BETWEEN PROOF CONSTRUCTION AND SAT-SOLVING

ALEKSY SCHUBERT, PAWEŁ URZYCZYN, AND KONRAD ZDANOWSKI

Abstract. The classical satisfiability problem (SAT) is used as a natural and general tool to express and solve combinatorial problems that are in NP. We postulate that provability for implicational intuitionistic propositional logic (IIPC) can serve as a similar natural tool to express problems in Pspace. We demonstrate it by proving two essential results concerning the system. One is a natural reduction from full IPC (with all connectives) to implicational formulas of order three. Another result is a convenient interpretation in terms of simple alternating automata. Additionally, we distinguish some natural subclasses of IIPC corresponding to the complexity classes NP and *co*-NP.

§1. Introduction. Everyone knows that classical propositional calculus (CPC) is a natural language to represent combinatorial problems (see, e.g., [15, 31]). Various decision problems can be easily encoded as instances of the formula satisfiability problem (SAT) and efficiently solved [1, 28].

In this article we would like to turn the reader's attention to the so far unexploited fact that intuitionistic implicational propositional calculus (IIPC) [18] is an interesting propositional formalism which is equally natural and simple in its nature as CPC, yet stronger in its expressive power. Indeed, while SAT and ASP [5] can express NP-complete problems, the decision problem for IIPC is complete for Pspace. Thus IIPC can accommodate a larger class of problems that may be encoded as formulas and solved using automated or interactive proof-search. In particular, the Sokoban puzzle [6, 11, 14] cannot be solved by means of SAT solving, but could be encoded in IIPC and examined by an interactive prover.

Of course the Pspace complexity is enormous, but the general case of NP is infeasible anyway. And not every polynomial space computation requires exponential time. We may only solve "easy" cases of hard problems, and then the increased expressiveness of the language can be useful rather than harmful. For example, since Pspace is closed under complements one can simultaneously attempt to prove a proposition and to disprove it by proving a dual one [43].

What is also important, this approach to Pspace avoids adding new syntactical forms such as Boolean quantification of QBF [37]. Moreover, we can syntactically distinguish subclasses of IIPC for which the decision problem is complete for P, NP, and *co*-NP.

Received May 10, 2024.
2020 *Mathematics Subject Classification.* Primary 03F03, 03F07, 03F50.
Key words and phrases. satisfiability, provability, simply typed lambda calculus, propositional logic.

0022-4812/00/0000-0000
DOI:10.1017/jsl.2025.10090

The strength of CPC and SAT-solving is in their conceptual simplicity —a propositional formula provides a specification of a configuration of interest while a solution provides a particular configuration that meets the specification. In the case of IIPC, as illustrated below, we are able to achieve the same goal. In addition, we obtain one more dimension of expressiveness: the proof we build represents the process of constructing the solution. For instance, a sequence of moves in the Sokoban game, or a computation of a machine corresponds to a sequence of proof steps (in the order of which the proof is being constructed).

Indeed, interestingly enough, IIPC offers not only a formalism to describe relationships, but also has a procedural view in the form of proof-search process. Moreover, the proof-search in IIPC does not have to be less convenient than processing SAT instances or computing in ASP-based paradigm [5]: normal proof search (Ben–Yelles algorithm) is intuitive and straightforward. While this observation has already been done in the context of λ-Prolog [24], it remained largely overlooked there that simplification of the formula format to order at most three does not restrict expressibility.[1]

The proof-search computational paradigm brings here an interesting, not always clearly expressed, facet to the Curry–Howard isomorphism. The Curry–Howard isomorphism states that systems of formal logic and computational calculi are in direct correspondence. It begun with the discovery of *formulas-as-types* and *proofs-as-terms* paradigm made by Curry [7] and was later expanded by Howard with the *computation-as-normalization* paradigm [17]. Later, various authors have contributed to the wider understanding of the *logic-as-computation* slogan, adding new facets to the general paradigm. For example, one very important aspect is the *formulas-as-games, proofs-as-strategies* view initiated by Lorenzen [21]. We think that the broad understanding of Curry–Howard might as well include yet another analogy: *computation-as-proof-search*. Virtually any algorithm can be expressed as a formula of some constructive logic in such a way that every proof of the formula is but an execution of the algorithm. Yet differently, finding a proof of a formula (or equivalently an inhabitant of a type) is the same as executing a program. This way we have a close relationship between *proof search* in the realm of logic or *program synthesis* [19, 30] in the realm of programming.

A simple illustration of the paradigm “proof construction as computation” is reading a logical consequence $\Gamma \vdash \tau$ as a configuration of a machine (a *monotonic automaton*). Under this reading the proof goal is the internal state, the assumptions Γ represent the memory. Variants of such *monotonic automata* were used in [32, 33]; in the present article we make this automata-theoretic semantics of (I)IPC very clear-cut.

We begin our presentation with Section 2 where we fix notation and recall some basic definitions. Then we enter the discussion of expressibility of IIPC, focusing mainly on the fact that the whole expressive strength is in formulas of order at most three. In Section 3 we demonstrate the natural equivalence between proof-search and computation: the monotonic automata directly implement the Wajsberg/Ben–

[1]A notable exception here is the work of Paul Tarau [38] where an IPC prover is presented that operates on formulas in form introduced by Mints [26], which is closely related to formulas of order three.

Yelles inhabitation algorithm for the full IPC (with all connectives). After showing that the halting problem for monotonic automata is PSPACE-complete, we reduce it to provability in IIPC. This yields a polynomial translation of the decision problem for the full IPC to IIPC formulas of order at most three. It follows from Section 4 however, that the translation does not preserve the equivalence of formulas. Still our reduction plays a similar role as that of the whole SAT to 3-CNF-SAT.

In Section 5 we define two subclasses of low-order IIPC corresponding to the complexity classes NP and *co*-NP.

We conclude in Section 6 with a few final comments.

§2. Preliminaries.

To make the article self-contained, we introduce here the necessary definitions and fix the basic notation. This section may be to large extent skipped and used as a reference. A more detailed account of the relevant notions can be found for instance in [35].

Propositional formulas. We assume an infinite set $\mathcal{X}$ of *propositional variables*, usually written as $p, q, r, \ldots$, possibly with indices. Propositional variables and the constant $\bot$ are called *atoms*.

The formulas of the full intuitionistic propositional logic, IPC, are generated by the grammar:

$$\varphi, \psi ::= p \mid \bot \mid \varphi \to \psi \mid \varphi \wedge \psi \mid \varphi \vee \psi,$$

where $p \in \mathcal{X}$. As usual, we use $\neg\varphi$ as a shorthand for $\varphi \to \bot$.

For clarity we do not include parentheses in the grammar. We adopt standard conventions that parentheses can be used anywhere to disambiguate understanding of the formula structure. Additionally, we assume that $\to$ is right-associative so that $\varphi_1 \to \varphi_2 \to \varphi_3$ is equivalent to $\varphi_1 \to (\varphi_2 \to \varphi_3)$.

We use the notation $\varphi[p := \psi]$ for substitution. If $\Gamma = \{\varphi_1, \ldots, \varphi_n\}$ then we write $\Gamma \to p$ for the formula $\varphi_1 \to \cdots \to \varphi_n \to p$.

A *literal* is either a propositional variable or a negated variable. Literals are written in typewriter font: $\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots$ If $\mathtt{p}$ is a literal, then $\overline{\mathtt{p}}$ is its dual literal, i.e., $\overline{p} = \neg p$ and $\overline{\neg p} = p$.

Proof terms. According to the Curry–Howard correspondence, formulas can be seen as types assigned to proof terms. In this view, IIPC is exactly the ordinary simply typed lambda-calculus. For the full IPC we need an extended calculus and we now define the syntax of it. We assume an infinite set Υ of *proof variables*, usually written as $x, y, z, \ldots$ with possible annotations. A *context* is a finite set of pairs $x : \varphi$, where x is a proof variable and φ is a formula, such that no proof variable occurs twice. Contexts are traditionally denoted by Γ, Δ, etc. If this does not lead to confusion we identify contexts with sets of formulas (forgetting the proof variables).

We define the Church style (raw) terms of intuitionistic propositional logic as follows:

$$\begin{gathered} M, N ::= x \mid M[\varphi] \mid \lambda x : \varphi.\, M \mid MN \mid \langle M, N \rangle \mid M\pi_1 \mid M\pi_2 \\ \mid \mathrm{in}_1 M \mid \mathrm{in}_2 M \mid M[x_1 : \varphi.\, N_1;\; x_2 : \psi.\, N_2] \end{gathered}$$

where $x, x_1, x_2 \in \Upsilon$. The set of λ-terms generated in this way is written Λ^p. Again we do not include parentheses in the grammar, but they can be used anywhere to disambiguate parsing. In case this does not lead to confusion, we omit type annotations from terms and write for example $\lambda x.\, M$ instead of $\lambda x\!:\!\varphi.\, M$ or $M[x_1.\, N_1;\; x_2.\, N_2]$ for $M[x_1\!:\!\varphi.\, N_1;\; x_2\!:\!\psi.\, N_2]$. We also use the common convention that application is left-associative: MNP stands for $(MN)P$. We often write e.g., ME not only for application of N to a term E but also for any elimination: E can be a projection π_1 or π_2, or a $\vee$-eliminator $[x\!:\!\varphi.\, P;\; y\!:\!\psi.\, Q]$ or a $\bot$-eliminator $[\varphi]$.

The set of *free variables* in a term is defined as

- $\mathrm{FV}(x) = \{x\}$,
- $\mathrm{FV}(\lambda x\!:\!\varphi.\, M) = \mathrm{FV}(M)\backslash\{x\}$,
- $\mathrm{FV}(MN) = \mathrm{FV}(\langle\, M, N\,\rangle) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$,
- $\mathrm{FV}(M\pi_i) = \mathrm{FV}(\mathrm{in}_i M) = \mathrm{FV}(M[\varphi]) = \mathrm{FV}(M)$ for $i = 1, 2$,
- $\mathrm{FV}(M[x\!:\!\varphi.\, N_1;\; y\!:\!\psi.\, N_2]) = \mathrm{FV}(M) \cup (\mathrm{FV}(N_1)\backslash\{x\}) \cup (\mathrm{FV}(N_2)\backslash\{y\})$.

As usual the terms are tacitly considered up to α-conversion so that the names of nonfree variables are not relevant. *Closed terms* are terms that have no occurrences of free variables. We use the notation $M[x := N]$ for capture-free substitution of N for the free occurrences of x in M.

