. Such a rule will be represented as a formula φ_J of the form $\text{Bus}(v) \rightarrow \text{Bus}(w)$. We define $\Gamma_{\mathcal{J}} = \{\varphi_J \mid J \in \mathcal{J}\}$.

To see the motivation, suppose we want to derive $\Gamma_{\mathcal{M}} \vdash \text{Bus}(bbbb)$, where \mathcal{M} is as in Example 5.1. We use the formula $\psi_{\langle I^*, I^*, I^*, I^* \rangle}$:

$$\forall \vec{x} \vec{y} (I^*(x_1, y_1) \rightarrow I^*(x_2, y_2) \rightarrow I^*(x_3, y_3) \rightarrow I^*(x_4, y_4) \rightarrow \text{Bus}(\vec{y}) \rightarrow \text{Bus}(\vec{x})),$$

instantiated by substituting b for x_i and c for y_i . Since the assumption $I^*(b, c)$ is in $\Gamma_{\mathcal{M}}$, the task of proving $\text{Bus}(bbbb)$ is reduced to proving $\text{Bus}(cccc)$.

Lemma 5.3. *A configuration $\langle w, \mathcal{J} \rangle$ is eventually accepting iff the judgment*

$$\Gamma_{\mathcal{M}}, \Gamma_{\mathcal{J}} \vdash \text{Bus}(w)$$

is derivable.

Proof. From left to right the proof is by induction with respect to the definition of an eventually accepting configuration. Let $\langle w, \mathcal{J} \rangle$ be eventually accepting. If it is final, the proof is trivial, because $\text{Bus}(w_1) \in \Gamma_{\mathcal{M}}$. Otherwise, assume for example that $\langle w, \mathcal{J} \rangle \Rightarrow_{\mathcal{M}}^{\mathbb{I}} \langle w', \mathcal{J}' \rangle$, where $\mathbb{I} = \langle I_1, \dots, I_m \rangle$ is a labeled instruction, and $\langle w', \mathcal{J}' \rangle$ is eventually accepting. Then $\mathcal{J}' = \mathcal{J} \cup \{J\}$, where J is a new local instruction. By the induction hypothesis we have $\Gamma_{\mathcal{M}}, \Gamma_{\mathcal{J}}, \varphi_J \vdash \text{Bus}(w')$. It follows that $\Gamma_{\mathcal{M}}, \Gamma_{\mathcal{J}} \vdash \varphi_J \rightarrow \text{Bus}(w')$. For $j = 1, \dots, m$, let $a_j \triangleright b_j (c_j \triangleright d_j)$ be the switches used in this step. Then $w = a_1 \dots a_m$, $w' = b_1 \dots b_m$, and $\varphi_J = \text{Bus}(d_1 \dots d_m) \rightarrow \text{Bus}(c_1 \dots c_m)$. Hence $\Gamma_{\mathcal{M}}, \Gamma_{\mathcal{J}} \vdash (\text{Bus}(\vec{d}) \rightarrow \text{Bus}(\vec{c})) \rightarrow \text{Bus}(\vec{b})$. We have all the $I_j(a_j, b_j, c_j, d_j)$ in $\Gamma_{\mathcal{M}}$, so we prove $\text{Bus}(\vec{a})$ using the appropriate axiom (2) instantiated with $\vec{x} := \vec{a}$, $\vec{y} := \vec{b}$, $\vec{z} := \vec{c}$, $\vec{u} := \vec{d}$. Other cases are similar.

The proof in the direction from right to left is by induction with respect to the length of long normal proofs. Assume that $\Gamma_{\mathcal{M}}, \Gamma_{\mathcal{J}} \vdash \text{Bus}(w)$. If w is not final then a long normal proof must begin with a variable of type (1), (2), or (3). Suppose for example that (3) is the case. For some instantiation $\vec{x} := \vec{a} = w$, $\vec{y} := \vec{b}$, $\vec{z} := \vec{c}$, there are proofs of $I_i(a_i, b_i, c_i)$ and of $\text{Bus}(\vec{b})$ and $\text{Bus}(\vec{c})$. A proof of $I_i(a_i, b_i, c_i)$ is only possible when $I_i(a_i, b_i, c_i)$ actually occurs in $\Gamma_{\mathcal{M}}$. This is because there are no other assumptions with target I_i . In particular this proves that variables b_i, c_i do correspond to actual bus symbols. Since $\text{Bus}(\vec{b})$ and $\text{Bus}(\vec{c})$ are provable, it follows from the induction hypothesis that $\langle \vec{b}, \mathcal{J} \rangle$ and $\langle \vec{c}, \mathcal{J} \rangle$ are eventually accepting. Therefore also $\langle w, \mathcal{J} \rangle$ is eventually accepting. \square

5.1. An upper bound for Σ_1 . A judgment of the form $\Gamma \vdash \varphi$, where φ is a Σ_1 formula and all assumptions in Γ are Π_1 formulas, is called a Σ_1 judgment. Observe that normal proofs of Σ_1 judgments are of the forms:

- a) $\Gamma \vdash \lambda X : \alpha. M : \alpha \rightarrow \beta$;
- b) $\Gamma \vdash X M_1 \dots M_r : \beta$,

where M is a normal proof term and each M_i , for $i = 1, \dots, r$, is a normal proof term or an object variable. Proofs of shape (b) are called *eliminators*. We say that N' is an *instance* of N when $N' = N[\vec{x} := \vec{y}]$, for some object variables \vec{x}, \vec{y} . The following is an easy consequence of Lemma 2.1.

