Logical Foundations of Inductive Theorem Proving

Stefan Hetzl
stefan.hetzl@tuwien.ac.at

TU Wien

Summer Term 2025

Contents

Introduction

Induction is a reasoning principle that is pervasive in mathematics and computer science. Therefore there is a strong desire to automate the search of proofs by induction. Automated inductive theorem proving is an area of research that is concerned with the development of algorithms which find proofs by induction automatically.

In this lecture we study the logical foundations of this topic. We use proof-theoretic and model-theoretic methods to analyse common approaches and algorithms in terms of mathematical logic. The emphasis is on delineating the strength and capabilities of these methods. In particular, we analyse straightforward induction proofs, equational theory exploration, saturation theorem proving with explicit induction axioms, and clause set cycles. The employed mathematical methods concern mostly weak arithmetical theories such as open induction or existential induction.

The order of the chapters in this lecture notes (roughly) follows the strengths of the methods considered.

Chapter 0

Preliminaries

In this chapter we briefly review important notions, definitions, and results which will be relevant for this lecture. Recommendable introductions to first-order logic can be found, e.g., in [1] and [24].

0.1 First-order logic

Throughout this lecture we will work with first-order logic with equality. A *first-order language* is a set L consisting of constant symbols, function symbols, and predicate symbols. An L *term* is composed of variables and function symbols and constant symbols from L. The free variables of a term are defined as usual. We will often write $t, u, v, \ldots$ for terms. The notation $t(x_1, \ldots, x_n)$ indicates that the free variables of t are among $x_1, \ldots, x_n$. An L *atom* consists of a predicate symbol from L or the equality symbol $=$ together with a tuple of L terms of appropriate arity. An L *formula* is composed from atoms using the quantifiers $\forall$, $\exists$ and the propositional connectives $\wedge$, $\vee$, $\neg$, and $\rightarrow$. We will write $\mathcal{F}(L)$ for the set of L formulas. The free variables of a formula are defined as usual. As for terms, the notation $\varphi(x_1, \ldots, x_n)$ indicates that the free variables of the formula φ are among $x_1, \ldots, x_n$. An L *sentence* is a formula without free variables. Often the language L will be clear from the context or irrelevant. Then we simply speak about terms, atoms, and formulas without mentioning L explicitly. We will use (the meta-level symbol) $\equiv$ to denote syntactic equality between terms, formulas, etc. which is not to be confused with (the object-level symbol) $=$ representing equality. For a formula F, a term t, and similiar objects we will write $\mathcal{L}(F)$, $\mathcal{L}(t)$, $\ldots$ for the first-order language consisting of all constant, function, and predicate symbols that occur in F, in t, etc. A quantifier $\forall x$ in a formula φ is called *strong quantifier* if it occurs under an even number of negations in φ (where the left-hand side of an implication is counted as a negation) and *weak quantifier* if it occurs under an odd number of negations in φ. Dually, a quantifier $\exists x$ in a formula φ is called *strong quantifier* if it occurs under an odd number of negations in φ and *weak quantifier* if it occurs under an even number of negations.

An L *structure* $\mathcal{M}$ is given by 1. a non-empty set M, the *domain* of $\mathcal{M}$, 2. for every constant symbol $c \in L$ an element $c^{\mathcal{M}} \in M$, 3. for every function symbol $f \in L$ a function $f^{\mathcal{M}} : M^k \to M$ where k is the arity of f, and 4. for every predicate symbol $P \in L$ a predicate $P^{\mathcal{M}} \subseteq M^k$ where k is the arity of P. Often we will write $a \in \mathcal{M}$ instead of $a \in M$ to indicate that a is an element of the domain of $\mathcal{M}$. A *variable assignment for* $\mathcal{M}$ is a mapping v from a set of variables to M. For an L structure $\mathcal{M}$, an L formula φ, and a variable assignment v that maps all free variables of φ to elements of M, we define $\mathcal{M}, I \models \varphi$ in the usual way by induction of φ. Then $\mathcal{M}, I \models \varphi$

means that φ is true in $\mathcal{M}$ under the interpretation I of its free variables. For a sentence σ we simply write $\mathcal{M} \models \sigma$ since I is not needed. If we write $\mathcal{M} \models \varphi$ for a formula φ we mean $\mathcal{M} \models \forall^* \varphi$ where $\forall^* \varphi$ denotes the universal closure of the formula φ. If $\mathcal{M} \models \varphi$ for a formula φ we also say that $\mathcal{M}$ is a *model* of φ.

An L formula φ is called *satisfiable* if there is an L structure $\mathcal{M}$ with $\mathcal{M} \models \varphi$. An L formula φ is called *valid* if $\mathcal{M} \models \varphi$ for every L structure $\mathcal{M}$. We also write $\models \varphi$ to state that φ is a valid formula.

Let L and L' be first-order languages with $L' \subseteq L$. For an L structure $\mathcal{M}$, the L' *reduct of* $\mathcal{M}$, written as $\mathcal{M}{\restriction_{L'}}$, is an L' structure with the same domain as $\mathcal{M}$ whose interpretation has been restricted to the symbols of L'. If $\mathcal{N}$ is the L' reduct of the L structure $\mathcal{M}$, we also say that $\mathcal{M}$ is an *expansion of* $\mathcal{N}$ *to* L.

There are many different proof systems for first-order logic with equality. They are all equivalent w.r.t. provability. For a set of sentences Γ and a sentence σ we write $\Gamma \vdash \sigma$ to denote that there is a proof of σ from Γ. We write $\Gamma \models \sigma$ if $\mathcal{M} \models \Gamma$ implies $\mathcal{M} \models \sigma$ for all structures $\mathcal{M}$.

Theorem 0.1 (Soundness)**.** *Let Γ be a set of sentences and let σ be a sentence. If $\Gamma \vdash \sigma$, then $\Gamma \models \sigma$.*

Theorem 0.2 (Completeness)**.** *Let Γ be a set of sentences and let σ be a sentence. If $\Gamma \models \sigma$, then $\Gamma \vdash \sigma$.*

A *theory* is a deductively closed set of sentences T, i.e., $T \vdash \sigma$ implies $\sigma \in T$. An *axiomatisation* of a theory T is a set of sentences A s.t. $A \vdash \sigma$ iff $T \vdash \sigma$. For a structure $\mathcal{M}$, the *theory of* $\mathcal{M}$ is defined as $\mathrm{Th}(\mathcal{M}) = \{\sigma \text{ sentence} \mid \mathcal{M} \models \sigma\}$. A theory T is called *complete* if for every sentence σ: $T \vdash \sigma$ or $T \vdash \neg\sigma$. A theory T is called *consistent* if there is no sentence σ s.t. $T \vdash \sigma$ and $T \vdash \neg\sigma$. A theory T' is called *extension* of a theory T if $T' \supseteq T$.

One of the most central results about first-order logic is the compactness theorem.

Theorem 0.3 (Compactness Theorem)**.** *Let Γ be a set of sentences. If every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable, then Γ is satisfiable.*

0.2 Theories of arithmetic

The *language of arithmetic* is $L_{\mathrm{A}} = \{0, s, +, \cdot, \leq\}$. Robinson's theory of minimal arithmetic Q consists of the universal closures of the following formulas:

$$s(x) \neq 0 \tag{Q1}$$
$$s(x) = s(y) \to x = y \tag{Q2}$$
$$x \neq 0 \to \exists y\, x = s(y) \tag{Q3}$$
$$x + 0 = x \tag{Q4}$$
$$x + s(y) = s(x + y) \tag{Q5}$$
$$x \cdot 0 = 0 \tag{Q6}$$
$$x \cdot s(y) = (x \cdot y) + x \tag{Q7}$$
$$x \leq y \leftrightarrow \exists z\, z + x = y \tag{Q8}$$

Let $\varphi(x, z_1, \ldots, z_k)$ be a formula. Then the *induction axiom* with induction formula $\varphi(x, \overline{z})$ is:

$$I_x\varphi(x, \overline{z}) \;\equiv\; \varphi(0, \overline{z}) \wedge \forall x\, (\varphi(x, \overline{z}) \to \varphi(s(x), \overline{z}) \to \forall x\, \varphi(x, \overline{z}).$$

The variables $z_1, \dots, z_k$ are called *parameters* of this induction axiom. Let Φ be a set of formulas. Then we define Φ-IND $= \{\forall \overline{z}\, I_x \varphi(x, \overline{z}) \mid \varphi(x, \overline{z}) \in \Phi\}$. The theory of Peano arithmetic (PA) is axiomatised by $Q + \mathcal{F}$-IND where $\mathcal{F}$ is the set of all L_{A} formulas.

For $i \in \mathbb{N}$ and a term t we define the term $s^i(t)$ by $s^0(t) :\equiv t$ and $s^{i+1}(t) :\equiv s(s^i(t))$. For $n \in \mathbb{N}$ the *numeral* $\underline{n}$ is defined as the term $s^n(0)$. Let L be a language containing 0 and s, let $\mathcal{M}$ be an L structure, and let $a \in \mathcal{M}$. Then a is called *standard number* if there is an $n \in \mathbb{N}$ s.t. $a = \underline{n}^{\mathcal{M}}$ and *nonstandard number* otherwise. The set of natural numbers $\mathbb{N}$ with the natural interpretation of the symbols in L_{A} is an L_{A} structure: the *standard model* (of arithmetic). This structure, which is written as $\mathbb{N}$, does not contain a nonstandard number. However, one can use the compactness theorem of first-order logic to establish the existence of a structure with nonstandard numbers.

The following proof establishes the existence of a nonstandard model of $\mathrm{Th}(\mathbb{N})$. It is a typical example for a compactness argument: let $L = L_{\mathrm{A}} \cup \{c/0\}$ and let $\Gamma = \mathrm{Th}(\mathbb{N}) \cup \{c \neq 0, c \neq \underline{1}, c \neq \underline{2}, \dots\}$. Let Γ_0 be a finite subset of Γ. Then Γ_0 contains only a finite numbers of formulas of the form $c \neq \underline{i}$. We define the L structure $\mathcal{M}_0$ as the expansion of the standard model $\mathbb{N}$ by setting $c^{\mathcal{M}_0} = n$ for an $n \in \mathbb{N}$ s.t. $c \neq \underline{n} \notin \Gamma_0$. Then $\mathcal{M}_0 \models \Gamma_0$, so, by the compactness theorem, there is an L structure $\mathcal{M}$ s.t. $\mathcal{M} \models \Gamma$. We define $\mathcal{N}$ as the L_{A} reduct of $\mathcal{M}$. Then $\mathcal{N}$ is a model of $\mathrm{Th}(\mathbb{N})$ which contains the nonstandard element $c^{\mathcal{M}}$.

0.3 Inductive data types

We will often work with a slight extension of first-order logic with equality: many-sorted first-order logic with equality. In this setting there is a fixed finite set of sorts: $s_1, \dots, s_n$. A many-sorted (first-order) language L is a set of typed constant symbols, functions symbols, and predicate symbols where the type of a constant symbols is a sort s_i, the type of a k-ary function symbols is an expression of the form $s_{i_1} \times \cdots \times s_{i_k} \to s_{i_{k+1}}$, and the type of a predicate symbol is an expression of the form $s_{i_1} \times \cdots \times s_{i_k}$. The set of L terms is defined inductively, allowing composition of terms only when they are typed correctly. When forming an L atom it is now also necessary to respect types, also in the case of equality atoms. The set of L formulas is defined inductively, with every variable being annotated with one of the sorts $s_1, \dots, s_n$. Usually, we will not write the sort annotations of variables as the sort of a variable will be clear from the context. An L structure $\mathcal{M}$ has a domain for each of the sorts $s_1, \dots, s_n$, written as $s_1^{\mathcal{M}}, \dots, s_n^{\mathcal{M}}$.

Example 0.4. Given two sorts s_1 and s_2 let $L = \{f : s_1 \times s_2 \to s_1, c : s_1, d : s_2\}$. Then $f(c, d) = c$ is an L atom but $f(c, c) = c$ and $f(c, d) = d$ are not.

Inductive data types are special sorts. Let $s_1, \dots, s_n$ be sorts. A definition of an inductive datatype D on top of $s_1, \dots, s_n$ is given by a finite set of *constructors* $c_1, \dots, c_k$ where, for $i = 1, \dots, k$, c_i is a many-sorted function symbol of type $\tau_i^1 \times \dots \times \tau_i^{m_i} \to D$ where $\tau_i^1, \dots, \tau_i^{m_i} \in \{D, s_1, \dots, s_n\}$. In order for such a set of constructors to be a valid definition of an inductive data type, there has to be an $i \in \{1, \dots, k\}$ with $\tau_i^1, \dots, \tau_i^{m_i} \in \{s_1, \dots, s_n\}$. This i is a base case. A term that consists of construtors only will also be called *constructor term.* It is useful to think of constructor terms as denotations of objects. Later we will define additional functions that manipulate these objects.

Example 0.5. The inductive data type Nat of natural numbers is defined by the constructors $0 : \mathsf{Nat}$ and $s : \mathsf{Nat} \to \mathsf{Nat}$. The constructor terms of Nat are exactly the numerals.

Example 0.6. Given a sort s, the inductive data type $\mathsf{List}(s)$ of lists with elements from s is defined on top of s by the constructors $\mathsf{nil} : \mathsf{List}(s)$ and $\mathsf{cons} : s \times \mathsf{List}(s) \to \mathsf{List}(s)$. The term nil

represents the empty list. A term of the form $\mathsf{cons}(a, L)$ represents the list whose first element is a and whose rest is L. In particular, $\mathsf{List}(\mathsf{Nat})$ is the type of lists of natural numbers.

Let D be an inductive data type defined on top of $s_1, \ldots, s_n$. Let $M_1, \ldots, M_n$ be sets, intended as interpretations of $s_1, \ldots, s_n$. Then the set of constructor terms of D w.r.t. $M_1, \ldots, M_n$, written as $\mathbb{T}^D(M_1, \ldots, M_n)$ is the smallest set X that satisfies

$$\frac{a_j \in X \text{ if } \tau_i^j = D \text{ and } a_j \in M_l \text{ if } \tau_i^j = s_l \text{ for all } j \in \{1, \ldots, m_i\}}{c_i(a_1, \ldots, a_{m_i}) \in X}$$

for all $i \in \{1, \ldots, k\}$.

Let $s_1, \ldots, s_n$ be sorts s.t. $s_{l+1}, \ldots, s_n$ are inductive data types with s_i being defined on top of $s_1, \ldots, s_{i-1}$. Let $M_1, \ldots, M_l$ be sets. Then the *standard model* $\mathcal{S}$ of $s_1, \ldots, s_n$ w.r.t. $M_1, \ldots, M_l$ is defined by

$$s_i^{\mathcal{S}} = \begin{cases} M_i & \text{for } i = 1, \ldots, l \\ \mathbb{T}^{s_i}(s_1^{\mathcal{S}}, \ldots, s_{i-1}^{\mathcal{S}}) & \text{for } i = l+1, \ldots, n \end{cases}$$

and the constructors being defined as themselves in the semantics.

Example 0.7. $\mathbb{T}^{\mathsf{Nat}}$ is the smallest set X closed under

$$\frac{}{0 \in X} \qquad\qquad \frac{a_1 \in X}{s(a_1) \in X}$$

$\mathbb{T}^{\mathsf{List}(s)}(\mathbb{T}^{\mathsf{Nat}})$ is the smallest set X closed under

$$\frac{}{\mathsf{nil} \in X} \qquad\qquad \frac{a_1 \in \mathbb{T}^{\mathsf{Nat}} \quad L_2 \in \mathsf{List}(\mathsf{Nat})}{\mathsf{cons}(a_1, L_2) \in \mathsf{List}(\mathsf{Nat})}$$

So the standard model $\mathcal{S}$ of $\mathsf{List}(\mathsf{Nat})$ has the domains $\mathsf{Nat}^{\mathcal{S}} = \mathbb{T}^{\mathsf{Nat}}$ and $\mathsf{List}(\mathsf{Nat})^{\mathcal{S}} = \mathbb{T}^{\mathsf{List}(s)}(\mathbb{T}^{\mathsf{Nat}})$.

Just as numerals denote numbers in the context of arithmetic, construtor terms denote data in the context of inductive data types. The role of programs will be played by functions defined by primitive recursion. A *primitive recursive function definition* over an inductive datatype has the following form: Let $s_1, \ldots, s_n$ be types and let D be an inductive data type on top of $s_1, \ldots, s_n$ with constructors $c_1, \ldots, c_k$ where $c_i : \tau_i^1 \times \ldots \times \tau_i^{m_i} \to D$ with $\tau_i^1, \ldots, \tau_i^{m_i} \in \{D, s_1, \ldots, s_n\}$. Let $f : \sigma_1 \times \cdots \times \sigma_l \times D \to D$ be a new function symbol. Then a primitive recursive definition of f consists of the set $D_f = \{D_f^{c_i} \mid 1 \leq i \leq k\}$ of equations where $D_f^{c_i}$ is

$$f(\overline{x}, c_i(y_1, \ldots, y_{m_i})) = t_i(\overline{x}, y_1, \ldots, y_{m_i}, f(\overline{x}, y_{j_1}), \ldots, f(\overline{x}, y_{j_p})) \qquad (D_f^{c_i})$$

for a term t_i which does not contain f and $\{j_1, \ldots, j_p\} = \{j \in \{1, \ldots, m_i\} \mid \tau_i^j = D\}$. Here we have defined f w.l.o.g. by recursion on the last argument. In fact we allow definition by recursion on any argument.

Example 0.8. In the inductive data type Nat, the following equations are primitive recursive definitions of the precessor function and of addition.

$$p(0) = 0 \qquad (D_p^0)$$

$$p(s(x)) = x \qquad (D_p^s)$$

$$x + 0 = x \qquad (D_+^0)$$

$$x + s(y) = s(x + y) \qquad (D_+^s)$$

Based on these definitions we can compute with the defined functions as in the following calculation:

$$p(s(s(0))) + s(0) =^{D_p^s} s(0) + s(0) =^{D_+^s} s(s(0) + 0) =^{D_+^0} s(s(0))$$

Example 0.9. In the inductive data type $\mathsf{List}(s)$, the following equations are a primitive recursive definition of the concatenation of lists.

$$\mathsf{nil} \circ L = L \tag{$D_\circ^{\mathsf{nil}}$}$$
$$\mathsf{cons}(x, K) \circ L = \mathsf{cons}(x, K \circ L) \tag{$D_\circ^{\mathsf{cons}}$}$$

We can compute the concatenation of two lists as in

$$\begin{aligned}\mathsf{cons}(a, \mathsf{cons}(b, \mathsf{nil})) \circ \mathsf{cons}(c, \mathsf{nil}) &=^{D_\circ^{\mathsf{cons}}} \mathsf{cons}(a, \mathsf{cons}(b, \mathsf{nil}) \circ \mathsf{cons}(c, \mathsf{nil}))\\ &=^{D_\circ^{\mathsf{cons}}} \mathsf{cons}(a, \mathsf{cons}(b, \mathsf{nil} \circ \mathsf{cons}(c, \mathsf{nil})))\\ &=^{D_\circ^{\mathsf{nil}}} \mathsf{cons}(a, \mathsf{cons}(b, \mathsf{cons}(c, \mathsf{nil}))).\end{aligned}$$

The observations at the end of examples 0.8 and 0.9 can be generalised in the sense of the following theorem.

Theorem 0.10. *Let $s_1, \ldots, s_n$ be types, let D be an inductive data type defined on top of $s_1, \ldots, s_n$, let $\mathcal{S}$ be the standard model of $s_1, \ldots, s_n, D$ w.r.t. the sets $M_1, \ldots, M_n$, and let D_f be a definition of f by primitive recursion on D. Then there is a unique function F s.t. $\mathcal{S}, f \mapsto F \models D_f$.*

Each inductive data type induces an *induction axiom for structural induction*. Let $c_1, \ldots, c_k$ be the constructors of an inductive data type D. Let $\varphi(x, \overline{z})$ be a formula. For $i = 1, \ldots, k$ define

$$\alpha_{i,x}\varphi(x, \overline{z}) \;\equiv\; \forall x_1 \cdots \forall x_{m_i} \left(\bigwedge_{\substack{j \in \{1,\ldots,m_i\} \\ \tau_i^j = D}} \varphi(x_j, \overline{z}) \to \varphi(c_i(x_1, \ldots, x_{m_i}), \overline{z}) \right)$$

Then the structural induction axiom for D is

$$I_x^D \varphi(x, \overline{z}) \;\equiv\; \bigwedge_{i=1}^{k} \alpha_{i,x}\varphi(x, \overline{z}) \to \forall x\, \varphi(x, \overline{z}).$$

Example 0.11. For the inductive data type Nat we have

$$\begin{aligned}&\alpha_{1,x}\varphi(x, \overline{z}) \equiv \varphi(0, \overline{z}),\\ &\alpha_{2,x}\varphi(x, \overline{z}) \equiv \forall x_1\, (\varphi(x_1, \overline{z}) \to \varphi(s(x_1), \overline{z})), \text{ and}\\ &I_x^{\mathsf{Nat}}\varphi(x, \overline{z}) \equiv \alpha_{1,x}\varphi(x, \overline{z}) \wedge \alpha_{2,x}\varphi(x, \overline{z}) \to \forall x\, \varphi(x, \overline{z}).\end{aligned}$$

Example 0.12. For the inductive data type $\mathsf{List}(s)$ we have

$$\begin{aligned}&\alpha_{1,x}\varphi(x, \overline{z}) \equiv \varphi(\mathsf{nil}, \overline{z}),\\ &\alpha_{2,x}\varphi(x, \overline{z}) \equiv \forall x_1 \forall L_2\, (\varphi(L_2, \overline{z}) \to \varphi(\mathsf{cons}(x_1, L_2), \overline{z})), \text{ and}\\ &I_x^{\mathsf{List}}\varphi(x, \overline{z}) \equiv \alpha_{1,x}\varphi(x, \overline{z}) \wedge \alpha_{2,x}\varphi(x, \overline{z}) \to \forall L\, \varphi(L, \overline{z}).\end{aligned}$$

0.4 Saturation theorem proving

The set of formulas which are valid in first-order logic is undecidable but computationally enumerable. So there are algorithms which, when given a first-order formula φ, will terminate with the information "φ is valid" whenever φ is indeed valid. But, in general, for non-valid φ they will not terminate. Automated deduction is a research area that is concerned with developing practically efficient such algorithms. Resolution-based saturation theorem proving is the most successful technique for automated deduction in pure first-order logic. In this section we briefly introduce some basic notions of this area in so far they are relevant for the rest of this course.

