







Synthesis of Recursive Programs in Saturation

Petra Hozzová, Daneshvar Amrollahi, Márton Hajdu,
Laura Kovács, Andrei Voronkov and Eva Maria Wagner

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

February 16, 2024

Synthesis of Recursive Programs in Saturation

Petra Hozzová¹, Daneshvar Amrollahi², Márton Hajdu¹, Laura Kovács¹, Andrei Voronkov^{1,3,4}, and Eva Maria Wagner¹

¹ TU Wien

² Stanford University

³ University of Manchester

⁴ EasyChair

Abstract. We turn saturation-based theorem proving into an automated framework for recursive program synthesis. We introduce magic axioms as valid induction axioms and use them together with answer literals in saturation. We introduce new inferences rules for induction in saturation and use answer literals to synthesize recursive functions from these proof steps. Our proof-of-concept implementation in the VAMPIRE theorem prover constructs recursive functions over algebraic data types, while proving inductive properties over these types.

Keywords: Program Synthesis · Saturation · Superposition · Induction · Recursion · Theorem Proving.

1 Introduction

Program synthesis is the task of constructing a program P satisfying a given specification F , ensuring that P is correct by design [17]. In this paper we work with a functional specification F of the input-output relation of a program P , where F is given as a $\forall\exists$ formula in first-order logic [17,1]. Validity of a specification formula F ensures that for every input value there exists an output value satisfying F , and therefore there is a function which for every input value gives such an output value. Our goal is to *automatically find a (possibly recursive) program that P computes the output, while preserving F .*

As a complementary approach to formal verification, synthesis is inherently more complex [25]. The complexity is further compounded when we consider reasoning about – and synthesizing – programs using recursion. As a remedy, in this paper we advocate for using automated first-order theorem proving as the reasoning back-end to (recursive) program synthesis.

The work [7] extended the saturation-based first-order theorem proving framework to *saturation-based synthesis framework*. The approach (i) uses saturation-based reasoning to prove that a specification F is valid; (ii) tracks the constructive parts of the proof of F ; (iii) and uses them to synthesize a program P satisfying F . In this paper we complement [7] with support for *recursive program* synthesis. We use recent developments on automating induction in saturation [4,8,6] and construct recursive programs based on applications of induction.

$$\begin{array}{ll}
\text{axioms: } \text{half}(0) \simeq 0 & \text{(H1)} \\
\text{half}(s(0)) \simeq 0 & \text{(H2)} \\
\forall x. \text{half}(s(s(x))) \simeq s(\text{half}(x)) & \text{(H3)} \\
\text{specification: } \forall x \exists y. \text{half}(y) \simeq x & \text{(SD)}
\end{array}$$

Fig. 1. Axioms of `half` and the $\forall\exists$ -specification for the function computing double.

Illustrative Example. Consider the specification (SD) of Figure 1, which describes the inverse of the `half` function over natural numbers. Given the axiomatization of `half` in Figure 1, our approach synthesizes the recursive function `double` as a solution of (SD), defined as:

$$\begin{array}{l}
\text{double}(0) \simeq 0 \\
\forall x. \text{double}(s(x)) \simeq s(s(\text{double}(x)))
\end{array} \tag{1}$$

The framework of [7] fails to synthesize a solution of (SD), as `double` is a recursive program. To the best of our knowledge, there exist no automated approach supporting recursive function synthesis from functional input-output specifications in full first-order logic.

This paper provides a solution in this respect by exploiting the constructive nature of induction. Intuitively, each case of an induction axiom tells us how to construct the desired program for the next recursive step using the program for the previous recursive step. We capture this construction recipe contained in the applications of induction in saturation-based proof search, by utilizing answer literals $\text{ans}(r)$ [3]. When we use an induction axiom in the proof, we introduce a special term into the answer literal, serving for tracking the program corresponding to the induction axiom. As we prove the cases of the induction axiom, we capture their corresponding programs in the answer literal. Finally, when we derive a clause $C \vee \text{ans}(r)$, where C only contains symbols allowed in a program, we convert the special tracker terms from r into recursive functions, and obtain a program for the initial specification conditioned on $\neg C$.

Contributions. We extend saturation-based first-order theorem proving with recursive program synthesis and bring the following contributions⁵:

- We introduce induction axioms, dubbed *magic axioms*, which capture the constructive nature of induction (Section 5).
- We convert the magic axioms into formulas used by a saturation-based framework to derive programs using recursion over algebraic datatypes, i.e., special cases of term algebras. We state necessary requirements for the calculus used in saturation and prove correctness of synthesized programs (Section 6).
- We present an extension of the superposition calculus that fulfills our necessary requirement and advocate for superposition reasoning for recursive function synthesis (Section 7).

⁵ proofs are given in the Appendix

- We show that our approach, illustrated initially for natural numbers, naturally extends to programs over arbitrary term algebras (Section 8).
- We implement our work in the VAMPIRE prover [13] and survey challenging examples it can synthesize (Section 9).

2 Preliminaries

We assume familiarity with standard multi-sorted first-order logic (FOL) with equality. We denote variables by x, y, z, w, u , terms by s, t, r , atoms by A , literals by L , clauses by C, D , formulas by F, G , all possibly with indices. Further, we write σ for Skolem constants. We reserve the symbol \square for the *empty clause* which is logically equivalent to \perp . We write \bar{L} for the literal complementary to L . By \simeq we denote the equality predicate and write $t \not\simeq s$ as a shorthand for $\neg t \simeq s$. We include a conditional term constructor *if – then – else* in the language, as follows: given a formula F and terms s, t of the same sort, we write *if F then s else t* to denote the term s if F is valid and t otherwise. An *expression* is a term, literal, clause or formula. We write $E[t]$ to denote that the expression E contains the term t . For simplicity, $E[s]$ denotes the expression E where all occurrences of t are replaced by the term s . Formulas with free variables are considered implicitly universally quantified, that is we consider closed formulas.

We use the standard semantics for FOL. For an interpretation function I , we denote the interpretation of a variable x , function symbol f and a predicate symbol p by x^I, f^I, p^I , respectively. We use the notation e^I, F^I also for interpretation of expressions e and formulas F , respectively. Further, for a variable or a constant a and a value v , we denote by $I\{a \mapsto v\}$ the interpretation function I' such that $a^{I'} = v$ and $b^{I'} = b^I$ for any constant or variable $b \neq a$.

We recall the standard notion of λ -expressions. Let t be a term and x a variable. Then $\lambda x.t$ denotes a λ -*expression*. For any interpretation I , we define $(\lambda x.t)^I$ as the function f given by $f(v) = t^{I\{x \mapsto v\}}$ for any value v . Moreover, we extend the notation of λ -expressions to also bind constants. Let c be a constant, then $\lambda c.t$ also denotes a λ -*expression*, and its interpretation $(\lambda c.t)^I$ is the function f given by $f(v) = t^{I\{c \mapsto v\}}$ for any value v .

A *substitution* θ is a mapping from variables to terms. A substitution θ is a *unifier* of two expressions E and E' if $E\theta = E'\theta$; θ is a *most general unifier (mgu)* if for every unifier η of E and E' , there exists substitution μ such that $\eta = \theta\mu$. We denote the mgu of E and E' with $\text{mgu}(E, E')$. We write $F_1, \dots, F_n \vdash G_1, \dots, G_m$ to denote that $F_1 \wedge \dots \wedge F_n \rightarrow G_1 \vee \dots \vee G_m$ is valid, and extend the notation also to validity modulo a theory T .

We work with term algebra [24], in particular with the special classes of the algebraically defined datatypes of the natural numbers \mathbb{N} , lists \mathbb{L} , and binary trees \mathbb{BT} . For reference we include the definitions of these term algebras in Figure 6 in Appendix D. We denote the sorts of symbols and terms by $:$ (colon), e.g., $f : \tau \rightarrow \alpha$ is a function symbol with domain τ and range α . To emphasize the sort τ of a quantified variable x , we write $\forall x \in \tau$ or $\exists x \in \tau$. For a term algebra sort τ , we denote its constructors with Σ_τ . We fix an arbitrary ordering on the constructors, and denote the i -th constructor in the order by c_i , i.e.,

Superposition (Sup):	Binary resolution (BR):	
$\frac{s \simeq t \vee C \quad L[s'] \vee D}{(L[t] \vee C \vee D)\theta}$	$\frac{A \vee C \quad \neg A' \vee D}{(C \vee D)\theta}$	
where $\theta := \text{mgu}(s, s')$.	where $\theta := \text{mgu}(A, A')$.	
Factoring (F):	Equality resolution (ER):	Equality factoring (EF):
$\frac{A \vee A' \vee C}{(A \vee C)\theta}$	$\frac{s \not\approx t \vee C}{C\theta}$	$\frac{s \simeq t \vee s' \simeq t' \vee C}{(s \simeq t \vee t \not\approx t' \vee C)\theta}$
where $\theta := \text{mgu}(A, A')$.	where $\theta := \text{mgu}(s, t)$.	where $\theta := \text{mgu}(s, s')$.

Fig. 2. Simplified superposition calculus Sup.

$\Sigma_\tau = \{c_1, \dots, c_{|\Sigma_\tau|}\}$. For each c_i , we denote its arity with n_{c_i} . We denote with P_{c_i} the set of argument positions of c_i of the sort τ . We only consider the standard models of term algebras. Programs we synthesize may contain terminating recursive functions $f : \tau \rightarrow \alpha$, where τ is a term algebra type. We define such function f by providing a set of equalities $\{f(c(\bar{x})) \simeq t[\bar{x}, f(x_{j_1}), \dots, f(x_{j_{|P_{c_i}}]})]\}_{c \in \Sigma_\tau}$, where $P_c = \{j_1, \dots, j_{|P_c|}\}$, and t contains no occurrences of f except for the distinguished ones. An example of such a definition is (1).

Saturation and Superposition. Saturation-based proof search implements *proving by refutation* [13]: validity of F is proved by establishing unsatisfiability of $\neg F$. Saturation-based first-order theorem provers work with clauses, rather than with arbitrary formulas. To prove a formula F , the provers negate F and further skolemize it and convert it to clausal normal form (CNF). The CNF of $\neg F$ is denoted by $\text{cnf}(\neg F)$, resulting in a set S of initial clauses. For example, the CNF of the negated and skolemized (SD) is

$$\text{half}(y) \not\approx \sigma, \tag{2}$$

where σ is a fresh constant used for skolemizing x , and y is implicitly universally quantified. Saturation provers *saturate* S by computing logical consequences of S with respect to a sound inference system \mathcal{I} . Whenever the empty clause \square is derived, the set S of clauses is unsatisfiable and F is valid. We may extend the initial set S with additional clauses C_1, \dots, C_n . If C is derived from this extended set, we say C is derived from S *under additional assumptions* C_1, \dots, C_n .

The *superposition calculus* Sup [19] is the most common inference system for first-order logic with equality. Figure 2 shows a simplified version⁶ of Sup. The Sup calculus is *sound* (if \square is derived from F , then F is unsatisfiable) and *refutationally complete* (if F is unsatisfiable, then \square can be derived from it).

⁶ see Appendix A.1 for the full Sup calculus.

3 Recent Developments in Saturation

In this section we summarize recent results relevant to our work.

Program Synthesis in Saturation. Synthesizing (non-recursive) programs in saturation has been initiated in [7]. Here, *computable* and *uncomputable* symbols in the signature are distinguished. Intuitively, computable symbols are those which are allowed to appear in a synthesized program. An expression is *computable* if all symbols it contains are computable. A symbol or an expression is *uncomputable* if it is not computable.

Let A_1, \dots, A_n be closed formulas. Then

$$A_1 \wedge \dots \wedge A_n \rightarrow \forall \bar{x} \exists y. F[\bar{x}, y] \quad (3)$$

is a (*synthesis*) *specification with inputs \bar{x} and output y* .