Lemma 5.4. *Fix an object variable x_0 , and let $\mathcal{W} = \text{FV}(\Gamma) \cup \text{FV}(\varphi) \cup \{x_0\}$. If $\Gamma \vdash N : \varphi$ then $\Gamma \vdash N' : \varphi$, for some instance N' of N such that $\text{FV}(N') \subseteq \mathcal{W}$.*

Proof. Let \vec{x} be the list of all variables in $\text{FV}(N) - \mathcal{W}$, and let \vec{y} be any variables in \mathcal{W} . (The latter is nonempty because of x_0 .) Then $\Gamma[\vec{x} := \vec{y}] \vdash N[\vec{x} := \vec{y}] : \varphi[\vec{x} := \vec{y}]$, by Lemma 2.1. But variables \vec{x} are neither free in Γ nor in φ , whence $\Gamma[\vec{x} := \vec{y}] = \Gamma$ and $\varphi[\vec{x} := \vec{y}] = \varphi$. \square

Note that if $\text{FV}(\Gamma) \cup \text{FV}(\varphi) \neq \emptyset$ then Lemma 5.4 yields $\text{FV}(N') \subseteq \text{FV}(\Gamma) \cup \text{FV}(\varphi)$.

Lemma 5.5. *Let $\Gamma \vdash N : \varphi$, where Γ consists of Π_1 formulas and N is normal. Assume in addition that either N is an eliminator or φ is a Σ_1 formula. Then the term N contains no occurrences of object abstraction. In addition, if N is an eliminator then φ is in Π_1 .*

Proof. Induction with respect to N . If $N = X$ then the type of X is in Π_1 , because X is declared in Γ .

If $N = \lambda X : \psi. P$ then ψ is in Π_1 and $\Gamma, X : \psi \vdash P : \vartheta$, for some $\vartheta \in \Sigma_1$. We use the induction hypothesis for P . Case $N = \lambda x N'$ is impossible. If $N = X \vec{N} M$, where M is

a proof term, then we have $\Gamma \vdash X\vec{N} : \psi \rightarrow \varphi$ and $\Gamma \vdash M : \psi$, for some ψ . Since $X\vec{N}$ is an eliminator, the formula $\psi \rightarrow \varphi$ is in Π_1 and so must be φ , while ψ is in Σ_1 . We apply induction to $X\vec{N}$ and M .

Finally, if $N = X\vec{N}y$, where y is an object variable, then we apply induction to $X\vec{N}$. \square

Lemma 5.6. *If $\Gamma \vdash M : \varphi$ then $\text{FV}(\varphi) \subseteq \text{FV}(\Gamma) \cup \text{FV}(M)$.*

Proof. Easy induction with respect to M . \square

Let \mathcal{W} be a set of variables. If $\text{FV}(\Gamma) \cup \text{FV}(\varphi) \cup \text{FV}(M) \subseteq \mathcal{W}$ then we say that $\Gamma \vdash M : \varphi$ is a \mathcal{W} -judgment. A judgment is \mathcal{W} -derivable when it is derivable using the rules in Figure 1 restricted to \mathcal{W} -judgments.

Lemma 5.7. *Let $\Gamma \vdash M : \varphi$ be a provable \mathcal{W} -judgment. If M contains no object abstraction then $\Gamma \vdash M : \varphi$ is \mathcal{W} -derivable.*

Proof. Easy induction with respect to M . In case of application one uses Lemma 5.6. \square

Lemma 5.8. *The decision problem for Σ_1 formulas is solvable in EXPSpace.*

Proof. To find a proof of a given Σ_1 formula φ one uses an obvious generalization of the Ben-Yelles algorithm [26] for simple types. It follows from Lemma 5.5 that a normal inhabitant N of a Σ_1 formula φ must not contain any object abstraction. In addition, by Lemma 5.4, one can assume that free variables of N are all in the set $\mathcal{W} = \text{FV}(\varphi) \cup \{x_0\}$. (The variable x_0 is added to make sure that the set is not empty.) By Lemma 5.7, the judgment $\vdash N : \varphi$ is \mathcal{W} -derivable. Therefore the algorithm needs only to consider judgments $\Gamma' \vdash M : \varphi$ where all object variables are in \mathcal{W} . The number of different formulas in Γ' is thus at most exponential in the size n of φ . (With at most n variables, every subformula of φ has at most n^n instances.) Using the same argument as for simple types we therefore obtain an alternating exponential time algorithm. \square

Theorem 5.9. *The decision problem for Σ_1 is EXPSpace-complete.*

Proof. Lemma 5.3 reduces the halting problem for bus machines to provability in Σ_1 . The upper bound is provided by Lemma 5.8. \square

6. ARITY-BOUNDED Σ_1

The undecidability of Δ_2 holds even if we require that all predicates in formulas are unary. Technically, it is the case because the formulas used in the proof are easy, and we can apply the translation defined in Section 4.1. But the proof of the EXPSpace-hardness of Σ_1 (Lemma 5.3) uses non-easy formulas of unbounded arity.

It turns out that the Σ_1 decision problem is actually “easier” if we set any fixed bound on the arity of formulas. For every such bound, in particular in the monadic case, the problem turns out only *co*-NEXPTIME-complete.