Resolution-based saturation theorem proving works with clause sets. A *literal* is an atom or a negated atom. A *clause* is a finite set of literals, interpreted as the universal closure of the disjunction of these literals. A *clause set* is simply a set of clauses and is interpreted as the conjunction of its clauses. Most of the clause sets we will be dealing with will be finite sets.

The standard transformation of a formula φ, whose validity we would like to check, proceeds as follows:

1. Negate φ to obtain $\neg\varphi$ (which is unsatisfiable iff φ is valid)
2. Skolemize $\neg\varphi$, i.e., remove all weak quantifiers, to obtain $\mathrm{SK}^{\exists}(\neg\varphi)$
3. Compute the clause normal form $\mathcal{C} = \mathrm{CNF}(\mathrm{SK}^{\exists}(\neg\varphi))$ of the Skolemization of $\neg\varphi$.

Then $\mathcal{C}$ is unsatisfiable iff φ is valid. We will now look at steps 2. and 3. in detail.

Skolemisation.

Skolemisation is a transformation of formulas that removes one type of quantifiers, either the weak or the strong quantifiers.

Definition 0.13. We write $s(L)$ for a set of constant and function symbols which consists of distinct and new function symbols f/n for every L formula $Qx\,\varphi$ with n free variables and $Q \in \{\forall, \exists\}$. We define $\mathrm{sk}(L) = L \cup s(L)$. We write $\mathrm{sk}^i(L)$ for the i-fold iteration of sk and define $\mathrm{sk}^\omega(L) = \bigcup_{i\in\mathbb{N}} sk^i(L)$.

Strictly speaking, the above definition is not well-formed: $s(\cdot)$ is not a function because the function symbols in $s(L)$ are not specified. The essential point if that the concrete symbols are not relevant as long as they are new and distinct. *New* means that f does not occur in L and *distinct* means that two different formulas $Q_1x_1\,\varphi_1$ and $Q_2x_2\,\varphi_2$ yield two different function symbols. If $n = 0$ then f, being a 0-ary function symbol, is a constant symbol. In abuse of notation, we will often write $f = s(Qx\,\varphi)$ to denote that f is the Skolem symbol corrensponding to $Qx\,\varphi$.

For $Q \in \{\forall, \exists\}$ we define

$$\overline{Q} = \begin{cases} \forall & \text{if } Q = \exists \\ \exists & \text{if } Q = \forall \end{cases}$$

Definition 0.14. We define the functions $\mathrm{SK}^{\exists}, \mathrm{SK}^{\forall} : \mathcal{F}(\mathrm{sk}^\omega(L)) \to \mathcal{F}(\mathrm{sk}^\omega(L))$ as follows:

$$\begin{aligned}
\mathrm{SK}^Q(A) &= A \text{ for an atom } A \\
\mathrm{SK}^Q(A \circ B) &= \mathrm{SK}^Q(A) \circ \mathrm{SK}^Q(B) \text{ for } \circ \in \{\forall, \exists\} \\
\mathrm{SK}^Q(\neg A) &= \neg \mathrm{SK}^{\overline{Q}}(A) \\
\mathrm{SK}^Q(A \to B) &= \mathrm{SK}^{\overline{Q}}(A) \to \mathrm{SK}^Q(B) \\
\mathrm{SK}^Q(Qy\,A(\overline{x}, y)) &= \mathrm{SK}^Q(A(\overline{x}, f(\overline{x}))) \qquad (*) \\
\mathrm{SK}^Q(\overline{Q}x\,A) &= \overline{Q}x\,\mathrm{SK}^Q(A)
\end{aligned}$$

where, in $(*)$, $\overline{x}$ are exactly the free variables of $Qy\, A(\overline{x}, y)$ and $f = s(Qy\, A(\overline{x}, y))$.

Example 0.15. Let $P/2$ and $Q/3$ be predicate symbols, then

$$\begin{aligned}
\mathrm{SK}^{\exists}(\exists x\,(\exists y\, P(x,y) \to \forall u \exists v\, Q(x,u,v))) &\equiv \mathrm{SK}^{\exists}(\exists y\, P(c,y) \to \forall u \exists v\, Q(c,u,v)) \\
&\equiv \mathrm{SK}^{\forall}(\exists y\, P(c,y)) \to \mathrm{SK}^{\exists}(\forall u \exists v\, Q(c,u,v)) \\
&\equiv \exists y\, \mathrm{SK}^{\forall}(P(c,y)) \to \forall u\, \mathrm{SK}^{\exists}(\exists v\, Q(c,u,v)) \\
&\equiv \exists y\, P(c,y) \to \forall u\, \mathrm{SK}^{\exists}(Q(c,u,f(u))) \\
&\equiv \exists y\, P(c,y) \to \forall u\, Q(c,u,f(u)).
\end{aligned}$$

Note that $\mathrm{SK}^{\exists}(F)$ does not contain weak quantifiers and $\mathrm{SK}^{\forall}(F)$ does not contain strong quantifiers.

Definition 0.16. The *Skolem axioms* for a formula $\varphi(\overline{x}, y)$ are:

$$\exists y\, \varphi(\overline{x}, y) \to \varphi(\overline{x}, f(\overline{x})) \tag{$S^{\exists}_{y}\varphi$}$$

$$\varphi(\overline{x}, g(\overline{x})) \to \forall y\, \varphi(\overline{x}, y) \tag{$S^{\forall}_{y}\varphi$}$$

where $f = s(\exists y\, \varphi(\overline{x}, y)$ and $g = s(\forall y\, \varphi(\overline{x}, y)$. For a language L we define

$$L\text{-SA} = \{\forall \overline{x}\, S^{Q}_{y}\varphi(\overline{x}, y) \mid Q \in \{\forall, \exists\}, \varphi(\overline{x}, y) \text{ is an } L \text{ formula}\}.$$

The Skolem axioms provide a good intuition for the meaning of the Skolem functions. For the existential quantifier: if there is a y s.t. $\varphi(\overline{x}, y)$, then $f(\overline{x})$ is such a way. For the universal quantifier, one can think of $g(\overline{x})$ as an object y for which $\varphi(\overline{x}, y)$ is not true, if such an object exists. Then, if $\varphi(\overline{x}, \cdot)$ is even true of this object of which it is not true (if such an object exists), then it is true for all y.

Lemma 0.17. *Let L be a first-order language, let φ be an $\mathrm{sk}^{\omega}(L)$ formula, and let $Q \in \{\forall, \exists\}$. Then we have $\mathrm{sk}^{\omega}(L)$-SA $\vdash \varphi \leftrightarrow \mathrm{SK}^{Q}(\varphi)$.*

: For sets of sentence Γ_1 and Γ_2 and a set $\mathcal{F}$ of formulas we write $\Gamma_1 \simeq_{\mathcal{F}} \Gamma_2$ if for all $\varphi \in \mathcal{F}$: $\Gamma_1 \vdash \varphi$ iff $\Gamma_2 \vdash \varphi$.

Theorem 0.18. *Let L be a first-order language and let Γ be a set of L sentences. Then*

1. $\mathrm{sk}^{\omega}(L)\text{-SA} + \Gamma \simeq_{\mathcal{F}(L)} \Gamma$,
2. $\mathrm{SK}^{\exists}(\Gamma) \simeq_{\mathcal{F}(L)} \Gamma$, *and*
3. Γ *is satisfiable iff* $\mathrm{SK}^{\exists}(\Gamma)$ *is satisfiable.*

Clause form transformation.

In transforming a formula without weak quantifiers into a clause set, there is a number of issues, mostly concerning complexity, to consider which are imporant in practical applications. For the purposes of this course, we do not have to deal with these aspects. So we will simply fix a function CNF that maps a formula without weak quantifers to a clause set. We assume that 1. $\mathcal{L}(\mathrm{CNF}(\varphi)) \subseteq \mathcal{L}(\varphi)$ and 2. $\models \varphi \leftrightarrow \mathrm{CNF}(\varphi)$. A straightforward way to realise such a function is to move all quantifiers to a prenex position, push negations to the bottom, and then distribute disjunctions of conjunctions. Note that, in practical applications, often normal form transformations are used which do not satisfy 1. In order to simplify the results, for the time being, we work with these assumptions.

Saturation systems.

A saturation system is, essentially, a set of rules for modifying a clause. More specifically:

Definition 0.19. A *reduction rule* is a set of instances of the form

$$\begin{array}{c} C_1 \quad \cdots \quad C_n \\ \hline\hline D_1, \ldots, D_m \end{array}$$

where $C_1, \ldots, C_n$ and $D_1, \ldots, D_m$ are clauses.

The above reduction rule is interpreted as replacing $C_1, \ldots, C_n$ with $D_1, \ldots, D_m$ in the current clause set.

Definition 0.20. An *inference rule* is a reduction rule that reintroduces its premises, i.e., a reduction rule of the form

$$\begin{array}{c} C_1 \quad \cdots \quad C_n \\ \hline\hline C_1, \ldots, C_n, D_1, \ldots, D_m \end{array}$$

As a shorthand notation we will write

$$\frac{C_1 \quad \cdots \quad C_n}{D_1, \ldots, D_m}$$

for this inference rule.

Definition 0.21. A saturation system is a set of reduction rules. Let $\mathcal{S}$ be a saturation system. Then $\mathcal{S}$ is called *sound* if, whenever a clause D is derivable from a clause set $\mathcal{C}$ in $\mathcal{S}$, then $\mathcal{C} \models D$. Moreover, $\mathcal{S}$ is called refutationally complete if, whenever a clause set $\mathcal{C}$ is inconsistent, then $\mathcal{S}$ derives the empty clause $\emptyset$ from $\mathcal{C}$.

The central idea of saturation theorem proving is the following: in order to determine the unsatisfiability of the input clause set $\mathcal{C}$ we start with setting $\mathcal{C}_0 = \mathcal{C}$. Then we iteratively add new clauses to the current clause set according to the rules of $\mathcal{S}$, resulting in a sequence $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots$. If $\mathcal{S}$ is sound then all $\mathcal{C}_i$ follow logically from $\mathcal{C}$. If $\mathcal{S}$ is refutationally complete, then one of the $\mathcal{C}_i$ will contain $\emptyset$ if $\mathcal{C}$ is unsatisfiable (and the rules are applied in a fair way). I.e. we try to saturate the clause set $\mathcal{C}$ with all its logical consequences. Therefore one of the following three things will happen when a sound and refutationally complete saturation system is applied to a concrete input clause set $\mathcal{C}$:

1. One of the $\mathcal{C}_i$ contains $\emptyset$. In this case we can conclude that $\mathcal{C}$ is unsatisfiable.

2. After a finite number of steps, no more new clauses can be added, i.e., the current clause set is saturated. In this case we can conclude that $\mathcal{C}$ is satisfiable.

3. The saturation process does not terminate.

Unification.

In order to describe a concrete saturation system we need to introduce the concept of unification. A *substitution* is a mapping of finitely many variables to terms. The application of a substitution σ to a term t is written as $t\sigma$. It replaces all variables in the domain of σ by the associated terms. All other variables remain unchanged. The concatenation of two substitutions σ and τ is written as $\sigma \circ \tau$. The application of $\sigma \circ \tau$ to a term t is defined as $t(\sigma \circ \tau) = (t\sigma)\tau$, i.e., first apply σ, then τ. We say that σ is more general than θ, written as $\sigma \leq \theta$, if there is a τ s.t. $\sigma \circ \tau = \theta$. A *unifier* of two terms t_1 and t_2 is a substitution σ s.t. $t_1\sigma = t_2\sigma$. A substitution σ is called *most general unifier* of t_1 and t_2 if σ is a unifier of t_1 and t_2 and, for every unifier θ of t_1 and t_2: $\sigma \leq \theta$. The central result about unification is the following

Theorem 0.22. *If two terms have a unifier, then they have a most general unifier.*

Example 0.23. Let $t_1 \equiv f(x, g(x))$ and $t_2 \equiv f(g(y), z)$. Then t_1 and t_2 are unifiable and the most general unifier of t_1 and t_2 is $\sigma = [x\backslash g(y), z\backslash g(g(y))]$.

Resolution and Paramodulation.

For a literal L we define

$$\overline{L} = \begin{cases} A & \text{if } L \equiv \neg A \\ \neg A & \text{if } L \text{ is an atom } A \end{cases}$$

Example 0.24. The resolution rule (including renaming) is

$$\frac{C \vee L \quad L' \vee D}{(C \vee D)\sigma}\ \text{Res}$$

where $C \vee L$ and $L' \vee D$ are w.l.o.g. variable-disjoint and σ is the most general unifier of L and $\overline{L'}$.

Example 0.25. The factoring rule is

$$\frac{L_1 \vee L_2 \vee C}{(L_1 \vee C)\sigma}\ \text{Fact}$$

where σ is the most general unifier of L_1 and L_2.

Example 0.26. Let $\mathcal{C} = \{\{P(c)\}, \{\neg P(x), P(f(x))\}, \{\neg P(f(f(c)))\}\}$. Then $\mathcal{C}$ is unsatisfiable and has the following resolution refutation.

$$\dfrac{\dfrac{\dfrac{P(c) \quad \neg P(x), P(f(x))}{P(f(c))}\ \text{Res}_{[x\backslash c]} \quad \neg P(x), P(f(x))}{P(f(f(c)))}\ \text{Res}_{[x\backslash f(c)]} \quad \neg P(f(f(c)))}{\emptyset}\ \text{Res}_{\text{id}}$$

Theorem 0.27. Res + Fact *is a sound and refutationally complete saturation system for first-order logic without equality.*

In order to deal with equality, we introduce the paramodulation rules.

Example 0.28. Paramodulation is the inference rule

$$\frac{C \vee s = t \quad D}{(C \vee D[t]_p)\sigma}\ \text{Paramod}$$

where σ is the most general unifier of s and $D|_p$.

Example 0.29. Reflexivity is the inference rule

$$\frac{}{t = t}\ \text{Ref}$$

for any term t.

Example 0.30. Let $G = \{\{u \cdot (v \cdot w) = (u \cdot v) \cdot w\}, \{e \cdot x = x\}, \{x^{-1} \cdot x = e\}, \{x \cdot x^{-1} = e\}\}$. We want to show that $G \vdash \forall x\, x \cdot e = x$ by resolution and paramodulation. To that aim, we negate and Skolemise the goal to obtain the clause set $\mathcal{C} = G \cup \{\{c \cdot e \neq c\}\}$ which has the following refutation:

$$\dfrac{e \cdot x = x \qquad \dfrac{y \cdot y^{-1} = e \qquad \dfrac{u \cdot (v \cdot w) = (u \cdot v) \cdot w \qquad \dfrac{x^{-1} \cdot x = e \quad c \cdot e \neq c}{c \cdot (x^{-1} \cdot x) \neq c}\,\text{Paramod}_{\text{id}}}{(c \cdot x^{-1}) \cdot x \neq c}\,\text{Paramod}_{[u\backslash c, v\backslash x^{-1}, w\backslash x]}}{e \cdot c \neq c}\,\text{Paramod}_{[y\backslash c, x\backslash c]}}{\emptyset}\,\text{Res}_{[x\backslash c]}$$

Throughout this course we will occasionally want to work with a particular saturation system that is sound and refutationally complete for first-order logic with equality. We define the saturation system $\mathcal{R} = \text{Res} + \text{Fact} + \text{Paramod} + \text{Ref}$.

Theorem 0.31. *$\mathcal{R}$ is a sound and refutationally complete saturation system for first-order logic with equality.*

Chapter 1

Straightforward induction proofs

1.1 The use of new formulas

We begin with a basic example, probably the most well-known[1] induction proof:

Theorem 1.1. *For all $n \geq 1$:* $\displaystyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

Proof. We proceed by induction on n. In the base case $n = 1$ we simply observe that $1 = \frac{1\cdot 2}{2}$. For the step case we have:

$$\begin{aligned} \text{Induction hypothesis:}\quad & \textstyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \\ \text{Claim:}\quad & \textstyle\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2} \end{aligned}$$

and we show that

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) =^{\text{IH}} \frac{n(n+1)}{2} + \frac{2(n+1)}{2} = \frac{(n+2)(n+1)}{2},$$

which finishes the induction step and hence completes the proof. $\square$

Let us look at a another example now. By summing up odd numbers as in the following table

$$\begin{aligned} 1+3&=4\\ 1+3+5&=9\\ 1+3+5+7&=16\\ 1+3+5+7+9&=25\\ &\vdots \end{aligned}$$

we notice that the first n consecutive odd numbers seem to always sum up to a square. We try to prove this result now.

Theorem 1.2. *The sum of the first n odd numbers is a square, i.e., for all $n \geq 1$ there is a $k \in \mathbb{N}$ s.t.* $\displaystyle\sum_{i=1}^{n}(2i-1) = k^2$.

[1] J. C. F. Gauss was a nine year old schoolboy when, given the task to sum up the numbers from 1 to 100, he amazed his teacher by giving the correct answer without any calculation. He did so, essentially, by reducing the problem to a single multipliation along the line of the above formula for $n = 100$.

We first try to prove this result as we did Theorem 1.1 above.

First proof attempt. For the base case $n = 1$ we have $1 = 1^2$. For the step case we have

Induction hypothesis: $\exists k_0 \sum_{i=1}^{n}(2i-1) = k_0^2$

Claim: $\exists k_1 \sum_{i=1}^{n+1}(2i-1) = k_1^2$

We try to prove the claim by the following calculuation:

$$\sum_{i=1}^{n+1}(2i-1) = \sum_{i=1}^{n}(2i-1) + (2n+1) =^{\mathrm{IH}} k_0^2 + 2n + 1 = \cdot \overset{?}{\cdot} \cdot = k_1^2.$$

However, we get stuck trying to define k_1 s.t. $k_0^2 + 2n + 1 = k_1^2$. ⊠

What is the solution? Let us examine the examples above one more time. They do show that the sum of the first n odd numbers is a square. But, beyond that, they also show the square of which number it is. In fact, a short look suffices to come up with the conjecture that the sum of the first n odd numbers is n^2. This conjecture can now be proved in a straightforward way by induction.

Proof of Theorem 1.2. For proving Theorem 1.2 it suffices to show that $\sum_{i=1}^{n}(2i-1) = n^2$ for all $n \geq 1$. We will prove this by induction on n. For the base case $n = 1$ we have $1 = 1^2$. For the step case we have

Induction hypothesis: $\sum_{i=1}^{n}(2i-1) = n^2$

Claim: $\sum_{i=1}^{n+1}(2i-1) = (n+1)^2$

and we prove the claim by the following calculation:

$$\sum_{i=1}^{n+1}(2i-1) = \sum_{i=1}^{n}(2i-1) + 2n + 1 =^{\mathrm{IH}} n^2 + 2n + 1 = (n+1)^2.$$

□

The situation we are in now is this: the first proof attempt got stuck. The second attempt has worked. But we do not know whether the first attempt is necessarily doomed to fail or we just did not find the right way to carry it out. In the next section we will show that, indeed, it is not possible to complete the first proof attempt. However, showing such an impossibility result requires a thorough logical analysis.

This example, although it is quite simple, already illustrates the fundamental problem when dealing with induction. Often, in order to prove some statement A, we have to carry out induction on another statement B (or several other statements). Finding suitable B's algorithmically is difficult. Put in simple terms, the relationship between such A's and B's is the central topic of this lecture.

1.2 The need for new formulas

In this section we carry out a thorough logical analysis of the failed proof attempt for Theorem 1.2. Our main result will be that, in a very strong sense, it is impossible to complete this partial proof. To that aim we introduce some logical notions and we will rely on some logical results and techniques covered in Chapter 0.

The most fundamental difference between the proof of Theorem 1.1 and the proof of Theorem 1.2 is that the first was by induction on the very formula of the theorem, while the latter was not. We make this distinction precise with the notion of a straightforward induction proof. For the sake of flexibility, we already treat the case of several universal quantifiers here.

Definition 1.3. Let T be a theory and let $\forall x_1 \cdots \forall x_n\, \varphi(x_1, \dots, x_n)$ be a sentence s.t. $\varphi(\overline{x})$ does not start with $\forall$. Then $\forall \overline{x}\, \varphi(\overline{x})$ has a *straightforward induction proof in T* if there is an $i \in \{1, \dots, n\}$ s.t. $T, I_{x_i} \forall \overline{x}_{<i} \forall \overline{x}_{>i} \varphi(\overline{x}) \vdash \forall \overline{x}\, \varphi(x)$.

For $n = 1$ this definition simplifies to: $\forall x\, \varphi(x)$ has a straightforward induction proof if $T, I_x \varphi(x) \vdash \forall x\, \varphi(x)$.

Example 1.4. We show that Theorem 1.1 has a straightforward induction proof. To that aim, we first have to give a logical formalisation of the theorem. We will take the language of arithmetic $L_{\mathrm{A}} = \{0, s, +, \cdot, \leq\}$ as basis. For representing the finite sum we extend L_{A} by a unary function symbol to obtain $L_g = L_{\mathrm{A}} \cup \{g/1\}$. Our intention is to axiomatise g in such a way that it represents the function $n \mapsto \sum_{i=1}^{n} i$ via the direct computation of the sum. Consequently, we define the following two axioms:

$$g(0) = 0 \tag{D_g^0}$$

$$\forall x\, g(s(x)) = g(x) + s(x) \tag{D_g^s}$$

As background theory we use $T_g = \mathrm{Th}(\mathbb{N}) \cup \{D_g^0, D_g^s\}$. Then $T_g \vdash g(\underline{n}) = \underline{\sum_{i=1}^{n} i}$ for all $n \in \mathbb{N}$. Note that the set $\mathrm{Th}(\mathbb{N})$ of all true arithmetical sentences is a very powerful set of axioms. For this example, a much weaker set would suffice. The reason for choosing $\mathrm{Th}(\mathbb{N})$ at this point is to facilitate the comparison with Theorem 1.11, which will establish that Theorem 1.2 does not have a straightforward induction proof.