Consider a computable term $r[\bar{x}]$ such that $A_1 \wedge \dots \wedge A_n \rightarrow \forall \bar{x}. F[\bar{x}, r[\bar{x}]]$ holds. Such an $r[\bar{x}]$ is called a *program* for (3) and a *witness for y in (3)*. If $A_1 \wedge \dots \wedge A_n \rightarrow \forall \bar{x}. (F_1 \wedge \dots \wedge F_n \rightarrow F[\bar{x}, r[\bar{x}]])$ holds for computable formulas F_1, \dots, F_n , then $\langle r[\bar{x}], \bigwedge_{i=1}^n F_i \rangle$ is a *program with conditions F_1, \dots, F_n* for (3).

The work of [7] extended saturation-based theorem proving to a *saturation-based program synthesis framework*. To this end, the classified negated specification (3) is extended by an *answer literal ans*:

$$A_1 \wedge \dots \wedge A_n \wedge \forall y. (\text{cnf}(\neg F[\bar{\sigma}, y]) \vee \text{ans}(y)) \quad (4)$$

The set of clauses (4) is then saturated. During saturation, upon deriving a clause $C[\bar{\sigma}] \vee \text{ans}(r[\bar{\sigma}])$, where $r[\bar{\sigma}]$ is computable and $C[\bar{\sigma}]$ is computable and does not contain *ans*, the program $\langle r[\bar{x}], \neg C[\bar{x}] \rangle$ with conditions for (3) is recorded and the clause is replaced by $C[\bar{\sigma}]$. This step is called *answer literal removal* within saturation. Once saturation terminates by deriving the empty clause \square , the final program for (3) is constructed by composing the relevant recorded programs with conditions in a nested if–then–else. To support derivation of such clauses $C[\bar{\sigma}] \vee \text{ans}(r[\bar{\sigma}])$ and to ensure that answer literals only have computable arguments, the work of [7] extended the superposition calculus Sup with new inference rules (as shown in Appendix A.2).

Induction in Saturation. Inductive reasoning has been integrated in saturation [23,4,8,6,5]. The main idea in this body of work is to apply induction by *theory lemma generation*: based on already derived formulas, generate a suitable induction axiom and add it to the search space. To this end, the following induction rule is used:

$$\frac{\bar{L}[t] \vee C}{F \rightarrow \forall x. L[x]} \text{ (Ind)},$$

where $L[t]$ is a ground literal, C is a clause, and $F \rightarrow \forall x. L[x]$ is a valid induction axiom. The conclusion of the **Ind** rule is classified, yielding $\text{cnf}(\neg F) \vee L[x]$. This clause is resolved with the premise $\bar{L}[t] \vee C$ immediately after applying the **Ind** rule and the resulting clause $\text{cnf}(\neg F) \vee C$ is added to the search space.

An example of a valid induction schema is the *structural induction axiom for natural numbers*, where $G[x]$ is any closed formula:

$$(G[0] \wedge \forall y.(G[y] \rightarrow G[s(y)])) \rightarrow \forall x.G[x] \quad (5)$$

When we instantiate the schema with $G[x] := L[x]$, we obtain an axiom that can be used in **Ind**. Since the rule requires $L[t]$ to be ground, this instance of **Ind** cannot be applied on (2) and thus is not sufficient for proving (SD) of Figure 1. To prove formulas with a free variable by induction, we extend **Ind** in Section 5.

Note that we can also use a complex formula $G[t]$ in place of the literal $L[t]$ in **Ind**, obtaining a more involved rule, possibly with multiple premises, similarly to a *multi-clause induction rule* [6] or a *induction with arbitrary formulas* [5].

4 Saturation with Induction in Constructive Logic

We first summarize the key challenges our work resolves towards recursive synthesis in saturation, and then present our synthesis approach in Sections 5–8.

The idea of extracting programs from proofs originates from results in constructive (intuitionistic) logic, starting with Kleene’s realizability [11]. In constructive logic, provability of a formula $\forall \bar{x} \exists y.F[\bar{x}, y]$ implies that there is an algorithm which, given values for \bar{x} , outputs a value for y satisfying $F[\bar{x}, y]$.

We note that the structural induction formula (5) over natural numbers has computational content, as follows. The program r for $\forall x.G[x]$ can be built from a program r_0 for $G[0]$ and a program r_s for $\forall y.(G[y] \rightarrow G[s(y)])$ as:

$$\begin{aligned} r(0) &\simeq r_0 \\ r(s(y)) &\simeq r_s(r(y)) \end{aligned}$$

For this to be useful, we need to first prove $G[0]$, then prove $\forall y.(G[y] \rightarrow G[s(y)])$, and then use the induction formula to derive $\forall x.G[x]$. Such an approach towards constructing programs does not however work in saturation-based theorem proving, as saturation does not reduce goals to subgoals [2]. Rather, we add the induction formula as a theory lemma to the proof search and continue saturation, so we do not have proofs of either $G[0]$ or $\forall y.(G[y] \rightarrow G[s(y)])$. Constructing programs during saturation becomes even more complex when using answer literals, because clauses generated during saturation may contain these literals. For example, if we try to extract a proof of $G[0]$, we may find a proof with an answer literal in it.

To capture the constructive nature of induction and address the above challenges of program synthesis in saturation, we use the the following trick. We modify the induction formula so that it indirectly stores information about the programs for $G[0]$ and $\forall y.(G[y] \rightarrow G[s(y)])$. To do this, instead of adding the induction axiom (5), in Section 5 we add what we call a *magic axiom for* (5), where G has an additional argument for storing the program. In Section 6 we further convert our magic axioms into formulas to be used to derive recursive programs in saturation.

5 Induction with Magic Formulas

We first present our approach to *proving* formulas with a free variable by induction. We further extend this approach to *synthesis* in Section 6. While our approach works the same way with arbitrary term algebras, for the sake of clarity we first introduce our work for natural numbers and then for general term algebras in Section 8.

We use the following *magic axiom*:

$$\left(\exists u_0. G[0, u_0] \wedge \forall y. (\exists w. G[y, w] \rightarrow \exists u_s. G[s(y), u_s]) \right) \rightarrow \forall z. \exists x. G[z, x] \quad (6)$$

Note that all magic axioms are valid, as they are instances of the structural induction axiom (5) with the quantified formula $\exists x. G[t, x]$ in place of $G[t]$. The magicness of (6) stems from its simple, yet powerful expressiveness: when used in proof search, the variables u_0, u_s in the antecedent capture the programs for the base and step cases, allowing us to construct a program for x in the consequent.

Using axiom (6), we introduce the following variant of the *Ind* rule:

$$\frac{\overline{L[t, x] \vee C}}{\left(\exists u_0. L[0, u_0] \wedge \forall y. (\exists w. L[y, w] \rightarrow \exists u_s. L[s(y), u_s]) \right) \rightarrow \forall z. \exists x. L[z, x]} \quad (\text{MagInd})$$

where the only free variable of $L[t, x]$ is x and C does not contain x .

Example 1. Consider the specification (SD) from Figure 1. To prove it using superposition, and not yet synthesize the function satisfying (SD), we use the following magic axiom:

$$\left(\exists u_0. \text{half}(u_0) \simeq 0 \wedge \forall y. (\exists w. \text{half}(w) \simeq y \rightarrow \exists u_s. \text{half}(u_s) \simeq s(y)) \right) \rightarrow \forall z. \exists x. \text{half}(x) \simeq z \quad (7)$$

To use (7) in saturation, we clasify it and skolemize the variables y, w, x as $\sigma_y, \sigma_w, \sigma_x(z)$, respectively. The following is a refutational proof of (SD):

1. $\text{half}(y) \not\approx \sigma$ [negated and skolemized specification (SD)]
2. $\text{half}(u_0) \not\approx 0 \vee \text{half}(\sigma_w) \simeq \sigma_y \vee \text{half}(\sigma_x(z)) \simeq z$ [MagInd with (7)]
3. $\text{half}(u_0) \not\approx 0 \vee \text{half}(u_s) \not\approx s(\sigma_y) \vee \text{half}(\sigma_x(z)) \simeq z$ [MagInd with (7)]
4. $\text{half}(u_0) \not\approx 0 \vee \text{half}(\sigma_w) \simeq \sigma_y$ [BR 1, 2]
5. $\text{half}(u_0) \not\approx 0 \vee \text{half}(u_s) \not\approx s(\sigma_y)$ [BR 1, 3]
6. $\text{half}(u_0) \not\approx 0 \vee \text{half}(u_s) \not\approx s(\text{half}(\sigma_w))$ [Sup 4, 5]
7. $\text{half}(u_0) \not\approx 0 \vee \text{half}(u_s) \not\approx \text{half}(s(s(\sigma_w)))$ [Sup (H3), 6]
8. $\text{half}(u_0) \not\approx 0$ [ER 7]
9. \square [BR 8, (H2)]

Hence, the magic axiom (6) is sufficient to prove (SD). However, (6) does not suffice to synthesize the program for (SD) from the above proof. Similarly to [7], for synthesis we would use

$$\text{half}(y) \not\approx \sigma \vee \text{ans}(y) \quad (8)$$

instead of clause 1 and obtain a derivation similar to the one above, but with the answer literal $\text{ans}(\sigma_x(\sigma))$. As σ_x is a fresh skolem function, it is uncomputable and not allowed in answer literals. Therefore, simply following the approach of [7] fails to synthesize a recursive program from the proof of (SD). We address the challenge of program construction for the skolem function σ_x in Section 6. \square

6 Programs with Primitive Recursion

We now construct recursive programs for proofs using induction over natural numbers (6). As mentioned in Section 4, the antecedent of the induction axiom gives us a recipe for constructing the program for the consequent. To capture this dependence of the consequent program x on the antecedent programs u_0, u_s , we convert the magic axiom (6) to its equivalent prenex normal form:

$$\exists y, w. \forall u_0, u_s, z. \exists x. \left((G[0, u_0] \wedge (G[y, w] \rightarrow G[s(y), u_s])) \rightarrow G[z, x] \right) \quad (9)$$

We next define a recursive operator to be used for constructing programs.

Definition 1 (Primitive Recursion Operator). Let $f_1 : \alpha$, and $f_2 : \mathbb{N} \times \alpha \rightarrow \alpha$. The *primitive recursion operator* R for natural numbers and α is:

$$\begin{aligned} R(f_1, f_2)(0) &\simeq f_1 \\ R(f_1, f_2)(s(y)) &\simeq f_2(y, R(f_1, f_2)(y)) \end{aligned}$$

Lemma 2 (Recursive Witness). The expression $R(u_0, \lambda y, w. u_s)(z)$ is a witness for the variable x in (9).

Lemma 2 ensures that we can construct a program for the consequent of the magic axiom given programs for the base case and the step case. We next integrate this construction into our synthesis framework using answer literals. For that we take a close look on skolemization of induction axiom (9), and define skolem symbols for the variable x , capturing the recursive program.

Definition 3 (rec-Symbols). Consider formulas $G[t, x]$ with a single free variable $x : \alpha$ containing a term $t : \mathbb{N}$. For each such formula we introduce a distinct computable function symbol $\text{rec}_{G[t, x]} : \alpha \times \alpha \rightarrow \alpha$. We will refer to such symbols $\text{rec}_{G[t, x]}$ as *rec-symbols*. When the formula $G[t, x]$ is clear from the context or unimportant for the context, we will simply write rec instead of $\text{rec}_{G[t, x]}$.

A term with a rec-symbol as the top-level functor is called a *rec-term*.