The natural deduction inference rules of IPC are presented in Figure 1 in the form of type-assignment system deriving judgements of the form $\Gamma \vdash M : \varphi$ (read: "M has type φ in Γ" or "M proves φ in Γ"), where Γ is a context and M is a proof term. From time to time we use the simplified notation $\Gamma \vdash \sigma$ to state that $\Gamma \vdash M : \sigma$ holds for some M. If Γ is known, implicit, or irrelevant we can simplify the statement $\Gamma \vdash M : \tau$ to $M : \tau$ (read "M has type τ").

Reductions. An introduction-elimination pair constitutes a *beta-redex*, and we have the following set of beta-reduction rules for all the logical connectives except $\bot$:

$$\begin{array}{lll}
(\lambda x\!:\!\sigma.M)N & \Rightarrow_\beta & M[x := N], \\
\langle\, M, N\,\rangle\pi_1 & \Rightarrow_\beta & M, \\
\langle\, M, N\,\rangle\pi_2 & \Rightarrow_\beta & N, \\
(\mathrm{in}_1 M)[x\!:\!\varphi.\, N_1;\; y\!:\!\psi.N_2] & \Rightarrow_\beta & N_1[x := M], \\
(\mathrm{in}_2 M)[x\!:\!\varphi.\, N_1;\; y\!:\!\psi.\, N_2] & \Rightarrow_\beta & N_2[y := M].
\end{array}$$

Other redexes represent elimination steps applied to a conclusion of a $\vee$- or $\bot$-elimination. The following rules, called *permutations* (aka commuting conversions), permute the "bad" elimination upwards. For the disjunction there is the following general scheme:

$$M[x\!:\!\varphi.\, N_1;\; y\!:\!\psi.\, N_2]E \;\;\Rightarrow_p\;\; M[x\!:\!\varphi.\, N_1 E;\; y\!:\!\psi.\, N_2 E],$$

where E is any eliminator, that means $E \in \Lambda^p$, or $E \in \{\pi_1, \pi_2\}$, $E = [\vartheta]$, or $E = [z\!:\!\vartheta.\, N;\; v\!:\!\varrho.\, Q]$.

Permutations for $M[\varphi]$ follow the pattern

$$M[\varphi]E \;\;\Rightarrow_p\;\; M[\psi],$$

where ψ is the type of $M[\varphi]E$. For example:

$$M[\varphi \Rightarrow \psi]N \Rightarrow_p M[\psi].$$

$$\frac{}{\Gamma, x\!:\!\varphi \vdash x : \varphi}\ (var)$$

$$\frac{\Gamma, x\!:\!\varphi \vdash M : \psi}{\Gamma \vdash \lambda x\!:\!\varphi.\, M : \varphi \to \psi}\ (\to I)$$

$$\frac{\Gamma \vdash M_1 : \varphi \to \psi \quad \Gamma \vdash M_2 : \varphi}{\Gamma \vdash M_1 M_2 : \psi}\ (\to E)$$

$$\frac{\Gamma \vdash M : \varphi \quad \Gamma \vdash N : \psi}{\Gamma \vdash \langle M, N \rangle : \varphi \wedge \psi}\ (\wedge I)$$

$$\frac{\Gamma \vdash M : \varphi \wedge \psi}{\Gamma \vdash M\pi_1 : \varphi}\ (\wedge E1) \qquad \frac{\Gamma \vdash M : \varphi \wedge \psi}{\Gamma \vdash M\pi_2 : \psi}\ (\wedge E2)$$

$$\frac{\Gamma \vdash M : \varphi}{\Gamma \vdash \mathrm{in}_1 M : \varphi \vee \psi}\ (\vee I1) \qquad \frac{\Gamma \vdash M : \psi}{\Gamma \vdash \mathrm{in}_2 M : \varphi \vee \psi}\ (\vee I2)$$

$$\frac{\Gamma \vdash M : \varphi \vee \psi \quad \Gamma, x\!:\!\varphi \vdash N_1 : \rho \quad \Gamma, y\!:\!\psi \vdash N_2 : \rho}{\Gamma \vdash M[x\!:\!\varphi.\, N_1;\ y\!:\!\psi.\, N_2] : \rho}\ (\vee E)$$

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash M[\varphi] : \varphi}\ (\bot E)$$

FIGURE 1. Proof assignment in IPC.

The relation $\to$ is the contextual closure of rules $\Rightarrow_\beta$ and $\Rightarrow_p$, and $\twoheadrightarrow$ stands for the reflexive-transitive closure of $\to$.

The system Λ^p has a number of important consistency features.

THEOREM 1. *The system Λ^p has the following properties*:

1. *Subject reduction*: *if $\Gamma \vdash M : \sigma$ and $M \twoheadrightarrow N$ then $\Gamma \vdash N : \sigma$.*
2. *Church–Rosser property*: *if $M \twoheadrightarrow N$ and $M \twoheadrightarrow P$ then there is a term Q such that $N \twoheadrightarrow Q$ and $P \twoheadrightarrow Q$.*
3. *Strong normalisation*: *every reduction $M_1 \to M_2 \to \cdots$ is finite.*

PROOF. Part (1) can be easily verified by observing that every reduction rule preserves typing. Part (2) follows from general results on higher-order rewriting [39, Chapter 11.6], because the rules are left-linear and nonoverlapping. For part (3), see [10]. $\dashv$

A type τ is *inhabited* iff there is a closed term M such that $\vdash M : \tau$ (an *inhabitant*).

Long normal forms. It follows from Theorem 1(3) that every inhabited type has a normal inhabitant. To organize and narrow proof search it is convenient to use a stricter notion of long normal form (lnf). In the lambda-calculus (or equivalently: in natural deduction) long normal forms play a role similar to *focusing* [20, 25] in sequent calculus.

We say that a term M such that $\Gamma \vdash M : \varphi$ is in *long normal form* when one of the following cases holds:

- M is a constructor $\lambda x.\, N$, $\langle N_1, N_2 \rangle$, $\mathrm{in}_1 N$, or $\mathrm{in}_2 N$, where terms N, N_1, and N_2 are lnf.
- $M = xE_1 \dots E_n$, where $E_1, \dots, E_n$ are projections or terms in long normal form, and φ is an atom.
- $M = xE_1 \dots E_n E$, where $E_1, \dots, E_n$ are projections or terms in long normal form, and E is a $\vee$- or $\bot$-eliminator, and φ is either an atom or a disjunction.

For example, let

$$\Gamma = \{x : \alpha \to p,\ y : \alpha,\ z : \alpha \to \beta \vee \gamma,\ u_1 : \beta \to p,\ u_2 : \gamma \to p\},$$

where p is an atom. In this context $\lambda w : \alpha.\, xw$ is an lnf of type $\alpha \to p$, and $zy[v_1 : \beta.\, u_1 v_1; v_2 : \gamma.\, u_2 v_2]$ is an lnf of type p. Also $zy[v_1 : \beta.\, \mathrm{in}_1 v_1;\ v_2 : \gamma.\, \mathrm{in}_2 v_2]$ is an lnf of type $\beta \vee \gamma$, while the mere application zy is not.

Lemma 2 [42]. *If $\Gamma \vdash \varphi$, then $\Gamma \vdash M : \varphi$, for some long normal form M.*

Kripke semantics. A Kripke model is a triple of the form

$$\mathcal{C} = \langle C, \leq, \Vdash \rangle,$$

where C is a nonempty set, the elements of which are called *states*, $\leq$ is a partial order in C and $\Vdash$ is a binary relation between elements of C and propositional variables. The relation $\Vdash$, read as *forces*, obeys the standard monotonicity condition: if $c \leq c'$ and $c \Vdash p$ then $c' \Vdash p$. Without loss of generality we may assume that C is finite, cf. [34], [8, Section 3].

Kripke semantics for IPC is defined as follows. If $\mathcal{C} = \langle C, \leq, \Vdash \rangle$ is a Kripke model then

- $c \Vdash \varphi \vee \psi$ if and only if $c \Vdash \varphi$ or $c \Vdash \psi$,
- $c \Vdash \varphi \wedge \psi$ if and only if $c \Vdash \varphi$ and $c \Vdash \psi$,
- $c \Vdash \varphi \to \psi$ if and only if for all $c' \geq c$ if $c' \Vdash \varphi$ then $c' \Vdash \psi$,
- $c \Vdash \bot$ does not hold.

We write $c \Vdash \Gamma$, when c forces all formulas in Γ. And $\Gamma \Vdash \varphi$ means that $c \Vdash \Gamma$ implies $c \Vdash \varphi$ for each Kripke model $\langle C, \leq, \Vdash \rangle$ and each $c \in C$.

The following completeness theorem holds (see, e.g., [12]):

Theorem 3. *For each Γ and φ, it holds that $\Gamma \vdash \varphi$ if and only if $\Gamma \Vdash \varphi$.*

The implicational fragment. In this article we are mostly interested in the implicational fragment IIPC of IPC. The formulas of IIPC (also known as simple types) are defined by the grammar

$$\sigma, \tau ::= p \mid \sigma \to \tau,$$

where $p \in \mathcal{X}$.

Any formula in IIPC can be written as $\sigma = \sigma_1 \to \cdots \to \sigma_n \to p$, where $n \geq 0$, and p is a type atom. Types $\sigma_1, \ldots, \sigma_n$ are the *arguments*, and the atom p is called the *target* of σ, written $p = \mathrm{tg}(\sigma)$.

The *order* $r(\sigma)$ of an implicational formula is defined as follows: an atom is of order 0, and the order of $\sigma \to \tau$ is the maximum of $r(\tau)$ and $r(\sigma) + 1$. In other words, if p is an atom, then

$$r(\sigma_1 \to \cdots \to \sigma_k \to p) = 1 + \max_i r(\sigma_i).$$

The restricted set $\Lambda_\to$ of IIPC proof-terms is defined by the pseudo-grammar:

$$M, N ::= x \mid \lambda x : \sigma.\, M \mid MN.$$

The relevant rules in Figure 1 are (var), $(\to I)$, and $(\to E)$, i.e., the type-assignment rules of the ordinary simply typed lambda-calculus.

§3. Automata for logic.

It follows from Lemma 2 that every provable formula has a long normal proof. This yields a simple proof-search algorithm, which is essentially implicit in [44], and hence called the Wajsberg algorithm (WA).[2]

To present the algorithm we first define the set $\mathsf{TG}(\varphi)$ of *targets of* φ. Targets are always atoms or disjunctions.