6.1. The lower bound. To obtain the *co*-NEXPTIME lower bound we encode a given branching puzzle \mathcal{G} and a constant s as an entailment problem $\Gamma_{\mathcal{G}} \vdash \text{start}$. This is partly similar to the construction in Section 4, in particular all formulas in $\Gamma_{\mathcal{G}}$ are easy. In addition, all these formulas are either quantifier-free or universal. From now on we assume that s is fixed. The idea of the encoding can easily be explained if we assume for a while that the language of arithmetic is in our disposal. Then $\Gamma_{\mathcal{G}}$ could be composed of the following assumptions:

Tiling step $\mathcal{G}(K, L, M, N) = \langle T, U \rangle$:

$$\begin{aligned} (0_l) \quad & \forall mt (K(m, t+1) \rightarrow L(m, t) \rightarrow M(m+1, t) \rightarrow N(m+2, t) \\ & \rightarrow Lt(t+1) \rightarrow (T(m+1, t+1) \rightarrow \text{loop}) \rightarrow \text{loop}). \\ (0_r) \quad & \forall mt (K(m, t+1) \rightarrow L(m, t) \rightarrow M(m+1, t) \rightarrow N(m+2, t) \\ & \rightarrow Rt(t+1) \rightarrow (U(m+1, t+1) \rightarrow \text{loop}) \rightarrow \text{loop}). \end{aligned}$$

First row:

$$\begin{aligned} (1) \quad & (E(0, 0) \rightarrow \text{loop}) \rightarrow \text{start}; \\ (2) \quad & \forall m (E(m, 0) \rightarrow (E(m+1, 0) \rightarrow \text{loop}) \rightarrow \text{loop}). \end{aligned}$$

First column:

$$(3) \quad \forall t (E(0, t) \rightarrow (Lt(t+1) \rightarrow E(0, t+1) \rightarrow \text{loop}) \rightarrow (Rt(t+1) \rightarrow E(0, t+1) \rightarrow \text{loop}) \rightarrow \text{loop}).$$

Conclusion:

$$(4) \quad \forall m, t \leq s (\text{OK}(m, t) \rightarrow \text{loop}).$$

Pairs (m, t) should be interpreted as tile locations. The space to tile is $\mathbb{N} \times \{0, 1\}^*$, not $\mathbb{N} \times \mathbb{N}$, so (m, t) does not identify a unique location in the tiling, but only in the local tiling associated to a certain path from $\{0, 1\}^*$. An assumption $K(m, t)$ only states that tile K is to be placed at node (m, t) in the present local tiling. This always refers to some particular location (m, w) , where $|w| = t$. The predicate $Lt(t)$ (resp. $Rt(t)$) indicate that node w is the left (resp. right) child of its parent w' , i.e., that $w = w'0$ (resp. $w = w'1$).

As in Section 4, we think of the formulas in $\Gamma_{\mathcal{G}}$ as of proof tactics. For example, formula (0_l) is used towards the proof goal *loop*, provided $K(m, t+1), \dots, Lt(t+1)$ can be verified. Applying this tactic will not change the proof goal but will add $T(m+1, t+1)$ as a new assumption.

We need to implement the above idea using unary relations and no arithmetic. In Section 4 we used different variables to represent coordinates of the grid. With a fixed supply of variables (cf. Lemma 5.4) we cannot do that. However, as long as we only need to encode bounded values of coordinates, this can be overcome by using many-argument predicates. Those can later be eliminated using Proposition 4.10 to a linear number of unary predicates.

The basic idea is this. Recall first that the highest number which occurs in pairs within $\mathcal{L}(s, s)$ is $2s$. Assume we have two free object variables x_0, x_1 , and let us fix a number $n > \log 2s$. Using x_0 as 0 and x_1 as 1 one can write a number $m \leq 2s$ as a sequence m_1, \dots, m_n , where each m_i is x_0 or x_1 . A $2n$ -ary predicate $K(m_1, \dots, m_n, t_1, \dots, t_n)$ can thus be read as $K(m, t)$, where “variables” m and t take values from 0 to $2^n - 1$. This suffices to represent coordinates of all points in the set $\mathcal{L}(s, s)$.

The formulas of $\Gamma_{\mathcal{G}}$ could now be rewritten using $2n$ -ary relation symbols $T \in \mathcal{T}$ and n -ary symbols Lt and Rt instead of the binary and unary symbols. In $T(m, t)$, the “variables” m and t are now understood as sequences built from x_0 and x_1 . For instance, $E(0, 0)$ means $E(x_0, \dots, x_0, x_0, \dots, x_0)$. The meaning of a quantifier $\forall m$ is $\forall m_1 m_2 \dots m_n$.

The remaining difficulty is the use of the bound “ $\leq s$ ” and the successor and predecessor operations $+1$ and -1 . The last two can be handled by observing that a number $t' < 2^n$ is a successor of a number t when, for some k , the last k bits in the representations of t' and t are, respectively, $011\dots 1$ and $100\dots 0$. There are n such patterns (one for each k) for binary strings of length n . Now, for instance, instead of the single formula

$$\forall m(\mathbf{E}(m, 0) \rightarrow (\mathbf{E}(m + 1, 0) \rightarrow \text{loop}) \rightarrow \text{loop}),$$

we can use n formulas, each following one of those patterns:

$$\forall \vec{z}(\mathbf{E}(\vec{z}, x_0, \vec{x}_1, \vec{x}_0) \rightarrow (\mathbf{E}(\vec{z}, x_1, \vec{x}_0, \vec{x}_0) \rightarrow \text{loop}) \rightarrow \text{loop}).$$

Above, \vec{z} is a sequence of $n - k$ bound variables, and \vec{x}_1 is a sequence of $k - 1$ occurrences of x_1 . The symbol \vec{x}_0 is loosely used for appropriately long sequences of x_0 .