The theorem we want to show is $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Since fractions are not in our language we simply multiply this equation with 2 so that our goal becomes $2 \cdot \sum_{i=1}^{n} i = n(n+1)$, or, formally: $\forall x\, \varphi(x)$ where $\varphi(x) \equiv \underline{2} \cdot g(x) = x \cdot s(x)$. We claim that $\forall x\, \varphi(x)$ has a straightforward induction proof in T_g. In order to show that, it suffices to show that $T_g \vdash \varphi(0)$ and $T_g \vdash \forall x\, (\varphi(x) \to \varphi(s(x)))$.

For the base case work in T_g: We have

$$\underline{2} \cdot g(0) =^{D_g^0} \underline{2} \cdot 0 = 0 = 0 \cdot \underline{1}.$$

For the step case work in T_g: Assuming $\underline{2} \cdot g(x) = x \cdot s(x)$ we have

$$\begin{aligned} \underline{2} \cdot g(s(x)) &=^{D_g^s} \underline{2} \cdot (g(x) + s(x)) = \underline{2} \cdot g(x) + \underline{2} \cdot s(x) \\ &=^{\mathrm{IH}} x \cdot s(x) + \underline{2} \cdot s(x) = (x + \underline{2}) \cdot (x + \underline{1}) = s(x) \cdot s(s(x)). \end{aligned}$$

Consequently $T_g, I_x \varphi(x) \vdash \forall x\, \varphi(x)$, i.e., $\forall x\, \varphi(x)$ has a straightforward induction proof in T_g.

We now carry out an analogous formalisation of Theorem 1.2 and set out to prove that, in contrast to Theorem 1.1, it does not have a straightforward induction proof. As in Example 1.4 we have to formalise a particular sum and we do so by introducing a new unary function symbol that will be used on top of L_{A}. We phrase the formalisation of Theorem 1.2 as a *challenge problem*, a format that we will use frequently in this course. A challenge problem typically consists of: 1. a first-order language L, 2. an L theory T, and 3. an L sentence σ. The challenge consists of proving σ from T by a suitable form of induction. What makes it a challenge is the fact that, in one way or another, it is "difficult" to find a suitable induction. A challenge problem is not a fully specified problem, since we do not want to make it precise what "suitable induction" means. The intention is that a challenge problem is useful as an input to an automated inductive theorem prover.

Challenge Problem 1.1 (Sum of odd numbers is a square)**.**

- Language: $L_f = L_A \cup \{f/1\}$
- Axioms: $T_f = \mathrm{Th}(\mathbb{N}) \cup \{D_f^0, D_f^s\}$ where

$$f(0) = 0 \tag{D_f^0}$$

$$\forall x\, f(s(x)) = f(x) + s(\underline{2} \cdot x) \tag{D_f^s}$$

- Goal: $\forall x\, \psi(x)$ where $\psi(x) \equiv \exists y\, f(x) = y \cdot y$

Note that $T_f \vdash f(\underline{n}) = \underline{\sum_{i=1}^{n}(2i-1)}$ for all $n \in \mathbb{N}$. In order to show that Challenge Problem 1.1 does not have a straightforward induction proof, it is helpful to first work out an alternative characterisation of the notion of straightforward induction proof in terms of inductive formulas.

Definition 1.5. Let L be a language containing 0 and s, let T be an L theory, and let $\varphi(x)$ be an L formula. Then we say that $\varphi(x)$ is *T-inductive* if $T \vdash \varphi(0)$ and $T \vdash \forall x\, (\varphi(x) \to \varphi(s(x)))$.

Note that the notion of T-inductive applies only to formulas with exactly one free variable. We start by observing some simple properties of T-inductive formulas.

Lemma 1.6. *Let $L \supseteq \{0, s\}$ and let T be an L theory.*

1. *If $\varphi(x)$ is T-inductive and $T \subseteq T'$, then $\varphi(x)$ is T'-inductive.*
2. *If $\varphi_1(x)$ and $\varphi_2(x)$ are T-inductive, then $\varphi_1(x) \wedge \varphi_2(x)$ and $\varphi_1(x) \vee \varphi_2(x)$ are T-inductive.*
3. *$\varphi(x)$ is T-inductive iff $T, I_x\varphi(x) \vdash \forall x\, \varphi(x)$.*
4. *$\varphi(x)$ is T-inductive iff $T \vdash I_x\varphi(x) \leftrightarrow \forall x\, \varphi(x)$.*

Proof. 1 follows directly from the definition. For 2 assume that $\varphi_1(x)$ and $\varphi_2(x)$ are T-inductive. Then

$$T \vdash \varphi_1(0) \tag{1.1}$$

$$T \vdash \forall x\, (\varphi_1(x) \to \varphi_1(s(x))) \tag{1.2}$$

$$T \vdash \varphi_2(0) \tag{1.3}$$

$$T \vdash \forall x\, (\varphi_2(x) \to \varphi_2(s(x))) \tag{1.4}$$

So, by (1.1) and (1.3), we have $T \vdash \varphi_1(0) \wedge \varphi_2(0)$ and $T \vdash \varphi_1(0) \vee \varphi_2(0)$. Moreover, by (1.2) and (1.4), we have $T \vdash \forall x\, (\varphi_1(x) \wedge \varphi_2(x) \to \varphi_1(s(x)) \wedge \varphi_2(s(x)))$ and $T \vdash \forall x\, (\varphi_1(x) \vee \varphi_2(x) \to \varphi_1(s(x)) \vee \varphi_2(s(x)))$ by a case distinction in T.

For the left-to-right direction of 3, assume that $\varphi(x)$ is T-inductive. Then $T \vdash \varphi(0)$ and $T \vdash \forall x\, (\varphi(x) \to \varphi(s(x)))$, so $T, I_x\varphi(x) \vdash \forall x\, \varphi(x)$. For the right-to-left direction of 3, we proceed by showing the contraposition. To that aim, assume that $\varphi(x)$ is not T-inductive. Then i) $T \nvdash \varphi(0)$ or ii) $T \nvdash \forall x\, (\varphi(x) \to \varphi(s(x)))$. In case i) there is an $\mathcal{M}_0 \models T$ with $\mathcal{M}_0 \not\models \varphi(0)$. So $\mathcal{M}_0 \not\models \forall x\, \varphi(x)$ but $\mathcal{M}_0 \models I_x\varphi(x)$ because $I_x\varphi(x)$ is equivalent to a formula of the form $\bot \wedge A \to B$ in $\mathcal{M}_0$. Hence $T, I_x\varphi(x) \nvdash \forall x\, \varphi(x)$. In case ii) there is an $\mathcal{M}_s \models T$ with $\mathcal{M}_s \not\models \forall x\,(\varphi(x) \to \varphi(s(x)))$, so there is an $a \in \mathcal{M}_s$ s.t. $\mathcal{M}_s \models \varphi(a)$ and $\mathcal{M}_s \not\models \varphi(s(a))$. So $\mathcal{M}_s \not\models \forall x\, \varphi(x)$. On the other hand, $\mathcal{M}_s \models I_x\varphi(x)$ because $I_x\varphi(x)$ is equivalent to a formula of the form $A \wedge \bot \to B$ in $\mathcal{M}_s$. Hence $T, I_x\varphi(x) \nvdash \forall x\, \varphi(x)$.

4 follows immediately from 3 with the missing implication being obtained from the observation that $\forall x\, \varphi(x) \to (A \to \forall x\, \varphi(x))$ is a tautology. □

The above properties 3 and 4 are useful characterisations of the T-inductive formulas. On the one hand, a formula $\varphi(x)$ is T-inductive iff $\forall x\, \varphi(x)$ can be proved from $I_x\varphi(x)$ in T (as in 3). Note that this is just the definition of having a straightforward induction proof for a formula with one free variable. On the other hand, a formula is T-inductive iff its universal closure is equivalent to its induction axiom in T (as in 4). We will sometime use one, sometimes the other characterisation. We now proceed to link straightforward induction proofs with inductive formulas by carrying these observations over to the case of several variables.

Lemma 1.7. *Let T be a theory and let $\forall x_1 \cdots \forall x_n\, \varphi(x_1, \ldots, x_n)$ be a sentence s.t. $\varphi(\overline{x})$ does not start with $\forall$. Then $\forall \overline{x}\, \varphi(\overline{x})$ has a straightforward induction proof in T iff there is an $i \in \{1, \ldots, n\}$ s.t. $\forall \overline{x}_{<i} \forall \overline{x}_{>i} \varphi(\overline{x})$ is T-inductive.*

Proof. For $i = 1, \ldots, n$ let $\psi_i(x_i) = \forall \overline{x}_{<i} \forall \overline{x}_{>i}\, \varphi(\overline{x})$. Then

$$T, I_{x_i}\psi_i(x_i) \vdash \forall \overline{x}\, \varphi(x) \quad \text{iff} \quad T, I_{x_i}\psi_i(x_i) \vdash \forall x_i\, \psi_i(x_i) \quad \text{iff}^{\text{Lem. 1.6/3}} \quad \psi_i(x_i) \text{ is } T\text{-inductive.} \quad \square$$

Now we come back to showing that Challenge Problem 1.1 does not have a straightforward induction proof. This will be essentially a compactness argument.

Definition 1.8. We define the language $L_{c,f} = L_f \cup \{c/0\}$ and the following sets of $L_{c,f}$ sentences:

$$\begin{aligned} \Gamma_c^{0,s} &= \mathrm{Th}(\mathbb{N}) \cup \{D_f^0, D_f^s, \psi(c), \neg\psi(s(c))\} \\ \Gamma_c^s &= \mathrm{Th}(\mathbb{N}) \cup \{D_f^s, \psi(c), \neg\psi(s(c)), c \neq 0, c \neq \underline{1}, c \neq \underline{2}, \ldots\} \end{aligned}$$

Lemma 1.9. *If Γ_c^s is satisfiable, then $\Gamma_c^{0,s}$ is satisfiable.*

Proof. Let $\mathcal{M} \models \Gamma_c^s$, then $c^{\mathcal{M}}$ is nonstandard. Define $\mathcal{N}$ from $\mathcal{M}$ by $\mathcal{N}\restriction_{L_A \cup \{c\}} = \mathcal{M}\restriction_{L_A \cup \{c\}}$ and

$$f^{\mathcal{N}}(x) = \begin{cases} x^2 & \text{if } x \text{ is standard} \\ f^{\mathcal{M}}(x) & \text{otherwise} \end{cases}$$

Then $\mathcal{N} \models \mathrm{Th}(\mathbb{N})$. $\mathcal{N} \models D_f^0$ because $f^{\mathbb{N}}(0) = 0$. $\mathcal{N} \models D_f^s$ because $n \mapsto n^2$ satisfies D_f^s on the standard numbers of $\mathcal{N}$ and $f^{\mathcal{M}}$ satisfies D_f^s on the nonstandard numbers of $\mathcal{N}$. Since $c^{\mathcal{M}}$ is nonstandard, $f(c)^{\mathcal{N}} = f(c)^{\mathcal{M}}$ so $\mathcal{N} \models \psi(c)$ and $f(s(c))^{\mathcal{N}} = f(s(c))^{\mathcal{M}}$ so $\mathcal{N} \models \psi(s(c))$. □

We can understand the above proof as, essentially, taking a model $\mathcal{M}$ of Γ_c^s and overwriting the interpretation of f on the standard numbers in order to obtain a model $\mathcal{N}$ of $\Gamma_c^{0,s}$. We need to overwrite the interpretation of $f(0)$ to satisfy D_f^0. But since we also want to keep D_f^s satisfied, this entails overwriting the interpretation of f on all standard numbers. Working with Γ_c^s instead of $\Gamma_c^{0,s}$ allows us, in a next step, a higher degree of flexibility in defining the interpretation of f on the standard numbers. This will be useful for obtaining a suitable counterexample by a compactness argument.

Lemma 1.10. *Γ_c^s is satisfiable.*

Proof. Let $\Gamma_0 \subseteq \Gamma_c^s$ be finite. Then there is an $m \in \mathbb{N}$ s.t. $c \neq \underline{m} \notin \Gamma_0$. Define the $L_{c,f}$ structure $\mathcal{M}_0$ by: $\mathcal{M}_0 \restriction_{L_A} = \mathbb{N}$, $c^{\mathcal{M}_0} = m$, $f^{\mathcal{M}_0} = \beta_m$ where $\beta_m : \mathbb{N} \to \mathbb{N}, n \mapsto n^2 + 2m + 1$. Then $\mathcal{M}_0 \models \mathrm{Th}(\mathbb{N})$. Moreover, $\beta_m(n+1) = (n+1)^2 + 2m + 1 = \beta_m(n) + 2n + 1$, so $\mathcal{M}_0 \models D_f^s$. Since $\beta_m(m) = (m+1)^2$, we have $\mathcal{M}_0 \models \psi(c)$. On the other hand $\beta_m(m+1)$ is not a square because

$$(m+1)^2 = m^2 + 2m + 1 < \beta_m(m+1) = m^2 + 4m + 2 < m^2 + 4m + 4 = (m+2)^2.$$

So $\mathcal{M}_0 \not\models \psi(s(c))$. The choice of m guarantees that $\mathcal{M}_0 \models c \neq \underline{i}$ for all $c \neq \underline{i} \in \Gamma_0$. Therefore Γ_0 is satisfiable. So, by the compactness theorem, Γ_c^s is satisfiable. □

Theorem 1.11. *Challenge Problem 1.1 does not have a straightforward induction proof.*

Proof. From Lemmas 1.10 and 1.9 we obtain an $L_{c,f}$ structure $\mathcal{M}$ with $\mathcal{M} \models \Gamma_c^{0,s}$. Let $\mathcal{N} = \mathcal{M}\restriction_{L_f}$. Then $\mathcal{N} \models T_f$ but $\mathcal{N} \not\models \forall x\,(\psi(x) \to \psi(s(x)))$ with counterexample $c^{\mathcal{M}}$. Therefore $\psi(x)$ is not T_f-inductive and thus, by Lemma 1.7, $\forall x\,\psi(x)$ does not have a straightforward induction proof in T_f. □

Using the very strong theory $\mathrm{Th}(\mathbb{N})$ as a basis here is quite unrealistic for applications in automated deduction. However, it gives the strongest result. In particular, we obtain the following

Corollary 1.12. *Let $T \subseteq \mathrm{Th}(\mathbb{N}) \cup \{D_f^0, D_f^s\}$. Then $\forall x\,\psi(x)$ does not have a straightforward induction proof in T.*

1.3 Weak base theories

In this section we study straightforward induction proofs in the very weak setting of linear arithmetic axiomatised with only a few simple axioms. Also in such a setting, the phenomenon of the existence and non-existence of straightforward induction proofs can be observed and plays an important role for understanding the difficulty of proving statements by induction. However, in contrast to the setting studied above we can show non-existence of a straightforward induction proof by handcrafting a suitable model without using the compactness theorem.

Definition 1.13. The language of linear arithmetic with predecessor is $L_{\mathrm{LA}} = \{0, s, p, +\}$. We define the basic L_{LA}-theory B by the following axioms:

$$s(x) \neq 0 \tag{A1}$$
$$p(0) = 0 \tag{A2}$$
$$p(s(x)) = x \tag{A3}$$
$$x + 0 = x \tag{A4}$$
$$x + s(y) = s(x+y) \tag{A5}$$

We start out by showing that associativity of addition has a straightforward induction proof. In order to do this in a comfortable way, it is useful to first introduce the notion of uniform straightforward induction proof.

Definition 1.14. Let L be a language containing 0 and s, let T be an L theory, and let $\forall x_1 \cdots \forall x_n\, \varphi(x_1, \ldots, x_n)$ be an L sentence s.t. $\varphi(\overline{x})$ does not start with $\forall$. Then $\forall \overline{x}\, \varphi(\overline{x})$ has a *uniform straightforward induction proof in T* if there is an $i \in \{1, \ldots, n\}$ s.t. the $L \cup \{c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n\}$ formula $\forall x_i\, \varphi(c_1, \ldots, c_{i-1}, x_i, c_{i+1}, \ldots, c_n)$ has a straightforward induction proof in T.

For justifying this terminology, we quickly make the following observation.

Lemma 1.15. *For L, T, and $\forall \overline{x}\, \varphi(\overline{x})$ as in Definition 1.14: If $\forall \overline{x}\, \varphi(\overline{x})$ has a uniform straightforward induction proof, then $\forall \overline{x}\, \varphi(\overline{x})$ has a straightforward induction proof.*

Proof. If $\forall \overline{x}\, \varphi(\overline{x})$ has a uniform straightforward induction proof in T, then there is an $i \in \{1, \ldots, n\}$ s.t. the $L \cup \{c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n\}$ formula $\forall x_i\, \varphi(c_1, \ldots, c_{i-1}, x_i, c_{i+1}, \ldots, c_n)$ has a straightforward induction proof in T. W.l.o.g. let $i = 1$. Then, by Lemma 1.7,

$$(1)\ T \vdash \varphi(0, c_2, \ldots, c_n) \quad \text{and} \quad (2)\ T \vdash \forall x_1\, (\varphi(x_1, c_2, \ldots, c_n) \to \varphi(s(x_1), c_2, \ldots, c_n)).$$

Since the c_i do not occur in T nor in $\varphi(\overline{x})$ we have

$$T \vdash \forall x_2 \cdots \forall x_n\, \varphi(0, x_2, \ldots, x_n)$$

from (1). Moreover, we have

$$T, \varphi(x_1, c_2, \ldots, c_n) \vdash \varphi(s(x_1), c_2, \ldots, c_n)$$

from (2). Therefore

$$T, \forall x_2 \cdots x_n\, \varphi(x_1, x_2, \ldots, x_n) \vdash \varphi(s(x_1), c_2, \ldots, c_n)$$

and, since the c_i do not occur in T nor in $\varphi(\overline{x})$, we have

$$T, \forall x_2 \cdots x_n\, \varphi(x_1, x_2, \ldots, x_n) \vdash \forall x_2 \cdots x_n\, \varphi(s(x_1), x_2, \ldots, x_n)$$

and thus

$$T \vdash \forall x_1\, (\forall x_2 \cdots x_n\, \varphi(x_1, x_2, \ldots, x_n) \to \forall x_2 \cdots x_n\, \varphi(s(x_1), x_2, \ldots, x_n))\,.$$

So $\forall x_2 \cdots \forall x_n\, \varphi(\overline{x})$ is T-inductive and thus, by Lemma 1.7, $\forall \overline{x}\, \varphi(\overline{x})$ has a straightforward induction proof in T. $\square$

Theorem 1.16. *$\forall x \forall y \forall z\, (x + y) + z = x + (y + z)$ has a straightforward induction proof in B.*

Proof. Let $\varphi(x, y, z) \equiv (x + y) + z = x + (y + z)$. We will show that $\forall x \forall y \forall z\, \varphi(x, y, z)$ has a uniform straightforward induction proof by induction on z. The result then follows from Lemma 1.15. For the induction base work in B: $(x + y) + 0 =^{(A4)} x + y =^{(A4)} x + (y + 0)$. Thus $B \vdash \varphi(x, y, 0)$. For the induction step work in B and assume $\varphi(x, y, z)$. Then $(x + y) + s(z) =^{(A5)} s((x + y) + z) =^{\mathrm{IH}} s(x + (y + z)) =^{(A5)} x + s(y + z) =^{(A5)} x + (y + s(z))$. So $B \vdash \forall z (\varphi(x, y, z) \to \varphi(x, y, s(z)))$. $\square$

We will now show that, in contrast, commutativity does not have a straightforward induction proof in B. To that aim we define the following model of B.

Definition 1.17. We define the L_{LA}-structure $\mathcal{M}_2$ as follows: the domain of $\mathcal{M}_2$ is $\mathbb{N} \cup \{a, b\}$, 0, s and p are interpreted as:

$$0^{\mathcal{M}_2} = 0 \qquad s^{\mathcal{M}_2}(m) = \begin{cases} m+1 & \text{if } m \in \mathbb{N} \\ m & \text{if } m \in \{a, b\} \end{cases} \qquad p^{\mathcal{M}_2}(m) = \begin{cases} m-1 & \text{if } m \in \mathbb{N}, m \geq 1 \\ 0 & \text{if } m = 0 \\ m & \text{if } m \in \{a, b\} \end{cases}$$

Addition is intrepreted according to the following table:

$+$	0	1	2	$\cdots$	a	b
0					b	a
1		as in $\mathbb{N}$			b	a
2					b	a
$\vdots$					$\vdots$	$\vdots$
a	a	a	a	$\cdots$	b	a
b	b	b	b	$\cdots$	b	a

Lemma 1.18. $\mathcal{M}_2 \models B$

Proof. $\mathcal{M}_2 \models \forall x\, s(x) \neq 0$ because 0 is not a successor. $\mathcal{M}_2 \models p(0) = 0$ by definition. For $\mathcal{M}_2 \models \forall x\, p(s(x)) = x$ we make a case distinction: if $m \in \mathbb{N}$, then $\mathcal{M}_2 \models p(s(m)) = p(m+1) = m$. If $m \in \{a, b\}$, then $\mathcal{M}_2 \models p(s(m)) = p(m) = m$. $\mathcal{M}_2 \models \forall x\, x + 0 = x$ follows immediately from the first column of the addition table. For showing $\mathcal{M}_2 \models \forall x \forall y\, x + s(y) = s(x+y)$ we make a case distinction:

1. If $m, n \in \mathbb{N}$ then we are done because $\mathbb{N} \models m + s(n) = s(m+n)$.
2. If $m \in \{a, b\}$ and $n \in \mathbb{N}$, then $\mathcal{M}_2 \models m + s(n) = m$ and $\mathcal{M}_2 \models s(m+n) = s(m) = m$.
3. If $m \in \mathbb{N} \cup \{a, b\}$ and $n \in \{a, b\}$, then $\mathcal{M}_2 \models m + s(n) = m + n = n'$ and $\mathcal{M}_2 \models s(m+n) = s(n') = n'$. where
$$n' = \begin{cases} a & \text{if } n = b \\ b & \text{if } n = a \end{cases}$$

□

Theorem 1.19. *$\forall x \forall y\, x + y = y + x$ does not have a straightforward induction proof in B.*

Proof. By Lemma 1.7 and the symmetry of the formula, it suffices to show that $\forall y\, x + y = y + x$ is not B-inductive. This follows immediately from Lemma 1.18 and the observation that $\mathcal{M}_2 \models a + 0 = a \neq b = 0 + a$. □

So commutativity of addition does not have a straightforward induction proof, a fact that we could show without the compactness theorem by handcrafting a suitable model. On the other hand, commutativity of addition does have a proof by induction in B.