Definition 4 (Magic Formula). The magic formula for $G[t, x]$ is:

$$\begin{aligned} &\forall u_0, u_s, z. \\ &\left((G[0, u_0] \wedge (G[\sigma_y, \sigma_w] \rightarrow G[s(\sigma_y), u_s])) \rightarrow G[z, \text{rec}_{G[t, x]}(u_0, u_s, z)] \right) \quad (10) \end{aligned}$$

It is easy to see that magic formula (10) is obtained by skolemizing the prenex normal form of magic axiom (9), where we replace the variables y, w by fresh constants σ_y, σ_w , and the variable x by a fresh $\text{rec}_{G[t,x]}$ -symbol. The constants σ_y, σ_w introduced in (10) are said to be *associated with the $\text{rec}_{G[t,x]}$ -term*. An occurrence of any skolem constant σ_y, σ_w is considered computable if it is an occurrence in the second argument of a $\text{rec}_{G[t,x]}$ -term which it is associated with.

We introduce additional requirements for reasoning with rec -terms to ensure that they always represent the recursive function to be synthesized.

Definition 5 (rec-Compliance). An inference system \mathcal{I} is *rec-compliant* if:

1. \mathcal{I} only introduces rec -terms in the instances of the magic formula (10),
2. \mathcal{I} does not introduce uncomputable symbols into arguments of rec -terms in clauses it derives.

Using a rec -compliant inference system \mathcal{I} , we derive clauses containing rec -terms. These terms correspond to functions constructed using the operator \mathbf{R} .

Definition 6 (Recursive Function Term). Let σ_y, σ_w be associated with $\text{rec}(s_1, s_2, t)$. Then we call the term $\mathbf{R}(s_1, \lambda\sigma_y, \sigma_w.s_2)(t)$ the *recursive function term corresponding to $\text{rec}(s_1, s_2, t)$* .

For a term r , we denote by $r^{\mathbf{R}}$ the expression obtained from r by iteratively replacing all rec -terms by their corresponding recursive function terms, starting from the innermost ones. Similarly, formula $F^{\mathbf{R}}$ denotes the formula F in which we replace all rec -terms by their corresponding recursive function terms.

Lemma 7 (Recursive Witness for Magic Formulas). Consider the formula obtained from (10) by replacing $\text{rec}_{G[t,x]}(u_0, u_s, z)$ by its corresponding recursive function term $\mathbf{R}(u_0, \lambda\sigma_y, \sigma_w.u_s)(z)$:

$$\forall u_0, u_s, z. \left((G[0, u_0] \wedge (G[\sigma_y, \sigma_w] \rightarrow G[s(\sigma_y), u_s])) \rightarrow G[z, \mathbf{R}(u_0, \lambda\sigma_y, \sigma_w.u_s)(z)] \right) \quad (11)$$

For every interpretation I , there exists a mapping of skolem constants to values $\{\sigma_y \mapsto v_y, \sigma_w \mapsto v_w\}$ such that I extended by this mapping is a model of (11). As a consequence, formula (11) is satisfiable.

Lemma 7 implies that we can use formula (11) instead of (10) in derivation, while preserving the soundness of the derivations. Soundness of our approach to recursive program synthesis is given next.

Theorem 8 (Semantics of Clauses with Answer Literals and rec -terms). Let C_1, \dots, C_m be clauses and F a formula containing no answer literals and no rec -symbols. Let C be a clause containing no answer literals. Let M_1, \dots, M_l be magic formulas. Assume that using a sound rec -compliant inference system \mathcal{I} , we derive $C \vee \text{ans}(r[\bar{\sigma}])$, where $r[\bar{\sigma}]$ is computable, from the set of clauses

$$\{ C_1, \dots, C_m, M_1, \dots, M_l, \text{cnf}(\neg F[\bar{\sigma}, y] \vee \text{ans}(y)) \}.$$

Then

$$M_1^R, \dots, M_l^R, C_1, \dots, C_m \vdash C^R, F[\bar{\sigma}, r^R[\bar{\sigma}]].$$

That is, under the assumptions $M_1^R, \dots, M_l^R, C_1, \dots, C_m, \neg C^R$, the computable expression $r^R[\bar{x}]$ is a witness for y in $\forall \bar{x} \exists y. F[\bar{x}, y]$.

Based on Theorem 8, if the CNF of A_1, \dots, A_n is among C_1, \dots, C_m , then $r^R[\bar{x}]$ is a witness for y in (3) under the assumptions $M_1^R, \dots, M_l^R, C_1, \dots, C_m, \neg C^R$. The following ensures that we can construct recursive programs with conditions.

Theorem 9 (Recursive Programs). Let $r[\bar{\sigma}]$ be a computable term, and $C[\bar{\sigma}], C_1[\bar{\sigma}], \dots, C_m[\bar{\sigma}]$ be ground computable clauses containing no answer literals and no rec-symbols. Assume that using a sound rec-compliant inference system \mathcal{I} , we derive the clause $C[\bar{\sigma}] \vee \text{ans}(r[\bar{\sigma}])$ from the CNF of

$$\{ A_1, \dots, A_n, C_1[\bar{\sigma}], \dots, C_m[\bar{\sigma}], M_1, \dots, M_l, \neg F[\bar{\sigma}, y] \vee \text{ans}(y) \}$$

where M_1, \dots, M_l are magic formulas. Then,

$$\langle r^R[\bar{x}], \bigwedge_{j=1}^m C_j[\bar{x}] \wedge \neg C[\bar{x}] \rangle$$

is a program with conditions for (3).

From Theorem 9 we obtain the following key result on program synthesis.

Theorem 10 (Recursive Program Synthesis). Let $P_1[\bar{x}], \dots, P_k[\bar{x}]$, where $P_i[\bar{x}] = \langle r_i^R[\bar{x}], \bigwedge_{j=1}^{i-1} C_j[\bar{x}] \wedge \neg C_i[\bar{x}] \rangle$, be programs with conditions for (3), such that $\bigwedge_{i=1}^n A_i \wedge \bigwedge_{i=1}^k C_i[\bar{x}]$ is unsatisfiable. Then the program $P[\bar{x}]$ defined as

$$\begin{aligned} P[\bar{x}] := & \text{if } \neg C_1[\bar{x}] \text{ then } r_1^R[\bar{x}] \\ & \text{else if } \neg C_2[\bar{x}] \text{ then } r_2^R[\bar{x}] \\ & \dots \\ & \text{else if } \neg C_{k-1}[\bar{x}] \text{ then } r_{k-1}^R[\bar{x}] \\ & \text{else } r_k^R[\bar{x}], \end{aligned}$$

is a program for (3).

7 Recursive Synthesis in Saturation

This section integrates the proving and synthesis steps of Sections 5–6 into saturation. The crux of our approach is that instead of adding standard induction formulas to the search space, we add magic formulas.

Theorems 9–10 imply that, to derive recursive programs, we can use any rec-compliant calculus, as long as the calculus supports derivation of clauses $C \vee \text{ans}(r)$, where r is computable and C is ground, computable, and contains no

rec-terms nor answer literals. In our work we rely on the extended Sup calculus of [7] (see Figure 4 in Appendix A.2), which we (i) further extend by adding magic formulas alongside standard induction formulas, (ii) make rec-compliant by disallowing inferences containing uncomputable rec-terms, and (iii) extend by adding more complex rules for introducing conditions into rec-terms (see Appendix A.3). We illustrate these steps by our running example.

Example 2. Using the extended Sup calculus, we synthesize the program for the specification of Figure 1. With the magic formula corresponding to (7),

$$\forall u_0, u_s, z. \left((\text{half}(u_0) \simeq 0 \wedge (\text{half}(\sigma_w) \simeq \sigma_y \rightarrow \text{half}(u_s) \simeq s(\sigma_y))) \rightarrow \text{half}(\text{rec}(u_0, u_s, z)) \simeq z \right), \quad (12)$$

we obtain the following derivation⁷:

1. $\text{half}(y) \not\simeq \sigma \vee \text{ans}(y)$ [negated, skolemized specification with answer literal]
2. $\text{half}(u_0) \not\simeq 0 \vee \text{half}(\sigma_w) \simeq \sigma_y \vee \text{half}(\sigma_x(z)) \simeq z$ [MagInd with (12)]
3. $\text{half}(u_0) \not\simeq 0 \vee \text{half}(u_s) \not\simeq s(\sigma_y) \vee \text{half}(\sigma_x(z)) \simeq z$ [MagInd with (12)]
4. $\text{half}(u_0) \not\simeq 0 \vee \text{half}(\sigma_w) \simeq \sigma_y \vee \text{ans}(\text{rec}(u_0, u_s, \sigma))$ [BR 1, 2]
5. $\text{half}(u_0) \not\simeq 0 \vee \text{half}(u_s) \not\simeq s(\sigma_y) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma))$ [BR 1, 3]
6. $\text{half}(u_0) \not\simeq 0 \vee \text{half}(u_s) \not\simeq s(\text{half}(\sigma_w)) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma))$ [Sup 4, 5]
7. $\text{half}(u_0) \not\simeq 0 \vee \text{half}(u_s) \not\simeq \text{half}(s(s(\sigma_w))) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma))$ [Sup (H3), 6]
8. $\text{half}(u_0) \not\simeq 0 \vee \text{ans}(\text{rec}(u_0, s(s(\sigma_w)), \sigma))$ [ER 7]
9. $\text{ans}(\text{rec}(s(0), s(s(\sigma_w))), \sigma)$ [BR 8, (H2)]
10. \square [answer literal removal 9]

The program recorded in step 10 of the proof is $\text{rec}(s(0), s(s(\sigma_w)), x)^R = R(s(0), \lambda \sigma_w. s(s(\sigma_w)))(x) = f(x)$, where f is defined as:

$$\begin{aligned} f(0) &\simeq s(0) \\ f(s(n)) &\simeq s(f(n)) \end{aligned}$$

Note that while the synthesized program satisfies the specification (SD), it does not match the expected definition of the double function from (1). Since the half function is rounding down, and the specification does not require the synthesized function to produce even results, the base case was resolved in step 9 with (H2), leading to $f(0) \simeq s(0)$. As a result, we have $f(n) = s(\text{double}(n))$ for any n . \square

Example 2 demonstrates that specification (SD) has multiple solutions and saturation can find a solution different from the intended one. In the next example we modify the specification to have a single solution and synthesize it.

Example 3. To synthesize the double function, we modify the specification:

$$\text{additional axioms: } \text{even}(0) \quad (\text{E1})$$

$$\neg \text{even}(s(0)) \quad (\text{E2})$$

$$\forall x. (\text{even}(s(s(x))) \leftrightarrow \text{even}(x)) \quad (\text{E3})$$

$$\text{new specification: } \forall x \exists y. (\text{half}(y) \simeq x \wedge \text{even}(y)) \quad (\text{SD}')$$

⁷ the fully detailed derivation is in Appendix E

After negating and skolemizing (SD') and adding the answer literal, we obtain:

$$\text{half}(y) \not\approx \sigma \vee \neg \text{even}(y) \vee \text{ans}(y) \quad (13)$$

In this case we use the magic axiom for the conjunction $G[t, x] := \text{half}(x) \simeq t \wedge \text{even}(x)$:

$$\begin{aligned} & \left(\exists u_0. (\text{half}(u_0) \simeq 0 \wedge \text{even}(u_0)) \wedge \right. \\ & \quad \left. \forall y. (\exists w. (\text{half}(w) \simeq y \wedge \text{even}(w)) \rightarrow \exists u_s. (\text{half}(u_s) \simeq s(y) \wedge \text{even}(u_s))) \right) \quad (14) \\ & \rightarrow \forall z. \exists x. (\text{half}(x) \simeq z \wedge \text{even}(x)) \end{aligned}$$

We classify the magic formula corresponding to (14), and further resolve it with the premise (13) to obtain:

$$\begin{aligned} & \text{half}(u_0) \not\approx 0 \vee \neg \text{even}(u_0) \vee \text{half}(\sigma_w) \simeq \sigma_y \vee \text{ans}(\text{rec}(u_0, u_s, \sigma)) \\ & \quad \text{half}(u_0) \not\approx 0 \vee \neg \text{even}(u_0) \vee \text{even}(\sigma_w) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma)) \\ & \text{half}(u_0) \not\approx 0 \vee \neg \text{even}(u_0) \vee \text{half}(u_s) \not\approx s(\sigma_y) \vee \neg \text{even}(u_s) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma)) \end{aligned}$$

The refutation of these clauses follows a similar course to the proof in Example 2. However, u_0 occurring in the literal $\neg \text{even}(u_0)$ forces the proof to use (H1) instead of (H2), and thus the final derived answer literal will be $\text{rec}(0, s(s(\sigma_w)), \sigma)$, corresponding exactly to the function definition of **double** from (1). Note that a derivation of this program in this case requires a saturation prover to apply induction on conjunctions of literals. \square

8 Generalization to Arbitrary Term Algebras

Our approach from Sections 5–7 generalizes naturally to arbitrary term algebras. This section summarizes the key parts of this generalization. We state all definitions, lemmas and theorems in Appendix C.