In a similar fashion we can handle the inequality occurring in (4). If s is written in binary as $b_1\dots b_n$ then a number m can be less or equal than s in as many different ways as there are numbers $i < n$ with $b_{i+1} = 1$. This happens when m has the form $b_1\dots b_i 0 \vec{z}$, for some \vec{z} consisting of $n - i + 1$ bits. So if $b_{i+1} = b_{j+1} = 1$ then we use the formula:

$$\forall \vec{z} \vec{y}(\text{OK}(b_1\dots b_i 0 \vec{z}, b_1\dots b_j 0 \vec{y}) \rightarrow \text{loop}).$$

This way we replace each of the assumptions (0_l) , (0_r) , $(1-4)$, by at most $2n^2$ formulas using $2n$ -ary predicates. The set $\Gamma_{\mathcal{G}}$ consists of all such formulas. To simplify the construction in the rest of this section we use the abbreviated notation with variables ranging over natural numbers.

Some definitions: A finite nonempty set $S \subseteq \mathcal{L}(s, s)$ is called a *base* when it is downward closed with respect to the relation \preceq . Then $S_2 := \{t \mid \exists m. (m, t) \in S\}$ is a finite initial segment of \mathbb{N} . A set of formulas Θ is *time-coherent for S* iff

- for each $t \in S_2$, if $t > 0$ then either $\text{Lt}(t)$ or $\text{Rt}(t)$ is in Θ , but not both.

For a time-coherent set Θ , we define $a_t^\Theta = 0$ when $\text{Lt}(t) \in \Theta$, and $a_t^\Theta = 1$ otherwise. Denote the word $a_1^\Theta \dots a_t^\Theta$ by w_t^Θ , and let $w_0^\Theta = \varepsilon$. If $t = \max S_2$ then we write w^Θ for w_t^Θ . We say that Θ is *very good for S* when it is time-coherent and the following holds:

- Formulas in Θ are only of the forms $\text{Lt}(t)$, $\text{Rt}(t)$, or $\text{T}(m, t)$, where $\text{T} \in \mathcal{T}$.
- $\text{T}(m, t) \in \Theta$ if and only if $(m, t) \in S$ and $\mathcal{G}^*(m, w_t^\Theta) = \text{T}$.
- If $\text{Lt}(t) \in \Theta$ or $\text{Rt}(t) \in \Theta$ then $t \in S_2$.

Lemma 6.1. *Let Θ be very good for base S . Assume that $\Gamma_{\mathcal{G}}, \Theta \vdash \text{loop}$. Then \mathcal{G} is s -solvable from w^Θ .*

Proof. Let $\Gamma_{\mathcal{G}}, \Theta \vdash N : \text{loop}$, where N is a long normal form. We proceed by induction with respect to N , in a similar style as we did in the proof of Lemma 4.1.

If N begins with a variable of type (2) then $\Gamma_{\mathcal{G}}, \Theta \vdash \mathbf{E}(m, 0)$ (whence the atom $\mathbf{E}(m, 0)$ is actually in Θ) and $\Gamma_{\mathcal{G}}, \Theta, \mathbf{E}(m + 1, 0) \vdash \text{loop}$, for some m . We apply the induction hypothesis to the set $\Theta \cup \{\mathbf{E}(m + 1, 0)\}$, very good for $S \cup \{(m + 1, 0)\}$.

In case N begins with a variable of type (4) we must have $\text{OK}(m, t) \in \Theta$. In addition, $m, t \leq s$, so the conclusion is immediate. The cases (0_l) and (0_r) are routine as well: for $\text{X} = \text{T}, \text{U}$, we consider the environment $\Theta, \text{X}(m + 1, t + 1)$, which is very good for $S \cup \{(m + 1, t + 1)\}$. Observe that the location $(m + 1, t + 1)$ may already belong to S , and in this case also the formula $\text{X}(m + 1, t + 1)$ is already in Θ , because our tiling is deterministic.

A crucial case is when N begins with an axiom of type (3). Then $\Gamma_{\mathcal{G}}, \Theta \vdash \mathbf{E}(0, t)$, and:

- $\Gamma_{\mathcal{G}}, \Theta, \text{Lt}(t + 1), \mathbf{E}(0, t + 1) \vdash \text{loop}$;

- $\Gamma_{\mathcal{G}}, \Theta, \text{Rt}(t+1), \text{E}(0, t+1) \vdash \text{loop}$.

If $t = |w^\ominus|$ then the location $(0, t+1)$ is “new”, that is neither $\text{Lt}(t+1)$ nor $\text{Rt}(t+1)$ occurs in Θ . Then both $\Theta, \text{Lt}(t+1), \text{E}(0, t+1)$ and $\Theta, \text{Rt}(t+1), \text{E}(0, t+1)$ are very good environments for $S \cup \{(0, t+1)\}$. By the induction hypothesis, \mathcal{G} is s -solvable from $w^\ominus 0$ and $w^\ominus 1$, so it is s -solvable from w^\ominus by Lemma 3.3.

If $t < |w^\ominus|$ then either $\text{Lt}(t+1)$ or $\text{Rt}(t+1)$ is already in Θ , and so is $\text{E}(0, t+1)$. Therefore one of the two environments is identical to $\Gamma_{\mathcal{G}}, \Theta$, and we can apply the induction hypothesis (there is a shorter proof). \square

Lemma 6.2. *If $\Gamma_{\mathcal{G}} \vdash \text{start}$ then \mathcal{G} is s -solvable.*

Proof. A proof of *start* is only possible when the judgment $\Gamma_{\mathcal{G}}, \text{E}(0, 0) \vdash \text{loop}$ is provable. Apply Lemma 6.1. \square

Now we address the question of the converse of Lemma 6.2. Put $(m, w) \subseteq (n, v)$ when either $w \subseteq v$ or $w = v$ and $m \leq n$. If $\mathcal{G}^*(m, w) = \text{OK}$, and (m, w) is minimal with respect to \subseteq , then we say that (m, w) is a *winning location*.