Theorem 1.20. *Let $\mathcal{F}$ be the set of L_{LA} formulas. Then $B + \mathcal{F}\text{-IND} \vdash \forall x \forall y\, x + y = y + x$*

Proof. We start by showing that $A_4' \equiv \forall x\, 0 + x = x$ has a straightforward induction proof in B. The base case is an instance of (A4). For the step case work in B: $0 + s(x) =^{(A5)} s(0 + x) =^{\mathrm{IH}} s(x)$.

Next we show that $A'_5 \equiv \forall x \forall y\, s(x) + y = s(x+y)$ has a uniform straightforward induction proof by induction on y in B. For the base case work in B: $s(x) + 0 =^{(A4)} s(x) =^{(A4)} s(x+0)$. For the step case work in B: $s(x) + s(y) =^{(A5)} s(s(x)+y) =^{\mathrm{IH}} s(s(x+y)) =^{(A5)} s(x+s(y))$.

Finally we show that $\forall x \forall y\, x + y = y + x$ has a uniform straightforward induction proof by induction on y in B, A'_4, A'_5. For the base case work in B, A'_4, A'_5: $x + 0 =^{(A4)} x =^{A'_4} 0 + x$. For the step case work in B, A'_4, A'_5: $x + s(y) =^{(A5)} s(x+y) =^{\mathrm{IH}} s(y+x) =^{A'_5} s(y) + x$. □

1.4 No need for many induction axioms

In the previous section we have seen that associativity of plus has a straightforward induction proof (over the base theory B) but commutativity of plus does not. Instead, we showed the commutativity of plus by showing the lemmas $\forall x\, 0 + x = x$ and $\forall x \forall y\, s(x) + y = s(x+y)$ by straightforward induction proofs. Proving a theorem by first showing several lemmas by straightforward induction proofs is indeed a common strategy, one that we will also see at work with considerable success in Chapter 2. This example may lead one to the assessment that, in general, it is necessary to carry out several inductions. This assessment is not correct. In this section we will see techniques for pulling several inductions together into a single one.

An induction axiom is called *parameter-free* if it is of the form $I_x\varphi(x)$, i.e., if the induction formula contains only a single free variable, and hence, no parameters. One way of transforming arbitrary induction axioms into parameter-free induction axioms, at the expense of increasing the quantifier complexity of the induction formula, is shown in the following lemma.

Lemma 1.21. *Let $L \supseteq \{0, s\}$, let $\varphi(x, \overline{z})$ be an L formula and define*

$$\varphi^-(x) \equiv \forall \overline{z} \left(\varphi(0, \overline{z}) \wedge \forall y \left(\varphi(y, \overline{z}) \to \varphi(s(y), \overline{z}) \right) \to \varphi(x, \overline{z}) \right).$$

Then $\varphi^-(x)$ is $\emptyset$-inductive and $\vdash \forall \overline{z}\, I_x\varphi(x, \overline{z}) \leftrightarrow I_x\varphi^-(x)$.

Proof. Observe that $\vdash \varphi^-(0)$ holds trivially. For showing $\vdash \forall x\, (\varphi^-(x) \to \varphi^-(s(x)))$, let x_0 be s.t. $\varphi^-(x_0)$ holds and let $\overline{z}$ be s.t. the premises (1) $\varphi(0, \overline{z})$ and (2) $\forall y\, (\varphi(y, \overline{z}) \to \varphi(s(y), \overline{z}))$ of $\varphi^-(s(x_0))$ hold. Then, from $\varphi^-(x_0)$, (1), and (2) we obtain $\varphi(x_0, \overline{z})$ and it remains to show $\varphi(s(x_0), \overline{z})$. This is done by letting $y = x_0$ in (2).

Then, by Lemma 1.6/4, $\vdash I_x\varphi^-(x) \leftrightarrow \forall x\, \varphi^-(x)$ and, by a quantifier shift, $\vdash \forall x\, \varphi^-(x) \leftrightarrow \forall \overline{z}\, I_x\varphi(x, \overline{z})$. □

The parameter-free induction axioms obtained in Lemma 1.21 can now be used to pull several induction together into a single one.

Theorem 1.22. *Let $L \supseteq \{0, s\}$ and let $\varphi_1(x, \overline{z}), \ldots, \varphi_n(x, \overline{z})$ be L formulas. Then there is an L formula $\varphi(x)$ s.t. $\vdash \bigwedge_{i=1}^n \forall \overline{z} I_x\varphi_i(x, \overline{z}) \leftrightarrow I_x\varphi(x)$.*

Proof. For $i = 1, \ldots, n$ apply Lemma 1.21 to obtain an $\emptyset$-inductive formula $\varphi_i^-(x)$ with $\vdash \forall \overline{z}\, I_x\varphi_i(x, \overline{z}) \leftrightarrow I_x\varphi_i^-(x)$. Define $\varphi(x) \equiv \bigwedge_{i=1}^n \varphi_i^-(x)$. Then $\varphi(x)$ is $\emptyset$-inductive by Lemma 1.6/2. So, by Lemma 1.6/4 we have $\vdash I_x\varphi_i^-(x) \leftrightarrow \forall x\, \varphi_i^-(x)$ and $\vdash I_x\varphi(x) \leftrightarrow \forall x\, \varphi(x)$. Therefore we obtain

$$\vdash \bigwedge_{i=1}^n \forall \overline{z}\, I_x\varphi_i(x, \overline{z}) \leftrightarrow \bigwedge_{i=1}^n I_x\varphi_i^-(x) \leftrightarrow \bigwedge_{i=1}^n \forall x\, \varphi_i^-(x) \leftrightarrow \forall x\, \varphi(x) \leftrightarrow I_x\varphi(x).$$

□

Lemma 1.21 increases the quantifier complexity of induction formulas in order to remove the parameters and to obtain $\emptyset$-inductive formulas that can be combined into a single one. In the following theorem we will see another strategy for merging several inductions into one. This strategy does not increase quantifier complexity but makes the (very mild) assumption that different numerals can be proved different in the background theory.

Theorem 1.23. *Let $L \supseteq \{0, s\}$, let T be an L theory s.t. $T \vdash \underline{i} \neq \underline{j}$ for all $i, j \in \mathbb{N}$ with $i \neq j$, let $\varphi_1(x, \overline{z}), \dots, \varphi_n(x, \overline{z})$ be L formulas, and let σ be an L sentence s.t. $T, \forall \overline{z}\, I_x\varphi_1(x, \overline{z}), \dots, \forall \overline{z}\, I_x\varphi_n(x, \overline{z}) \vdash \sigma$. Then there is an L formula $\varphi(x, y, \overline{z})$ s.t. $T, \forall y \forall \overline{z}\, I_x\varphi(x, y, \overline{z}) \vdash \sigma$.*

Proof. Define $\varphi(x, y, \overline{z}) \equiv \bigwedge_{j=1}^{n}(y = \underline{j} \to \varphi_j(x, \overline{z}))$. Then we have

$$T \vdash \varphi(x, \underline{i}, \overline{z}) \leftrightarrow \top \wedge \cdots \wedge \top \wedge \varphi_i(x, \overline{z}) \wedge \top \wedge \cdots \wedge \top \leftrightarrow \varphi_i(x, \overline{z})$$

and hence

$$T \vdash I_x\varphi(x, \underline{i}, \overline{z}) \leftrightarrow I_x\varphi_i(x, \overline{z}).$$

Therefore

$$T, \forall y \forall \overline{z}\, I_x\varphi(x, y, \overline{z}) \vdash \bigwedge_{i=1}^{n} \forall \overline{z}\, I_x\varphi_i(x, \overline{z})$$

and thus

$$T, \forall y \forall \overline{z} I_x\varphi(x, y, \overline{z}) \vdash \sigma.$$

□

Example 1.24. Towards proving commutativity of addition with a single induction we first observe that $i \neq j$ implies $B \vdash \underline{i} \neq \underline{j}$. To see this, let w.l.o.g. $i < j$ and proceed by induction on i. If $i = 0$ then $B \vdash 0 \neq s(\underline{j-1})$ by (A1). For the case $i > 0$ we have $B \vdash \underline{i-1} \neq \underline{j-1}$ by induction hypothesis, so it suffices to show that $B \vdash \underline{i-1} \neq \underline{j-1} \to \underline{i} \neq \underline{j}$. To that aim show the contraposition in B: if $\underline{i} = \underline{j}$ then $p(\underline{i}) = p(\underline{j})$ and so, by (A3), $\underline{i-1} = \underline{j-1}$.

Then an application of Theorem 1.23 to the proof of Theorem 1.20 shows that the formula

$$\varphi(x, y) \equiv (y = \underline{1} \to 0 + x = x) \wedge (y = \underline{2} \to \forall z\, s(z) + x = s(z + x)) \wedge (y = \underline{3} \to \forall z\, x + z = z + x)$$

satisfies

$$B, \forall y\, I_x\varphi(x, y) \vdash \forall x \forall y\, x + y = y + x.$$

1.5 No need for strict strenghtenings

In Section 1.2 we have seen that $\forall n \exists k \sum_{i=1}^{n}(2i - 1) = k^2$ does not have a straightforward induction proof. The solution was to show the stronger statement $\forall n \sum_{i=1}^{n}(2i - 1) = n^2$ with a straightforward induction proof. This is an experience one often makes when trying to prove a statement by induction: the original statement cannot be shown by a straightforward induction proof, but a slight generalisation or strengthening can be. As useful as this technique is for manually finding proofs by induction, it is also somewhat misleading, since it seems to suggest that a logical strenghtening is often necessary. In fact, it is quite easy to show that, in the sense of the following theorem, a strict logical strenghtening is never necessary.

Theorem 1.25. *If $T, I_x\varphi(x) \vdash \sigma$ then there is a $\psi(x)$ s.t. $T, I_x\psi(x) \vdash \sigma$ and $T \vdash \forall x\, \psi(x) \leftrightarrow \sigma$.*

Proof. Let $T, I_x\varphi(x) \vdash \sigma$. Then we claim that

$$T \vdash \varphi(0) \vee \sigma \tag{1.5}$$
$$T \vdash \forall x\,(\varphi(x) \to \varphi(s(x))) \vee \sigma, \text{ and} \tag{1.6}$$
$$T \vdash \forall x\,\varphi(x) \to \sigma. \tag{1.7}$$

To show this let A, B, and C be sentences and assume $T, A \wedge B \to C \vdash \sigma$. Let $\mathcal{M} \models T$. Then $\mathcal{M} \not\models A$ and $\mathcal{M} \not\models \sigma$ is impossible which shows (1.5) and (1.6). Moreover, $\mathcal{M} \models C$ and $\mathcal{M} \not\models \sigma$ is impossible which shows (1.7).

Let $\psi(x) \equiv \varphi(x) \vee \sigma$. Then by (1.5) we have $T \vdash \psi(0)$. Moreover, we claim that $T \vdash \forall x(\psi(x) \to \psi(s(x)))$. To see this work in T: let x_0 be s.t. $\psi(x_0)$, i.e., $\varphi(x_0) \vee \sigma$. If σ then $\psi(s(x_0))$ and we are done. If $\neg\sigma$ then, by (1.6), $\forall x\,(\varphi(x) \to \varphi(s(x)))$, which, by letting $x := x_0$, yields $\varphi(s(x_0))$ and thus $\psi(s(x_0))$.

So $\psi(x)$ is T-inductive and therefore $T, I_x\psi(x) \vdash \forall x\,\psi(x)$ and thus, by (1.7), we obtain $T, I_x\psi(x) \vdash \sigma$. Moreover, we have $\vdash \sigma \to \forall x\,\psi(x)$. $T \vdash \forall x\,\psi(x) \to \sigma$ follows from (1.7). □

Example 1.26. Using the notation of Challenge Problem 1.1 let $\theta(x, y) \equiv f(x) = y \cdot y$. Then we have shown in Theorem 1.11 that $\forall x \exists y\,\theta(x, y)$ does not have a straightforward induction proof in T_f. On the other hand, the natural formalisation of the proof of Theorem 1.2 shows that $\forall x\,\theta(x, x)$ does have a straightforward induction proof in T_f, i.e., $T_f, I_x\theta(x, x) \vdash \forall x\,\theta(x, x)$. Thus we also have $T_f, I_x\theta(x, x) \vdash \forall x \exists y\,\theta(x, y)$. Theorem 1.25 now shows that the formula $\psi(x) \equiv \theta(x, x) \vee \forall x \exists y\,\theta(x, y)$ satisfies

$$T_f, I_x\psi(x) \vdash \forall x \exists y\,\theta(x, y) \text{ and}$$
$$T_f \vdash \forall x\,\psi(x) \leftrightarrow \forall x \exists y\,\theta(x, y).$$

Chapter notes

Theorem 1.11 is due to Lundstedt [20]. The notion of a T-inductive formula has been studied in the context of automated inductive theorem proving in [15] for different induction schemes. It is closely related to the notion of cut in models of arithmetic, see, e.g., [17, 11]. The basic theory B of linear arithmetic was studied in [22] which gave a simple finite axiomatisation of B together with open induction. Section 1.4 is based on Section 2.4 of [15]. The proof of Theorem 1.23 is due to Gentzen [7]. Theorem 1.25 is a slight generalisation of [15, Theorem 3.1.].

Chapter 2

Equational theory exploration

It is quite common for proofs by induction to proceed by establishing some lemmas by straightforward induction and then using these lemmas to prove the main theorem, possibly by induction again (as we have seen, e.g., in the case of commutativity of addition). This observation has prompted an interest in applying the approach of theory exploration to inductive theorem proving. The basic idea of theory exploration is that a user specifies a theory, i.e., some axioms and definitions, and the computer system explores this theory, i.e., it proves some interesting facts about these notions. In contrast to traditional theorem proving which is usually goal-oriented, theory exploration proceeds in the other direction, starting from the axioms trying to discover useful lemmas. In this chapter we study a simple such algorithm that is restricted to equational lemmas. This class is already very useful for simple examples in functional programming based on primitive recursive functions over inductive data types. The algorithm has two phases:

1. Find a set of conjectures (equations that could not be falsified).

2. Try to prove these conjectures by induction, possibly using other conjectures which have already been proved.

2.1 Finding conjectures

The first phase consists in conjecturing a set of equations. We start out with inductive data types $D_1, \ldots, D_n$ together with primitive recursive definitions for functions symbols over these data types. In this setting we have the following procedures at our disposal.

1. $\textsc{Value}(t)$ which, given a variable-free term t computes the value of t.

2. $\textsc{GenerateRandomTuple}(\overline{x})$ which, given a tuple of typed variables, generates a tuple of random values, i.e., constructor terms, of the appropriate types.

The value of a variable-free term t is simply a constructor term t^* which is the result of computing the function defined by primitive recursion in the standard model as in Theorem 0.10. What random means here exactly is not very important (it should be rather small values and reasonably random in the sense that there is a high probability that n consecutive calls result in n pairwise different tuples of variables). We will write $|t|$ for the size of a term t which is defined inductively as follows:

$$|c| = 1 \qquad \text{for a constant symbol } c$$

$$|f(t_1, \dots, t_n)| = 1 + \sum_{i=1}^{n} |t_i| \qquad \text{for an } n\text{-ary function symbol } f$$

But again, what exactly size means in this context is not too important. What is important is the fact that a bound on the size will yield a finite set of terms.

The procedure $\textsc{Conjecture}(k, \overline{x}, N)$ takes the arguments k, $\overline{x}$, and N where $k \in \mathbb{N}$ is a term size parameter, $\overline{x}$ is a tuple of typed variables, and $N \in \mathbb{N}$ is the number of counterexamples to generate for trying to separate terms. The algorithm works as follows: for each $i \in \{1, \dots, n\}$ it maintains an equivalence relation E_i over a set T_i of small terms of type D_i in the variables $\overline{x}$. The interpretation of $t \; E_i \; t'$ is that we conjecture that $\forall \overline{x} \, t = t'$ is true in the standard model. In the beginning it (optimistically) conjectures all equations between terms of T_i to be true. In order to arrive at a more realistic picture it generates N random tuples $\overline{a}$ matching the types of $\overline{x}$. Each random tuple $\overline{a}$ is used to compute the value of both terms t and t' of a conjectured equality. If $t[\overline{x} \backslash \overline{a}] \neq t'[\overline{x} \backslash \overline{a}]$ in the standard model, we have found a counterexample and do no longer conjecture $\forall \overline{x} \, t = t'$. If $t[\overline{x} \backslash \overline{a}] = t'[\overline{x} \backslash \overline{a}]$ in the standard model, we continue to conjecture $\forall \overline{x} \, t = t'$. The equivalence relation is updated to reflect these changes. At the end

Algorithm 2.1 Forming equational conjectures

```
procedure $\textsc{Conjecture}(k, \overline{x}, N)$
    for all $i \in \{1, \dots, n\}$ do
        $T_i := \{t \text{ term of sort } D_i \mid |t| \leq k, \mathrm{Var}(t) \subseteq \{\overline{x}\}\}$
        $E_i := \{(t_1, t_2) \mid t_1, t_2 \in T_i\}$
    end for
    loop $N$ times
        $\overline{a} := \textsc{GenerateRandomTuple}(\overline{x})$
        for each equivalence class $C$ of each $E_i$ do
            $E' := \{(t_1, t_2) \in C \mid \textsc{Value}(t_1[\overline{x} \backslash \overline{a}]) = \textsc{Value}(t_2[\overline{x} \backslash \overline{a}])\}$
            Replace $C$ by $E'$ in $E_i$
        end for
    end loop
    return $\{t_1 = t_2 \mid (t_1, t_2) \in E_i, 1 \leq i \leq n\}$
end procedure
```

of this algorithm, for all $i \in \{1, \dots, n\}$, (T_i, E_i) is an equivalence relation where two terms are in the same equivalence class iff they withstood N tests. So while $t \; E_i \; t'$ does not imply that $\forall \overline{x} \, t = t'$ is true in the standard model, it is considered a conjecture because it has withstood N tests. In particular, $t \not\!E_i \; t'$ implies that $\forall \overline{x} \, t = t'$ is false in the standard model because a counterexample has been found among the N randomly generated tuples.

Example 2.1. Consider the sort Nat with the usual definition of addition D_+ by primitive recursion on the right argument. Let $\overline{x} = x_1 : \mathsf{Nat}, x_2 : \mathsf{Nat}$ and $k = 4$, then $|T_1| = 48$. While this is easily in the scope of implementations, for applying the algorithm by hand we will here work with the following subset T of T_1.

$$T = \{x_1, x_2, s(x_1), s(x_2), 0 + x_1, x_1 + x_2, x_2 + x_1, s(x_1) + x_2, s(x_1 + x_2)\}.$$

In the beginning everyhing is conjectured to be equal, so we have one equivalence class $C = T$.

The algorithm now picks a first tuple $\overline{a}$ of values, say $\overline{a} = (0, 1)$, and evalutes the terms in T on these values.

t	$\text{VALUE}(t[\overline{x}\backslash\overline{a}])$
x_1	0
x_2	1
$s(x_1)$	1
$s(x_2)$	2
$0 + x_1$	0
$x_1 + x_2$	1
$x_2 + x_1$	1
$s(x_1) + x_2$	2
$s(x_1 + x_2)$	2

This leads to a separation of C into the three classes

$C_0 = \{x_1, 0+x_1\}$, $\quad C_1 = \{x_2, s(x_1), x_1+x_2, x_2+x_1\}$, $\quad$ and $C_2 = \{s(x_2), s(x_1)+x_2, s(x_1+x_2)\}$.

Now the algorithm picks another tuple at random, say $\overline{a} = (1, 0)$, and evalutes the members of each equivalence class seperately yielding:

t	$\text{VALUE}(t[\overline{x}\backslash\overline{a}])$
x_1	1
$0 + x_1$	1

t	$\text{VALUE}(t[\overline{x}\backslash\overline{a}])$
x_2	0
$s(x_1)$	2
$x_1 + x_2$	1
$x_2 + x_1$	1

t	$\text{VALUE}(t[\overline{x}\backslash\overline{a}])$
$s(x_2)$	1
$s(x_1) + x_2$	2
$s(x_1 + x_2)$	2

At this point the only equivalence classes of size strictly greater than one are $\{x_1, 0 + x_1\}$, $\{s(x_1) + x_2, s(x_1 + x_2)\}$, and $\{x_1 + x_2, x_2 + x_1\}$. Thus the remaining conjectures are $\forall x_1\, x_1 = 0 + x_1$, $\forall x_1 \forall x_2\, s(x_1) + x_2 = s(x_1 + x_2)$, and $\forall x_1 \forall x_2\, x_1 + x_2 = x_2 + x_1$. These three equations are indeed true.

2.2 Proving conjectures

The second phase consists in trying to prove the conjectures obtained from the first phase. This will involve checking whether certain formulas are provable in first-order logic with equality. Since provability is undecidable, we introduce a timeout parameter $t \in \mathbb{N}$ to interrupt a proof search if it is running for too long. That way we loose completeness, but for practical purposes, which are the focus of this chapter, this is justifiable.