- $\mathsf{TG}(\mathbf{a}) = \{\mathbf{a}\}$, when $\mathbf{a}$ is an atom (a propositional variable or $\bot$).
- $\mathsf{TG}(\tau \to \sigma) = \mathsf{TG}(\sigma)$.
- $\mathsf{TG}(\tau \vee \sigma) = \{\tau \vee \sigma\}$.
- $\mathsf{TG}(\tau \wedge \sigma) = \mathsf{TG}(\tau) \cup \mathsf{TG}(\sigma)$.

Clearly, $\mathsf{TG}(\varphi) = \{\mathrm{tg}(\varphi)\}$, when φ is an implicational formula.

We define the family $tr(\alpha, \varphi)$ of *traces to* α *in* φ. Each trace is a set of formulas.

- $tr(\alpha, \varphi) = \varnothing$ if $\alpha \notin \mathsf{TG}(\varphi)$.
- $tr(\alpha, \alpha) = \{\varnothing\}$.
- $tr(\alpha, \tau \to \sigma) = \{\{\tau\} \cup T \mid T \in tr(\alpha, \sigma)\}$.
- $tr(\alpha, \tau \wedge \sigma) = tr(\alpha, \tau) \cup tr(\alpha, \sigma)$.

For example, $tr(p, r \to (p \wedge (q \to p) \wedge (s \to p \vee q)) = \{\{r\}, \{r, q\}\}$.

LEMMA 4. *Let* $(x : \psi) \in \Gamma$ *and* $T \in tr(\alpha, \psi)$. *If* $\Gamma \vdash \rho$, *for all* $\rho \in T$, *then* $\Gamma \vdash xE_1 \ldots E_n : \alpha$, *where* $n \geq 0$ *and* $E_1, \ldots, E_n$ *are some terms or projections.*

PROOF. Induction with respect to ψ. For example, assume $\psi = \psi_1 \wedge \psi_2$, and let $T \in tr(\alpha, \psi_1)$. Given that, we apply the induction hypothesis so that we obtain $\Gamma, y : \psi_1 \vdash yE_1 \ldots E_n : \alpha$, where $n \geq 0$, so $\Gamma \vdash x\pi_1 E_1 \ldots E_n : \alpha$. ⊣

LEMMA 5. *Assume that* $(x : \psi) \in \Gamma$ *and* $\Gamma \vdash xE_1 \ldots E_m : \varphi$, *where* $E_1, \ldots, E_m$ *are terms or projections and* φ *is an atom or a disjunction. Let* $J = \{j \mid E_j \text{ is a term}\}$ *and let* $\Gamma \vdash E_j : \sigma_j$, *for all* $j \in J$. *Define* $T = \{\sigma_j \mid j \in J\}$. *Then* $\varphi \in \mathsf{TG}(\psi)$, *and* $T \in tr(\varphi, \psi)$.

PROOF. Induction with respect to m. For example, if $\psi = \psi_1 \to \psi_2$, then we apply the induction hypothesis to $\Gamma, y : \psi_1 \vdash yE_2 \ldots E_m : \varphi$. Consequently we obtain $T' = \{\sigma_j \mid j \in J \wedge j > 1\} \in tr(\varphi, \psi_2)$, and $T = T' \cup \{\psi_1\} \in tr(\varphi, \psi)$. ⊣

[2]See [3] for a correction of the proof in [44].

For a given judgement $\Gamma \vdash \varphi$, the WA attempts (implicitly) to construct a long normal proof term. It proceeds as follows:

1. If $\varphi = \tau \wedge \sigma$, call $\Gamma \vdash \tau$ and $\Gamma \vdash \sigma$.
2. If $\varphi = \tau \to \sigma$, call $\Gamma, \tau \vdash \sigma$.
3. If φ is an atom or a disjunction, choose $\psi \in \Gamma$ and $\alpha \in \mathsf{TG}(\psi)$ such that either α is a disjunction, or $\alpha = \bot$, or α is a propositional variable and $\alpha = \varphi$. Then choose $T \in tr(\alpha, \psi)$, and:
   - Call $\Gamma \vdash \rho$, for every $\rho \in T$;
   - If $\alpha = \beta \vee \gamma$, call $\Gamma, \beta \vdash \varphi$ and $\Gamma, \gamma \vdash \varphi$ in addition.

The procedure accepts in case (3), when φ is an atom in Γ, as there is nothing to call.

With respect to IIPC case (1) disappears and case (3) simplifies to

3'. If φ is an atom then choose $\rho_1 \to \cdots \to \rho_n \to \varphi \in \Gamma$ and call $\Gamma \vdash \rho_i$, for all $i = 1, \dots, n$.

We thus obtain the algorithm for inhabitation in the simply typed lambda-calculus known as the *Ben–Yelles algorithm* [2].

The most important properties of WA are the following.

LEMMA 6.

1. *The algorithm WA accepts an IPC judgement if and only if it is provable.*
2. *All formulas occurring in any run of the algorithm are subformulas of the formulas occurring in the initial judgement.*

PROOF. (1) We prove that a judgement $\Gamma \vdash \varphi$ is accepted if and only if there exists a long normal form of type φ in Γ. From left to right we proceed by induction with respect to the definition of the algorithm, using Lemma 4. In cases (1) and (2) the term M is a constructor, in case (3) it is an eliminator with a head variable x of type ψ. For example, if $\varphi = \tau \vee \sigma$ and $\psi = \gamma_1 \to \gamma_2 \to \alpha \vee \beta$ then $M = xN_1N_2[z : \alpha.\, P;\ v : \beta.\, Q]$, where N_1, N_2, P, Q are long normal forms obtained in the four recursive (or parallel) calls.

From right to left we work by induction with respect to the size of the lnf using Lemma 5. For example, in the case of the term $xE_1 \dots E_m[u : \alpha.\, P;\ v : \beta.\, Q]$, types of $E_1, \dots, E_m$ make a trace T to $\alpha \vee \beta$ in ψ, and we can use induction for $\Gamma, u : \alpha \vdash P : \varphi$ and $\Gamma, v : \alpha \vdash Q : \varphi$.

(2) In each of the steps of WA each new formula must be a subformula of either the present proof goal or one of the assumptions. ⊣

Monotonic automata. We define here a natural notion of automaton used as operational semantics of IPC. This notion is a simplification of the automata introduced by Barendregt, Dekkers, and Schubert [32] and of those used in [33] (but differs significantly from the notion introduced by Tzevelekos [41]).

The idea is simple. If we read a proof task $\Gamma \vdash \varphi$ as a configuration of a machine, then any action taken by WA results in expanding the memory Γ and proceeding to a new internal state, yielding a new task (or a new configuration) $\Gamma' \vdash \varphi'$. For example, if an assumption of the form $(p \to q) \to r \in \Gamma$ is used to derive $\Gamma \vdash r$, then the next task $\Gamma, p \vdash q$ is a result of executing an instruction that can be written as

r : $\texttt{check}\ (p \to q) \to r$; $\texttt{set}\ p$; $\texttt{jmp}\ q$ ("in state r check the presence of $(p \to q) \to r$ in memory, add p to the storage and go to state q").

A *monotonic automaton* is therefore defined as $\mathcal{M} = \langle Q, R, f, \mathcal{I} \rangle$, where

- Q is a finite set of states, with $f \in Q$ as the final state;
- R is a finite set of registers;
- $\mathcal{I}$ is a finite set of instructions of the form:
  (1) q : $\texttt{check}\ S_1$; $\texttt{set}\ S_2$; $\texttt{jmp}\ p$, or
  (2) q : $\texttt{jmp}\ p_1\ \texttt{and}\ p_2$,
  where $q, p, p_1, p_2 \in Q$ and $S_1, S_2 \subseteq R$.

We define a *configuration* of $\mathcal{M}$ as a pair $\langle q, S \rangle$, where $q \in Q$ and $S \subseteq R$. Let $I \in \mathcal{I}$. The transition relation $\langle q, S \rangle \to_I \langle p, S' \rangle$ holds

- for I of type (1), when $S_1 \subseteq S$, $S' = S \cup S_2$;
- for I of type (2), when $S = S'$, and $p = p_1$ or $p = p_2$.

A configuration $\langle q, S \rangle$ is *accepting* when either $q = f$, or

- $\langle q, S \rangle \to_I \langle p, S' \rangle$, where I is of type (1), and $\langle p, S' \rangle$ is accepting, or
- $\langle q, S \rangle \to_I \langle p_1, S \rangle$ and $\langle q, S \rangle \to_I \langle p_2, S \rangle$, where I is of type (2), and both $\langle p_1, S \rangle$ and $\langle p_2, S \rangle$ are accepting.

Observe that a monotonic automaton is an alternating machine. Instructions of type (2) introduce *universal branching*, and *nondeterminism* occurs when more than one instruction is applicable in a state.[3] The name "monotonic" is justified by the memory usage: registers are write-once devices, once raised (set to 1) they cannot be reset to zero. Note also that all tests are positive: the machine cannot see that a register is off. A *nondeterministic* automaton is one without universal branching (cf. Section 5.2).

It should be clear that our definition is motivated by proof search. Indeed, the algorithm WA is almost immediately implemented as an automaton.

PROPOSITION 7. *Given a formula Φ in IPC, one can construct (in logspace) a monotonic automaton $\mathcal{M}_\Phi$ and state q so that $\vdash \Phi$ if and only if the configuration $\langle q, \varnothing \rangle$ of $\mathcal{M}_\Phi$ is accepting.*

PROOF (SKETCH). Let S be the set of all subformulas of Φ. Define automaton $\mathcal{M} = \langle Q, R, f, \mathcal{I} \rangle$, where

- $R = S$ is the set of registers.
- The set of states Q contains S and some auxiliary states.

Under this definition, a judgement $\Gamma \vdash \varphi$ corresponds directly to a configuration $\langle \varphi, \Gamma \rangle$ of $\mathcal{M}$. The instructions of the automaton now implement cases (1–3) of WA. Of course the following instructions are in $\mathcal{I}$:

1. φ : $\texttt{jmp}\ \tau\ \texttt{and}\ \sigma$, for each $\varphi = \tau \wedge \sigma \in S$;
2. φ : $\texttt{check}\ \varnothing$; $\texttt{set}\ \tau$; $\texttt{jmp}\ \sigma$, for each $\varphi = \tau \to \sigma \in S$.

Case (3) of WA splits into three subcases handled with help of auxiliary states, and depending on a choice of a formula $\psi \in S$.