Let τ be a (possibly polymorphic) term algebra with constructors $\{c_1, \dots, c_n\}$, where we denote the sort of each c_i by $\tau_{i,1} \times \dots \times \tau_{i,n_{c_i}} \rightarrow \tau$, and $P_{c_i} = \{j_1, \dots, j_{|P_{c_i}|}\}$ for each $i = 1, \dots, n$. Let α be any sort. The *magic axiom* for $G[t, x]$, where $t : \tau, x : \alpha$, is:

$$\left(\bigwedge_{c \in \Sigma_\tau} \bigvee_{i=1}^{n_c} y_{c,i}. \left(\bigwedge_{j \in P_c} \exists w_{c,j}. G[y_{c,j}, w_{c,j}] \rightarrow \exists u_c. G[c(\bar{y}_c), u_c] \right) \right) \rightarrow \forall z. \exists x. G[z, x] \quad (15)$$

The corresponding *magic formula* uses the skolem function $\text{rec}_{G[t,x]} : \alpha^{n_c} \times \tau \rightarrow \alpha$:

$$\forall_{c \in \Sigma_\tau} u_c. \forall z. \left(\bigwedge_{c \in \Sigma_\tau} \left(\bigwedge_{j \in P_c} G[\sigma_{y_{c,j}}, \sigma_{w_{c,j}}] \rightarrow G[c(\bar{\sigma}_{y_c}), u_c] \right) \rightarrow G[z, \text{rec}_{G[t,x]}(\bar{u}, z)] \right) \quad (16)$$

Note that each $\sigma_{y_{c_i,j}}, \sigma_{w_{c_i,j}}$ introduced in (16) is considered computable only in the i th argument of its associated rec -term. We define the *recursion operator* R

for τ and α analogously to Definition 1:

$$\begin{aligned} \mathbf{R}(f_1, \dots, f_n)(c_1(\bar{x})) &\simeq f_1(x_1, \dots, x_{n_{c_1}}, \mathbf{R}(f_1, \dots, f_n)(x_{j_1}), \dots, \mathbf{R}(f_1, \dots, f_n)(x_{j_{|P_{c_1}|}})) \\ &\quad \dots \\ \mathbf{R}(f_1, \dots, f_n)(c_n(\bar{x})) &\simeq f_n(x_1, \dots, x_{n_{c_n}}, \mathbf{R}(f_1, \dots, f_n)(x_{j_1}), \dots, \mathbf{R}(f_1, \dots, f_n)(x_{j_{|P_{c_n}|}})) \end{aligned}$$

where for each i we have $f_i : \tau_{i,1} \times \dots \times \tau_{i,n_{c_i}} \times \alpha^{|P_{c_i}|} \rightarrow \alpha$. Using \mathbf{R} , we state an analogue of Lemma 7:

Lemma 11 (Recursive Witness for Magic Formulas Using τ). Consider the formula obtained from (16) by replacing $\text{rec}_{G[t,x]}(\bar{u}, z)$ by its corresponding recursive function term:

$$\begin{aligned} \forall_{c \in \Sigma_\tau} u_c. \forall z. &\left(\bigwedge_{c \in \Sigma_\tau} \left(\bigwedge_{j \in P_c} G[\sigma_{y_{c,j}}, \sigma_{w_{c,j}}] \rightarrow G[c(\bar{\sigma}_{y_c}), u_c] \right) \right. \\ &\left. \rightarrow G[z, \mathbf{R}(\lambda_{i=1}^{n_{c_1}} \sigma_{y_{c_1,i}} \cdot \lambda_{k \in P_{c_1}} \sigma_{w_{c_1,k}} \cdot u_{c_1}, \dots, \lambda_{i=1}^{n_{c_n}} \sigma_{y_{c_n,i}} \cdot \lambda_{k \in P_{c_n}} \sigma_{w_{c_n,k}} \cdot u_{c_n})(z)] \right) \end{aligned} \quad (17)$$

For every interpretation, there exists its extension by some $\{\sigma_{y_{c,i}} \mapsto v_{y,c,i}, \sigma_{w_{c,k}} \mapsto v_{w,c,k}\}_{c \in \Sigma_\tau, i \in \{1, \dots, n_c\}, k \in P_c}$ such that the extension is a model of (17). As a consequence, formula (17) is satisfiable.

Using Lemma 11, we derive the analogues of Theorems 8–10 for an arbitrary term algebra τ . We then employ magic formulas (16) in $\text{MagI}nd$ when in the premise $L[t, x] \vee C \vee \text{ans}(r[x])$ we have $t : \tau$. We finally note that our synthesis method generalizes also to sorts other than term algebras, as long as the induction axiom used for the sort carries the constructive meaning described in Section 4.

9 Implementation and Examples

Implementation. We extended the first-order theorem prover `VAMPIRE` [13] with a proof-of-concept implementation of our method for recursive program synthesis in saturation. Our implementation consists of approximately 1,100 lines of C++ code and is available online at <https://github.com/vprover/vampire/tree/synthesis-recursive>.

We implemented the $\text{MagI}nd$ rule as well as a version of $\text{MagI}nd$ using a magic axiom with base case $s(0)$ for natural numbers and $\text{cons}(a, \text{nil})$ for any a for lists. To support synthesis requiring induction on specifications $\neg F[t, x]$, where $F[t, x]$ is an arbitrary formula with the only free variable x , we use an encoding as follows. We change the specification $\forall \bar{x} \exists y. F[\bar{x}, y]$ to $\forall \bar{x} \exists y. p(\bar{x}, y)$, where p is a fresh uncomputable predicate, and we add an axiom $\forall \bar{x}, y. (p(\bar{x}, y) \leftrightarrow F[\bar{x}, y])$.

Examples. Our implementation can synthesize the programs for the specifications (SD) and (SD'). We also synthesize further examples over the term algebras⁸ of natural numbers \mathbb{N} , lists \mathbb{L} , and binary trees \mathbb{BT} . We display the

⁸ See Appendix D for term algebra constructors and signatures (Figure 6), and for axiomatization and lemmas for the used predicates and functions (Figures 7, 8).

Specification	Program	Synthesized definitions	VAMPIRE
Double: $\forall x \in \mathbb{N}. \exists y \in \mathbb{N}.$ $(\text{half}(y) \simeq x \wedge \text{even}(y))$	$f(x)$	$f(0) \simeq 0$ $f(s(n)) \simeq s(s(f(n)))$	✓
Associativity of addition: $\forall x_1, x_2, x_3 \in \mathbb{N}. \exists y \in \mathbb{N}.$ $(x_1 + x_2) + x_3 \simeq x_1 + y$	$f(x_3)$	$f(0) \simeq x_2$ $f(s(n)) \simeq s(f(n))$	✓
Subtraction with condition: $\forall x_1, x_2 \in \mathbb{N}. \exists y \in \mathbb{N}.$ $(x_2 < x_1 \rightarrow x_2 + y = x_1)$	$f(x_2)$	$f(0) \simeq x_1$ $f(s(n)) \simeq p(f(n))$	✓
Floored square root: $\forall x \in \mathbb{N}. \exists y \in \mathbb{N}.$ $(y \cdot y \leq x \wedge x < s(y) \cdot s(y))$	$f(x)$	$f(0) \simeq 0$ $f(s(n)) \simeq \text{if } s(n) \simeq s(f(n)) \cdot s(f(n))$ $\text{ then } s(f(n)) \text{ else } f(n)$	✗
Floored division: $\forall x_1, x_2 \in \mathbb{N}. \exists y \in \mathbb{N}. (x_2 \neq 0 \rightarrow$ $(y \cdot x_2 \leq x_1 \wedge x_1 < s(y) \cdot x_2))$	$f(x_1)$	$f(0) \simeq 0$ $f(s(n)) \simeq \text{if } s(n) \simeq s(f(n)) \cdot x_2$ $\text{ then } s(f(n)) \text{ else } f(n)$	✗
Length of 2 concatenated lists: $\forall x_1, x_2 \in \mathbb{L}. \exists y \in \mathbb{N}.$ $y \simeq \text{len}(x_1 ++ x_2)$	$f(x_1)$	$f(\text{nil}) \simeq \text{len}(x_2)$ $f(\text{cons}(n, l)) \simeq s(f(l))$	✓
Last element of a list: $\forall x \in \mathbb{L}. \exists y \in \mathbb{N}. (x \neq \text{nil} \rightarrow$ $\exists z \in \mathbb{L}. x \simeq z ++ \text{cons}(z, \text{nil}))$	$f(x)$	$f(\text{cons}(n, \text{nil})) \simeq n$ $l \neq \text{nil} \rightarrow f(\text{cons}(n, l)) \simeq f(l)$	✓
Prefix of a list given its suffix: $\forall x_1, x_2 \in \mathbb{L}. \exists y \in \mathbb{L}.$ $(\text{suff}(x_2, x_1) \rightarrow x_1 \simeq y ++ x_2)$	$f(x_2)$	$f(\text{nil}) \simeq x_1$ $f(\text{cons}(n, l)) \simeq g(f(l))$ $g(\text{cons}(n, \text{nil})) \simeq \text{nil}$ $l \neq \text{nil} \rightarrow g(\text{cons}(n, l)) \simeq \text{cons}(n, g(l))$	✗
Maximum element of a list: $\forall x \in \mathbb{L}. \exists y \in \mathbb{N}. (x \neq \text{nil} \rightarrow$ $(\text{in}(y, x) \wedge \forall k \in \mathbb{N}. (\text{in}(k, x) \rightarrow k \leq y))$	$f(x)$	$f(\text{cons}(n, \text{nil})) \simeq n$ $l \neq \text{nil} \rightarrow f(\text{cons}(n, l)) \simeq \text{if } f(l) < n$ $\text{ then } n \text{ else } f(l)$	✗
Maximum element of a tree: $\forall x \in \mathbb{BT}. \exists y \in \mathbb{N}.$ $(\text{in}(y, x) \wedge \forall k \in \mathbb{N}. (\text{in}(k, x) \rightarrow k \leq y))$	$f(x)$	$f(\text{leaf}(n)) \simeq n$ $f(\text{bt}(l, n, r)) \simeq$ $\text{if } f(l) < f(r) \text{ then}$ $\text{if } f(l) < n \text{ then}$ $\text{if } f(r) < n \text{ then } n \text{ else } f(r)$ $\text{ else } f(r)$ $\text{else if } f(r) < n \text{ then}$ $\text{if } f(l) < n \text{ then } n \text{ else } f(l)$ $\text{ else } f(l)$	✗

Table 1. Synthesis examples using natural numbers \mathbb{N} , lists \mathbb{L} and binary trees \mathbb{BT} . The x -variables in the program and synthesized definitions are the inputs. While our framework synthesizes all these examples (see Appendix D for the derivations), our implementation in VAMPIRE only synthesizes those marked with “✓”. Note that for “Length of 2 concatenated lists” we consider $++$ to be uncomputable.

specifications alongside the programs synthesized by our framework in Table 1, and provide the full derivations of the synthesized programs in Appendix D. Our framework synthesizes programs for each of the examples, yet our implementation supports so far only a limited set of magic formulas; therefore, the “VAMPIRE” column of Table 1 lists which examples are solved in practice.