Definition 2.2. We fix an algorithm $\mathcal{A}$ for proof search in first-order logic with equality. Then, for a theory T, a sentence σ, and a timeout parameter $t \in \mathbb{N}$ we write $T \vdash^t \sigma$ to state that $\mathcal{A}$ finds a proof of σ in T within time t.

In this notation, $\mathcal{A}$ is not referenced explicitly anymore. This is justified because we will only use the relation $\vdash^t$ in a way that does not depend on $\mathcal{A}$. Moreover, it is not relevant for our purposes wether t measures wall-clock time, CPU time, or the number of steps in the execution of $\mathcal{A}$. Therefore we do not specify it any further. Also note that $T \nvdash^t \sigma$ means that within time t no proof of $T \vdash \sigma$ is found (by $\mathcal{A}$). For any $t \in \mathbb{N}$: $T \vdash^t \sigma$ implies $T \vdash \sigma$ and thus, by contraposition, $T \nvdash \sigma$ implies $T \nvdash^t \sigma$.

The procedure EXPLORE$(k, \overline{x}, N, A, t)$ takes the arguments k, $\overline{x}$, N, A, and t where, as above, k is a term size parameter, $\overline{x}$ is a tuple of typed variables, and N is the number of counterexamples. Moreover, A is a set of axioms consisting of the primitive recursive definitions of all function symbols which are not constructors, and t is a timeout parameter to limit the prover. The algorithm works as follows: it obtains a set of conjectures C from calling CONJECTURE$(k, \overline{x}, N)$. It goes through this set of conjectures and tries to prove each one of them. For each particular conjecture $\forall\overline{x}\,\varphi(\overline{x})$ it first checks whether $\forall\overline{x}\,\varphi(\overline{x})$ is already provable from the current lemmas and A in pure first-order logic. In order to avoid undecidability issues, this check is done with the timeout t. If $A, L \vdash^t \forall\overline{x}\,\varphi(\overline{x})$, it proceeds to the next conjecture and does not add $\forall\overline{x}\,\varphi(\overline{x})$ to the set of lemmas. In this case, $\forall\overline{x}\,\varphi(\overline{x})$ is not considered interesting enough since it follows from other lemmas by pure first-order logic. This filters out trivial modifications like equality up to variable renamings, equations modulo symmetry, etc. If $A, L \nvdash^t \forall\overline{x}\,\varphi(\overline{x})$, the algorithm checks whether $\forall\overline{x}\,\varphi(\overline{x})$ has a uniform straightforward induction proof from A and the already proven set of lemmas L. Again, to avoid undecidability issues, this is done with timeout t and is a check in pure first-order logic that can be done with any FOL theorem prover. If a straightforward induction proof is found, this conjecture is added to the lemma set. If no straightforward induction proof is found, it is discarded. In the end the set L of lemmas

Algorithm 2.2 Equational theory exploration

```
1:  procedure EXPLORE($k, \overline{x}, N, A, t$)
2:      $C :=$ CONJECTURE($k, \overline{x}, N$)
3:      $L := \emptyset$
4:      while $C \neq \emptyset$ do
5:          Select $\varphi(\overline{x}) \in C$
6:          $C := C \setminus \{\varphi(\overline{x})\}$
7:          if $A, L \nvdash^t \forall\overline{x}\,\varphi(\overline{x})$ then
8:              if $\exists i \in \{1, \dots, m\}$ s.t. $\varphi(c_1, \dots, c_{i-1}, x_i, c_{i+1}, \dots, c_m)$ is $A, L$-inductive then
9:                  $L := L \cup \{\forall\overline{x}\,\varphi(\overline{x})\}$
10:             end if
11:         end if
12:     end while
13:     return $L$
14: end procedure
```

contains enough lemmas so that all conjectures that could be proved can be proved from the lemmas in L by pure first-order logic, i.e., without induction.

Example 2.3. Continuing Example 2.1 we have $A = \{\forall x\, x + 0 = x, \forall x \forall y\, x + s(y) = s(x + y)\}$. The call to CONJECTURE$(k, \overline{x}, N)$ will fill C with a set of conjectures including $0 + x_1 = x_1$, $s(x_1) + x_2 = s(x_1 + x_2)$, and $x_1 + x_2 = x_2 + x_1$.

We assume that the algorithm selects $0 + x_1 = x_1$ first. Then $A \nvdash \forall x_1\, 0 + x_1 = x_1$, so the algorithm tries a straightforward induction proof next. This succeeds since $0 + x_1 = x_1$ is inductive. So $L = \{\forall x_1\, 0 + x_1 = x_1\}$ after this step.

We assume that the algorithm now selects $s(x_1) + x_2 = s(x_1 + x_2)$ from C. Since $A, \forall x_1\, 0 + x_1 = x_1 \nvdash \forall x_1 \forall x_2\, s(x_1) + x_2 = s(x_1 + x_2)$, the algorithm tries to find a straightforward induction proof and succeeds. This yields $L = \{\forall x_1\, 0 + x_1 = x_1, \forall x_1 \forall x_2\, x_1 + s(x_2) = s(x_1 + x_2)\}$

Finally, when the algorithm selects $x_1 + x_2 = x_2 + x_1$, the same repeats and it returns a set L containing, among other lemmas, the commutativity of plus.

Note that the result of this algorithm is sensitive to the order in which conjectures are selected from C for processing. In practice this is dealt with by heuristics which select simple equations first. Moreover, in principle it is possible to modify the algorithm in such a way that, after the first pass through C it reexamines the conjecutres it could not prove and iterates this re-examination until no more new conjectures can be proved in an interation.

This algorithm has several advantages which make it very useful in practice: the search space for induction formulas is quite limited, essentially by the number of conjectures. Moreover, the number of conjectures is limited, which is due to the fact that the counterexamples from the first phase filter out many wrong equations very quickly. This leads to a limited number of provability checks with timeout in pure first-order logic. Moreover, these provability checks are in a logical context (equational reasoning for primitive recursive functions over inductive data types) where first-order theorem provers are quite efficient.

2.3 Analysis

What can we prove with this method? We have seen that we can prove, e.g., the commutativity of addition, so it goes beyond straightforward induction. The central observation is: the proofs this methods allows us to generate are either straightforward induction proofs in the original theory A or straightforward induction proofs in a theory $A + L'$ where L' are lemmas which have in turn been proved by this method. This naturally leads to the notion of iterated straightforward induction proof.

Definition 2.4. Let L be a first-order language, let T be an L theory. A finite sequence $\pi = \varphi_1(c_1, \dots, x_{i_1}, \dots, c_{k_1}), \dots, \varphi_n(c_1, \dots, x_{i_n}, \dots, c_{k_n})$ is called *iterated uniform straightforward induction proof of* $\forall \overline{x}\, \varphi_n(\overline{x})$ *in* T if $\varphi_1(x_1, \dots, x_{k_1}), \dots, \varphi_n(x_1, \dots, x_{k_n})$ are L formulas and, for $j = 1, \dots, n$, the formula $\varphi_j(c_1, \dots, x_{i_j}, \dots, c_{k_n})$ is $T + \{\forall \overline{x}\, \varphi_1(\overline{x}), \dots, \forall \overline{x}\, \varphi_{j-1}(\overline{x})\}$-inductive. If $\varphi_1(\overline{x}), \dots, \varphi_n(\overline{x})$ are equations, we call π an *equational iterated uniform straighforward induction proof (e.i.u.s.i. proof) in* T.

Theorem 2.5. *Let* $k, \overline{x}, N, A, t$ *be arguments of* Explore, *let* $L = \text{Explore}(k, \overline{x}, N, A, t)$, *and let* $\forall \overline{x}\, \varphi(\overline{x}) \in L$. *Then* $\forall \overline{x}\, \varphi(\overline{x})$ *has an e.i.u.s.i. proof in* A.

Proof. During the execution of the main loop of $\text{Explore}(k, \overline{x}, N, A, t)$ we maintain an e.i.u.s.i. proof $\varphi_1(c_1, \dots, x_{i_1}, \dots, c_m), \dots, \varphi_j(c_1, \dots, x_{i_j}, \dots, c_m)$ s.t. $L = \{\forall \overline{x}\, \varphi_1(x), \dots, \forall \overline{x}\, \varphi_j(x)\}$. In the beginning it is initialised as empty list. Whenever Explore adds a new sentence $\forall \overline{x}\, \varphi(\overline{x})$ to L in line 9 that was proved by uniform straightforward induction on x_i in line 8, we set $i_{j+1} := i$ and $\varphi_{j+1} :\equiv \varphi$, and add $\varphi_{j+1}(c_1, \dots, x_{i_{j+1}}, \dots, c_m)$ to our e.i.u.s.i. proof. Then $\varphi_{j+1}(c_1, \dots, x_{i_{j+1}}, \dots, c_m)$ is $A, \forall \overline{x}\, \varphi_1(\overline{x}), \dots, \forall \overline{x}\, \varphi_j(\overline{x})$-inductive and, after execution of line 9, $L = \{\forall \overline{x}\, \varphi_1(\overline{x}), \dots, \forall \overline{x}\, \varphi_{j+1}(\overline{x})\}$. $\square$

At this point it is not clear how to prove negative results of the form: the sentence σ does not have an e.i.u.s.i. proof in the theory T. Model-theoretic techniques do not seem to apply in a straightforward way. The strategy we employ for obtaining negative results is to work with an over-approximation of the sentences provable by e.i.u.s.i. proofs. The central observation is: we only prove equations and we only carry out induction on equations. This motivates us to study, with methods from mathematical logic, what can be proved when induction is restricted to induction on equations, or, more generally, on atoms in Chapter 3.

Chapter notes

Theory exploration is a large subject. The methods described in this chapter are mostly abstractions of algorithms used in the HipSpec system. Algorithm 2.2 is a simplified version of the algorithm underlying HipSpec [3]. Algorithm 2.1 is a simplified version of the algorithm underlying QuickSpec [5]. These algorithms work quite well on many problems concerning simple primitive reursive functions over inductive data types, such as those found, e.g., in [16, 2, 4]. There are also extensions of these algorithms that go beyond equational lemmas [23].

Chapter 3

Atomic induction

In this chapter we will consider atomic induction, i.e., the induction scheme restricted to atoms as induction formulas. The induction scheme for a set of formulas Φ is defined as Φ-IND $= \{\forall \overline{z}\, I_x\varphi(x, \overline{z}) \mid \varphi(x, \overline{z}) \in \Phi\}$. *Atomic induction* is $\mathrm{IAtom}_L = A_L$-IND where A_L is the set of atoms in the first-order language L. Often, when the language L is clear from the context or irrelevant, we will simply write IAtom for IAtom_L. Restricting the induction scheme to a certain set of induction formulas is the most important way of defining subsystems of Peano arithmetic and related theories in mathematical logic. The restriction considered here, to atoms, results in one of the weakest classes of theories with induction axioms that have been considered in the literature.

3.1 E.i.u.s.i. proofs and atomic induction

The first observation of this chapter will be that every sentence that has an e.i.u.s.i. proof also has a proof by atomic induction, as hinted at in the end of Chapter 2.

Theorem 3.1. *Let T be a theory and let σ be a sentence that has an e.i.u.s.i. proof in T. Then $T + \mathrm{IAtom} \vdash \sigma$.*

Proof. Let $\varphi_1(c_1, \ldots, x_{i_1}, \ldots, c_{k_1}), \ldots, \varphi_n(c_1, \ldots, x_{i_n}, \ldots, c_{k_n})$ be an e.i.u.s.i. proof. We show, by induction on $j \in \{1, \ldots, n\}$, that $T + \mathrm{IAtom} \vdash \forall \overline{x}\, \varphi_j(\overline{x})$. To that aim let $T_j = T + \{\forall \overline{x}\, \varphi_1(\overline{x}), \ldots, \forall \overline{x}\, \varphi_{j-1}(\overline{x})\}$. Then, by induction hypothesis, $T + \mathrm{IAtom} \vdash T_j$. For $j = 1, \ldots, n$ the formula $\varphi_j(c_1, \ldots, x_{i_j}, \ldots, c_{k_j})$ is T_j-inductive, i.e.,

$$T_j \vdash \varphi_j(c_1, \ldots, 0, \ldots, c_{k_j}) \text{ and}$$
$$T_j \vdash \forall x_i\, (\varphi_j(c_1, \ldots, x_i, \ldots, c_{k_j}) \to \varphi_j(c_1, \ldots, s(x_i), \ldots, c_{k_j})).$$

Since $\varphi_j(\overline{x})$ is an equation, we have

$$T_j + \mathrm{IAtom} \vdash \forall x_i\, \varphi_j(c_1, \ldots, x_i, \ldots, c_{k_j})$$

and thus

$$T_j + \mathrm{IAtom} \vdash \forall \overline{x}\, \varphi_j(\overline{x}).$$

So, by induction hypothesis, $T + \mathrm{IAtom} \vdash \forall \overline{x}\, \varphi_j(\overline{x})$. □

Corollary 3.2. *Let $k, \overline{x}, N, A, t$ be arguments of* Explore*, let $L = \text{Explore}(k, \overline{x}, N, A, t)$, and let $\forall \overline{x}\, \varphi(\overline{x}) \in L$. Then $A + \text{IAtom} \vdash \forall \overline{x}\, \varphi(\overline{x})$.*

At this point we can stop and observe that we already know several sentences that are provable by atomic induction, for example: the associativity and the commutativity of plus are provable in $B + \text{IAtom}_{L_{\text{LA}}}$, the first has a straighforward induction proof, the second does not. The formula $\forall x\, \varphi(x)$ from Example 1.4 is provable in $T_g + \text{IAtom}_{L_g}$. Moreover, the formula $\forall x\, \psi(x)$ from Challenge Problem 1.1, which does not have a straightforward induction proof, is provable in $T_f + \text{IAtom}_{L_f}$. On the other hand, we will later see that there are also statements which have straightforward induction proofs but are not provable with open induction, see Corollary 3.8.

3.2 The simplest nonstandard model of induction

In order to show that a certain sentence σ is not provable from atomic induction, we need a model of atomic induction that does not satisfy σ. So far, the nonstandard models we have constructed were either for very weak theories without induction, such as B from Section 1.3, or they were constructed by strong tools such as the compactness theorem. In this section we will consider a simple nonstandard model of $B + \text{IAtom}_{L_{\text{LA}}}$. Showing that a certain (handcrafted) structure satisfies an induction scheme, even if it is for a very restricted set of formulas, is usually more difficult than showing that it satisfies a finitely axiomatised base theory like B because there are infinitely many induction formulas to consider. Our model will essentially look like the standard model $\mathbb{N}$ to which we add a single nonstandard element, which we call ∞, that behaves like infinity.

Definition 3.3. We define the L_{LA}-structure $\mathbb{N}^\infty$ as follows: The domain of $\mathbb{N}^\infty$ is $\mathbb{N} \cup \{\infty\}$. On $\mathbb{N}$ the constant and function symbols 0, s, p, and $+$ are defined in the standard way. If ∞ is involved we define $s^{\mathbb{N}^\infty}(\infty) = p^{\mathbb{N}^\infty}(\infty) = \infty$ and $a +^{\mathbb{N}^\infty} \infty = \infty +^{\mathbb{N}^\infty} a = \infty$ for any $a \in \mathbb{N} \cup \{\infty\}$.

One way to think about this is that ∞ absorbs all finite numbers. So if ∞ occurs as an argument to an L_{LA} term, then, over $\mathbb{N}^\infty$, the entire term will evaluate to ∞. We then have the following theorem.

Theorem 3.4. $\mathbb{N}^\infty \models B + \text{IAtom}_{L_{\text{LA}}}$

For showing that $\mathbb{N}^\infty \models B$ we have to check all the axioms. While this can be a little tedious sometimes, it is usually not very difficult. The more interesting, and more difficult, part is to show that $\mathbb{N}^\infty \models \text{IAtom}_{L_{\text{LA}}}$. In order to prepare this second part, we need to prove two lemmas first.

Lemma 3.5. *Let $t(x)$ be an L_{LA} term. Then there is a $k_t \in \mathbb{N}$ and a linear polynomial $P_t(X) \in \{\sum_{i=0}^n a_i X^i \mid a_0 \in \mathbb{Z}, a_1, \ldots, a_n \in \mathbb{N}\}$ s.t. $t^{\mathbb{N}}(n) = P_t(n)$ for all $n \geq k_t$. Moreover, if $t(x)$ contains x, then $P(X)$ is not constant.*

Proof. We assign $P_t(X)$ and k_t to t by induction on the structure of t. If $t \equiv 0$ then $P_t(X) = 0$ and $k_t = 0$. If $t \equiv x$ then $P_t(X) = X$ and $k_t = 0$. If $t \equiv s(t')$ then $P_t(X) = P_{t'}(X) + 1$ and $k_t = k_{t'}$. If $t \equiv t_1 + t_2$ then $P_t(X) = P_{t_1}(X) + P_{t_2}(X)$ and $k_t = \max\{k_{t_1}, k_{t_2}\}$. If t contains x, then so does one of the t_i. Since $P_{t_i}(X)$ is non-constant, so is $P_t(X)$. Let $t \equiv p(t')$. If $P_{t'}(X) = 0$ then $P_t(X) = 0$ and $k_t = k_{t'}$. Otherwise $P_{t'}(n) = t'^{\mathbb{N}}(n) \geq 0$ for all $n \geq k_{t'}$ by induction hypothesis and thus $P_{t'}(n) = t'^{\mathbb{N}}(n) \geq 1$ for all $n \geq k_{t'} + 1$. Let $P_t(X) = P_{t'}(X) - 1$ and $k_t = k_{t'} + 1$. Then for $n \geq k_t = k_{t'} + 1$: $P_t(n) = P_{t'}(n) - 1 = t'^{\mathbb{N}}(n) - 1 = p(t'^{\mathbb{N}}(n)) = t^{\mathbb{N}}(n)$. $\square$

Lemma 3.6. *Let $\overline{z} = z_1, \ldots, z_k$, let $t_1(x, \overline{z})$ and $t_2(x, \overline{z})$ be L_{LA} terms, and let $a_1, \ldots, a_k \in \mathbb{N} \cup \{\infty\}$. If $\mathbb{N}^\infty \models t_1(\underline{n}, \overline{a}) = t_2(\underline{n}, \overline{a})$ for all $n \in \mathbb{N}$, then $\mathbb{N}^\infty \models \forall x \, t_1(x, \overline{a}) = t_2(x, \overline{a})$.*

Proof. 1. If x does not occur in t_1 nor in t_2, then $\mathbb{N}^\infty \models t_1(\overline{a}) = t_2(\overline{a}) \leftrightarrow \forall x \, t_1(\overline{a}) = t_2(\overline{a})$. 2. If x occurs in exactly one of the t_i, say in t_1, then our atom has the form $t_1(x, \overline{a}) = t_2(\overline{a})$ and we make a further case distinction. 2a. If there is an $i \in \{1, \ldots, k\}$ s.t. z_i occurs in t_1 and $a_i = \infty$, then $\mathbb{N}^\infty \models t_1(b, \overline{a}) = \infty$ for all $b \in \mathbb{N} \cup \{\infty\}$. So $\mathbb{N}^\infty \models t_1(0, \overline{a}) = t_2(\overline{a}) \rightarrow t_1(\infty, \overline{a}) = t_2(\overline{a})$. 2b. If there is no such i then, by Lemma 3.5, $n \mapsto t_1^{\mathbb{N}^\infty}(n, \overline{a})$ is a polynomial almost everywhere in the standard part of $\mathbb{N}^\infty$. Therefore it is not equal to the constant function $n \mapsto t_2(\overline{a})$. 3. If x occurs on both sides, then $\mathbb{N}^\infty \models t_1(\infty, \overline{a}) = \infty = t_2(\infty, \overline{a})$ and we are done. □

Proof of Theorem 3.4. We have $\mathbb{N}^\infty \models s(x) \neq 0$ because 0 is not a successor in $\mathbb{N}^\infty$. We have $\mathbb{N}^\infty \models p(0) = 0$ by definition. We have $\mathbb{N}^\infty \models p(s(a)) = a$ for standard a as in the standard model. If $a = \infty$, then $\mathbb{N}^\infty \models p(s(a)) = p(a) = a$. We have $\mathbb{N}^\infty \models a + 0 = a$ for standard a as in the standard model. If $a = \infty$, then $\mathbb{N}^\infty \models a + 0 = a$ by definition. We have $\mathbb{N}^\infty \models a + s(b) = s(a + b)$ if both, a and b are standard as in the standard model. If one of a or b is ∞, then $\mathbb{N}^\infty \models a + s(b) = \infty = s(a + b)$.

For showing that $\mathbb{N}^\infty \models \mathrm{IAtom}_{L_{\mathrm{LA}}}$, let $\overline{z} = z_1, \ldots, z_k$, let $t_1(x, \overline{z}) = t_2(x, \overline{z})$ be an atom, let $\overline{a} \in (\mathbb{N} \cup \{\infty\})^k$, and assume

1. $\mathbb{N}^\infty \models t_1(0, \overline{a}) = t_2(0, \overline{a})$ and
2. $\mathbb{N}^\infty \models \forall x \, (t_1(x, \overline{a}) = t_2(x, \overline{a}) \rightarrow t_1(s(x), \overline{a}) = t_2(s(x), \overline{a}))$.

Then $\mathbb{N}^\infty \models t_1(\underline{n}, \overline{a}) = t_2(\underline{n}, \overline{a})$ for all $n \in \mathbb{N}$. Therefore, by Lemma 3.6, $\mathbb{N}^\infty \models \forall x \, t_1(x, \overline{a}) = t_2(x, \overline{a})$ and we are done. □

There are a number of simple properties of $\mathbb{N}$ which are not true in $\mathbb{N}^\infty$.