[3]But states themselves are not classified as existential or universal.

If φ is an atom, $\varphi \in \mathsf{TG}(\psi)$, for some $\psi \in S$, and $\{\rho_1, \dots, \rho_m\} \in tr(\varphi, \psi)$, then $\mathcal{I}$ contains a sequence of instructions (using $m - 2$ brand new states) abbreviated as:

3a. φ : `check` ψ; `set` $\varnothing$; `jmp` $\rho_1, \dots, \rho_m$.

If φ is an atom or a disjunction, and $\bot \in \mathsf{TG}(\psi)$, for some $\psi \in S$, and $\{\rho_1, \dots, \rho_m\} \in tr(\bot, \psi)$, then $\mathcal{I}$ also contains similar instructions:

3b. φ : `check` ψ; `set` $\varnothing$; `jmp` $\rho_1, \dots, \rho_m$.

If φ is an atom or a disjunction, $\alpha \vee \beta \in \mathsf{TG}(\psi)$, for some $\psi \in S$, and $\{\rho_1, \dots, \rho_m\} \in tr(\alpha \vee \beta, \psi)$, then $\mathcal{I}$ contains instructions (using m auxiliary states including s_1 and s_2):

3c. φ : `check` ψ; `set` $\varnothing$; `jmp` $\rho_1, \dots, \rho_m, s_1, s_2$;
s_1 : `check` $\varnothing$; `set` α; `jmp` φ;
s_2 : `check` $\varnothing$; `set` β; `jmp` φ.

For example, if $\psi = \alpha \to \beta \vee \gamma \in \Gamma$, and $\varphi \in S$ is an atom, then the following instructions are in $\mathcal{I}$ (where p_1, p_2, p_3, p_4 are fresh auxiliary states):

$$\begin{aligned}
\varphi &: \texttt{check } \psi;\ \texttt{set } \varnothing;\ \texttt{jmp } p_1\,;\\
p_1 &: \texttt{jmp } \alpha \texttt{ and } p_2\,;\\
p_2 &: \texttt{jmp } p_3 \texttt{ and } p_4\,;\\
p_3 &: \texttt{check } \varnothing;\ \texttt{set } \beta;\ \texttt{jmp } \varphi\,;\\
p_4 &: \texttt{check } \varnothing;\ \texttt{set } \gamma;\ \texttt{jmp } \varphi\,.
\end{aligned}$$

By straightforward induction one proves that a configuration of the form $\langle \varphi, \Gamma \rangle$ is accepting if and only if $\Gamma \vdash M : \varphi$ for some lnf M. It remains to define q as Φ, and observe that by Lemma 6(2) the automaton can be computed in logspace. $\dashv$

Complexity. The *halting problem* for monotonic automata is

"*Given $\mathcal{M}, q, S$, is $\langle q, S \rangle$ an accepting configuration of $\mathcal{M}$?*"

In the remainder of this section we show that this problem is PSPACE-complete. The upper bound is routine.

LEMMA 8. *It is decidable in polynomial space if a given configuration of a monotone automaton is accepting. For nondeterministic automata* (*with no universal branching*) *the problem is in* NP.

PROOF. An accepting computation of an alternating automaton can be seen as a tree. Every branch of the tree is a (finite or infinite) sequence $\langle q_0, S_0 \rangle, \langle q_1, S_1 \rangle, \langle q_2, S_2 \rangle, \dots$ of configurations, such that $S_0 \subseteq S_1 \subseteq S_2 \subseteq \cdots$. If the number of states and the number of registers are bounded by n then a configuration must necessarily be repeated after at most n^2 steps. An alternating Turing Machine working in time n^3 can therefore verify if a given configuration is accepting, and our halting problem is in APTIME $\subseteq$ PSPACE, cf. [27, Chapter 19]. In case of no universal branching, a nondeterministic Turing Machine suffices. $\dashv$

The next example hints on the technique used to show the lower bound.

Example 9. Consider a finite automaton $\mathcal{A}$, with states $\{0, \dots, k\}$, the initial state 0, and the final state k. Given a string $w = a_1 \dots a_n$, we define a monotonic $\mathcal{M}$ such that $\mathcal{A}$ accepts w if and only if the initial configuration $\langle q^0, r_0^0 \rangle$ of $\mathcal{M}$ is accepting.[4]

States of $\mathcal{M}$ are $q^0, q^1, \dots, q^n, f$, where q^0 is initial and f is final. Registers are r_i^t, for $t \leq n$ and $i \leq k$. For all $t = 0, \dots, n-1$, we have an instruction

$$q^t : \texttt{check}\ r_i^t;\ \texttt{set}\ r_j^{t+1};\ \texttt{jmp}\ q^{t+1},$$

whenever $\mathcal{A}$, reading a_{t+1} in state i, can enter state j. For $t = n$, we take at last

$$q^n : \texttt{check}\ r_k^n;\ \texttt{set}\ \varnothing;\ \texttt{jmp}\ f.$$

Then an accepting computation of the automaton $\mathcal{A}$, consisting of states $0, i_1, i_2, \dots, i_n = k$, is represented by a computation of $\mathcal{M}$, ending in $\langle f, r_0^0, r_{i_1}^1, \dots, r_{i_n}^n \rangle$. Note that the instructions of $\mathcal{M}$ are all of type (1), i.e., there is no alternation.

Correctness: By induction with respect to $n - t$ one shows that a configuration of the form $\langle q^t, r_0^0, \dots, r_{i_t}^t \rangle$ is accepting if and only if $\mathcal{A}$ accepts the suffix $a_{t+1} \dots a_n$ of w from state i_t.

In order to simplify the proof of Pspace-hardness, let us begin with the following observation. Every language $L \in$ Pspace reduces in logarithmic space to some context-sensitive language L', recognizable by a linear bounded automaton (LBA), cf. [16, Chapter 9.3]. Indeed, for any $L \in \mathrm{Dspace}(n^k)$, take the language $L' = \{w\$^{n^k} \mid w \in L \wedge |w| = n\}$, where $|w|$ denotes the length of the word w. Hence it suffices to reduce the halting problem for LBA (aka In-place Acceptance problem, cf. [27, Chapter 19]) to the halting problem of monotonic automata. This retains the essence of Pspace but reduces the amount of bookkeeping.

Given a linear bounded automaton $\mathcal{A}$ and an input string $w = x^1 \dots x^n$, we construct a monotonic automaton $\mathcal{M}$ and an initial configuration C_0 such that

$$\mathcal{A} \text{ accepts } w \text{ if and only if } C_0 \text{ is an accepting configuration of } \mathcal{M}.$$

Let p be a polynomial such that $\mathcal{A}$ works in time $2^{p(n)}$. The alternating automaton $\mathcal{M}$ simulates $\mathcal{A}$ by splitting the $2^{p(n)}$ steps of computation recursively into halves and executing the obtained fractions concurrently. The "history" of each branch of the computation tree of $\mathcal{M}$ is recorded in its registers. For every $d = 0, \dots, p(n)$, there are three groups of registers (marked B, E, H) representing $\mathcal{A}$'s configurations at the beginning (B) and at the end (E) of a computation segment of up to 2^d steps, and halfway (H) through that segment. That is, for any $i = 1, \dots, n$, $d = 0, \dots, p(n)$, for any state q of $\mathcal{A}$, and for any tape symbol a of $\mathcal{A}$, the automaton $\mathcal{M}$ has the following registers:

- $s(B, d, q), s(H, d, q), s(E, d, q)$ – "the current state of $\mathcal{A}$ is q";
- $c(B, d, i, a), c(H, d, i, a), c(E, d, i, a)$ – "the symbol at position i is a";
- $h(B, d, i), h(H, d, i), h(E, d, i)$ – "the machine head scans position i".

By B_d, H_d, E_d we denote the sets of all registers indexed by d and, respectively, by B, H, E. A set of registers $S \subseteq X_d$ is an X, d-*code* of a configuration of $\mathcal{A}$, when

[4]We write sets without { } whenever it is convenient.

S contains exactly one register of the form $s(X, d, q)$, exactly one $h(X, d, j)$, and, for every i, exactly one $c(X, d, i, a)$.

The initial configuration of $\mathcal{M}$ is $C_0 = \langle 0, S_0 \rangle$, where S_0 is the $B, p(n)$-code of the initial configuration of $\mathcal{A}$, that is,

– $S = \{s(B, p(n), q_0), c(B, p(n), 1, x^1), \dots, c(B, p(n), n, x^n), h(B, p(n), 1)\}$.

The machine $\mathcal{M}$ works as follows.

In the initial phase (commencing in state 0) it guesses the final configuration of $\mathcal{A}$, and sets the appropriate registers in $E_{p(n)}$ to obtain the $E, p(n)$-code of that final configuration. Then $\mathcal{M}$ enters state $Q_{p(n)}$.

Assume now that the machine is in configuration $\langle Q_d, S \rangle$, where $d > 0$, and S contains:

– a B, d-code of some configuration C^b of $\mathcal{A}$;
– an E, d-code of some configuration C^e of $\mathcal{A}$.

The following steps are now executed.

(1) An intermediate configuration C^h is guessed, i.e., registers in H_d are nondeterministically set to obtain an H, d-code of C^h. The machine selects an adequate sequence of instructions from the set below (where q', a, and j are arbitrary):

$$\begin{aligned} Q_d &: \texttt{check}\ \varnothing;\ \texttt{set}\ s(H, d, q');\ \texttt{jmp}\ Q_d^1; \\ Q_d^i &: \texttt{check}\ \varnothing;\ \texttt{set}\ c(H, d, i, a);\ \texttt{jmp}\ Q_d^{i+1}, \quad \text{for } i = 1, \dots, n; \\ Q_d^{n+1} &: \texttt{check}\ \varnothing;\ \texttt{set}\ h(H, d, j);\ \texttt{jmp}\ Q'_d. \end{aligned}$$

(2) The machine makes a universal split into states Q_d^B and Q_d^E.