10 Related Work

We only discuss approaches that support full automation, without templates or user guidance. We extend the recursion-free synthesis framework of [7] and exploit ideas from deductive synthesis [17,14,27] using answer literals [3]. We bring recursive program synthesis into the landscape of saturation-based proving and construct programs from saturation proofs with magic axioms.

The works of [16,27] construct recursive programs from proofs by induction, by reducing the program specification to subgoals corresponding to the cases of the induction axiom. Modern first-order theorem provers mostly implement saturation-based proof search, which however does not support a goal-subgoal architecture. Our approach integrates induction directly into saturation and enables automated reasoning with term algebras.

Fully automated methods supporting recursive program synthesis include SYNQUID [20], LEON [12], JENNISYS [15], SUSLIK [21], CYPRESS [9], and BURST [18]. Except for BURST, all these works decompose goals into subgoals. Our work complements these methods, by turning saturation into a recursive synthesis framework over first-order theories. As such, our work also differs from SYNQUID, where term enumeration combined with type checking is used over program specifications within decidable logics. LEON uses recursive schemas corresponding to our recursive operator R , instantiates them by candidate program terms, and checks if they satisfy the specification. Unlike LEON, we support a complete handling of quantifiers via superposition reasoning. JENNISYS uses a verifier to generate input-output examples, which differs from our setting of using inductive formulas as logical specifications. BURST generates programs by composition from existing ones, using quantifier-free fragments of first-order logic. Contrarily to this, we support full first-order logic and induction, without using subgoal proof strategies.

The syntax-guided synthesis (SyGuS) framework [1] supports specifications for recursive functions and can encode our examples from Section 9. However, to the best of our knowledge, SyGuS methods do not support recursive synthesis. While the semantics-guided synthesis framework [10] also supports recursive functions, its (to the best of our knowledge) only solver MESSY synthesizes programs from input-output examples rather than from logical specifications.

11 Conclusions

We extend saturation-based framework to recursive program synthesis by utilizing the constructive nature of induction axioms. We introduce magic axioms as a tracking mechanism and seamlessly integrate these axioms into saturation. We then construct correct recursive programs using answer literals in saturation, as also demonstrated by our proof-of-concept implementation. Extending our work with tailored handling of (more general) magic axioms, and respective superposition inferences, is an interesting line for future work.

Acknowledgements. We acknowledge funding from the ERC Consolidator Grant ARTIST 101002685, the TU Wien SecInt Doctoral College, the FWF SFB project SpyCoDe F8504, and the WWTF ICT22-007 grant ForSmart.

References

1. Alur, R., Bodík, R., Dallal, E., Fisman, D., Garg, P., Juniwal, G., Kress-Gazit, H., Madhusudan, P., Martin, M.M.K., Raghthaman, M., Saha, S., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-Guided Synthesis. In: Dependable Software Systems Engineering, pp. 1–25 (2015)
2. Bonacina, M.P.: A Taxonomy of Theorem-Proving Strategies. In: Artificial Intelligence Today: Recent Trends and Developments, pp. 43–84 (1999). https://doi.org/10.1007/3-540-48317-9_3
3. Green, C.: Theorem-Proving by Resolution as a Basis for Question-Answering Systems. *Machine Intelligence* **4**, 183–205 (1969)
4. Hajdu, M., Hozzová, P., Kovács, L., Schoisswohl, J., Voronkov, A.: Induction with Generalization in Superposition Reasoning. In: CICM. pp. 123–137 (2020). https://doi.org/10.1007/978-3-030-53518-6_8
5. Hajdu, M., Kovács, L., Rawson, M., Voronkov, A.: The Vampire Approach to Induction. In: Practical Aspects of Automated Reasoning (2022)
6. Hajdu, M., Hozzová, P., Kovács, L., Voronkov, A.: Induction with Recursive Definitions in Superposition. In: FMCAD. pp. 1–10 (2021). https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_34
7. Hozzová, P., Kovács, L., Norman, C., Voronkov, A.: Program Synthesis in Saturation. In: CADE. pp. 307–324 (2023)
8. Hozzová, P., Kovács, L., Voronkov, A.: Integer Induction in Saturation. In: CADE. pp. 361–377 (2021)
9. Itzhaky, S., Peleg, H., Polikarpova, N., Rowe, R.N.S., Sergey, I.: Cyclic program synthesis. In: PLDI. p. 944–959 (2021). <https://doi.org/10.1145/3453483.3454087>, <https://doi.org/10.1145/3453483.3454087>
10. Kim, J., Hu, Q., D’Antoni, L., Reps, T.: Semantics-Guided Synthesis. *Proc. ACM Program. Lang.* **5**(POPL) (2021). <https://doi.org/10.1145/3434311>, <https://doi.org/10.1145/3434311>
11. Kleene, S.: On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic* **10**, 109–124 (1945)
12. Kneuss, E., Kurač, I., Kuncak, V., Suter, P.: Synthesis Modulo Recursive Functions. In: OOPSLA. p. 407–426 (2013). <https://doi.org/10.1145/2509136.2509555>, <https://doi.org/10.1145/2509136.2509555>
13. Kovács, L., Voronkov, A.: First-Order Theorem Proving and Vampire. In: CAV. pp. 1–35 (2013)
14. Lee, R.C.T., Waldinger, R.J., Chang, C.L.: An Improved Program-Synthesizing Algorithm and Its Correctness. *Commun. ACM* (4), 211–217 (1974). <https://doi.org/10.1145/360924.360967>
15. Leino, K.R.M., Milicevic, A.: Program Extrapolation with Jennisys. In: OOPSLA. p. 411–430. OOPSLA ’12 (2012). <https://doi.org/10.1145/2384616.2384646>, <https://doi.org/10.1145/2384616.2384646>
16. Manna, Z., Waldinger, R.: Fundamentals of Deductive Program Synthesis. *IEEE Transactions on Software Engineering* **18**(8), 674–704 (1992). <https://doi.org/10.1109/32.153379>

17. Manna, Z., Waldinger, R.: A Deductive Approach to Program Synthesis. *ACM Trans. Program. Lang. Syst.* **2**(1), 90–121 (1980). <https://doi.org/10.1145/357084.357090>
18. Miltner, A., Nuñez, A.T., Brendel, A., Chaudhuri, S., Dillig, I.: Bottom-up Synthesis of Recursive Functional Programs using Angelic Execution **6**(POPL) (2022). <https://doi.org/10.1145/3498682>, <https://doi.org/10.1145/3498682>
19. Nieuwenhuis, R., Rubio, A.: Paramodulation-Based Theorem Proving. In: *Handbook of Automated Reasonings*, vol. I, pp. 371–443. Elsevier and MIT Press (2001)
20. Polikarpova, N., Kuraj, I., Solar-Lezama, A.: Program Synthesis from Polymorphic Refinement Types. *ACM SIGPLAN Notices* **51**(6), 522–538 (2016). <https://doi.org/10.1145/2980983.2908093>, <https://doi.org/10.1145/2980983.2908093>
21. Polikarpova, N., Sergey, I.: Structuring the Synthesis of Heap-Manipulating Programs **3**(POPL) (2019). <https://doi.org/10.1145/3290385>, <https://doi.org/10.1145/3290385>
22. Reger, G., Suda, M., Voronkov, A.: Unification with Abstraction and Theory Instantiation in Saturation-Based Reasoning. In: *TACAS*. pp. 3–22 (2018)
23. Reger, G., Voronkov, A.: Induction in Saturation-Based Proof Search. In: *CADE*. pp. 477–494 (2019)
24. Rybina, T., Voronkov, A.: A Decision Procedure for Term Algebras with Queues. *ACM Transactions on Computational Logic* **2**(2), 155–181 (2001). <https://doi.org/10.1145/371316.371494>
25. Srivastava, S., Gulwani, S., Foster, J.S.: From Program Verification to Program Synthesis. In: *POPL*. p. 313–326 (2010). <https://doi.org/10.1145/1706299.1706337>
26. Sutcliffe, G.: The Logic Languages of the TPTP World. *Logic Journal of the IGPL* (2022). <https://doi.org/10.1093/jigpal/jzac068>
27. Tammet, T.: Completeness of Resolution for Definite Answers. *J. of Logic and Computation* **5**(4), 449–471 (08 1995)

A Inference Rules

In this appendix we display the calculi used in the paper: the standard superposition calculus $\mathbb{S}up$, the extended version from [7], as well as the rules introduced in this paper to support reasoning with *rec*-terms.

A.1 Standard Superposition Calculus $\mathbb{S}up$

See Figure 3 for the rules of the standard superposition calculus $\mathbb{S}up$. The calculus is parametrized by a *simplification ordering* \succ on terms and a *selection function*, which selects in each non-empty clause a non-empty subset of literals. The selection function can possibly select any subset of literals. Using different selection functions and different orderings results in different derivations.

We denote selected literals by underlining them. An inference rule can be applied on the given premise(s) if the literals that are underlined in the rule are also selected in the premise(s). For a certain class of selection functions, the superposition calculus $\mathbb{S}up$ is *sound* (if \square is derived from F , then F is unsatisfiable) and *refutationally complete* (if F is unsatisfiable, then \square can be derived from it).

Superposition:			
$\frac{s \simeq t \vee C \quad \underline{L[s']} \vee D}{(L[t] \vee C \vee D)\theta}$	$\frac{s \simeq t \vee C \quad \underline{u[s']} \not\simeq u' \vee D}{(u[t] \not\simeq u' \vee C \vee D)\theta}$	$\frac{s \simeq t \vee C \quad \underline{u[s']} \simeq u' \vee D}{(u[t] \simeq u' \vee C \vee D)\theta}$	
where $\theta := \text{mgu}(s, s')$; $t\theta \not\simeq s\theta$; (first rule only) $L[s']$ is not an equality literal; and (second and third rules only) $u'\theta \not\simeq u[s']\theta$.			
Binary resolution:	Factoring:	Equality resolution:	Equality factoring:
$\frac{\underline{A} \vee C \quad \neg \underline{A'} \vee D}{(C \vee D)\theta}$	$\frac{\underline{A} \vee \underline{A'} \vee C}{(A \vee C)\theta}$	$\frac{s \not\simeq t \vee C}{C\theta}$	$\frac{s \simeq t \vee s' \simeq t' \vee C}{(s \simeq t \vee t \not\simeq t' \vee C)\theta}$
where $\theta := \text{mgu}(A, A')$.	where $\theta := \text{mgu}(A, A')$.	where $\theta := \text{mgu}(s, t)$.	where $\theta := \text{mgu}(s, s')$; $t\theta \not\simeq s\theta$; and $t'\theta \not\simeq t\theta$.

Fig. 3. Superposition calculus $\mathbb{S}up$. The underlined literals are selected.