Theorem 3.7. *For all $k \geq 1$: $\mathbb{N}^\infty \not\models \forall x \, x \neq s^k(x)$. Moreover, $\mathbb{N}^\infty \not\models \forall x \forall y \forall z \, (x + y = x + z \rightarrow y = z)$ and $\mathbb{N}^\infty \not\models \forall x \forall y \forall z \, (x + z = y + z \rightarrow x = y)$.*

Proof. For acyclicity observe that $\mathbb{N}^\infty \models s^k(\infty) = \infty$. For left cancellation we have $\mathbb{N}^\infty \models 0 + \infty = \infty = 1 + \infty$ but $\mathbb{N}^\infty \not\models 0 = 1$. For right cancellation we have $\mathbb{N}^\infty \models \infty + 0 = \infty = \infty + 1$ but $\mathbb{N}^\infty \not\models 0 = 1$. □

Corollary 3.8. *For all $k \geq 1$: $B + \mathrm{IAtom}_{L_{\mathrm{LA}}} \nvdash \forall x \, x \neq s^k(x)$. $B + \mathrm{IAtom}_{L_{\mathrm{LA}}} \nvdash \forall x \forall y \forall z \, (x + y = x + z \rightarrow y = z)$. $B + \mathrm{IAtom}_{L_{\mathrm{LA}}} \nvdash \forall x \forall y \forall z \, (x + z = y + z \rightarrow x = y)$.*

The formulas $\forall x \, x \neq s^k(x)$ and $\forall x \forall y \forall z \, (x + z = y + z \rightarrow x = y)$ have straightforward induction proofs but, by the above corollary, are not provable by atomic induction (with base theory B). The formula $\forall x \forall y \forall z \, (x + y = x + z \rightarrow y = z)$ does neither have a straightforward induction proof nor one by atomic induction with base theory B.

In order to relate these results to algorithms for automated inductive theorem proving, we formulate the following challenge problems.

Challenge Problem 3.1 (Additive Left Cancellation)**.**

- Language: L_{LA} (see Definition 1.13)
- Axioms: B (see Definition 1.13)

- Goal: $\forall x \forall y \forall z\, (x + y = x + z \to y = z)$

Challenge Problem 3.2 (Additive Right Cancellation)**.**

- Language: L_{LA} (see Definition 1.13)
- Axioms: B (see Definition 1.13)
- Goal: $\forall x \forall y \forall z\, (x + z = y + z \to x = y)$

Challenge Problem 3.3 (Acyclicity)**.**

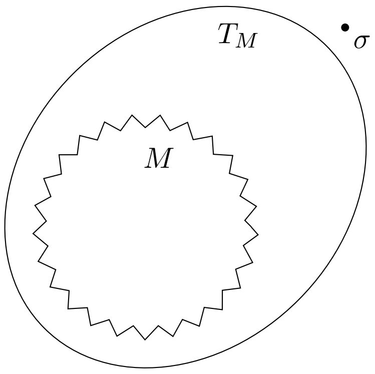
- Language: L_{LA} (see Definition 1.13)
- Axioms: B (see Definition 1.13)
- Goal: $x \neq s^k(x)$ for $k \geq 1$

Corollary 3.9. *Let $k, \overline{x}, N$ and t be arguments for the theory exploration method* Explore *defined in Algorithm 2.2, let $L = \text{Explore}(k, \overline{x}, N, B, t)$. Then $B + L$ does not prove any of the challenge problems 3.1, 3.2, or 3.3.*

3.3 A general methodology

The relationship between Theorem 2.5, Theorem 3.1, Corollary 3.2, Corollary 3.8, and Corollary 3.9 is an instance of a methodology that we will apply repeatedly throughout this course: in order to understand the limits of a practical method M for inductive theorem proving, such as the algorithm Explore, we overapproximate M by a theory T_M, such as atomic induction, possibly via some intermediate proof structure, such as e.i.u.s.i. proofs (as in Corollary 3.2 via Theorems 2.5 and 3.1). In this context, overapproximation means that everything provably in M is provable in T_M (but not necessarily vice versa). The methods for obtaining such a relationship are typically of a proof-theoretical nature. Usually, some form of proof translation from the method M to the theory T_M is carried out.

Such a result allows to understand the power of M in terms of the theory T_M. In particular, it shows that independence results transfer from T_M to M. An *independence result* for a theory T is a result of the form: $T \nvdash \sigma$ and $T \nvdash \neg\sigma$ for some sentence σ. Typically this is formulated for a sentence σ which is true (in a suitable sense). Now, if everything that is provable by M is also provable by T_M, then a (true) sentence σ that is unprovable by T_M is also unprovable by M. This has been exploited for obtaining Corollary 3.9 from Corollary 3.8. In the context of this course, the proof of the indepence result itself (Corollary 3.8) is typically a construction of a countermodel.



Now, in principle, this methodolgy would be applicable in a very generous way by, e.g., using Peano arithmetic for T_M and the consistency of Peano arithmetic for σ. However, such results are not of great interest in computer science since such σ would be considered outside the scope of automation anyway. Therefore we will mostly work with *practically meaningful independence results*, i.e., such σ which would typically be considered to be within the scope of automation. This emphasis entails the need of much tighter translations in the proof-theoretic translation and a significant amount of specific additional work in the construction of the countermodel.

3.4 Simple nonstandard models for inductive data types

In this section we will prove a simple practically meaningful independence result that is applicable to (most) inductive data types. To that aim we first define some properties of inductive data types.

Definition 3.10. Let D be an inductive data type defined on top of the sorts $s_1, \dots, s_n$ with the constructors $c_1, \dots, c_k$. The language of D is $L_D = \{c_1, \dots, c_k\}$. We define the following L_D formulas:

$$\begin{aligned}
\mathrm{DISJ}^D_{i,j} &\equiv \forall \overline{x} \forall \overline{y}\, c_i(\overline{x}) \neq c_j(\overline{y}) \text{ for } 1 \leq i < j \leq k && \text{(disjointness)}\\
\mathrm{INJ}^D_i &\equiv \forall \overline{x} \forall \overline{y}\, (c_i(\overline{x}) = c_i(\overline{y}) \to \overline{x} = \overline{y}) \text{ for } 1 \leq i \leq k && \text{(injectivity)}\\
\mathrm{ACY}^D_t &\equiv \forall x \forall \overline{z}\, x \neq t(x, \overline{z}) \text{ where } t(x,\overline{z}) \text{ contains } x \text{ and is different from } x && \text{(acyclicity)}\\
\mathrm{SUR}^D &\equiv \forall x \exists \overline{y} \bigvee_{i=1}^{k} x = c_i(\overline{y}) && \text{(surjectivity)}
\end{aligned}$$

The basic theory of the inductive data type D is the L_D theory $T_D = \{\mathrm{DISJ}^D_{i,j} \mid 1 \leq i < j \leq k\} \cup \{\mathrm{INJ}^D_i \mid 1 \leq i \leq k\}$.

We also write DISJ^D for $\bigwedge_{1 \leq i < j \leq k} \mathrm{DISJ}^D_{i,j}$, INJ^D for $\bigwedge_{i=1}^{k} \mathrm{INJ}^i_D$, and ACY^D for the set of sentences $\{\mathrm{ACY}^D_t \mid t(x,\overline{z}) \text{ contains } x \text{ and is different from } x\}$.

Definition 3.11. Let $s_1, \dots, s_n$ be sorts and let D be an inductive data type defined on top of $s_1, \dots, s_n$. Then the set $\mathbb{T}^D(M, M_1, \dots, M_n)$ is defined as the smallest set X that satisfies

$$\frac{a_j \in X \cup M \text{ if } \tau_i^j = D \text{ and } a_j \in M_l \text{ if } \tau_i^j = s_l \text{ for all } j \in \{1, \dots, m_i\}}{c_i(a_1, \dots, a_{m_i}) \in X}$$

for all $i \in \{1, \dots, k\}$.

We define the L_D structure $\mathcal{M}'(D, s_1, \dots, s_n)$, abbreviated as $\mathcal{M}'$, as follows. Every parameter sort s_i is interpreted as a singleton set consisting of a new object a_i

$$s_i^{\mathcal{M}'} = \{a_i\}$$

The sort D is the closure of $\{a\}$, where a is a new object, under the constructors.

$$D^{\mathcal{M}'} = \mathbb{T}^D(\{a\}, s_1^{\mathcal{M}'}, \dots, s_n^{\mathcal{M}'})$$

The constructors are interpreted as themselves.

$$c_i^{\mathcal{M}'}(b_1, \dots, b_{m_i}) = c_i(b_1, \dots, b_{m_i})$$

This model adds a new element, a to $D^{\mathcal{M}'}$ that is not obtainable from any constructor. We therefore have:

Lemma 3.12. *Let D be an inductive data type defined on top of the sorts $s_1, \dots, s_n$. Then*

$$\begin{aligned}\mathcal{M}'(D, s_1, \dots, s_n) &\models \mathrm{DISJ}^D\\ \mathcal{M}'(D, s_1, \dots, s_n) &\models \mathrm{INJ}^D\\ \mathcal{M}'(D, s_1, \dots, s_n) &\models \mathrm{ACY}^D\\ \mathcal{M}'(D, s_1, \dots, s_n) &\not\models \mathrm{SUR}^D\end{aligned}$$

Proof. By construction via $\mathbb{T}^D(\{a\}, \{a_1\}, \dots, \{a_n\})$, $\mathcal{M}'(D, s_1, \dots, s_n)$ and the interpretation of the constructors as themselves, $\mathcal{M}'(D, s_1, \dots, s_n)$ satisfies disjointness, injectivity, and acyclicity. For surjectivity, note that $\mathcal{M}'(D, s_1, \dots, s_n) \not\models \exists \overline{y} \bigvee_{i=1}^{k} a = c_i(\overline{y})$. □

We now want to show the following

Theorem 3.13. *Let D be an inductive data type with at least two constructors defined on top of the sorts $s_1, \dots, s_n$. Then $\mathcal{M}'(D, s_1, \dots, s_n) \models T_D + \mathrm{IAtom}_{L_D}$.*

This theorem, together with the observation that $\mathcal{M}'(D, s_1, \dots, s_n) \not\models \mathrm{SUR}^D$, yields a practically meaningful indepence result: the basic theory T_D of the inductive data type D, together with atomic induction, does not prove surjectivity, see Corollary 3.16. In order to prove Theorem 3.13, we need some preparatory lemmas.

Lemma 3.14. *Let D be an inductive data type defined on top of the sorts $s_1, \dots, s_n$, and let $t(x, \overline{z})$ be an L_D term that contains x. Then $\mathrm{INJ}^D \vdash \forall x \forall y \forall \overline{z}(t(x, \overline{z}) = t(y, \overline{z}) \to x = y)$.*

Proof. By induction on the structure of t. □

Lemma 3.15. *Let D be an inductive data type defined on top of the sorts $s_1, \dots, s_n$ and let $A(x, \overline{z})$ be an equation of type D. Then*

$$\mathcal{M}'(D, s_1, \dots, s_n) \models \forall x \forall y \forall \overline{z}\,(x \neq y \land A(x, \overline{z}) \land A(y, \overline{z}) \to \forall x\, A(x, \overline{z})).$$

Proof. Abbreviate $\mathcal{M}'(D, s_1, \dots, s_n)$ as $\mathcal{M}'$, let $A(x, \overline{z}) \equiv t_1(x, \overline{z}) = t_2(x, \overline{z})$. Then it suffices to show $\mathcal{M}' \models t_1(b, \overline{d}) = t_2(b, \overline{d}) \land t_1(c, \overline{d}) = t_2(c, \overline{d}) \to \forall x\, t_1(x, \overline{d}) = t_2(x, \overline{d})$ for all $b, c, \overline{d}$ with $b \neq c$. We make a case distinction. 1. If x does not occur in t_1 nor in t_2, then $\mathcal{M}' \models t_1(\overline{d}) = t_2(\overline{d}) \to \forall x\, t_1(x, \overline{d}) = t_2(x, \overline{d})$. 2. If x occurs in exactly one of the t_i, say in t_1, then $A(x, \overline{z})$ is of the form $t_1(x, \overline{z}) = t_2(\overline{z})$. By Lemma 3.14, $\mathcal{M}' \models t_1(b, \overline{d}) \neq t_1(c, \overline{d})$ and therefore $\mathcal{M}' \not\models t_1(b, \overline{d}) = t_2(\overline{d}) \land t_1(c, \overline{d}) = t_2(\overline{d})$. 3. If x occurs in both t_1 and t_2, we proceed by induction on t_1. The base case is $t_1(x, \overline{z}) = x$. Since $\mathcal{M}' \models \mathrm{ACY}^D_{t_2}$ we have $t_2 \equiv x$ and thus $\mathcal{M}' \models \forall x\, t_1(x, \overline{d}) = t_2(x, \overline{d})$. For the induction step let $t_1(x, \overline{z}) \equiv c_{l_1}(t_{1,1}, \dots, t_{1,m_{l_1}})$. Then $t_2 \equiv x$ is impossible since $\mathcal{M}' \models \mathrm{ACY}^D_{t_1}$. So $t_2 \equiv c_{l_2}(t_{2,1}, \dots, t_{2,m_{l_2}})$. Since $\mathcal{M}' \models \mathrm{DISJ}^D_{l_1,l_2}$ we have $l_1 = l_2$. Since $\mathcal{M}' \models \mathrm{INJ}^D_{l_1}$ we have $\mathcal{M}' \models t_{1,i}(b, \overline{d}) = t_{2,i}(b, \overline{d})$ and $\mathcal{M}' \models t_{1,i}(c, \overline{d}) = t_{2,i}(c, \overline{d})$ for $i = 1, \dots, l_1$. So, by induction hypothesis, $\mathcal{M}' \models \forall x\, t_{1,i}(x, \overline{d}) = t_{2,i}(x, \overline{d})$ for $i = 1, \dots, l_1$. Thus $\mathcal{M}' \models \forall x\, t_1(x, \overline{d}) = t_2(x, \overline{d})$. □

Proof of Theorem 3.13. Write $\mathcal{M}'$ for $\mathcal{M}'(D, s_1, \dots, s_n)$. We have already observed that $\mathcal{M}' \models T_D$ in Lemma 3.12. In order to show that $\mathcal{M}' \models \mathrm{IAtom}$ let $A(x, \overline{z})$ be an equation of type D. Since D contains at least two constructors, there are L_D terms $u(\overline{y})$ and $v(\overline{y})$ that start with constructors and elements $\overline{c}$ of the domains of $\mathcal{M}'$ s.t. $u(\overline{c})^{\mathcal{M}'} \neq v(\overline{c})^{\mathcal{M}'}$. So if $\mathcal{M}' \models \bigwedge_{i=1}^{k} \alpha_{i,k} A(x, \overline{d})$ for some $\overline{d}$, then $\mathcal{M}' \models A(u(\overline{c}), \overline{d})$ and $\mathcal{M}' \models A(v(\overline{c}), \overline{d})$. So, by Lemma 3.15, $\mathcal{M}' \models \forall x\, \varphi(x, \overline{d})$. $\square$

Corollary 3.16. *Let D be an inductive data type with at least two constructors. Then $T_D +$ IAtom $\nvdash$ SUR^D*

Proof. Let D be defined on top of the sorts $s_1, \ldots, s_n$. Then, by Theorem 3.13, $\mathcal{M}'(D, s_1, \ldots, s_n) \models T_D +$ IAtom and, by Lemma 3.12, $\mathcal{M}'(D, s_1, \ldots, s_n) \not\models$ SUR^D. $\square$

So in particular: equational theory exploration does not prove surjectivity in an inductive data type with at least two constructors. We formulate this in terms of a challenge problem as a corollary.

Challenge Problem 3.4 (Surjectivity)**.**

- Language: L_D for D being an inductive data type
- Axioms: T_D, see Definition 3.10
- Goal: SUR^D, see Definition 3.10

Corollary 3.17. *Equational theory exploration does not solve Challenge Problem 3.4 for any inductive data type with at least two constructors.*

Chapter notes

The methodology for analysing methods for inductive theorem proving described in Section 3.3 was pioneered and applied to several methods in [25], see also [12, 13, 14]. Theorem 3.13 was proven in [26]. The simplest nonstandard model $\mathbb{N}^\infty$, as well as more simple nonstandard models for arithmetic and inductive data types can be found in [26].

Chapter 4

Saturation theorem proving with explicit induction axioms

4.1 The induction rule

The integration of induction axioms, or any other type of axioms, into a saturation system $\mathcal{S}$ can, in principle, be achieved in the following way: we simply add a new inference rule without premises that adds the clause normal form of an instance of the axiom scheme to the current clause set, as in:

$$\frac{}{\mathrm{CNF}(\mathrm{SK}^{\exists}(I_x\varphi(x,\overline{z})))}$$

There is a number of problems with such an inference rule, the first is purely formal: the first-order language from which to take $\varphi(x,\overline{z})$ is not specified. There are various ways of fixing this problem. In our context, it is most useful to simply mention part of the context explicitly as in the following definition.

Definition 4.1. The *induction rule for saturation systems* is given by

$$\frac{C_1 \quad \cdots \quad C_n}{\mathrm{CNF}(\mathrm{SK}^{\exists}(I_x\varphi(x,\overline{z})))}\ \text{Induction}$$

where $C_1,\dots,C_n$ are clauses and $\varphi(x,\overline{z})$ is an $\mathcal{L}(C_1,\dots,C_n)$ formula.

Adding this rule to a sound saturation systems results in a system that is sound w.r.t. the standard model in the following sense.

Definition 4.2. The second-order induction axiom is the second order sentence IND_2:

$$\forall X\,(X(0) \land \forall u\,(X(u) \to X(s(u)))) \to \forall v\,X(v))$$

The second-order induction axiom is very powerful. For example, up to isomorphism, the standard model $\mathbb{N}$ is the only model of $Q + \mathrm{IND}_2$.

Theorem 4.3. *Let $\mathcal{S}$ be a sound saturation system, let L be a language that contains 0 and s, let Γ be a set of L sentences, and let σ be an L sentence. If $\mathcal{S}$+*Induction *refutes* $\mathrm{CNF}(\mathrm{SK}^{\exists}(\Gamma+\neg\sigma))$, *then* $\Gamma, \mathrm{IND}_2 \models \sigma$.

Proof Sketch. The proof proceeds, essentially, by a proof translation and deskolemisation. $\square$

This means that every rule that can be simulated by this rule can be added to an inductive theorem prover and will result in a saturation system that is sound w.r.t. the standard model. However, the following problems still remain with this rule:

1. A (serious) practical problem and, in a sense, again, the central problem of automated inductive theorem proving: How does the automated theorem prover choose $\varphi(x, \overline{z})$?
2. Another practical problem (that we will not deal with in this course): When, i.e., at what times in the saturation process does the automated theorem prover choose to add an induction axiom?
3. A (subtle) theoretical problem: this induction rule increases the langauge by adding new Skolem symbols. By iterating this induction rule, this effect proliferates to create yet more Skolem symbols coming form induction axioms with existing Skolem symbols.

4.2 Parameter-free literal induction with generalisation

One way to deal with the first problem is to strongly restrict the set of formulas $\varphi(x, \overline{z})$ that may be introduced as induction formulas. In this section we will consider an induction rule that adds a clausified version of the induction axiom for literals.

Definition 4.4. The rule of *parameter-free literal induction with generalisation* for saturation systems is

$$\frac{\overline{L(a)} \vee C}{\mathrm{CNF}(\mathrm{SK}^{\exists}(I_x L(x)))}\ \mathrm{LIND}_{\lambda}^{-}$$

where a is a constant symbol and $L(x)$ is a literal.

As we will see soon, this rule also considerably simplifies dealing with the third problem.

First note that $L(a)$ is variable-free. The clause $\overline{L(a)} \vee C$ can be interpreted as the implication $L(a) \to C$. The clause normal form of the literal induction axiom for $L(x)$ is computed as:

$$\begin{aligned} I_x L(x) &\equiv L(0) \wedge \forall x\, (L(x) \to L(s(x))) \to \forall x\, L(x) \\ \mathrm{SK}^{\exists}(I_x L(x)) &\equiv L(0) \wedge (L(c) \to L(s(c))) \to \forall x\, L(x) \\ \mathrm{CNF}(\mathrm{SK}^{\exists}(I_x L(x))) &\equiv \{\{\neg L(0), L(c), L(x)\}, \{\neg L(0), \neg L(s(c)), L(x)\}\} \end{aligned}$$

Usually this rule is applied immediately followed by resolutions against the source literal $\overline{L(a)}$ yielding the clauses $\neg L(0) \vee L(c) \vee C$ and $\neg L(0) \vee \neg L(s(c)) \vee C$. This can be interpreted as saying: in order to prove C it suffices to prove $L(0)$ and the implication $L(c) \to L(s(c))$. And indeed, resolving against the clauses $L(0)$ and $\{\neg L(c), L(s(c))\}$ then yields C.