Lemma 6.3. *Let Θ be very good for base S . If \mathcal{G} is s -solvable from w^\ominus then $\Gamma_{\mathcal{G}}, \Theta \vdash \text{loop}$.*

Proof. The obvious case is when $\mathcal{G}^*(m, w_r^\ominus) = \text{OK}$, for some $(m, r) \in S$, and $m, r \leq s$. Otherwise, for every w with $|w| = s$, $w^\ominus \subseteq w$, there is a winning location (m, w') such that $m \leq s$ and $w' \subseteq w$. (Some of these winning locations may be equal, in particular if $w' \subseteq w^\ominus$ then all of them are equal.) For every such (m, w') , the *distance* from S to (m, w') is the cardinality of the difference $\mathcal{L}(m, |w'|) - S$.

The proof of the lemma is by induction with respect to the sum of distances from S to all winning locations (m, w') such that $w' \subseteq w^\ominus$ or $w^\ominus \subseteq w'$. (This is equivalent to saying that there is w of length s such that $w', w^\ominus \subseteq w$.)

The base case has already been treated, so assume that we have a winning location (m, w') . If $w' \subseteq w^\ominus$ then it is the only winning location of interest. If $(m, |w'|) \notin S$ then the difference $\mathcal{L}(m, |w'|) - S$ has a minimal element (m_1, t_1) . As in the proof of Lemma 4.3, we apply the induction hypothesis to the set $S \cup \{(m_1, t_1)\}$ and apply an assumption (0_l) or (0_r) . This we can do because either $\text{Lt}(t_1)$ or $\text{Rt}(t_1)$ must belong to Θ .

The remaining case is when there are at least two winning locations, all of them of the form (m, w_v^\ominus) with $v \neq \varepsilon$. In addition, at least one such v begins with 0 and at least one with 1. Take $t = |w^\ominus|$ and let $S' = S \cup \{(0, t+1)\}$; then the environments $\Theta_0 = \Theta \cup \{\text{Lt}(t+1), \text{E}(0, t+1)\}$ and $\Theta_1 = \Theta \cup \{\text{Rt}(t+1), \text{E}(0, t+1)\}$ are very good for S' , so $\Gamma_{\mathcal{G}}, \Theta_0 \vdash \text{loop}$ and $\Gamma_{\mathcal{G}}, \Theta_1 \vdash \text{loop}$ by the induction hypothesis. (In each case, we have fewer winning positions and thus fewer components in our sum.) We now use assumption (3). \square

Lemma 6.4. *If \mathcal{G} is s -solvable then $\Gamma_{\mathcal{G}} \vdash \text{start}$.*

Proof. Apply Lemma 6.3 to the base $S = \emptyset$. \square

Theorem 6.5. *The decision problem for monadic Σ_1 is co-NEXPTIME-hard.*

Proof. Hardness for arbitrary predicates follows from Lemmas 3.4 and 6.4. Translation to the monadic case is possible by Proposition 4.10 because all formulas we use are easy. \square

In fact our hardness result applies to a “shallow” fragment of monadic Σ_1 . This fragment consists of formulas of the form $\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow \mathbf{a}$, where τ_i are universal formulas, i.e., we have $\tau_i = \forall \vec{x}_i \tau'_i$ with quantifier-free τ'_i .

6.2. Arity-bounded refutation soup. For the matching upper bound we show that provability of arity-bounded Σ_1 formulas is solvable in *co*-NEXPTIME. More precisely, let us fix a number r and consider only formulas involving predicates of arity at most r . We will demonstrate a nondeterministic exponential time algorithm for non-provability, i.e., refutability of such formulas.

Lemma 6.6.

- (1) Every Σ_1 formula has the form $\tau_1 \rightarrow \tau_2 \rightarrow \dots \rightarrow \tau_n \rightarrow \mathbf{a}$, where $\tau_i \in \Pi_1$, and \mathbf{a} is an atom.
- (2) Every Π_1 formula has the form $\forall \vec{y}_1(\sigma_1 \rightarrow \forall \vec{y}_2(\sigma_2 \rightarrow \dots \rightarrow \forall \vec{y}_k(\sigma_k \rightarrow \mathbf{a}) \dots))$, where $\sigma_i \in \Sigma_1$, and \mathbf{a} is an atom.

Proof. The pseudo-grammar of Section 2.2 simplifies as follows. The metavariable \mathbf{a} now stands for an atom of arity r or less.

- $\Sigma_1 ::= \mathbf{a} \mid \Pi_1 \rightarrow \Sigma_1$;
- $\Pi_1 ::= \mathbf{a} \mid \Sigma_1 \rightarrow \Pi_1 \mid \forall x \Pi_1$.