A.2 Extended Superposition Calculus for Reasoning With Answer literals

In Figure 4 we display the calculus from [7]. It uses the notion of computable unifier based on an abstract unifier, which we recall from [7]:

- An *abstract unifier* [22] of two expressions E_1, E_2 is a pair (θ, D) such that:

Superposition (Sup):		
$\frac{s \simeq t \vee C \vee \text{ans}(r) \quad \underline{L[s'] \vee C' \vee \text{ans}(r')}}{(\underline{D} \vee L[t] \vee C \vee C' \vee \text{ans}(\text{if } s \simeq t \text{ then } r' \text{ else } r))\theta}$	$\frac{s \simeq t \vee C \vee \text{ans}(r) \quad \underline{L[s'] \vee C' \vee \text{ans}(r')}}{(\underline{D} \vee r \not\simeq r' \vee L[t] \vee C \vee C' \vee \text{ans}(r))\theta}$	$\frac{s \simeq t \vee C \vee \text{ans}(r) \quad \underline{L[s'] \vee C' \vee \text{ans}(r')}}{(\underline{D} \vee r \not\simeq r' \vee u[t] \simeq u' \vee C \vee C' \vee \text{ans}(r))\theta}$
$\frac{s \simeq t \vee C \vee \text{ans}(r) \quad \underline{u[s'] \not\simeq u' \vee C' \vee \text{ans}(r')}}{(\underline{D} \vee u[t] \not\simeq u' \vee C \vee C' \vee \text{ans}(\text{if } s \simeq t \text{ then } r' \text{ else } r))\theta}$	$\frac{s \simeq t \vee C \vee \text{ans}(r) \quad \underline{u[s'] \simeq u' \vee C' \vee \text{ans}(r')}}{(\underline{D} \vee r \not\simeq r' \vee u[t] \simeq u' \vee C \vee C' \vee \text{ans}(r))\theta}$	$\frac{s \simeq t \vee C \vee \text{ans}(r) \quad \underline{u[s'] \not\simeq u' \vee C' \vee \text{ans}(r')}}{(\underline{D} \vee r \not\simeq r' \vee u[t] \not\simeq u' \vee C \vee C' \vee \text{ans}(r))\theta}$
<p>where (θ, D) is a computable unifier of s, s' w.r.t. the argument of the answer literal in the rule conclusion (i.e. if $s \simeq t$ then r' else r for the left-column rules, and r for the others); (rules on the first line only) $L[s']$ is not an equality literal; and (rules on the second and third line only) $u'\theta \not\simeq u[s']\theta$.</p>		
Binary resolution (BR):		
$\frac{\underline{A} \vee C \vee \text{ans}(r) \quad \neg \underline{A'} \vee C' \vee \text{ans}(r')}{(\underline{D} \vee C \vee C' \vee \text{ans}(\text{if } A \text{ then } r' \text{ else } r))\theta}$	$\frac{\underline{A} \vee C \vee \text{ans}(r) \quad \neg \underline{A'} \vee C' \vee \text{ans}(r')}{(\underline{D} \vee r \not\simeq r' \vee C \vee C' \vee \text{ans}(r))\theta}$	
<p>where (θ, D) is a computable unifier of A, A' w.r.t. (first rule) if A then r' else r or (second rule) r.</p>		
Factoring (F):	Equality resolution (ER):	Equality factoring (EF):
$\frac{\underline{A} \vee \underline{A'} \vee C \vee \text{ans}(r)}{(\underline{D} \vee A \vee C \vee \text{ans}(r))\theta}$	$\frac{s \not\simeq t \vee C \vee \text{ans}(r)}{(\underline{D} \vee C \vee \text{ans}(r))\theta}$	$\frac{s \simeq t \vee s' \simeq t' \vee C \vee \text{ans}(r)}{(\underline{D} \vee s \simeq t \vee t \not\simeq t' \vee C \vee \text{ans}(r))\theta}$
<p>where (θ, D) is a computable unifier of A, A' w.r.t. r.</p>	<p>where (θ, D) is a computable unifier of s, t w.r.t. r.</p>	<p>where (θ, D) is a computable unifier of s, s' w.r.t. r; $t\theta \not\simeq s\theta$; and $t'\theta \not\simeq t\theta$.</p>

Fig. 4. Selected rules of the extended superposition calculus Sup for reasoning with answer literals [7], with underlined literals being selected.

1. θ is a substitution and D is a (possibly empty) disjunction of disequalities,
2. $(D \vee E_1 \simeq E_2)\theta$ is valid in the underlying theory.

Intuitively speaking, an abstract unifier combines disequality constraints D with a substitution θ such that the substitution is a unifier of E_1, E_2 if the constraints D are not satisfied.

- A *computable unifier* [7] of two expressions E_1, E_2 with respect to an expression E_3 is an abstract unifier (θ, D) of E_1, E_2 such that the expression $E_3\theta$ is computable.

For example, let f be computable and g uncomputable. Then $(\{y \mapsto f(z)\}, z \not\simeq g(x))$ is a computable unifier of the terms $f(g(x)), y$ with respect to $f(y)$. Further, $(\{y \mapsto f(g(x))\}, \emptyset)$ is an abstract unifier of the same terms, but not a computable unifier with respect to $f(y)$.

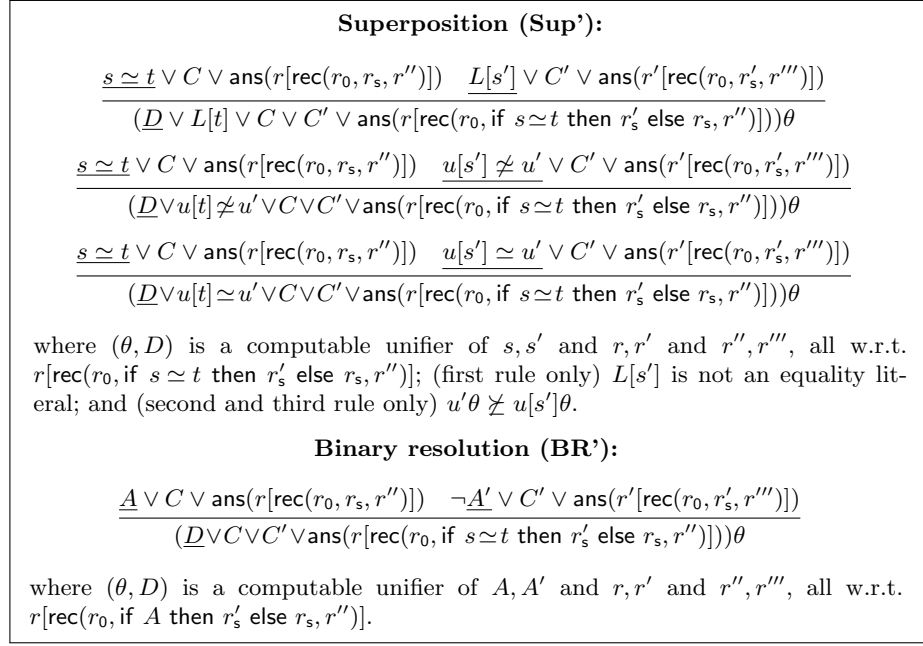


Fig. 5. Newly introduced rules of the extended superposition calculus Sup for reasoning with answer literals with rec-terms. The underlined literals are selected.

A.3 Additional Rules for Reasoning with rec-terms

In Figure 5 we introduce a variation of the superposition and binary resolution rules for reasoning with answer literals containing rec-terms. They apply when both premises have answer literals which are unifiable except for the second argument of a rec-term they contain. In this case the rules trigger to produce a clause containing an if–then–else in the second argument of the rec-term in the answer literal. These rules are useful when the BR and Sup rules from the left-hand column of Figure 4 do not apply, because the condition of the if–then–else is not computable outside of the second argument of the rec-term – i.e., when the condition contains skolem constants associated with the rec-term. Applying these new rules results into conditions which are local to the recursive branch of the synthesized recursive function, and use the argument of the recursive call, or the result of the recursive call.

B Lemma and Theorem Proofs

Lemma 2 (Recursive Witness). The expression $R(u_0, \lambda y, w.u_s)(z)$ is a witness for the variable x in (9).

Proof. Let us consider the following formula obtained by replacing x in (9) by $R(u_0, \lambda y, w.u_s)(z)$:

$$\exists y, w. \forall u_0, u_s, z. \left((G[0, u_0] \wedge (G[y, w] \rightarrow G[s(y), u_s])) \rightarrow G[z, R(u_0, \lambda y, w.u_s)(z)] \right) \quad (18)$$

We will prove that every interpretation I is a model of (18).

By contradiction, let us assume that (18) is false in I . That would mean that for all values a, b and an interpretation $I\{y \mapsto a, w \mapsto b\}$, there exists an extension $J_{a,b}$ of $I\{y \mapsto a, w \mapsto b\}$ by $\{u_0 \mapsto v_0, u_s \mapsto v_s, z \mapsto v_z\}$ for some values v_0, v_s, v_z such that:

$$\left((G[0, u_0] \wedge (G[y, w] \rightarrow G[s(y), u_s])) \rightarrow G[z, R(u_0, \lambda y, w.u_s)(z)] \right)^{J_{a,b}} = \perp \quad (19)$$

This means that for any values a, b :⁹

$$G^{J_{a,b}}[0, u_0^{J_{a,b}}] = \top \quad (20)$$

$$G^{J_{a,b}}[a, b] = \perp \quad \text{or} \quad G^{J_{a,b}}[s(a), u_s^{J_{a,b}}] = \top \quad (21)$$

$$G^{J_{a,b}}[z^{J_{a,b}}, R^{J_{a,b}}(u_0^{J_{a,b}}, (\lambda y, w.u_s)^{J_{a,b}})(z^{J_{a,b}})] = \perp \quad (22)$$

Since the operator R has a fixed interpretation and since the variables y, w, u_0, u_s, z do not occur in $G[x_1, x_2]$ (where x_1, x_2 are fresh), from (20)-(22) we obtain for any values a, b :

$$G^I[0, u_0^{J_{a,b}}] = \top \quad (23)$$

$$G^I[a, b] = \perp \quad \text{or} \quad G^I[s(a), u_s^{J_{a,b}}] = \top \quad (24)$$

$$G^I[z^{J_{a,b}}, R^I(u_0^{J_{a,b}}, (\lambda y, w.u_s)^{J_{a,b}})(z^{J_{a,b}})] = \perp \quad (25)$$

By definition $(\lambda y, w.u_s)^{J_{a,b}} = f_{a,b}$ where for any v_1, v_2 : $f_{a,b}(v_1, v_2) = u_s^{J_{a,b}\{y \mapsto v_1, w \mapsto v_2\}}$. Since $J_{a,b}\{y \mapsto v_1, w \mapsto v_2\} = J_{v_1, v_2}$, the function $f_{a,b}$ actually does not depend on a, b and thus we write $f \stackrel{\text{def}}{=} f_{a,b}$ and obtain:

$$f(v_1, v_2) = u_s^{J_{v_1, v_2}} \quad (26)$$

Using (26) we obtain from (24) and (25) for any a, b :

$$G^I[a, b] = \perp \quad \text{or} \quad G^I[s(a), f(a, b)] = \top \quad (27)$$

$$G^I[z^{J_{a,b}}, R^I(u_0^{J_{a,b}}, f)(z^{J_{a,b}})] = \perp \quad (28)$$

To simplify the notation, we now consider (28) for arbitrarily fixed values $a := v_y, b := v_w$. We denote $v_z := z^{J_{v_y, v_w}}, v_0 := u_0^{J_{v_y, v_w}}$ and obtain:

$$G^I[v_z, R^I(v_0, f)(v_z)] = \perp \quad (29)$$

Assume there is a smallest value of v_z such that (29) holds. Either this value is 0, or a successor of some v , i.e., $v_z = s(v)$:

⁹ In the following, we write $u_0^{J_{a,b}}, u_s^{J_{a,b}}$ instead of v_0, v_s to emphasize the dependence of the interpretation of u_0, u_s on the values a, b .