Example 4.5. $\mathrm{LIND}_{\lambda}^{-}$ is powerful enough to solve Challenge Problem 3.3, i.e.,$\mathcal{R} + \mathrm{LIND}_{\lambda}^{-}$ refutes $\mathrm{CNF}(\mathrm{SK}^{\exists}(B + \neg\forall x\, x \neq s^k(x)))$ for all $k \geq 1$. This is unsolvable by atomic induction. We start the saturation process with the clause set

$$\begin{aligned} \mathcal{C}_0 &= \mathrm{CNF}(\mathrm{SK}^{\exists}(B)) \cup \mathrm{CNF}(\mathrm{SK}^{\exists}(\neg\forall x\, x \neq s^k(x))) \\ &= \{\{s(x) \neq 0\}, \{p(0) = 0\}, \{p(s(x)) = x\}, \{x + 0 = x\}, \{x + s(y) = s(x + y)\}\} \cup \{\{a = s^k(a)\}\}. \end{aligned}$$

for the new Skolem constant $a = s(\forall x\, x \neq s^k(x))$. The application

$$\frac{\overline{a \neq s^k(a)}}{\mathrm{CNF}(\mathrm{SK}^{\exists}(I_x x \neq s^k(x)))}\ \mathrm{LIND}_{\lambda}^{-}$$

of LIND_λ^-, followed by resolution against the source literal $a = s^k(a)$, yields

$$\mathcal{C}_1 = \mathcal{C}_0 \cup \{\{0 = s^k(0), c \neq s^k(c)\}, \{0 = s^k(0), s(c) = s^{k+1}(c)\}\}.$$

Since $k \geq 1$, resolution with $s(x) \neq 0$ yields

$$\mathcal{C}_2 = \mathcal{C}_1 \cup \{\{c \neq s^k(c)\}, \{s(c) = s^{k+1}(c)\}\},$$

from which we can obtain the empty clause by:

$$\dfrac{\dfrac{}{p(s(c)) = p(s(c))}\;\text{Ref} \qquad \dfrac{s(c) = s^{k+1}(c) \qquad \dfrac{p(s(x)) = x \qquad \dfrac{p(s(x)) = x \quad c \neq s^k(c)}{p(s(c)) \neq s^k(c)}\;\text{Paramod}_{[x \backslash c]}}{p(s(c)) \neq p(s^{k+1}(c))}\;\text{Paramod}_{[x \backslash s^k(c)]}}{p(s(c)) \neq p(s(c))}\;\text{Paramod}_{\text{id}}}{\emptyset}\;\text{Res}_{\text{id}}$$

Example 4.6. Consider the sentence $\sigma \equiv \forall x\, x + (x + x) = (x + x) + x$. It is an instance of both, associativity and commutativity and thus $B + \mathrm{IAtom} \vdash \sigma$. However, if we try to prove σ by a straightforward induction proof in B, the proof attempt gets stuck at the induction step:

$$s(x) + (s(x) + s(x)) =^{(A5)} s(x) + s(s(x) + x) = \cdots = (s(x) + s(x)) + s(x).$$

The generalisation feature of the LIND_λ^- rule allows to prove σ from B without finding a suitable lemma, such as associativity or commutativity, explicitly (as theory exploration would do). Let $\mathcal{S}$ be a sound and refutationally complete saturation system. We start with

$$\begin{aligned}\mathcal{C}_0 &= \mathrm{CNF}(\mathrm{SK}^\exists(B)) \cup \mathrm{CNF}(\mathrm{SK}^\exists(\forall x\, x + (x + x) = (x + x) + x)) \\ &= \mathrm{CNF}(\mathrm{SK}^\exists(B)) \cup \{\{a + (a + a) \neq (a + a) + a\}\}\end{aligned}$$

There are $2^6 = 64$ possibilities for generalising the literal $L(a) \equiv a + (a + a) = (a + a) + a$ to a literal $L(x)$. The rule application

$$\dfrac{a + (a + a) \neq (a + a) + a}{\mathrm{CNF}(\mathrm{SK}^\exists(I_x a + (a + x) = (a + a) + x))}\;\mathrm{LIND}_\lambda^-$$

followed by resolution against the source literal yields

$$\begin{aligned}\mathcal{C}_1 = \mathcal{C}_0 \cup \{&\{a + (a + 0) \neq (a + a) + 0, a + (a + c) = (a + a) + c\}, \\ &\{a + (a + 0) \neq (a + a) + 0, a + (a + s(c)) \neq (a + a) + s(c)\}\}.\end{aligned}$$

Then we have

$$\mathcal{C}_1 \models a + (a + 0) =^{(A4)} a + a =^{(A4)} (a + a) + 0$$

and thus

$$\begin{aligned}&\mathcal{C}_1 \models a + (a + c) = (a + a) + c \text{ and} \\ &\mathcal{C}_1 \models a + (a + s(c)) \neq (a + a) + s(c).\end{aligned} \tag{IH}$$

Moreover, we have

$$\mathcal{C}_1 \models a + (a + s(c)) =^{(A5)} a + s(a + c) =^{(A5)} s(a + (a + c)) =^{(\mathrm{IH})} s((a + a) + c) =^{(A5)} (a + a) + s(c)$$

and thus $\mathcal{C}_1 \models \bot$. A refutationally complete saturation system will hence prove $\emptyset$ from $\mathcal{C}_1$.

4.3 Analysis of LIND_λ^-

We now set out to analyse the power of the LIND_λ^- rule. To this aim we introduce the ground induction rule and observe that every instance of LIND_λ^- is also an instance of ground induction.

Definition 4.7. For a set of formulas Φ we define the *ground induction rule* Φ-GIND by

$$\frac{C_1 \quad \cdots \quad C_n}{\mathrm{CNF}(\mathrm{SK}^\exists(I_x\varphi(x,\bar{t})))}\ \Phi\text{-GIND}$$

where $\varphi(x,\bar{z}) \in \Phi$ and $\bar{t}$ is a vector of ground $\mathcal{L}(C_1,\dots,C_n)$ terms.

Lemma 4.8. *For any saturation system $\mathcal{S}$ and any clause set $\mathcal{C}$: if $\mathcal{S} + \mathrm{LIND}_\lambda^-$ refutes $\mathcal{C}$, then $\mathcal{S} + \mathrm{Literal}(\mathcal{L}(\mathcal{C}))$-GIND refutes $\mathcal{C}$.*

Proof. If a clause D occurs in an $\mathcal{S} + \mathrm{LIND}_\lambda^-$ derivation, then $\mathcal{L}(D) \subseteq \mathcal{L}(\mathcal{C}) \cup L'$ where L' is a set of constants, the Skolem constants introduced by LIND_λ^--inferences. Therefore, given

$$\frac{\overline{L(a)} \vee C}{\mathrm{CNF}(\mathrm{SK}^\exists(I_xL(x)))}\ \mathrm{LIND}_\lambda^-$$

there is an $\mathcal{L}(\mathcal{C})$ literal $L'(x,\overline{w})$ and constants $\overline{c} \in L'$ s.t. $L(x) \equiv L'(x,\overline{c})$, so $I_xL(x) \equiv I_xL'(x,\overline{c})$ and we can write this inference as

$$\frac{\overline{L'(a,\overline{c})} \vee C}{\mathrm{CNF}(\mathrm{SK}^\exists(I_xL'(x,\overline{c})))}\ \mathrm{Literal}(\mathcal{L}(\mathcal{C})))$$

□

This lemma allows to replace the cascade of growing languages of LIND_λ^- applications by induction in one global language that is known from the very beginning.

Lemma 4.9. *Let $\mathcal{S}$ be a sound saturation system, let $\mathcal{C}$ be a clause set, and let Φ be a set of formulas. If $\mathcal{S} + \Phi$-GIND refutes $\mathcal{C}$, then the theory*

$$\mathrm{sk}^\omega(\mathcal{L}(\mathcal{C}) \cup \mathcal{L}(\Phi) \cup \{0,s\})\text{-SA} + \mathcal{C} + \Phi\text{-IND}$$

is inconsistent.

Proof. Let $L' = \mathrm{sk}^\omega(\mathcal{L}(\mathcal{C}) \cup \mathcal{L}(\Phi) \cup \{0,s\})$, and let $\mathcal{C} = \mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_n$ be an $\mathcal{S}+\Phi$-GIND deduction. We show that $\mathcal{L}(\mathcal{C}_n) \subseteq L'$ and L'-SA $+ \mathcal{C} + \Phi$-IND $\models \mathcal{C}_n$ by induction on n. For the base case we have $\mathcal{C} \models \mathcal{C}_0$. For the induction step we assume $\mathcal{L}(\mathcal{C}_n) \subseteq L'$ and L'-SA $+ \mathcal{C} + \Phi$-IND $\models \mathcal{C}_n$. If $\mathcal{C}_{n+1}$ is obtained by $\mathcal{S}$, then the claim follows from the soundness of $\mathcal{S}$. If $\mathcal{C}_{n+1}$ is obtained by Φ-GIND, then $\mathcal{C}_{n+1} = \mathcal{C}_n \cup \mathrm{CNF}(\mathrm{SK}^\exists(I_x\varphi(x,\bar{t})))$ where $\varphi(x,\bar{z}) \in \Phi$ and $\bar{t}$ is a vector of ground $\mathcal{L}(\mathcal{C}_n)$ terms. Since Φ-IND $\vdash \forall\bar{z}\, I_x\varphi(x,\bar{z})$, we also have Φ-IND $\vdash I_x\varphi(x,\bar{t})$. By the induction hypothesis $\mathcal{L}(I_x\varphi(x,\bar{t})) \subseteq L'$, so, by Lemma 0.17, L'-SA $+ \Phi$-IND $\vdash \mathrm{SK}^\exists(I_x\varphi(x,\bar{t}))$, so L'-SA$+\Phi$-IND $\vdash \mathrm{CNF}(\mathrm{SK}^\exists(I_x\varphi(x,\bar{t})))$ and thus L'-SA$+\mathcal{C}+\Phi$-IND $\models \mathcal{C}_{n+1}$ and $\mathcal{L}(\mathcal{C}_{n+1}) \subseteq L'$. □

Theorem 4.10. *Let $\mathcal{S}$ be a sound saturation system and let $\mathcal{C}$ be a clause set. If $\mathcal{S} + \mathrm{LIND}_\lambda^-$ refutes $\mathcal{C}$, then $\mathcal{C} + \mathrm{ILiteral}_{\mathcal{L}(\mathcal{C})}$ is inconsistent.*

Proof. Since $\mathcal{S} + \mathrm{LIND}^-_\lambda$ refutes $\mathcal{C}$, by Lemma 4.8, also $\mathcal{S} + \mathrm{Literal}(\mathcal{L}(\mathcal{C}))$-GIND refutes $\mathcal{C}$. So, by Lemma 4.9,

$$\mathrm{sk}^\omega(\mathcal{L}(\mathcal{C}) \cup \{0, s\})\text{-SA} + \mathcal{C} + \mathrm{Literal}(\mathcal{L}(\mathcal{C}))\text{-IND}$$

is inconsistent. So, by Theorem 0.18/1, also

$$\mathcal{C} + \mathrm{ILiteral}_{\mathcal{L}(\mathcal{C})}$$

is inconsistent. □

This result is another example for an embedding of a method for automated inductive theorem proving into a theory in the sense of mathematical logic, cf. Section 3.3. In this case we see that the growing languages of Skolem symbols in $\mathcal{S} + \mathrm{LIND}^-_\lambda$ refutations are absorbed by the parameters of the ILiteral induction axioms. We will later use Theorem 4.10 to show that the following Challenge Problem 4.1 is not provable using LIND^-_λ by showing that it is not provable by literal induction.

Challenge Problem 4.1 (Every number is even or odd)**.**

- Language: $\{0/0, s/1, E/1, O/1\}$
- Axioms: $\{E(0), \forall x\,(E(x) \to O(s(x))), \forall x\,(O(x) \to E(s(x)))\}$
- Goal: $\forall x\,(E(x) \lor O(x))$

4.4 Multi-clause induction

Definition 4.11. Let $\varphi(\overline{x}, \overline{z})$ be a formula. Then the *diagonal induction axiom* $I^{\mathrm{Diag}}_{\overline{x}}\varphi(\overline{x}, \overline{z})$ is

$$\left(\bigwedge_{i=1}^{|\overline{x}|} \forall \overline{x}_{<i} \forall \overline{x}_{>i}\, \varphi(\overline{x}_{<i}, 0, \overline{x}_{>i}, \overline{z}) \land \forall \overline{x}\,(\varphi(\overline{x}, \overline{z}) \to \varphi(s(\overline{x}), \overline{z})\right) \to \forall \overline{x}\, \varphi(\overline{x}, \overline{z})$$

Let $m \in \mathbb{N}$ and let Φ be a set of formulas. Then Φ-$\mathrm{Diag}_m\mathrm{IND}$ is the set of universal closures of the formulas $I^{\mathrm{Diag}}_{\overline{x}}\varphi$ where $\overline{x}$ is a tuple of m variables and $\varphi \in \Phi$. Moreover, we define Φ-$\mathrm{Diag}_\omega\mathrm{IND} = \bigcup_{m\in\mathbb{N}} \Phi$-$\mathrm{Diag}_m\mathrm{IND}$.

Definition 4.12. The *multi-clause induction rule* for saturation systems is

$$\frac{\overline{L_1(\overline{t})} \lor C_1 \quad \cdots \quad \overline{L_n(\overline{t})} \lor C_n}{\mathrm{CNF}(\mathrm{SK}^\exists(I^{\mathrm{Diag}}_{\overline{x}} \bigvee_{i=1}^n L_i(\overline{x})))}\ \mathrm{MCIND}^-_\lambda$$

where $L_1(\overline{x}), \ldots, L_n(\overline{x})$ are literals, $\overline{t}$ is a tuple of ground terms, and $C_1, \ldots, C_n$ are clauses.

Example 4.13. Let $\mathcal{S}$ be a sound and refutationally complete saturation system. Then $\mathcal{S} + \mathrm{MCIND}^-_\lambda$ solves Challenge Problem 4.1. Challenge Problem 4.1, written as a clause set, is

$$\mathcal{C} = \{\{E(0)\}, \{\neg E(x), O(s(x))\}, \{\neg O(x), E(s(x))\}, \{\neg E(a)\}, \{\neg O(a)\}\}$$

where $a = s(\forall x\,(E(x) \lor O(x)))$ The application

$$\frac{\overline{E(a)} \qquad\qquad \overline{O(a)}}{\mathrm{CNF}(\mathrm{SK}^\exists(I^{\mathrm{Diag}}_x E(x) \lor O(x)))}\ \mathrm{MCIND}^-_\lambda$$

of MCIND_λ^- gives $\mathcal{C}_1 = \mathcal{C} \cup \mathrm{CNF}(\mathrm{SK}^\exists(I_x^{\mathrm{Diag}} E(x) \vee O(x)))$. Now observe that Challenge Problem 4.1 has a straightforward induction proof, i.e.,

$$E(0), \forall x\,(E(x) \to O(s(x))), \forall x\,(O(x) \to E(s(x))), I_x(E(x) \vee O(x)) \vdash \forall x\,(E(x) \vee O(x)).$$

So, by Skolemisation,

$$E(0), \forall x\,(E(x) \to O(s(x))), \forall x\,(O(x) \to E(s(x))), \mathrm{SK}^\exists(I_x(E(x) \vee O(x))) \vdash E(a) \vee O(a).$$

and, since CNF preserves logical equivalence and $I_x(E(x) \vee O(x)) \equiv I_x^{\mathrm{Diag}}(E(x) \vee O(x))$, we have

$$E(0), \forall x\,(E(x) \to O(s(x))), \forall x\,(O(x) \to E(s(x))), \mathrm{CNF}(\mathrm{SK}^\exists(I_x^{\mathrm{Diag}}(E(x) \vee O(x)))), \neg E(a), \neg O(a) \vdash \bot.$$

which, when expressed as a clause set, is exactly $\mathcal{C}_1$.

4.5 Analysis of multi-clause induction

Lemma 4.14. *Let $\mathcal{S}$ be a sound saturation system, let $\mathcal{C}$ be a clause set. If $\mathcal{S} + \mathrm{MCIND}_\lambda^-$ refutes $\mathcal{C}$, then the theory*

$$\mathrm{sk}^\omega(\mathcal{L}(\mathcal{C}) \cup \{0, s\})\text{-SA} + \mathcal{C} + \mathrm{Clause}(\mathcal{L}(\mathcal{C}))\text{-Diag}_\omega\mathrm{IND}$$

is inconsistent.

Proof. Let $L' = \mathrm{sk}^\omega(\mathcal{L}(\mathcal{C}) \cup \{0, s\})$ and let $\mathcal{C} = \mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_n$ be an $\mathcal{S} + \mathrm{MCIND}_\lambda^-$ deduction. We show that there is a set L_n of constant symbols s.t. $\mathcal{L}(\mathcal{C}_n) = \mathcal{L}(\mathcal{C}) \cup L_n \subseteq L'$ and L'-SA $+ \mathcal{C} + \mathrm{Clause}(\mathcal{L}(\mathcal{C}))\text{-Diag}_\omega\mathrm{IND} \models \mathcal{C}_n$ by induction on n. For the base case we have $\mathcal{C} \models \mathcal{C}_0$. For the induction step we assume $\mathcal{L}(\mathcal{C}_n) = \mathcal{L}(\mathcal{C}) \cup L_n \subseteq L'$ and L'-SA $+ \mathcal{C} + \mathrm{Clause}(\mathcal{L}(\mathcal{C}))\text{-Diag}_\omega\mathrm{IND} \models \mathcal{C}_n$. If $\mathcal{C}_{n+1}$ is obtained from $\mathcal{C}_n$ by an inference from $\mathcal{S}$, then the claim follows from the soundness of $\mathcal{S}$. If $\mathcal{C}_{n+1}$ is obtained from $\mathcal{C}_n$ by an application of MCIND_λ^-, then $\mathcal{C}_{n+1} = \mathcal{C}_n \cup \mathrm{CNF}(\mathrm{SK}^\exists(I_{\overline{x}}^{\mathrm{Diag}} \bigvee_{i=1}^n L_i(\overline{x})))$ for some literals $L_1(\overline{x}), \dots, L_n(\overline{x})$ in $\mathcal{L}(\mathcal{C}_n) = \mathcal{L}(\mathcal{C}) \cup L_n$. So there are $\mathcal{L}(\mathcal{C})$ literals $L_1'(\overline{x}, \overline{y}), \dots, L_n'(\overline{x}, \overline{y})$ and constant symbols $\overline{c}$ in L' s.t. $L_i(\overline{x}) \equiv L_i'(\overline{x}, \overline{c})$ for $i = 1, \dots, n$. So we have $\mathrm{Clause}(\mathcal{L}(\mathcal{C}))\text{-Diag}_\omega\mathrm{IND} \vdash I_{\overline{x}}^{\mathrm{Diag}} \bigvee_{i=1}^n L_i'(\overline{x}, \overline{c})$ and, by Lemma 0.17, L'-SA$+$ $\mathrm{Clause}(\mathcal{L}(\mathcal{C}_0))\text{-Diag}_\omega\mathrm{IND} \vdash \mathrm{SK}^\exists(I_{\overline{x}}^{\mathrm{Diag}} \bigvee_{i=1}^n L_i'(\overline{x}, \overline{c}))$. Since $I_{\overline{x}}^{\mathrm{Diag}} \bigvee_{i=1}^n L_i(\overline{x})$ is parameter-free, we have $\mathcal{L}(\mathcal{C}_{n+1}) = \mathcal{L}(\mathcal{C}_n) \cup \{d_1, \dots, d_k\}$ for some new Skolem constants $d_1, \dots, d_k$. Moreover, L'-SA $+ \mathcal{C} + \mathrm{Clause}(\mathcal{L}(\mathcal{C}_0))\text{-Diag}_\omega\mathrm{IND} \models \mathcal{C}_{n+1}$. □

Theorem 4.15. *Let $\mathcal{S}$ be a sound saturation system and let $\mathcal{C}$ be a clause set. If $\mathcal{S} + \mathrm{MCIND}_\lambda^-$ refutes $\mathcal{C}$, then $\mathcal{C} + \mathrm{Clause}(\mathcal{L}(\mathcal{C}))\text{-Diag}_\omega\mathrm{IND}$ is inconsistent.*

Proof. Since $\mathcal{S} + \mathrm{MCIND}_\lambda^-$ refutes $\mathcal{C}$, by Lemma 4.14,

$$\mathrm{sk}^\omega(\mathcal{L}(\mathcal{C}) \cup \{0, s\})\text{-SA} + \mathcal{C} + \mathrm{Clause}(\mathcal{L}(\mathcal{C}))\text{-Diag}_\omega\mathrm{IND}$$

is inconsistent. So, by Theorem 0.18/1,

$$\mathcal{C} + \mathrm{Clause}(\mathcal{L}(\mathcal{C}))\text{-Diag}_\omega\mathrm{IND}$$

is inconsistent. □

Challenge Problem 4.2 (Cross-Conjunction)**.**

- Language: $L_{\mathrm{cc}} = \{0, s, P/1, Q/1\}$
- Axioms: $\Gamma_{\mathrm{cc}} = \{\mathrm{DISJ}^{\mathsf{Nat}}, \mathrm{INJ}^{\mathsf{Nat}}, P(0), Q(0), \forall x\,(P(x) \to Q(s(x))), \forall x\,(Q(x) \to P(s(x)))\}$
- Goal: $\forall x\, P(x)$

It is easy to see that $\forall x\,(P(x) \wedge Q(x))$ has a straightforward induction proof in Γ_{cc}. An alternative solution of this challenge problem is based on using two-step induction as follows: since $\Gamma_{\mathrm{cc}} \vdash P(s(0))$ and $\Gamma_{\mathrm{cc}} \vdash \forall x\,(P(x) \to P(s(s(x))))$, we have

$$\Gamma_{\mathrm{cc}}, P(0) \wedge P(s(0)) \wedge \forall x\,(P(x) \to P(s(s(x)))) \to \forall x\, P(x) \vdash \forall x\, P(x)$$

On the other hand:

Theorem 4.16. $\Gamma_{\mathrm{cc}} + \mathrm{Clause}(L_{\mathrm{cc}})\text{-}\mathrm{Diag}_\omega\mathrm{IND} \nvdash \forall x\, P(x)$.

Without Proof. □

Corollary 4.17. *Let $\mathcal{S}$ be a sound saturation system. Then $\mathcal{S} + \mathrm{MCIND}_\lambda^-$ does not solve Challenge Problem 4.2 (Cross-Conjunction).*

Chapter notes

The approach of adding (clausifications of) induction axioms to a resolution-based theorem prover dates back to, at least, [6]. The rule for parameter-free literal induction with generalisation has been described in [21, 9]. The rule for multi-clause induction has been introduced in [8, 10]. Much of this chapter is based on Chapters 3 and 4 of [25]. A proof of Theorem 4.3 can be found in [25, Corollary 3.1.11]. The proof of Theorem 4.10 follows that of [25, Corollary 4.1.6]. The proof of Theorem 4.15 follows that of [25, Corollary 4.2.8]. A proof of Theorem 4.16 can be found in [25, Proposition 4.2.15].

Chapter 5

Literal induction

There are a number of problems that are solvable by literal induction, but not by atomic induction.

Proposition 5.1.