□

As it suffices to deal with long normal proofs, we are mostly interested in judgments of the form $\Gamma \vdash N : \mathbf{a}$, where Γ consists of Π_1 formulas, and \mathbf{a} is an atom. The long normal proof N must begin with a proof variable X ; assume that a declaration of the form $X : \forall \vec{y}_1(\sigma_1 \rightarrow \forall \vec{y}_2(\sigma_2 \rightarrow \dots \rightarrow \forall \vec{y}_k(\sigma_k \rightarrow \mathbf{b}) \dots))$ is in Γ . Types σ_i are in Σ_1 and therefore $\sigma_j = \tau_{j1} \rightarrow \tau_{j2} \rightarrow \dots \rightarrow \tau_{jr_j} \rightarrow \mathbf{a}_j$, for $j = 1, \dots, k$. To maintain some basic hygiene we assume that all variables in $\vec{y}_1 \vec{y}_2 \dots \vec{y}_k$ are different and not free in Γ . Then we have $N = X \vec{x}_1(\lambda Y_{11} \dots Y_{1r_1}. N_1) \dots \vec{x}_k(\lambda Y_{k1} \dots Y_{kr_k}. N_k)$, for some variables $\vec{x}_1 \vec{x}_2 \dots \vec{x}_k$ (assumed different from \vec{y}_i). Write S for the substitution $[\vec{y}_1 := \vec{x}_1, \vec{y}_2 := \vec{x}_2, \dots, \vec{y}_k := \vec{x}_k]$. Then we must have $\mathbf{a} = \mathbf{b}[S]$, and $\Gamma, Y_{j1} : \tau_{j1}[S], Y_{j2} : \tau_{j2}[S], \dots, Y_{jr_j} : \tau_{jr_j}[S] \vdash N_j : \mathbf{a}_j[S]$, for all j .

We know from Lemma 5.4 that if $\Gamma \vdash \varphi$ is inhabited, then there exists a long normal inhabitant N with $\text{FV}(N) \subseteq \text{FV}(\Gamma) \cup \text{FV}(\varphi)$. Therefore, the above analysis can be strengthened by the requirement that all the variables $\vec{x}_1 \vec{x}_2 \dots \vec{x}_k$ are in $\text{FV}(\Gamma) \cup \text{FV}(\mathbf{a})$. The next lemma is a contraposition of the above taking this additional requirement into account.

Lemma 6.7. *Let Γ consist of Π_1 formulas and let \mathbf{a} be an atom. Then $\Gamma \not\vdash \mathbf{a}$ if and only if for every $X \in \text{Dom}(\Gamma)$ of type $\forall \vec{y}_1(\sigma_1 \rightarrow \forall \vec{y}_2(\sigma_2 \rightarrow \dots \rightarrow \forall \vec{y}_k(\sigma_k \rightarrow \mathbf{b}) \dots))$, every S with $\text{Dom}(S) = \vec{y}_1, \dots, \vec{y}_n$, $\text{Rg}(S) \subseteq \text{FV}(\Gamma) \cup \text{FV}(\mathbf{a})$, and $\mathbf{a} = \mathbf{b}[S]$, there is $j \in \{1, \dots, k\}$ with $\sigma_j = \tau_{j1} \rightarrow \tau_{j2} \rightarrow \dots \rightarrow \tau_{jr_j} \rightarrow \mathbf{a}_j$ such that $\Gamma, \tau_{j1}[S], \tau_{j2}[S], \dots, \tau_{jr_j}[S] \not\vdash \mathbf{a}_j[S]$.*

□

Morally, Lemma 6.7 states that in a certain proof-construction game one of the players has a winning strategy: either the Prover, trying to construct a long normal proof (always a finite one) or the Reviewer, attempting to build a (possibly infinite) *refutation*, cf. [25]. Indeed, let $\Gamma \vdash \mathbf{a}$ be a Σ_1 judgment as above and assume the notation from Lemma 6.7. Every pair (X, S) such that a declaration $X : \forall \vec{y}_1(\sigma_1 \rightarrow \forall \vec{y}_2(\sigma_2 \rightarrow \dots \rightarrow \forall \vec{y}_k(\sigma_k \rightarrow \mathbf{b}) \dots))$ is in Γ , and S is a variable substitution satisfying $\mathbf{a} = \mathbf{b}[S]$, is called a *question induced by $\Gamma \vdash \mathbf{a}$* . For any $j \in \{1, \dots, k\}$, a *j -th answer* to the question (X, S) is any judgment $\Gamma', \tau_{j1}[S], \tau_{j2}[S], \dots, \tau_{jr_j}[S] \vdash \mathbf{a}_j[S]$, where $\Gamma \subseteq \Gamma'$. (Note that we do not require $\Gamma = \Gamma'$. This extra flexibility is used in the proof of Lemma 6.9.)

Lemma 6.7 may now be read as: $\Gamma \not\vdash \mathbf{a}$ if and only if Reviewer can answer every question (and in addition $\Gamma' = \Gamma$ always holds). This constitutes a refutation seen as a Reviewer's winning strategy. A compact way to represent such a refutation is simply a set of judgments.

A *refutation soup* is a non-empty set \mathcal{Z} of Σ_1 judgments such that

- If $\Gamma \vdash \mathbf{a}$ is in \mathcal{Z} then for every question induced by $\Gamma \vdash \mathbf{a}$ there is an answer in \mathcal{Z} .

We say that \mathcal{Z} *refutes* $\Gamma_0 \vdash \mathbf{a}_0$ whenever the judgment $\Gamma_0 \vdash \mathbf{a}_0$ belongs to \mathcal{Z} . Then we also say that $\Gamma_0 \vdash \mathbf{a}_0$ is *refutable*. Observe that a judgment of the form $\Gamma, X : \mathbf{a} \vdash \mathbf{a}$ cannot occur in a soup. Indeed, there is no answer to the question $\langle X, \emptyset \rangle$.