(3) In state Q_d^B registers in $S \cap B_d$ are copied to corresponding registers in B_{d-1}. This has to be done nondeterministically, by guessing which registers in $S \cap B_d$ are set. The relevant instructions are:

$$\begin{aligned} Q_d^B &: \texttt{check}\ s(B, d, q);\ \texttt{set}\ s(B, d-1, q);\ \texttt{jmp}\ Q_d^{B,1}; \\ Q_d^{B,i} &: \texttt{check}\ c(B, d, i, a);\ \texttt{set}\ c(B, d-1, i, a);\ \texttt{jmp}\ Q_d^{B,i+1}, \quad \text{for } i = 1, \dots, n; \\ Q_d^{B,n+1} &: \texttt{check}\ h(B, d, j);\ \texttt{set}\ h(B, d-1, j);\ \texttt{jmp}\ Q_d^{BE}. \end{aligned}$$

Then registers in $S \cap H_d$ are copied to E_{d-1} in a similar way. In short, this can be informally written as $B_{d-1} := B_d; E_{d-1} := H_d$. Then the machine enters state Q_{d-1}.

(4) In state Q_d^E, the operations follow a similar scheme, that can be written in short as

$$B_{d-1} := H_d; E_{d-1} := E_d;\ \texttt{jmp}\ Q_{d-1}.$$

The above iteration splits the computation of $\mathcal{M}$ into $2^{p(n)}$ branches, each eventually entering state Q_0. At this point we verify the correctness. The sets $S \cap B_0$ and $S \cap E_0$ should now encode some configurations C^b and C^e of $\mathcal{A}$ such that either $C^b = C^e$, or C^e is obtained from C^b in one step. This can be nondeterministically verified, and afterwards $\mathcal{M}$ enters its final state.

This last phase uses, in case $C^b = C^e$, the supply of instructions below (the other variant can be handled similarly).

$$\begin{aligned}
Q_0 &: \texttt{check}\ s(B, d, q);\ \texttt{set}\ \varnothing;\ \texttt{jmp}\ Q_d^s(q);\\
Q_d^s(q) &: \texttt{check}\ s(E, d, q);\ \texttt{set}\ \varnothing;\ \texttt{jmp}\ Q_d^{c,1};\\
Q_d^{c,i} &: \texttt{check}\ c(B, d, i, a);\ \texttt{set}\ \varnothing;\ \texttt{jmp}\ Q_d^{c,i}(a);\\
Q_d^{c,i+1}(a) &: \texttt{check}\ c(E, d, i, a);\ \texttt{set}\ \varnothing;\ \texttt{jmp}\ Q_d^{c,i+1};\\
Q_d^{c,n+1} &: \texttt{check}\ h(B, d, j);\ \texttt{set}\ \varnothing;\ \texttt{jmp}\ Q_d^h(j);\\
Q_d^h(j) &: \texttt{check}\ h(E, d, j);\ \texttt{set}\ \varnothing;\ \texttt{jmp}\ f.
\end{aligned}$$

The main property of the above construction is the following.

LEMMA 10. *Let S be a set of registers such that:*

– *$S \cap B_d$ is a B, d-code of some configuration C^b of $\mathcal{A}$;*
– *$S \cap E_d$ is an E, d-code of some configuration C^e of $\mathcal{A}$.*

In addition, assume that $S \cap H_d = \varnothing$, as well as $S \cap (B_e \cup H_e \cup E_e) = \varnothing$, for all $e < d$. Then the following are equivalent:

1. *$\langle Q_d, S \rangle$ is an accepting configuration of $\mathcal{M}$;*
2. *C^e is reachable from C^b in at most 2^d steps of $\mathcal{A}$.*

PROOF. (1) $\Rightarrow$ (2): Induction with respect to the definition of acceptance.
(2) $\Rightarrow$ (1): Induction with respect to d. ⊣

THEOREM 11. *The halting problem for monotonic automata is* PSPACE*-complete.*

PROOF. Lemma 8 implies that the problem belongs to PSPACE. The hardness part is a consequence of Lemma 10 applied for $d = p(n)$ with C^b and C^e being, respectively, the initial and final configuration of $\mathcal{A}$. ⊣

Automata to formulas. In order to finish our reduction of provability in IPC to provability in IIPC we need to prove a specific converse of Proposition 7. Consider a monotonic automaton $\mathcal{M} = \langle Q, R, f, \mathcal{I} \rangle$, and an ID of the form $C_0 = \langle q_0, S_0 \rangle$. Without loss of generality we can assume that $Q \cap R = \varnothing$. Using states and registers of $\mathcal{M}$ as propositional atoms, we define a set of axioms Γ so that $\Gamma \vdash q_0$ if and only if C_0 is accepting. The set Γ contains the atoms $S_0 \cup \{f\}$; other axioms in Γ represent instructions of $\mathcal{M}$.

An axiom representing q : $\texttt{check}\ S_1$; $\texttt{set}\ S_2$; $\texttt{jmp}\ p$, where $S_1 = \{s_1^1, \dots, s_1^k\}$ and $S_2 = \{s_2^1, \dots, s_2^\ell\}$, is:

$$s_1^1 \to \cdots \to s_1^k \to (s_2^1 \to \cdots \to s_2^\ell \to p) \to q. \tag{1}$$

And for every instruction q : $\texttt{jmp}\ p_1\ \texttt{and}\ p_2$, there is an axiom

$$p_1 \to p_2 \to q. \tag{2}$$

Observe that all the above axioms are of order at most two, hence the formula $\Gamma \to q_0$ has order at most three.

LEMMA 12. *Given the above definitions, the configuration $\langle q_0, S_0 \rangle$ is accepting if and only if $\Gamma \vdash q_0$ holds.*

Proof. For every $S \subseteq R$ and $q \in Q$, we prove that $\Gamma, S \vdash q$ if and only if $C = \langle\, q, S \cup S_0 \,\rangle$ is accepting. We think of Γ as of a type environment where each axiom is a declaration of a variable.

($\Leftarrow$) Induction with respect to long normal proofs. With $\to$ as the only connective, any normal proof T of an atom q must be a variable or an application, say $T = xN_1 \dots N_m$. The case of $x : f$ (i.e., $q = f$) is obvious; otherwise the type of x corresponds to an instruction. There are two possibilities:

(1) If $x : s_1^1 \to \cdots \to s_1^k \to (s_2^1 \to \cdots \to s_2^\ell \to p) \to q$,
then actually we obtain that $T = xD_1 \dots D_k(\lambda u_1 : s_2^1 \dots \lambda u_\ell : s_2^\ell . P)$. Terms $D_1, \dots, D_k$ are, respectively, of types $s_1^1, \dots, s_1^k$, and they must be variables declared in S, as there are no other assumptions with targets $s_1^1, \dots, s_1^k$. Hence the instruction corresponding to x is applicable at $C = \langle\, q, S \,\rangle$ and yields $C' = \langle\, p, S \cup S' \,\rangle$, where $S' = S \cup \{s_2^1, \dots, s_2^\ell\}$. In addition we have $\Gamma, S \cup S' \vdash P : p$, whence C' is accepting by the induction hypothesis.

(2) If x has type $p_1 \to p_2 \to q$, where $p_1, p_2 \in Q$,
then $T = xT_1T_2$. The appropriate universal instruction leads to two IDs: $C_1 = \langle\, p_1, S \,\rangle$ and $C_2 = \langle\, p_2, S \,\rangle$. The judgements $\Gamma, S \vdash T_1 : p_1$ and $\Gamma, S \vdash T_2 : p_2$ obey the induction hypothesis. Thus C_1, C_2 are accepting and so is C.

($\Rightarrow$) Induction with respect to the definition of acceptance. $\dashv$

Proposition 13. *The halting problem for monotonic automata reduces to the provability problem for formulas in IIPC of order at most three.*

Putting together Propositions 7 and 13 and Theorem 11 we obtain a number of consequences.

Theorem 14. *Provability in IPC, IIPC and IIPC restricted to formulas of order* 3 *are* Pspace-*complete.*

While the statement of Theorem 14 is well-known [36], even for similarly restricted IIPC formulas [26, Theorem 1], the present automata-theoretic proof directly demonstrates that the computational content of proof-search is exactly the same in the full IPC and in the implicational fragment of order 3. Monotonic automata serve here as the natural computational device to illustrate this, and furthermore, they are computationally equivalent to polynomial-space Turing Machines.

Without loss of generality we can interpret problems in Pspace as reachability questions concerning some objects or configurations of polynomial size. The construction used in the proof of Theorem 11 (the simulation of LBA) reflects a natural, possibly interactive, approach to solve such questions: split the reachability task into two, by choosing some intermediate configuration. An example that comes to mind is the Sokoban game: the set of winning positions is a context-sensitive language and one can try to solve the puzzle by determining some milestone states.

Another consequence of our development is that the computational power of IPC is fully contained in IIPC, and in an apparently small fragment.

Theorem 15. *For every formula φ of full IPC one can construct (in logspace) an implicational formula ψ of order at most three such that ψ is provable iff so is φ.*

§4. An intuitionistic order hierarchy.

In Section 3, we observed that provability in the whole IPC is faithfully reflected by provability for formulas of IIPC of that have order at most three. Proving any formula is therefore at most as difficult as proving some formula of order three. But is every formula *equivalent* to one of order three? The answer is negative: in the case of IPC we have a strict order hierarchy of formulas. Define by induction $\varphi^1 = p_1$ and $\varphi^{k+1} = \varphi^k \to p_{k+1}$. That is,

$$\varphi^k = (\cdots((p_1 \to p_2) \to p_3) \to \cdots \to p_{k-1}) \to p_k.$$

LEMMA 16. *Every proof of $\varphi^k \to \varphi^k$ is $\beta\eta$-convertible to the identity combinator $\lambda x.x$.*

PROOF. We prove the following generalized statement by induction with respect to the number k. Let $\mathrm{tg}(\gamma) \notin \{p_1, \dots, p_k\}$, for all $\gamma \in \Gamma$, and let $\Gamma, X : \varphi^k \vdash M : \varphi^k$, where M is in normal form. Then $M =_{\beta\eta} X$. Indeed, first note that X is the only assumption with target p_k, hence for $k = 1$ the claim follows immediately. Otherwise either $M = X$ or $M = \lambda Y. M'$ with a derivation $\Gamma, X : \varphi^k, Y : \varphi^{k-1} \vdash M' : p_k$. This is only possible when $M' = XM''$, where $\Gamma, X : \varphi^k, Y : \varphi^{k-1} \vdash M'' : \varphi^{k-1}$. By the induction hypothesis for $\Gamma' = \Gamma \cup \{x : \varphi^k\}$ and $Y : \varphi^{k-1}$, we have $M'' =_{\beta\eta} Y$, hence $M = \lambda Y. XM'' =_{\beta\eta} \lambda Y. XY =_{\beta\eta} X$. ⊣

THEOREM 17. *For every k, no implicational formula of order less than k is intuitionistically equivalent to φ^k.*

PROOF. If $\vdash \varphi^k \leftrightarrow \alpha$ then there are closed terms $T : \varphi^k \to \alpha$ and $N : \varphi^k \to \alpha$. The composition $\lambda x. N(Tx)$ is a combinator of type $\varphi^k \to \varphi^k$, and by Lemma 16 it must be $\beta\eta$-equivalent to identity. That is, φ^k is a *retract* of α, in the sense of [29]. It thus follows from [29, Proposition 4.5] that α must be of order at least k. ⊣

Interestingly enough, Theorem 17 stays in contrast with the situation in classical logic. Every propositional formula is classically equivalent to a formula in conjunctive normal form (CNF). If implication is the only connective then we have a similar property.