1. If $v_z = 0$, then $R^I(v_0, f)(v_z) = R^I(v_0, f)(0)$ is by definition of R equal to v_0 . Therefore,

$$\perp \stackrel{(29)}{=} G^I[0, R^I(v_0, f)(0)] = G^I[0, v_0] \stackrel{(23)}{\text{with } a:=v_y, b:=v_w}{=} \top.$$

This is a contradiction and thus it has to be the second case:

2. $v_z = s(v)$, therefore from (29):

$$G^I[s(v), R^I(v_0, f)(s(v))] = \perp$$

By definition of R we have $R^I(v_0, f)(s(v)) = f(v, R^I(v_0, f)(v))$, and thus:

$$G^I[s(v), f(v, R^I(v_0, f)(v))] = \perp \quad (30)$$

From (27) with $a := v, b := R^I(v_0, f)(v)$ we obtain:

$$G^I[v, R^I(v_0, f)(v)] = \perp \quad \text{or} \quad G^I[s(v), f(v, R^I(v_0, f)(v))] = \top$$

From that and (30) we get:

$$G^I[v, R^I(v_0, f)(v)] = \perp$$

That is, v satisfies (29), which contradicts the assumption that $s(v)$ is the smallest value satisfying (29).

Therefore, there is no value v_z satisfying (29). Thus, since the argument above works for arbitrary v_y, v_w , the formula from (19) cannot be false in any $J_{a,b}$. This means that (18) cannot be false in any I , meaning that it is valid and $R(u_0, \lambda y, w.u_s)(z)$ is indeed a witness for the variable x in (9). \square

Lemma 7 (Recursive Witness for Magic Formulas). Consider the formula obtained from (10) by replacing $\text{rec}_{G[t,x]}(u_0, u_s, z)$ by its corresponding recursive function term $R(u_0, \lambda\sigma_y, \sigma_w.u_s)(z)$:

$$\forall u_0, u_s, z. \left((G[0, u_0] \wedge (G[\sigma_y, \sigma_w] \rightarrow G[s(\sigma_y), u_s])) \rightarrow G[z, R(u_0, \lambda\sigma_y, \sigma_w.u_s)(z)] \right) \quad (11)$$

For every interpretation I , there exists a mapping of skolem constants to values $\{\sigma_y \mapsto v_y, \sigma_w \mapsto v_w\}$ such that I extended by this mapping is a model of (11). As a consequence, formula (11) is satisfiable.

Proof. The lemma immediately follows from the fact that formula (10) is a skolemization of

$$\exists y, w. \forall u_0, u_s, z. \left((G[0, u_0] \wedge (G[y, w] \rightarrow G[s(y), u_s])) \rightarrow G[z, R(u_0, \lambda y, w.u_s)(z)] \right),$$

which is by Lemma 2 valid. \square

Theorem 8 (Semantics of Clauses with Answer Literals and rec-terms).

Let C_1, \dots, C_m be clauses and F a formula containing no answer literals and no rec-symbols. Let C be a clause containing no answer literals. Let M_1, \dots, M_l be magic formulas. Assume that using a sound rec-compliant inference system \mathcal{I} , we derive $C \vee \text{ans}(r[\bar{\sigma}])$, where $r[\bar{\sigma}]$ is computable, from the set of clauses

$$\{ C_1, \dots, C_m, M_1, \dots, M_l, \text{cnf}(\neg F[\bar{\sigma}, y] \vee \text{ans}(y)) \}.$$

Then

$$M_1^R, \dots, M_l^R, C_1, \dots, C_m \vdash C^R, F[\bar{\sigma}, r^R[\bar{\sigma}]].$$

That is, under the assumptions $M_1^R, \dots, M_l^R, C_1, \dots, C_m, \neg C^R$, the computable expression $r^R[\bar{x}]$ is a witness for y in $\forall \bar{x} \exists y. F[\bar{x}, y]$.

Proof. The proof mirrors the proof of Theorem 1 of [7].

We consider the calculus which was used for deriving $C \vee \text{ans}(r[\bar{\sigma}])$, but with lifted ordering and selection constraints. Since the soundness of the calculus does not depend on these constraints, the calculus without the constraints is sound as well. Now, since ans is uninterpreted, we can replace $\text{ans}(y)$ by $y \not\approx r^R[\bar{\sigma}]$. Further, since also all rec-symbols are uninterpreted, we can also replace each $\text{rec}(\bar{u}, z)$ in each induction formula M_i by its corresponding recursive function term. Since the calculus is rec-compliant, all rec-terms were introduced by the induction formulas M_1, \dots, M_l , and therefore after the replacements we obtain a derivation of $C^R \vee r^R[\bar{\sigma}] \not\approx r^R[\bar{\sigma}]$ from $\forall y. \text{cnf}(\neg F[\bar{\sigma}, y] \vee y \not\approx r^R[\bar{\sigma}]), M_1^R, \dots, M_l^R, C_1, \dots, C_m$ using the calculus without the constraints.¹⁰

We want to show that

$$\bigwedge_{i=1}^l M_i^R \wedge \bigwedge_{i=1}^m C_i \rightarrow C^R \vee F[\bar{\sigma}, r^R[\bar{\sigma}]] \quad (31)$$

is valid. Hence, we need to show that in each interpretation, in which the antecedent is true, also the consequent is true. Let us consider such an interpretation I . We distinguish two cases:

1. First, assume that $\forall y. \text{cnf}(\neg F[\bar{\sigma}, y] \vee y \not\approx r^R[\bar{\sigma}])$ is true in I . Then since all assumptions from which we derived $C^R \vee r^R[\bar{\sigma}] \not\approx r^R[\bar{\sigma}]$ are true in I and since the inference system is sound, also $C^R \vee r^R[\bar{\sigma}] \not\approx r^R[\bar{\sigma}]$ is true. That clause is equivalent to C^R , hence C^R is true, which makes the consequent of (31) true.
2. Second, assume that $\forall y. \text{cnf}(\neg F[\bar{\sigma}, y] \vee y \not\approx r^R[\bar{\sigma}])$ is false in I . Then its negation, $\neg \forall y. \text{cnf}(\neg F[\bar{\sigma}, y] \vee y \not\approx r^R[\bar{\sigma}])$, equivalent to $\exists y. (F[\bar{\sigma}, y] \wedge y \simeq r^R[\bar{\sigma}])$, equivalent to $F[\bar{\sigma}, r^R[\bar{\sigma}]]$ must be true in I . Hence, the consequent of (31) is true also in this case.

¹⁰ The derivation might not have been possible in the calculus with the ordering and selection constraints due to replacing the positive literal $\text{ans}(y)$ with the negative literal $y \not\approx r^R[\bar{\sigma}]$ containing different symbols, and replacing rec-terms by terms with R and λ .

Therefore (31) is valid. Since $\bar{\sigma}$ are fresh uninterpreted constants, we obtain that $\bigwedge_{i=1}^l M_i^R \wedge \bigwedge_{i=1}^m C_i \rightarrow C^R \vee F[\bar{x}, r^R[\bar{x}]]$ is valid too, and hence $r^R[\bar{x}]$ is a witness for $\forall \bar{x}. \exists y. F[\bar{x}, y]$ under the assumptions $M_1^R, \dots, M_l^R, C_1, \dots, C_m, \neg C^R$.

Finally, note that since $r[\bar{\sigma}]$ is computable, so is $r[\bar{x}]$. The only skolem constants $r[\bar{x}]$ contains are skolem constants within the respective arguments of rec-terms they are associated with. Since $r^R[\bar{x}]$ lambda-binds exactly those skolem constants from the rec-terms, we have that $r^R[\bar{x}]$ is computable too. \square

Theorem 9 (Recursive Programs). Let $r[\bar{\sigma}]$ be a computable term, and $C[\bar{\sigma}], C_1[\bar{\sigma}], \dots, C_m[\bar{\sigma}]$ be ground computable clauses containing no answer literals and no rec-symbols. Assume that using a sound rec-compliant inference system \mathcal{I} , we derive the clause $C[\bar{\sigma}] \vee \text{ans}(r[\bar{\sigma}])$ from the CNF of

$$\{ A_1, \dots, A_n, C_1[\bar{\sigma}], \dots, C_m[\bar{\sigma}], M_1, \dots, M_l, \neg F[\bar{\sigma}, y] \vee \text{ans}(y) \}$$

where M_1, \dots, M_l are magic formulas. Then,

$$\langle r^R[\bar{x}], \bigwedge_{j=1}^m C_j[\bar{x}] \wedge \neg C[\bar{x}] \rangle$$

is a program with conditions for (3).

Proof. From Theorem 8 follows that

$$\bigwedge_{i=1}^n A_i \wedge \bigwedge_{i=1}^l M_i^R \wedge \bigwedge_{i=1}^m C_i[\bar{\sigma}] \rightarrow C^R[\bar{\sigma}] \vee F[\bar{\sigma}, r^R[\bar{\sigma}]] \quad (32)$$

is valid. Since $C[\bar{\sigma}]$ does not contain any rec-terms, $C^R[\bar{\sigma}] = C[\bar{\sigma}]$. We can therefore equivalently rewrite (32) as

$$\bigwedge_{i=1}^l M_i^R \rightarrow \left(\bigwedge_{i=1}^n A_i \wedge \bigwedge_{i=1}^m C_i[\bar{\sigma}] \rightarrow C[\bar{\sigma}] \vee F[\bar{\sigma}, r^R[\bar{\sigma}]] \right). \quad (33)$$

Let us consider an arbitrary interpretation I of

$$\bigwedge_{i=1}^n A_i \wedge \bigwedge_{i=1}^m C_i[\bar{\sigma}] \rightarrow C[\bar{\sigma}] \vee F[\bar{\sigma}, r^R[\bar{\sigma}]]. \quad (34)$$

From Lemma 7 follows that we can extend I to I' which is a model of $\bigwedge_{i=1}^l M_i^R$ by choosing suitable values for skolem constants in each M_i (each of these skolems only occurs in one M_i , and they do not occur in (34)). Since (33) is valid and $\bigwedge_{i=1}^l M_i^R$ is true in I' , also (34) has to be true in I' . However, since (34) does not contain any of those skolem constants by which I was extended to I' , we get that (34) is also true in I . Therefore (34) is valid.

Next, since $\bar{\sigma}$ are fresh uninterpreted constants, we obtain that the formula $\bigwedge_{i=1}^n A_i \wedge \bigwedge_{i=1}^m C_i[\bar{x}] \rightarrow C[\bar{x}] \vee F[\bar{x}, r^R[\bar{x}]]$ is valid as well, and this formula is equivalent to $\bigwedge_{j=1}^m C_j[\bar{x}] \wedge \neg C[\bar{x}] \rightarrow (\bigwedge_{i=1}^n A_i \rightarrow F[\bar{x}, r^R[\bar{x}]])$. Therefore $\langle r^R[\bar{x}], \bigwedge_{j=1}^m C_j[\bar{x}] \wedge \neg C[\bar{x}] \rangle$ is a program with conditions for $A_1 \wedge \dots \wedge A_n \rightarrow \forall \bar{x}. \exists y. F[\bar{x}, y]$. \square

Theorem 10 (Recursive Program Synthesis). Let $P_1[\bar{x}], \dots, P_k[\bar{x}]$, where $P_i[\bar{x}] = \langle r_i^R[\bar{x}], \bigwedge_{j=1}^{i-1} C_j[\bar{x}] \wedge \neg C_i[\bar{x}] \rangle$, be programs with conditions for (3), such that $\bigwedge_{i=1}^n A_i \wedge \bigwedge_{i=1}^k C_i[\bar{x}]$ is unsatisfiable. Then the program $P[\bar{x}]$ defined as

$$\begin{aligned} P[\bar{x}] := & \text{if } \neg C_1[\bar{x}] \text{ then } r_1^R[\bar{x}] \\ & \text{else if } \neg C_2[\bar{x}] \text{ then } r_2^R[\bar{x}] \\ & \dots \\ & \text{else if } \neg C_{k-1}[\bar{x}] \text{ then } r_{k-1}^R[\bar{x}] \\ & \text{else } r_k^R[\bar{x}], \end{aligned}$$

is a program for (3).

Proof. The proof is given in the extended version of [7].