Lemma 6.8. *A judgment $\Gamma_0 \vdash \mathbf{a}_0$ is refutable if and only if $\Gamma_0 \not\vdash \mathbf{a}_0$.*

Proof. (\Rightarrow) Let \mathcal{Z} be a refutation soup such that some judgments in \mathcal{Z} are provable. Among such judgments there is one which has a shortest long normal proof. Let $\Gamma_0 \vdash \mathbf{a}_0$ be this judgment. Assume that $N = X\vec{x}_1(\lambda Y_{11} \dots Y_{1r_1}^1. N_1) \dots \vec{x}_k(\lambda Y_{k1} \dots Y_{kr_k}. N_k)$ is the proof. Note that $k \neq 0$, as otherwise the question (X, \emptyset) has no answer. Consider the question (X, S) , where $S = [\vec{y}_1 := \vec{x}_1, \vec{y}_2 := \vec{x}_2, \dots, \vec{y}_k := \vec{x}_k]$. In the refutation \mathcal{Z} there is an answer of the form $\Gamma', \tau_{j1}[S], \tau_{j2}[S], \dots, \tau_{jr_j}[S] \vdash \mathbf{a}_j[S]$. Clearly, \mathcal{Z} refutes this judgment as well. But on the other hand, $\Gamma', \tau_{j1}[S], \tau_{j2}[S], \dots, \tau_{jr_j}[S] \vdash N_j : \mathbf{a}_j[S]$, i.e., the refuted judgment has a proof, shorter than N . This contradicts our assumption about N .

(\Leftarrow) A soup may be defined as a sum of an ascending sequence of sets \mathcal{Z}_n . The set \mathcal{Z}_0 consists only of the initial judgment $\Gamma_0 \vdash \mathbf{a}_0$. Then, for any n , we select a judgment in \mathcal{Z}_n and a question induced by this judgment which does not have an answer in \mathcal{Z}_n . By Lemma 6.7 there is always a non-provable answer. We obtain \mathcal{Z}_{n+1} by adding this answer to \mathcal{Z}_n . This process must end because only a finite number of judgments may occur in the construction. \square

We now show that every refutable judgment has a small soup.

Lemma 6.9. *If $\Gamma_0 \not\vdash \mathbf{a}_0$ then there is a refutation soup of size exponential in the length n of the judgment $\Gamma_0 \vdash \mathbf{a}_0$.*

Proof. Let \mathcal{F} be the set of all formulas of the form $\tau[S]$, where τ is a subformula of a formula in Γ_0 , and S is a substitution such that $\text{Rg}(S) \subseteq \text{FV}(\Gamma_0) \cup \text{FV}(\mathbf{a}_0)$. A judgment $\Gamma \vdash \mathbf{a}$ is *reasonable* when $\Gamma \cup \{\mathbf{a}\} \subseteq \mathcal{F}$.

As in the proof of Lemma 6.8 we construct a soup by induction, but now we introduce some structure: rather than just a set of judgments we define a tree labeled by judgments. The rule is that children of every node are answers to questions induced by that node. We begin from the root labeled $\Gamma_0 \vdash \mathbf{a}_0$. At every step we select a leaf node $\Gamma \vdash \mathbf{a}$ (a judgment not processed before) and for every question (X, S) induced by that node we choose an unprovable answer to that question, say $\Gamma' \vdash \mathbf{a}'$, which is reasonable and *maximal* in the following sense: whenever $\Gamma' \subsetneq \Gamma'' \subseteq \mathcal{F}$ then $\Gamma'' \vdash \mathbf{a}_j[S]$ has a proof. Then we add $\Gamma' \vdash \mathbf{a}'$ as a new child of $\Gamma \vdash \mathbf{a}$, unless $\Gamma' \vdash \mathbf{a}'$ already occurs on the path from the root to $\Gamma \vdash \mathbf{a}$. It should be clear that the set of all labels in our tree is a soup.

If a non-root judgment $\Gamma \vdash \mathbf{a}$ is an ancestor of $\Gamma' \vdash \mathbf{b}$ in our tree then $\Gamma \subseteq \Gamma'$. Therefore $\mathbf{a} \neq \mathbf{b}$, as otherwise $\Gamma \vdash \mathbf{a}$ would not be selected as maximal, or the same judgment would occur twice on a path. It follows that every path of the tree is of length at most n^{r+1} , where r is the maximum arity of predicates in \mathcal{F} . Indeed, every judgment in a non-root position along the path addresses a different target, and there is at most $n \cdot n^r$ of those (up to n predicates times up to n^r ways in which n variables can occur at r positions). Since the maximal branching is $n \cdot n^n \cdot n^n$ (an upper bound for the number of questions), the total number of nodes does not exceed $(n \cdot n^n \cdot n^n)^{n^{r+1}} \leq 2^{n^{r+4}}$. \square

Proposition 6.10. *For every r , non-provability of Σ_1 formulas using at most r -ary predicates is solvable in NEXPTIME.*

Proof. A nondeterministic algorithm can generate a refutation soup and verify its correctness in exponential time. \square

In particular we have:

Corollary 6.11. *The decision problem for Σ_1 formulas of any fixed finite signature is in the class $co\text{-NEXPTIME}$.* \square

Together with Theorem 6.5 we obtain the final result.

Theorem 6.12. *For every $r \in \mathbb{N}$, the decision problem for Σ_1 formulas using at most r -ary predicates is $co\text{-NEXPTIME}$ -complete.* \square

7. CONCLUSION AND FUTURE WORK

We proved that derivability of universally-implicational formulas for the class Δ_2 of Mints hierarchy (and therefore for all larger classes) is undecidable even for unary predicate symbols. In case of Σ_1 the problem is in general EXPSPACE -complete, but it turns out only $co\text{-NEXPTIME}$ -complete if we restrict the arity of predicates (this applies e.g., to the monadic fragment). In particular the exponential upper bound holds for every finite signature.