PROPOSITION 18. *Every implicational formula is classically equivalent to a formula of order at most three.*

PROOF. Given a formula of the form $\varphi = \alpha_1 \to \cdots \to \alpha_n \to p$, we first translate the conjunction $\alpha_1 \wedge \cdots \wedge \alpha_n$ into a conjuntive normal form $\beta_1 \wedge \cdots \wedge \beta_m$, so that φ is equivalent to a formula $\psi = \beta_1 \to \cdots \to \beta_m \to p$. Each β_i is a disjunction of literals. For every i, there are two possibilities.

Case 1: At least one component of β_i is a variable, say

$$\beta_i = \neg q_1 \vee \cdots \vee \neg q_r \vee r_1 \vee \cdots \vee r_k \vee s.$$

We replace β_i in ψ by the formula

$$\beta_i' = q_1 \to \cdots \to q_r \to (r_1 \to p) \to \cdots \to (r_k \to p) \to s.$$

Case 2: All components of the formula β_i are negated variables, that means $\beta_i = \neg q_1 \vee \cdots \vee \neg q_r$. Then we replace such β_i by the formula $q_1 \to \cdots \to q_r \to p$.

For example, if $\psi = (s \vee q \vee \neg r) \to (\neg q \vee \neg r \vee \neg s) \to p$ then we rewrite ψ as the formula $(r \to (q \to p) \to s) \to (q \to r \to s \to p) \to p$. It is a routine exercise to see that the final result is a formula of rank at most 3 which is classically equivalent to the initial formula φ (note that if a Boolean valuation falsifies p then it satisfies $p \leftrightarrow \bot$). ⊣

EXAMPLE 19. The formula $\varphi^5 = (((p_1 \to p_2) \to p_3) \to p_4) \to p_5$ is classically equivalent to this "normal form":

$$(p_1 \to (p_2 \to p_5) \to p_4) \to (p_3 \to p_4) \to p_5.$$

REMARK 20. Despite the contrast between the classical CNF collapse and order hierarchy in intuitionistic logic, there is still a strong analogy between CNF and order three fragment of IIPC. The CNF formulas do indeed exhaust the whole expressive power of classical propositional logic, but for a heavy price. The value-preserving translation of a formula to CNF is exponential, hence useless with respect to NP-completeness of CNF-SAT. However, there is a polynomial time translation of SAT to CNF-SAT that preserves satisfiability (probably first formulated by Tseitin about 1966 [40], in its modern formulation available in [16, Theorem 13.2]).

§5. Below order three.

In this section we identify fragments of IIPC corresponding to the complexity classes P, NP, and *co*-NP.

5.1. Formulas of order two: deterministic polynomial time.

Implicational formulas of rank 1 are the same as propositional clauses in logic programming. Therefore decision problem for rank 2 formulas (no matter, classical or intuitionistic) amounts to standard propositional logic programming based on Horn clauses, which is known to be P-complete [9] with respect to logspace reductions.

5.2. Order three minus: class NP.

We define here a subclass of IIPC for which the provability problem is NP-complete.

We split the set $\mathcal{X}$ of propositional variables into two disjoint infinite subsets $\mathcal{X}_0, \mathcal{X}_1 \subseteq \mathcal{X}$, called, respectively, *data* and *control* variables. The role of control variables is to occur as targets, the data variables only occur as arguments. The set of formulas $\mathbb{T}_{3^-}$ is defined by the grammar:

$$\begin{aligned}
\mathbb{T}_{3^-} &::= \mathcal{X}_1 \mid \mathbb{T}_{2^-} \to \mathbb{T}_{3^-} \mid \mathcal{X}_0 \to \mathbb{T}_{3^-}\\
\mathbb{T}_{2^-} &::= \mathcal{X}_1 \mid \mathcal{X}_0 \to \mathbb{T}_{2^-} \mid \mathbb{T}_{1^-} \to \mathbb{T}_{1^-}\\
\mathbb{T}_{1^-} &::= \mathcal{X}_1 \mid \mathcal{X}_0 \to \mathbb{T}_{1^-}.
\end{aligned}$$

Formulas in $\mathbb{T}_{1^-}$ are of the form $p_1 \to \cdots \to p_n \to q$, where $p_i \in \mathcal{X}_0$ and $q \in \mathcal{X}_1$. The set $\mathbb{T}_{2^-}$ consists of formulas of order at most two, with an $\mathcal{X}_1$ target, and with at most one argument in $\mathbb{T}_{1^-}$, and all other arguments being variables in $\mathcal{X}_0$. Finally the $\mathbb{T}_{3^-}$ formulas are of shape $\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to q$, where $q \in \mathcal{X}_1$ and $\sigma_i \in \mathbb{T}_{2^-} \cup \mathcal{X}_0$, for $i = 1, \dots, n$.

LEMMA 21. *Proof search for formulas in* $\mathbb{T}_{3^-}$ *is in* NP.

PROOF. Proving an implicational formula amounts to proving its target in the context consisting of all its arguments. In the case of $\mathbb{T}_{3^-}$ we are interested in contexts built from atoms in $\mathcal{X}_0$ and formulas in $\mathbb{T}_{2^-}$ (some of those can be atoms in $\mathcal{X}_1$). Such contexts are called NP-*contexts*. If Γ is an NP-context, and $\Gamma \vdash M : q$, with M an lnf, then M is either a variable or it has the form $M = XY_1Y_2 \dots Y_k(\lambda V_1 \dots V_m.\, N)Z_1 \dots Z_\ell$, where:

– the type of X is a $\mathbb{T}_{2-}$ formula of the form

$$p_1 \to p_2 \to \cdots \to p_k \to \alpha \to p'_1 \to \cdots \to p'_\ell \to q;$$

– $Y_1 : p_1, Y_2 : p_2, \dots, Y_k : p_k, Z_1 : p'_1, \dots, Z_\ell : p'_\ell$ are declared in Γ;
– the term $\lambda V_1 \dots V_m . N$ has a $\mathbb{T}_{1-}$ type $\alpha = s_1 \to \cdots \to s_m \to q'$.

We then have $\Gamma, V_1 : s_1, \dots, V_m : s_m \vdash N : q'$, with $s_1, \dots, s_m \in \mathcal{X}_0$ and $q' \in \mathcal{X}_1$, and the context $\Gamma, V_1 : s_1, \dots, V_m : s_m$ is an NP-context.

In terms of a monotonic automaton this proof construction step amounts to executing this instruction:

$$q : \texttt{check}\ p_1, \dots, p_k, p'_1, \dots, p'_\ell;\ \texttt{set}\ s_1, \dots, s_m;\ \texttt{jmp}\ q'.$$

No other actions need to be performed by the automaton except a final step, which takes up the form $q : \texttt{check}\ q;\ \texttt{set}\ \varnothing;\ \texttt{jmp}\ f$, where f is a final state (this corresponds to the case of $M = X$).

It follows that $\mathbb{T}_{3-}$ proof search can be handled by a nondeterministic automaton (with no universal branching). By Lemma 8 provability in $\mathbb{T}_{3-}$ belongs to NP. $\dashv$

Remark 22. To exclude universal branching, only one argument in a $\mathbb{T}_{2-}$ formula can be nonatomic. Note however that formulas used in the proof of Proposition 13 satisfy a similar restriction. Hence the separation between “data atoms” $\mathcal{X}_0$ and “control atoms” in $\mathcal{X}_1$ is essential too.

Similarly, sole separation between “data atoms” and “control atoms” does not reduce the complexity either, as it directly corresponds to separation between registers and states of automata.

Lemma 23. *Provability in $\mathbb{T}_{3-}$ is* NP-*hard.*

Proof. We reduce the 3-CNF-SAT problem to provability in $\mathbb{T}_{3-}$. For every 3-CNF formula

$$\Psi = (\mathtt{r}_{11} \vee \mathtt{r}_{12} \vee \mathtt{r}_{13}) \wedge \cdots \wedge (\mathtt{r}_{k1} \vee \mathtt{r}_{k2} \vee \mathtt{r}_{k3}), \qquad (*)$$

where $\mathtt{r}_{ij}$ are literals, we construct a $\mathbb{T}_{3-}$ formula ψ so that Ψ is classically satisfiable if and only if ψ has a proof. Assume that $\{p_1, \dots, p_n\}$ are all propositional variables occurring in Ψ, and that $p_1, \dots, p_n, p'_1, \dots, p'_n \in \mathcal{X}_0$. Other atoms of the formula ψ are $q_1, \dots, q_n, c_1, \dots, c_k \in \mathcal{X}_1$.

Define $\rho_{ij} = p_\ell$ when $\mathtt{r}_{ij} = p_\ell$, and $\rho_{ij} = p'_\ell$ when $\mathtt{r}_{ij} = \neg p_\ell$. The formula ψ has the form $\Gamma \to q_1$, where Γ consists of the following axioms:

1. $(p_i \to q_{i+1}) \to q_i$ and $(p'_i \to q_{i+1}) \to q_i$, for $i = 1, \dots, n-1$;
2. $(p_n \to c_1) \to q_n$ and $(p'_n \to c_1) \to q_n$;
3. $\rho_{i1} \to c_{i+1} \to c_i$, $\rho_{i2} \to c_{i+1} \to c_i$, and $\rho_{i3} \to c_{i+1} \to c_i$, for $i = 1, \dots, k-1$;
4. $\rho_{k1} \to c_k$, $\rho_{k2} \to c_k$, and $\rho_{k3} \to c_k$.

For a binary valuation v, let Δ_v be such that $p_i \in \Delta_v$ when $v(p_i) = 1$ and $p'_i \in \Delta_v$ otherwise. Suppose that Ψ is satisfied by some v. Then, for every i there is j with $\rho_{ij} \in \Delta_v$ and one can readily see that $\Gamma, \Delta_v \vdash c_1$ using axioms (4) and (3).