1. *$B + \mathrm{ILiteral} \vdash \forall x\, x \neq s^k(x)$ (Acyclicity)*
2. *$B + \mathrm{ILiteral} \vdash \forall x \forall y \forall z\, (x + y = x + z \to y = z)$ (Additive Left Cancellation)*
3. *$B + \mathrm{ILiteral} \vdash \forall x \forall y \forall z\, (x + z = y + z \to x = y)$ (Additive Right Cancellation)*
4. *$\mathrm{ILiteral} \vdash \mathrm{SUR}^D$ for an inductive data type D*

Proof. For 1, note that $\forall x\, x \neq s^k(x)$ has a straightforward induction proof in B, cf. exercise sheet 3, exercise 2. For 4 cf. exercise sheet 6, exercise 5. Since $B + \mathrm{ILiteral}$ proves the commutativity of $+$, 2 and 3 are equivalent. For 3 first note that (*) $B \vdash s(u) = s(v) \to u = v$. Work in $B + \mathrm{ILiteral}$. Assume $x \neq y$ and consider the literal $L(x, y, z) \equiv x + z \neq y + z$. We have $L(x, y, 0)$ by $x + 0 =^{(A4)} x \neq y =^{(A4)} y + 0$. For the induction step assume $L(x, y, z)$. Then $x + s(z) =^{(A5)} s(x + z) \neq^{(*),\mathrm{IH}} s(y + z) =^{(A5)} y + s(z)$, i.e., $L(x, y, s(z))$. □

5.1 Unsolvability of the even/odd challenge problem

In this section we will show that literal induction does not solve Challenge Problem 4.1 (Even/Odd). For the rest of this section fix the language $L_{\mathrm{EO}} = \{0, s, E, O\}$ and define the set of sentences $\Gamma_{\mathrm{EO}} = \{E(0), \forall x\, (E(x) \to O(s(x))), \forall x\, (O(x) \to E(s(x)))\}$.

Definition 5.2. We define an L_{EO}-structure $\mathcal{M}_1$ as follows. The domain of $\mathcal{M}_1$ is $(\{0\} \times \mathbb{N}) \cup (\{1\} \times \mathbb{Z})$. The language is interpreted as follows:

$$\begin{aligned} 0^{\mathcal{M}_1} &= (0, 0) \\ s^{\mathcal{M}_1}(k, n) &= (k, n + 1) \\ E^{\mathcal{M}_1} &= \{(0, n) \mid n \text{ is even}\} \\ O^{\mathcal{M}_1} &= \{(0, n) \mid n \text{ is odd}\} \end{aligned}$$

For an inductive data type we have already defined the theory $T_D = \mathrm{DISJ}^D + \mathrm{INJ}^D$. We define $T_D^* = T_D + \mathrm{ACY}^D + \mathrm{SUR}^D$. We start with a simple observations concerning the structure $\mathcal{M}_1$.

Lemma 5.3. $\mathcal{M}_1 \models T^*_{\mathsf{Nat}} + \Gamma_{\mathrm{EO}}$

Proof Sketch. By a straightforward verification. $\square$

A useful result from the literature is the following:

Theorem 5.4. *T^*_{Nat} is complete.*

Proof Sketch. Completeness of T^*_{Nat} is essentially shown by quantifier-elimination. $\square$

Corollary 5.5. $T^*_{\mathsf{Nat}} \models \{0, s\}$-IND.

Proof. Since T^*_{Nat} is complete and true in the standard model we have $\mathrm{Th}(\mathbb{N}) = T^*_{\mathsf{Nat}}$ and hence $T^*_{\mathsf{Nat}} \models \{0, s\}$-IND. $\square$

Lemma 5.6. *Let $L(x)$ be a non-equational L_{EO} literal containing x. Then $\mathcal{M}_1 \not\models L(0)$ or $\mathcal{M}_1 \not\models \forall x\,(L(x) \rightarrow L(s(x)))$.*

Proof. Suppose that $\mathcal{M}_1 \models L(0)$ and $\mathcal{M}_1 \models \forall x\,(L(x) \rightarrow L(s(x)))$. Then $\mathcal{M}_1 \models L(\underline{n})$ for all $n \in \mathbb{N}$. If $L(x)$ is of the form $E(s^k(x))$ for some $k \in \mathbb{N}$, then $\mathcal{M}_1 \models E((0,k))$ and $\mathcal{M}_1 \models E((0,k+1))$ which contradicts the definition of $\mathcal{M}_1$. If $L(x)$ is of the form $\neg E(s^k(x))$ for some $k \in \mathbb{N}$, then $\mathcal{M}_1 \models \neg E((0,k))$ and $\mathcal{M}_1 \models \neg E((0,k+1))$ which contradicts the definition of $\mathcal{M}_1$. The cases $O(s^k(x))$ and $\neg O(s^k(x))$ are analogous. $\square$

Lemma 5.7. $\mathcal{M}_1 \models \mathrm{Literal}(L_{\mathrm{EO}})$-IND.

Proof. Let $L(x, \overline{z})$ be a literal. If $L(x, \overline{z})$ is an equation, then, by Lemma 5.3 and Corollary 5.5, we have $\mathcal{M}_1 \models \forall \overline{z}\, I_x L(x, \overline{z})$. If $L(x, \overline{z})$ does not contain x, then $\forall \overline{z}\, I_x L(\overline{z})$ is valid. If $L(x, \overline{z})$ contains x and is not an equation, we are done by Lemma 5.6. $\square$

Theorem 5.8. $\Gamma_{\mathrm{EO}} + \mathrm{Literal}(L_{\mathrm{EO}})\text{-IND} \not\models \forall x\,(E(x) \vee O(x))$.

Proof. $\mathcal{M}_1 \models \Gamma_{\mathrm{EO}}$ by Lemma 5.3 and $\mathcal{M}_1 \models \mathrm{Literal}(L)$-IND by Lemma 5.7. However, $\mathcal{M}_1 \not\models \forall x\,(E(x) \vee O(x))$ since every nonstandard number is neither even nor odd. $\square$

Corollary 5.9. *Let $\mathcal{S}$ be a sound refutation system. Then $\mathcal{S} + \mathrm{LIND}^-_\lambda$ does not solve Challenge Problem 4.1 (Even/Odd).*

Chapter 6

Clause set cycles

In this chapter we will study clause set cycles, a form of reasoning by infinite descent.

6.1 Definition and variants

Definition 6.1. Let L be a first-order language and let η be a new constant symbol. A finite $L \cup \{\eta\}$ clause set $\mathcal{C}(\eta)$ is called L *clause set cycle* it it satisfies

$$\mathcal{C}(s(\eta)) \models \mathcal{C}(\eta) \text{ and}$$
$$\mathcal{C}(0) \models \bot.$$

An $L \cup \{\eta\}$ clause set $\mathcal{D}(\eta)$ is *refuted by an L clause set cycle* $\mathcal{C}(\eta)$ if

$$\mathcal{D}(\eta) \models \mathcal{C}(\eta).$$

Example 6.2. Clause set cycles solve Challenge Problem 4.1 (Even/Odd). Let

$$\mathcal{C}(\eta) = \{\{E(0)\}, \{\neg E(x), O(s(x))\}, \{\neg O(x), E(s(x))\}, \{\neg E(\eta)\}, \{\neg O(\eta)\}\}$$

Then $\mathcal{C}(0) \models E(0) \land \neg E(0)$, so $\mathcal{C}(0) \models \bot$. For the descent step let $\mathcal{M} \models \mathcal{C}(s(\eta))$. Then $\mathcal{M} \models \neg E(s(\eta))$ and $\mathcal{M} \models \neg O(s(\eta))$, so, by one resolution step each, we have $\mathcal{M} \models \neg O(\eta)$ and $\mathcal{M} \models \neg E(\eta)$. Thus $\mathcal{M} \models \mathcal{C}(\eta)$. Therefore $\mathcal{C}(\eta)$ is refuted by the clause set cycle $\mathcal{C}(\eta)$.

The notion of refutation by a clause set cycle can be made considerably more flexible by allowing parameters for the external and internal offset and the descent step size. All of these notions coincide.

6.2 Logical characterisation

Definition 6.3. Let Φ be a set of formulas. Then the *parameter-free induction rule with the η-restriction* is

$$\frac{\varphi(0) \quad \forall x\,(\varphi(x) \to \varphi(s(x)))}{\varphi(\eta)}\ \Phi\text{-IND}_{\eta}^{\mathrm{R}-}$$

where $\varphi(x) \in \Phi$.

Definition 6.4. For a theory T and a rule R we define $[T, R] = T + \{\varphi \mid T \vdash \Gamma, \Gamma/\varphi \in R\}$.

Theorem 6.5. *Let L be a language not containing η, let $\mathcal{D}(\eta)$ be an $L \cup \{\eta\}$ clause set. Then $\mathcal{D}$ is refuted by an L clause set cycle iff $[\emptyset, \exists_1(L)\text{-IND}_\eta^{\text{R}-}] \cup \mathcal{D}(\eta)$ is inconsistent.*

There are a number "cheap" ways to exploit weaknesses of clause set cycles, e.g., by the eta restriction, by rule vs. axiom. In the next section we develop an independence result based on parameter-freeness of $\exists_1(L)\text{-IND}_\eta^{\text{R}-}$.

6.3 A practically relevant independence result

Definition 6.6. For $k, n, m \in \mathbb{N}$ with $0 < n < m$ we define the L_{LA} formula

$$n \cdot x + \underline{(m-n)k} = m \cdot x \to x = \underline{k} \qquad (E_{k,n,m}(x))$$

An superficial intuition on how one might go about proving $E_{k,n,m}(x)$ is the following: bring $n \cdot x$ to the right-hand side to obtain "$(m-n) \cdot k = (m-n) \cdot x$" and then divide by $m - n$. This proof sketch generously disregards the difference between variables and numerals, as well as the fact that there is no multiplication in L_{LA} and hence no multiplicative cancellation. A simple instance is $E_{0,1,2}(x) \equiv x + 0 = x + x \to x = 0$.

The main result is the following theorem:

Theorem 6.7. *Let $k, n, m \in \mathbb{N}$ with $0 < n < m$. Write A for the associativity of $+$ and C for the commutativity of $+$. Then $B + A + C + \exists_1(L_{\text{LA}})\text{-IND}^- \nvdash \forall x\, E_{k,n,m}(x)$.*

Proof Sketch. The proof is rather lengthy. The model $\mathcal{M}$ which plays a central role is defined as follows: the domain of $\mathcal{M}$ is $\{(i, n) \in \mathbb{N} \times \mathbb{Z} \mid i = 0 \text{ implies } n \in \mathbb{N}\}$. The language is interpreted as:

$$\begin{aligned} 0^{\mathcal{M}} &= (0,0) & p^{\mathcal{M}}((0,n)) &= (0, n \dot{-} 1) \\ s^{\mathcal{M}}(i,n) &= (i, n+1) & p^{\mathcal{M}}((i,n)) &= (i, n-1) \text{ if } i > 0 \\ (i,n) +^{\mathcal{M}} (j,m) &= (\max(i,j), n+m) \end{aligned}$$

Then $\mathcal{M} \models A + B + C$ which is a straightforward verification. Moreover, $\mathcal{M} \models \exists_1(L_{\text{LA}})\text{-IND}^-$ which is the difficult part. And finally $\mathcal{M} \not\models E_{k,n,m}$ because we have

$$n \cdot (1,k) +^{\mathcal{M}} (\underline{(m-n)k})^{\mathcal{M}} = (1, nk) +^{\mathcal{M}} (0, (m-n)k) = (1, mk) = m \cdot (1,k)$$

but $(1,k) \neq (0,k)$. ☐

On the other hand, open induction (with parameters) suffices in the sense that $B+\text{Open}(L_{\text{LA}})\text{-IND} \vdash \forall x\, E_{k,n,m}(x)$. This example shows how powerful parameters can be in induction axioms.

Challenge Problem 6.1 ($E_{k,n,m}$)**.**

- Language: L_{LA}
- Axioms: $A + B + C$ (see Theorem 6.7)
- Goal: $E_{k,n,m}$ for $k, n, m \in \mathbb{N}$ with $0 < n < m$ (see Definition 6.6)

Corollary 6.8. *Clause set cycles do not solve Challenge Problem 6.1 ($E_{k,n,m}$).*

Proof. Let $k, n, m \in \mathbb{N}$ with $0 < n < m$ and let $\mathcal{E}_{k,n,m}(\eta) = \text{CNF}(A \wedge B \wedge C \wedge \neg E_{k,n,m}(\eta))$. Suppose that $\mathcal{E}_{k,n,m}(\eta)$ is refuted by a clause set cycle. Then, by Theorem 6.5, $[\emptyset, \exists_1(L_{\text{LA}})\text{-IND}_\eta^{\text{R}-} \cup \mathcal{E}_{k,n,m}(\eta)$ is inconsistent. So $A + B + C + \exists_1(L_{\text{LA}})\text{-IND} \vdash \forall x\, E_{k,n,m}(x)$ which contradicts Theorem 6.7. ☐

Chapter notes

The notion of clause set cycle was introduced in [12] to capture methods for automated inductive theorem proving such as the n-clause calculus [19, 18]. Theorem 6.7 has been shown in [13]. Much of this chapter is based on [25, Chapter 6]

Chapter 7

First-order induction

The results we have seen so far may lead to the impression that overcoming a practically relevant independence result is always possible by using a sufficiently strong class of formulas in the language of the original problem as induction formulas. For example, Challenge Problem 3.4 (Surjectivity) is unsolvable by atomic induction, but it is solvable by literal induction (in the same language L_D). Moreover, Challenge Problem 4.1 (Even/Odd) is unsolvable by literal induction, but it is solvable by open induction (in the same language L_{EO}). Furthermore, Challenge Problem 6.1 ($E_{k,n,m}$) is unsolvable by parameter-free $\exists_1$ induction but is solvable by open induction with parameters (in the same language L_{LA}).

The mere existence of statements which are true but not provable by induction on any first-order formula in the language of the statement is not surprising. Indeed, it is a corollary of the second icompleteness theorem that PA $\nvdash \mathsf{Con}_{\mathrm{PA}}$ which is an statement of this form since $\mathsf{Con}_{\mathrm{PA}}$ is an L_{A} sentence and PA is $Q + L_{\mathrm{A}}$-IND. However, $\mathsf{Con}_{\mathrm{PA}}$ is a formal statement that would generally be regarded to be outside the scope of automation anyway. In this chapter we will present such a result for a much simpler statement.

Definition 7.1. We define the first-order language $L_{\mathrm{DH}} = \{0/0, s/1, p/1, d/1, \left\lfloor \frac{\cdot}{2} \right\rfloor /1\}$.

The intended interpretation of the symbols in L_{DH} are, respectively, zero, the successor function, the truncated predecessor function, the function $n \mapsto 2 \cdot n$, and the function $n \mapsto \left\lfloor \frac{n}{2} \right\rfloor$.

Definition 7.2. We define the L_{DH} theory T_{DH} by the following axioms:

$$\begin{aligned}
0 &\neq s(x) \\
p(0) &= 0 \\
p(s(x)) &= x \\
d(0) &= 0 \\
d(s(x)) &= s(s(d(x))) \\
\left\lfloor \frac{0}{2} \right\rfloor &= 0 \\
\left\lfloor \frac{1}{2} \right\rfloor &= 0 \\
\left\lfloor \frac{s(s(x))}{2} \right\rfloor &= s\left(\left\lfloor \frac{x}{2} \right\rfloor\right)
\end{aligned}$$

Challenge Problem 7.1 (Double/Half).

- Language: L_{DH} (see Definition 7.1)
- Axioms: T_{DH} (see Definition 7.2)
- Goal: $\forall x\, x \neq s(d(x))$

The main theorem of this chapter is:

Theorem 7.3. $T_{\mathrm{DH}} + L_{\mathrm{DH}}$-IND $\nvdash \forall x\, x \neq s(d(x))$.

The consequence of this theorem for automated inductive theorem proving is that *any* method that generates only induction formulas in the language of the original problem will fail to solve Challenge Problem 7.1 (Double/Half).

The following structure plays the central role in the proof of Theorem 7.3.

Definition 7.4. We define the L_{DH} structure $\mathcal{M}$ as follows: the domain of $\mathcal{M}$ is $(\{0\} \times \mathbb{N}) \cup (\{1\} \times \mathbb{Z})$. All symbols are defined in the natural way, i.e.,

$$\begin{aligned}
0^{\mathcal{M}} &= (0,0)\\
s^{\mathcal{M}}(i,n) &= (i, n+1)\\
p^{\mathcal{M}}(i,n) &= \begin{cases} (0,0) & \text{if } (i,n) = (0,0)\\ (i, n-1) & \text{otherwise}\end{cases}\\
d^{\mathcal{M}}((i,n)) &= (i, 2\cdot n)\\
\left\lfloor \frac{(i,n)}{2} \right\rfloor^{\mathcal{M}} &= (i, \left\lfloor \frac{n}{2} \right\rfloor)
\end{aligned}$$

It is then straightforward to check that $\mathcal{M} \models T_{\mathrm{DH}}$ and it is easy to observe that $\mathcal{M} \not\models \forall x\, x \neq s(d(x))$ because $\mathcal{M} \models s(d(1,-1)) = s(1,-2) = (1,-1)$. The difficult part is to show that $\mathcal{M} \models L_{\mathrm{DH}}$-IND. This is done based on the following two lemmas.

Lemma 7.5. Th$(\mathcal{M})$ *has quantifier elimination, i.e., for every* L_{DH} *formula* $\varphi(\overline{x})$ *there is a quantifier-free* L_{DH} *formula* $\psi(\overline{x})$ *s.t.* $\mathcal{M} \models \varphi(\overline{x}) \leftrightarrow \psi(\overline{x})$.

Lemma 7.6. $\mathcal{M} \models \mathrm{Open}(L_{\mathrm{DH}})$-IND.

Bibliography

[1] Jeremy Avigad. *Mathematical Logic and Computation.* Cambridge University Press, 2023.

[2] Alan Bundy, David Basin, Dieter Hutter, and Andrew Ireland. *Rippling: Meta-Level Guidance for Mathematical Reasoning.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2005.

[3] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating inductive proofs using theory exploration. In Maria Paola Bonacina, editor, *24th International Conference on Automated Deduction (CADE-24), Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 392–406. Springer, 2013.

[4] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. TIP: tons of inductive problems. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *International Conference on Intelligent Computer Mathematics (CICM) 2015*, volume 9150 of *Lecture Notes in Computer Science*, pages 333–337. Springer, 2015.

[5] Koen Claessen, Nicholas Smallbone, and John Hughes. Quickspec: Guessing formal specifications using testing. In Gordon Fraser and Angelo Gargantini, editors, *4th International Conference on Tests and Proofs (TAP)*, volume 6143 of *Lecture Notes in Computer Science*, pages 6–21. Springer, 2010.

[6] Jared L. Darlington. Automatic theorem proving with equality substitutions and mathematical induction. volume 3 of *Machine Intelligence*, pages 113–127. Edinburgh University Press, 1968.

[7] Gerhard Gentzen. Zusammenfassung von mehreren volländigen Induktionen zu einer einzigen. *Archiv für mathematische Logik und Grundlagenforschung*, 2(1):81–83, 1954.

[8] Márton Hajdú. Automating inductive reasoning with recursive functions. Master's thesis, Technische Universität Wien, 2021.

[9] Márton Hajdú, Petra Hozzová, Laura Kovács, Johannes Schoisswohl, and Andrei Voronkov. Induction with generalization in superposition reasoning. In Christoph Benzmüller and Bruce R. Miller, editors, *13th International Conference on Intelligent Computer Mathematics CICM*, volume 12236 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2020.

[10] Márton Hajdú, Petra Hozzová, Laura Kovács, and Andrei Voronkov. Induction with Recursive Definitions in Superposition. In *Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, 2021*, pages 1–10. IEEE, 2021.

[11] Petr Hájek and Pavel Pudlák. *Metamathematics of First-Order Arithmetic.* Springer, 1993.

[12] Stefan Hetzl and Jannik Vierling. Clause Set Cycles and Induction. *Logical Methods in Computer Science*, 16(4), 2020.

[13] Stefan Hetzl and Jannik Vierling. Unprovability results for clause set cycles. *Theoretical Computer Science*, 935:21–46, 2022.

[14] Stefan Hetzl and Jannik Vierling. Induction and Skolemization in saturation theorem proving. *Annals of Pure and Applied Logic*, 174(1):103167, 2023.

[15] Stefan Hetzl and Tin Lok Wong. Some observations on the logical foundations of inductive theorem proving. *Logical Methods in Computer Science*, 13(4), 2018.

[16] Andrew Ireland and Alan Bundy. Productive Use of Failure in Inductive Proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996.

[17] Richard Kaye. *Models of Peano Arithmetic*, volume 15 of *Oxford Logic Guides*. Clarendon Press, 1991.

[18] Abdelkader Kersani. *Preuves par induction dans le calcul de superposition.* PhD thesis, Université de Grenoble, 2014.

[19] Abdelkader Kersani and Nicolas Peltier. Combining superposition and induction: A practical realization. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *9th International Symposium on Frontiers of Combining Systems (FroCoS)*, volume 8152 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2013.

[20] Anders Lundstedt. Necessarily non-analytic induction proofs – summary of some results. Stockholm University, available at `https://anderslundstedt.com/research/`, 2020.

[21] Giles Reger and Andrei Voronkov. Induction in saturation-based proof search. In Pascal Fontaine, editor, *27th International Conference on Automated Deduction (CADE)*, volume 11716 of *Lecture Notes in Computer Science*, pages 477–494. Springer, 2019.

[22] Joseph R. Shoenfield. Open sentences and the induction axiom. *Journal of Symbolic Logic*, 23(1):7–12, 1958.

[23] Irene Lobo Valbuena and Moa Johansson. Conditional lemma discovery and recursion induction in hipster. *Electronic Communications of the EASST*, 72, 2015.

[24] Dirk van Dalen. *Logic and Structure.* Springer, 2008.

[25] Jannik Vierling. *The limits of automated inductive theorem provers.* PhD thesis, TU Wien, Austria, 2024.

[26] Johannes Weiser. Subsystems of Open Induction. Master's thesis, TU Wien, Austria, 2025. available at `https://www.dmg.tuwien.ac.at/hetzl/teaching/m_weiser.pdf`.