These results combined with an earlier analysis [23] give the picture of complexity of provability in Mints hierarchy in which the level of a formula φ is determined by the level of a prenex formula classically equivalent to φ . Observe that all the hardness results were obtained for formulas with a fixed depth of quantifiers.

The fragment of intuitionistic logic discussed in this paper only involves the two basic connectives, \forall and \rightarrow . By conservativity, all our lower bounds extend to the full first-order language with \exists , \vee , \wedge , and \perp . It is not necessarily so with the upper bounds. The exponential space algorithm for Σ_1 extends to the general case, but the refutation soup argument does not (because targets in judgments can be disjunctions). We conjecture that the Σ_1 fragment of the full first-order logic will turn out EXPSPACE -complete even in the monadic case. On the other hand, we believe that the number of predicates matters: perhaps Corollary 6.11 can be improved down to PSPACE ?

Another issue demanding future work is the exact complexity of the class Π_1 [22, 23].

REFERENCES

- [1] Daniel Bonevac. A history of quantification. In *Logic: A History of its Central Concepts*, volume 11 of *Handbook of the History of Logic*. North Holland, 2012.
- [2] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.
- [3] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda – a functional language with dependent types. In *Theorem Proving in Higher Order Logics*, volume 5674 of *LNCS*, pages 73–78. Springer, 2009.
- [4] Wolfgang Burr. The intuitionistic arithmetical hierarchy. In *Logic Colloquium '99*, volume 17 of *Lecture Notes in Logic*, pages 51–59. ASL, 1999.
- [5] Alonzo Church. *Introduction to Mathematical Logic*. Princeton, 1944.
- [6] Coq Development Team. *The Coq Proof Assistant Reference Manual V8.4*, March 2012. <http://coq.inria.fr/distrib/V8.4/refman/>.
- [7] Anatoli Degtyarev, Yuri Gurevich, Paliath Narendran, Margus Veanes, and Andrei Voronkov. Decidability and complexity of simultaneous rigid E-unification with one variable and related results. *Theoretical Computer Science*, 243(1-2):167–184, 2000.

- [8] Gilles Dowek and Ying Jiang. Eigenvariables, bracketing and the decidability of positive minimal predicate logic. *Theoret. Comput. Sci.*, 360(1–3):193–208, 2006.
- [9] M. Fitting. *Fundamentals of Generalized Recursion Theory*. Elsevier, 1981.
- [10] Jonathan Fleischmann. Syntactic preservation theorems for intuitionistic predicate logic. *Notre Dame Journal of Formal Logic*, 51(2):225–245, 2010.
- [11] Neil Immerman. *Descriptive Complexity*. Springer, 1999.
- [12] G. Kreisel. Elementary completeness properties of intuitionistic logic with a note on negations of prenex formulae. *J. Symbolic Logic*, 23(3):pp. 317–330, 1958.
- [13] Dariusz Kuśmierk. The inhabitation problem for rank two intersection types. In *TLCA*, volume 4583 of *LNCS*, pages 240–254. Springer, 2007.
- [14] G.E. Mints. Solvability of the problem of deducibility in LJ for a class of formulas not containing negative occurrences of quantifiers. *Steklov Inst.*, 98:135–145, 1968.
- [15] V.P. Orevkov. The undecidability in the constructive predicate calculus of the class of formulas of the form $\neg\neg\forall\exists$. *Doklady AN SSSR*, 163(3):581–583, 1965.
- [16] V.P. Orevkov. Solvable classes of pseudoprenex formulas. *Zapiski nauchnyh Seminarov LOMI*, 60:109–170, 1976.
- [17] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.
- [18] H. Rasiowa and R. Sikorski. On existential theorems in non-classical functional calculi. *Fundamenta Mathematicae*, 41:21–28, 1954.
- [19] Jakob Rehof and Paweł Urzyczyn. The complexity of inhabitation with explicit intersection. In *Logic and Program Semantics*, volume 7230 of *LNCS*, pages 256–270. Springer, 2012.
- [20] Eric Rosen. On the first-order prefix hierarchy. *Notre Dame Journal of Formal Logic*, 46(2):147–164, 2005.
- [21] Ivar Rummelhoff. *Polymorphic Π 1 Types and a Simple Approach to Propositions, Types and Sets*. PhD thesis, University of Oslo, 2007.
- [22] Aleksy Schubert, Paweł Urzyczyn, and Daria Walukiewicz-Chrząszcz. Restricted positive quantification is not elementary. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *Proc. TYPES 2014*, volume 39 of *LIPICs*, pages 251–273. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015.
- [23] Aleksy Schubert, Paweł Urzyczyn, and Daria Walukiewicz-Chrząszcz. How hard is positive quantification? To appear in *ACM ToPLaS*, 2016.
- [24] Aleksy Schubert, Paweł Urzyczyn, and Konrad Zdanowski. On the Mints hierarchy in first-order intuitionistic logic. In A. Pitts, editor, *Foundations of Software Science and Computation Structures 2015*, volume 9034 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2015.
- [25] Tomasz Skura. Refutation systems in propositional logic. In Dov M. Gabbay and Franz Guenther, editors, *Handbook of Philosophical Logic*, volume 16, pages 115–157. Springer, second edition, 2011.
- [26] M.H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149. Elsevier, 2006.
- [27] P. Urzyczyn. Inhabitation of low-rank intersection types. In P.-L. Curien, editor, *TLCA*, volume 5608 of *LNCS*, pages 356–370. Springer, 2009.
- [28] Hao Wang. Toward mechanical mathematics. *IBM J. Res. Dev.*, 4(1):2–22, January 1960.