Let $\Delta^i_v = \Delta_v \cap (\{p_j \mid j < i\} \cup \{p'_j \mid j < i\})$. Since $\Gamma, \Delta_v \vdash c_1$ we obtain $\Gamma, \Delta^n_v \vdash q_n$ using (2), and then we use (1) to prove $\Gamma, \Delta^i_v \vdash q_i$ by induction, for $n-1 \geq i \geq 1$. Since $\Delta^1_v = \varnothing$, we eventually get $\Gamma \vdash q_1$.

For the converse, a proof of $\Gamma \vdash q_1$ in long normal form must begin with a head variable of type $(p_1 \to q_2) \to q_1$ or $(p_1' \to q_2) \to q_1$ applied to an abstraction $\lambda x.\, N$ with N of type q_2. The shape of N is also determined by axioms (1–2), and it must inevitably contain a proof of $\Gamma, \Delta_v \vdash c_1$ for some v. Such a proof is only possible if each of the k clauses is satisfied by v. The fun of checking the details is left to the reader. $\dashv$

We can put together Lemmas 21 and 23 to obtain the conclusion of this section: a very limited fragment of IIPC is of the same expressive power as SAT.

Theorem 24. *Proof search for $\mathbb{T}_{3-}$ formulas is* NP-*complete.*

5.3. Order two plus. We distinguish another natural class of formulas of low order for which the provability problem is *co*-NP-complete. We consider here implicational formulas built from literals, and we restrict attention to formulas of order two, where all literals are counted as of order zero. We call this fragment *order two plus*. Note that if we use the standard definition of order, these formulas are of order three.

It is convenient and illustrative to work with literals (using negation), but formulas of order two plus make in fact a fragment of IIPC. Indeed, $\neg p = p \to \bot$ by definition, and in all our proofs below the constant $\bot$ can be understood merely as a distinguished atom with no particular meaning. In other words, the *ex falso* rule, i.e., $\bot$-elimination is not involved.

Lemma 25. *Formulas of order two plus have the linear size model property: if $\nvdash \varphi$ then there is a Kripke model of depth at most 2 and of cardinality not exceeding the length of φ.*

Proof. Let $\varphi = \xi_1 \to \cdots \to \xi_n \to \mathtt{p}$, where $\xi_i = \mathtt{q}_i^1 \to \cdots \to \mathtt{q}_i^{n_i} \to \mathtt{r}_i$. Without loss of generality we may assume that literals $\mathtt{p}, \mathtt{r}_1, \dots, \mathtt{r}_n$ are all either propositional variables or $\bot$. Suppose that $\nvdash \varphi$. Then there exists a finite Kripke model C and a state c_0 of C such that $C, c_0 \nVdash \varphi$. That is, $C, c_0 \Vdash \xi_i$, for all $i = 1, \dots, n$, and $C, c_0 \nVdash \mathtt{p}$. For every $i = 1, \dots, n$ we now select a final state c_i of C as follows. Since $C, c_0 \Vdash \xi_i$, there are two possibilities: either $C, c_0 \Vdash \mathtt{r}_i$, or $C, c_0 \nVdash \mathtt{q}_i^j$, for some j. The important case is when $C, c_0 \nVdash \mathtt{q}^j$ and $\mathtt{q}^j = \neg s$, for some propositional variable s. Then there is a successor state c' of c_0 with $C, c' \Vdash s$, hence there also exists a final state forcing s. We define c_i as one of such final states. In other cases the choice of c_i is irrelevant and we can choose any final state.

Now define a new model C' with the set of states $\{c_0\} \cup \{c_1, \dots, c_n\}$ and the relation $\Vdash$ inherited from C, i.e., $C', c \Vdash s$ iff $C, c \Vdash s$, for any state c of C' and any propositional variable s. Note that so defined C' has depth at most 2.

We claim that $C', c_0 \nVdash \varphi$. Clearly $C', c_0 \nVdash \mathtt{p}$, so we should prove that all states in C' force all formulas ξ_i. Forcing in any state only depends on its successor states, hence if we had $C, c_i \Vdash \xi_i$ then we still have $C', c_i \Vdash \xi_i$, for all $i = 1, \dots, n$, because nothing has changed at the final states. But also nothing has changed at c_0 with respect to ξ_i. Indeed, if $C, c_0 \Vdash \mathtt{r}_i$ then $C', c_0 \Vdash \mathtt{r}_i$, and if $C, c_0 \nVdash \mathtt{q}_i^j$ for some j, where $\mathtt{q}_i^j$ is a propositional variable, then $C', c_0 \nVdash \mathtt{q}_i^j$ as well. Otherwise, for some s, j, we have $\neg s = \mathtt{q}_i^j$ and $C', c_i \Vdash s$, so $C', c_0 \nVdash \mathtt{q}_i^j$. $\dashv$

EXAMPLE 26. Lemma 25 cannot be improved to 2-state models: the formula

$$(\neg p \to q) \to (\neg r \to q) \to (p \to \neg r) \to q$$

requires a countermodel with at least 3 states.

THEOREM 27. *Order two plus fragment of IPC is co*-NP-*complete.*

PROOF. That the problem is in *co*-NP follows from Lemma 25: the small countermodel can be guessed and verified in polynomial time.

The *co*-NP-hardness of order two plus is shown by a reduction from non-3-CNF-SAT. Let us begin with a formula in 3-CNF:

$$\Psi = (\mathtt{r}_{11} \vee \mathtt{r}_{12} \vee \mathtt{r}_{13}) \wedge \cdots \wedge (\mathtt{r}_{k1} \vee \mathtt{r}_{k2} \vee \mathtt{r}_{k3}),$$

where $\mathtt{r}_{ij}$ are literals. Assume that $\{p_1, \dots, p_n\}$ are all propositional variables in Ψ. We define a set Γ_Ψ of formulas using propositional variables $p_1, \dots, p_n, p'_1 \dots, p'_n$. For any literal $\mathtt{r}_{jm}$ occurring in Ψ we write $\mathtt{r}'_{jm}$ to denote:

– the variable p'_i, when $\mathtt{r}_{jm} = p_i$;
– the variable p_i, when $\mathtt{r}_{jm} = \neg p_i$.

Members of Γ_Ψ are as follows (for all $i = 1, \dots, n$ and $j = 1, \dots, k$):

– $X_i : \neg p_i \to \neg p'_i \to \bot$;
– $Y_j : \mathtt{r}'_{j1} \to \mathtt{r}'_{j2} \to \mathtt{r}'_{j3} \to \bot$.

For example, if the first component of the formula Ψ was $(p \vee \neg q \vee \neg s)$ then we obtain $Y_1 : p' \to q \to s \to \bot$. We shall prove that:

Ψ is classically unsatisfiable if and only if $\Gamma_\Psi \vdash \bot$.

($\Rightarrow$) Let $m \leq n$ and let v be a Boolean valuation of variables $p_1, \dots, p_m$. Define an environment $\Gamma_v = \Gamma_\Psi \cup \{x_1 : p_1^v, \dots x_m : p_m^v\}$, where

$$p_i^v = \begin{cases} p_i, & \text{if } v(p_i) = 1; \\ p'_i, & \text{otherwise.} \end{cases}$$

By a reverse induction with respect to m we prove that $\Gamma_v \vdash \bot$, for every such v. We begin with $m = n$. Then v is defined on all variables in Ψ and does not satisfy Ψ. Therefore the value under v of at least one clause $\mathtt{r}_{j1} \vee \mathtt{r}_{j2} \vee \mathtt{r}_{j3}$ is zero, in which case we have $\mathtt{r}'_{j1}, \mathtt{r}'_{j2}, \mathtt{r}'_{j3} \in \Gamma_v$, hence $\bot$ is derivable using the assumption Y_j. (For example, if the unsatisfied component of Ψ were $(p \vee \neg q \vee \neg s)$ then we would have $p^v = p'$, $q^v = q$, $s^v = s$.)

For the induction step assume the claim holds for some $m \leq n$, and let v be a valuation of $p_1, \dots, p_{m-1}$. For $b = 0, 1$, define v_b as v extended by $v_b(p_m) = b$. By the induction hypothesis there are proofs $\Gamma_{v_0} \vdash M_0 : \bot$ and $\Gamma_{v_1} \vdash M_1 : \bot$. Then one proves $\bot$ from Γ_v using the assumption X_m; the lambda term in question has the form $X_m(\lambda x_m : p_m.\, M_1)(\lambda x_m : p'_m.\, M_0)$.

($\Leftarrow$) By contraposition suppose that v satisfies Ψ. We extend it to primed propositional variables by letting $v(p') = 1 - v(p)$. Since v satisfies all the clauses $\mathtt{r}_{11} \vee \mathtt{r}_{12} \vee \mathtt{r}_{13}$ of Ψ, it satisfies all the formulas in Γ_Ψ. Consequently, $\Gamma \nvdash \bot$ even in classical logic.

For any given Boolean valuation v of $p_1, \dots, p_n$, we prove that v does not satisfy Ψ. Let again $\Gamma_v = \Gamma_\Psi \cup \{x_1 : p_1^v, \dots, x_n : p_n^v\}$. Since $\Gamma_\Psi \vdash \bot$, also $\Gamma_v \vdash \bot$, so let M

be the shortest possible normal lambda-term such that $\Gamma_v \vdash M : \bot$. The proof must begin with either some X_i or some Y_j. In the first of the two cases it must be of the form $M = X_i(\lambda y_i : p_i . M_1)(\lambda y_i : p_i' . M_0)$, where $\Gamma_v, y_i :, p_i \vdash M_1 : \bot$ and $\Gamma_v, y_i : p_i' \vdash M_0 : \bot$. But in the context Γ_v we have either $x_i : p_i$ or $x_i : p_i'$. Thus either $M_1[y_i := x_i]$ or $M_0[y_i := x_i]$ makes a proof of $\bot$ shorter than M.

It follows that the shortest proof of $\bot$ is of the form $M = Y_j N_1 N_2 N_3$, where $\Gamma_v \vdash N_1 : \mathtt{r}'_{j1}$, $\Gamma_v \vdash N_2 : \mathtt{r}'_{j2}$, and $\Gamma_v \vdash N_3 : \mathtt{r}'_{j3}$. Then N_1, N_2, N_3 must be variables declared in Γ_v which is only possible when the literals $\mathtt{r}_{j1}, \mathtt{r}_{j2}, \mathtt{r}_{j3}$ are zero-valued under v. $\dashv$

§6. Conclusions and further research. We have demonstrated the strength of implicational intuitionistic propositional logic (IIPC) as a reasonable language to express problems solvable in PSPACE. Moreover, some natural subclasses of IIPC, called *order three minus* and *order two plus*, correspond, respectively, to complexity classes NP and *co*-NP (Section 5).

The situation in IIPC can be related to the one in modal logic S4 through the standard embedding [23] (see [4] for a modern account of the embedding). Each subsequent order corresponds through this embedding to one application of the modal operator. In particular, formulas of order three in IIPC translate to formulas of modal depth four. Interestingly enough, satisfiability for S4 formulas already of modal depth $k \geq 2$ is PSPACE-complete [13, Theorem 4.2].

Acknowledgements. We would like to thank anonymous referees who substantially helped in making the article better.

Funding. The work was partly supported by IDUB POB 3 programme at the University of Warsaw.

REFERENCES

[1] S. Alouneh, S. Abed, M. H. Al Shayeji, and R. Mesleh, *A comprehensive study and analysis on SAT-solvers: Advances, usages and achievements*. ***Artificial Intelligence Review***, vol. 52 (2019), no. 4, pp. 2575–2601.

[2] C.-B. Ben-Yelles, ***Type-assignment in the lambda-calculus; syntax and semantics***, Ph.D. thesis, Mathematics Department, University of Wales, 1979.

[3] M. N. Bezhanishvili, *Notes on Wajsberg's proof of the separation theorem*, ***Initiatives in Logic*** (J. Srzednicki, editor), Springer, Dordrecht, 1987, pp. 116–127.

[4] F. Bou, *Complexity of strict implication*, ***Advances in Modal Logic 5, Papers from the Fifth Conference on "Advances in Modal Logic," Held in Manchester, UK, 9-11 September 2004*** (R. A. Schmidt, I. Pratt-Hartmann, M. Reynolds, and H. Wansing, editors), King's College Publications, 2004, pp. 1–16.

[5] G. Brewka, T. Eiter, and M. Truszczyński, *Answer set programming at a glance*. ***Communications of the ACM***, vol. 54 (2011), no. 12, pp. 92–103.

[6] J. C. Culberson, *Sokoban is* PSPACE*-complete*, Department of Computing Science, The University of Alberta, Technical Report TR 97-02, 1997.

[7] H. B. Curry, *The combinatory foundations of mathematical logic*. ***The Journal of Symbolic Logic***, vol. 7 (1942), no. 2, pp. 49–64.

[8] D. van Dalen, *Intuitionistic logic*, ***Handbook of Philosophical Logic: Volume III: Alternatives in Classical Logic*** (D. Gabbay and F. Guenthner, editors), Springer, Dordrecht, 1986, pp. 225–339.

[9] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, *Complexity and expressive power of logic programming*. ***ACM Computing Surveys***, vol. 33 (2001), no. 3, pp. 374–425.

[10] P. de Groote, *On the strong normalisation of intuitionistic natural deduction with permutation-conversions*. ***Information and Computation***, vol. 178 (2002), no. 2, pp. 441–464.

[11] D. Dor and U. Zwick, *SOKOBAN and other motion planning problems*. ***Information and Computation***, vol. 13 (1999), no. 4, pp. 215–228.

[12] M. Fitting, ***Intuitionistic Logic, Model Theory and Forcing***, Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam-London, 1969.

[13] J. Y. Halpern, *The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic*. ***Artificial Intelligence***, vol. 75 (1995), no. 2, pp. 361–372.

[14] R. A. Hearn and E. D. Demaine, *Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation*. ***Theoretical Computer Science***, vol. 343 (2005), nos. 1–2, pp. 72–96.

[15] H. H. Hoos and T. Stützle, ***Stochastic Local Search***, **Foundations and Applications**, Elsevier, Amsterdam, 2005.

[16] J. E. Hopcroft and J. D. Ullman, ***Introduction to Automata Theory, Languages, and Computation***, Addison-Wesley Publishing Company, Reading, MA, 1979.

[17] W. A. Howard, *Assignment of ordinals to terms for primitive recursive functionals of finite type*, ***Intuitionism and Proof Theory: Proceedings of the Summer Conference at Buffalo N.Y. 1968*** (A. Kino, J. Myhill, and R. E. Vesley, editors), Studies in Logic and the Foundations of Mathematics, 60, Elsevier, 1970, pp. 443–458.

[18] I. Johansson, *Der Minimalkalkül, ein reduzierter intuitionistischer Formalismus*. ***Compositio Mathematica***, vol. 4 (1936), pp. 119–136.

[19] O. Kupferman and M. Y. Vardi, *Church's problem revisited*. ***The Bulletin of Symbolic Logic***, vol. 5 (1999), no. 2, pp. 245–263.

[20] C. Liang and D. Miller, *Focusing and polarization in linear, intuitionistic, and classical logics*. ***Theoretical Computer Science***, vol. 410 (2009), no. 46, pp. 4747–4768, Abstract Interpretation and Logic Programming: In honor of professor Giorgio Levi.

[21] P. Lorenzen, *Ein dialogisches Konstruktivitätskriterium*, ***Infinitistic Methods***, Pergamon Press, Oxford, 1961, pp. 9–16. Reprinted in Lorenzen and Lorenz [22, pp. 193–200].

[22] P. Lorenzen and K. Lorenz, ***Dialogische Logik***, Wissenschaftliche Buchgesellschaft, Darmstadt, 1978.

[23] J. C. C. McKinsey and A. Tarski, *Some theorems about the sentential calculi of Lewis and Heyting*. ***The Bulletin of Symbolic Logic***, vol. 13 (1948), no. 1, pp. 1–15.

[24] D. Miller, *A survey of the proof-theoretic foundations of logic programming*. ***Theory and Practice of Logic Programming***, vol. 22 (2022), no. 6, pp. 859–904.

[25] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov, *Uniform proofs as a foundation for logic programming*. ***Annals of Pure and Applied Logic***, vol. 51 (1991), no. 1, pp. 125–157.

[26] G. Mints, *Complexity of subclasses of the intuitionistic propositional calculus*. ***BIT Numerical Mathematics***, vol. 32 (1992), no. 1, pp. 64–69.

[27] C. H. Papadimitriou, ***Computational Complexity***, Addison-Wesley Publishing Company, Inc., Reading, MA, 1995.

[28] M. R. Prasad, A. Biere, and A. Gupta, *A survey of recent advances in SAT-based formal verification*. ***International Journal on Software Tools for Technology Transfer***, vol. 7 (2005), no. 2, pp. 156–173.

[29] L. Regnier and P. Urzyczyn, *Retractions of types with many atoms*. preprint, 2002. CoRR, vol. cs.LO/0212005, https://doi.org/10.48550/arXiv.cs/0212005.

[30] J. Rehof and M. Y. Vardi, *Design and synthesis from components (Dagstuhl seminar 14232)*. ***Dagstuhl Reports***, vol. 4 (2014), no. 6, pp. 29–47.

[31] I. A. Roig, ***Solving hard industrial combinatorial problems with SAT***, Ph.D. thesis, Universitat Politècnica de Catalunya, Software Department, 2013.

[32] A. Schubert, W. Dekkers, and H. P. Barendregt, *Automata theoretic account of proof search*, ***24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*** (S. Kreutzer, editor), Leibniz International Proceedings in Informatics, 41, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Saarbrücken/Wadern, 2015, pp. 128–143.

[33] A. Schubert, P. Urzyczyn, and D. Walukiewicz-Chrząszcz, *How hard is positive quantification?* ***ACM Transactions on Computational Logic***, vol. 17 (2016), no. 4, pp. 30:1–30:29.

[34] C. A. Smoryński, ***Investigation of intuitionistic formal systems by means of Kripke models***, Ph.D. thesis, University of Illinois, 1973.

[35] M. H. Sèrensen and P. Urzyczyn, ***Lectures on the Curry-Howard Isomorphism***, Studies in Logic and the Foundations of Mathematics, 149, Elsevier Science Inc., New York, 2006.

[36] R. Statman, *Intuitionistic propositional logic is polynomial-space complete*. ***Theoretical Computer Science***, vol. 9 (1979), no. 1, pp. 67–72.

[37] L. J. Stockmeyer and A. R. Meyer, *Word problems requiring exponential time (preliminary report)*. ***In Proceedings of the fifth annual ACM symposium on Theory of computing (STOC '73).*** Association for Computing Machinery, New York, NY, 1973, pp. 1–9.

[38] P. Tarau, *Abductive reasoning in intuitionistic propositional logic via theorem synthesis*, ***Theory and Practice of Logic Programming***, vol. 22 (2022), no. 5, pp. 693–707.

[39] Terese, ***Term Rewriting Systems***, Cambridge University Press, Cambridge, 2003.

[40] G. S. Tseitin, *On the complexity of derivation in propositional calculus*, ***Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*** (J. H. Siekmann and G. Wrightson, editors), Springer, Berlin, 1983, pp. 466–483.

[41] N. Tzevelekos, *Fresh-register automata*, ***Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*** (T. Ball and M. Sagiv, editors), ACM, New York, NY, 2011, pp. 295–306.

[42] P. Urzyczyn, *Intuitionistic games: Determinacy, completeness, and normalization*. ***Studia Logica***, vol. 104 (2016), no. 5, pp. 957–1001.

[43] ———, *Duality in intuitionistic propositional logic*, ***26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*** (U. de'Liguoro, S. Berardi, and T. Altenkirch, editors), Leibniz International Proceedings in Informatics, 188, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Saarbrücken/Wadern, 2020, pp. 11:1–11:10.

[44] M. Wajsberg, *Untersuchungen über den Aussagenkalkül von A. Heyting*. ***Wiadomości Matematyczne***, vol. 46 (1938), pp. 45–101. English translation: On A. Heyting's propositional calculus, in Mordchaj Wajsberg, Logical Works (S. J. Surma, editor), Ossolineum, Wrocław, 1977, pages 132–171.

FACULTY OF MATHEMATICS
INFORMATICS AND MECHANICS
UNIVERSITY OF WARSAW
UL. STEFANA BANACHA 2, 202-097 WARSAW
POLAND
E-mail: alx@mimuw.edu.pl
E-mail: urzy@mimuw.edu.pl

CARDINAL STEFAN WYSZYŃSKI
UNIVERSITY IN WARSAW
UL. DEWAJTIS 5, 501-815 WARSAW
POLAND
E-mail: k.zdanowski@uksw.edu.pl