C Generalization to Arbitrary Term Algebras

In this appendix we expand on Section 8 and present all definitions, lemmas, and theorems for recursive synthesis using arbitrary term algebras. We will work with an arbitrary (possibly polymorphic) term algebra τ with constructors $\{c_1, \dots, c_n\}$, where we denote the sort of each c_i by $\tau_{i,1} \times \dots \times \tau_{i,n_{c_i}} \rightarrow \tau$, and $P_{c_i} = \{j_1, \dots, j_{|P_{c_i}|}\}$ for each $i = 1, \dots, n$. Let α be any sort.

We recall the *magic axiom for $G[t, x]$* , where $t : \tau, x : \alpha$, and by \bar{y}_c we denote $y_{c,1}, \dots, y_{c,n_c}$:

$$\left(\bigwedge_{c \in \Sigma_\tau} \forall_{i=1}^{n_c} y_{c,i} \cdot \left(\bigwedge_{j \in P_c} \exists w_{c,j} \cdot G[y_{c,j}, w_{c,j}] \right) \rightarrow \exists u_c \cdot G[c(\bar{y}_c), u_c] \right) \rightarrow \forall z. \exists x. G[z, x] \quad (15)$$

We use the magic axiom in **MagInd**, when $L[t, x]$ is a literal with the only free variable x :

$$\frac{\bar{L}[t, x] \vee C}{\left(\bigwedge_{c \in \Sigma_\tau} \forall_{i=1}^{n_c} y_{c,i} \cdot \left(\bigwedge_{j \in P_c} \exists w_{c,j} \cdot L[y_{c,j}, w_{c,j}] \right) \rightarrow \exists u_c \cdot L[c(\bar{y}_c), u_c] \right) \rightarrow \forall z. \exists x. L[z, x]} \text{(MagInd)}$$

We convert (15) to prenex normal form:

$$\begin{aligned} & \exists_{c \in \Sigma_\tau, i \in \{1, \dots, n_c\}} y_{c,i} \cdot \exists_{c \in \Sigma_\tau, k \in P_c} w_{c,k} \cdot \forall_{c \in \Sigma_\tau} u_c \cdot \forall z. \exists x. \\ & \left[\left(\bigwedge_{c \in \Sigma_\tau} \left(\bigwedge_{j \in P_c} G[y_{c,j}, w_{c,j}] \right) \rightarrow G[c(\bar{y}_c), u_c] \right) \rightarrow G[z, x] \right] \quad (35) \end{aligned}$$

We define the *primitive recursion operator* R for τ and α analogously to Definition 1:

$$\begin{aligned} R(f_1, \dots, f_n)(c_1(\bar{x})) & \simeq f_1(x_1, \dots, x_{n_{c_1}}, R(f_1, \dots, f_n)(x_{j_1}), \dots, R(f_1, \dots, f_n)(x_{j_{|P_{c_1}|}})) \\ & \dots \\ R(f_1, \dots, f_n)(c_n(\bar{x})) & \simeq f_n(x_1, \dots, x_{n_{c_n}}, R(f_1, \dots, f_n)(x_{j_1}), \dots, R(f_1, \dots, f_n)(x_{j_{|P_{c_n}|}})) \end{aligned}$$

where for each i we have $f_i : \tau_{i,1} \times \cdots \times \tau_{i,n_{c_i}} \times \alpha^{|P_{c_i}|} \rightarrow \alpha$.

Using the recursion operator \mathbf{R} , we state the analogue of Lemma 2:

Lemma 12 (Recursive Witness for Term Algebra τ). The expression

$$\mathbf{R}(\lambda_{i=1}^{n_{c_1}} y_{c_1,i} \cdot \lambda_{k \in P_{c_1}} w_{c_1,k} \cdot u_{c_1}, \dots, \lambda_{i=1}^{n_{c_n}} y_{c_n,i} \cdot \lambda_{k \in P_{c_n}} w_{c_n,k} \cdot u_{c_n})(z) \quad (36)$$

is a witness for the variable x in axiom (35).

Proof. The proof is analogous to the proof of Lemma 2. We consider an interpretation I under which (36) is not a witness for x in (35), and extend it to $J_{\bar{a}, \bar{b}}$, parametrized by values \bar{a}, \bar{b} assigned to \bar{y}, \bar{w} . Under this interpretation the antecedent of (35) is true. Hence, we obtain one assumption per each of the cases of the antecedent (corresponding to constructors of τ), similarly as we obtained (23) for 0 and (27) for \mathbf{s} in the proof of Lemma 2. We use these assumptions to refute that there is a smallest value v_z for which

$$G^I[v_z, \mathbf{R}^I(\bar{f})(v_z)] = \perp,$$

where $v_z = z^{J_{\bar{a}, \bar{b}}}$ and each element of \bar{f} is defined analogously to f in the original proof. \square

For $G[t, x]$ we introduce a distinct computable function symbol $\text{rec}_{G[t, x]} : \alpha^{n_c} \times \tau \rightarrow \alpha$. As for natural numbers, we call such symbols for any $G[t, x]$ the *rec-symbols*, and terms with a *rec-symbol* as the top-level functor the *rec-terms*.

We recall the *magic formula for $G[t, x]$* corresponding to magic axiom (15)

$$\forall_{c \in \Sigma_\tau} u_c. \forall z. \left(\bigwedge_{c \in \Sigma_\tau} \left(\bigwedge_{j \in P_c} G[\sigma_{y_{c,j}}, \sigma_{w_{c,j}}] \rightarrow G[c(\bar{\sigma}_{y_c}), u_c] \rightarrow G[z, \text{rec}_{G[t, x]}(\bar{u}, z)] \right) \right), \quad (16)$$

which uses skolem constants $\sigma_{y_{c_i,j}}, \sigma_{w_{c_i,j}}$ to skolemize the variables $y_{c_i,j}, w_{c_i,j}$, and the skolem function $\text{rec}_{G[t, x]}$ to skolemize the variable x , and where by \bar{u} we denote u_{c_1}, \dots, u_{c_n} , and by $\bar{\sigma}_{y_c}$ we denote $\sigma_{y_{c,1}}, \dots, \sigma_{y_{c,n_c}}$. As for natural numbers, we say that the skolem constants $\sigma_{y_{c_i,j}}, \sigma_{w_{c_i,j}}$ introduced in the same (16) as the $\text{rec}_{G[t, x]}$ -term are *associated with the $\text{rec}_{G[t, x]}$ -term*. Each $\sigma_{y_{c_i,j}}, \sigma_{w_{c_i,j}}$ introduced in (16) is considered computable only in the i th argument of its associated *rec-term*.

Exactly as for natural numbers, an inference system \mathcal{I} is *rec-compliant* if:

1. \mathcal{I} only introduces *rec-terms* in the instances of the magic formula (16),
2. \mathcal{I} does not introduce uncomputable symbols into arguments of *rec-terms* in clauses it derives.

When $\sigma_{y_{c_i,j}}, \sigma_{w_{c_i,j}}$ are associated with $\text{rec}(\bar{s}, t)$, then the term

$$\mathbf{R}(\lambda_{i=1}^{n_{c_1}} \sigma_{y_{c_1,i}} \cdot \lambda_{k \in P_{c_1}} \sigma_{w_{c_1,k}} \cdot s_1, \dots, \lambda_{i=1}^{n_{c_n}} \sigma_{y_{c_n,i}} \cdot \lambda_{k \in P_{c_n}} \sigma_{w_{c_n,k}} \cdot s_n)(t)$$

is the *recursive function term corresponding to $\text{rec}(\bar{s}, t)$* . As for natural numbers, for a term r , we denote by $r^{\mathbf{R}}$ the expression obtained from r by iteratively replacing all *rec-terms* by their corresponding recursive function terms, starting from the innermost ones. Similarly, formula $F^{\mathbf{R}}$ denotes the formula F in which we replace all *rec-terms* by their corresponding recursive function terms. We recall Lemma 11 from Section 8:

Lemma 11 (Recursive Witness for Magic Formulas Using τ). Consider the formula obtained from (16) by replacing $\text{rec}_{G[t,x]}(\bar{u}, z)$ by its corresponding recursive function term:

$$\begin{aligned} & \forall_{c \in \Sigma_\tau} u_c. \forall z. \left(\bigwedge_{c \in \Sigma_\tau} \left(\bigwedge_{j \in P_c} G[\sigma_{y_{c,j}}, \sigma_{w_{c,j}}] \rightarrow G[c(\overline{\sigma_{y_c}}), u_c] \right) \right. \\ & \left. \rightarrow G[z, \mathbf{R}(\lambda_{i=1}^{n_{c_1}} \sigma_{y_{c_1,i}} \cdot \lambda_{k \in P_{c_1}} \sigma_{w_{c_1,k}} \cdot u_{c_1}, \dots, \lambda_{i=1}^{n_{c_n}} \sigma_{y_{c_n,i}} \cdot \lambda_{k \in P_{c_n}} \sigma_{w_{c_n,k}} \cdot u_{c_n})(z)] \right) \end{aligned} \quad (17)$$

For every interpretation, there exists its extension by some $\{\sigma_{y_{c,i}} \mapsto v_{y,c,i}, \sigma_{w_{c,k}} \mapsto v_{w,c,k}\}_{c \in \Sigma_\tau, i \in \{1, \dots, n_c\}, k \in P_c}$ such that the extension is a model of (17). As a consequence, formula (17) is satisfiable.

Proof. Immediately follows from (17) being a skolemization of

$$\begin{aligned} & \exists_{c \in \Sigma_\tau, i \in \{1, \dots, n_c\}} y_{c,i}. \exists_{c \in \Sigma_\tau, k \in P_c} w_{c,k}. \forall_{c \in \Sigma_\tau} u_c. \forall z. \\ & \left[\left(\bigwedge_{c \in \Sigma_\tau} \left(\left(\bigwedge_{j \in P_c} G[y_{c,j}, w_{c,j}] \right) \rightarrow G[c(\overline{y_c}), u_c] \right) \right) \right. \\ & \left. \rightarrow G[z, \mathbf{R}(\lambda_{i=1}^{n_{c_1}} y_{c_1,i} \cdot \lambda_{k \in P_{c_1}} w_{c_1,k} \cdot u_{c_1}, \dots, \lambda_{i=1}^{n_{c_n}} y_{c_n,i} \cdot \lambda_{k \in P_{c_n}} w_{c_n,k} \cdot u_{c_n})(z)] \right], \end{aligned}$$

which is by Lemma 12 valid. \square

Using Lemma 11, we derive the analogues of Theorems 8–10 for an arbitrary term algebra τ . Note that since we extended the definition of a rec -compliant system and $r^{\mathbf{R}}, F^{\mathbf{R}}$ for τ , the statements and proofs of the theorems do not change.

We finally note that our synthesis method generalizes also to other sorts as term algebras, as long as the induction axiom used for the sort carries the constructive meaning described in Section 4.

D Example Derivations

In this appendix we display the definitions of the term algebra constructors, functions and predicates used in the paper (see Figure 6). We also detail specifications of the examples from Table 1 of Section 9, and show the full derivations of their programs. Note that these derivations may differ from derivations found using VAMPIRE.¹¹

In the following derivations we use definitions of functions and predicates listed in Figure 7. We also use auxiliary lemmas (listed in Figure 8), which only serve the purpose of making the derivations shorter and easier to read. Each of the lemmas can be individually proven (e.g. using superposition calculus extended with induction). For the sake of readability, we use $x \leq y$ as a syntactic macro for $\neg(y < x)$.

¹¹ All our examples as well as the instructions to run VAMPIRE are available online at https://github.com/vprover/vampire_benchmarks/tree/master/synthesis/recursive.

	Natural numbers \mathbb{N}	Natural lists \mathbb{L}	Natural binary trees \mathbb{BT}
Constructors	$0 : \mathbb{N}$ $s : \mathbb{N} \rightarrow \mathbb{N}$	$nil : \mathbb{L}$ $cons : \mathbb{N} \times \mathbb{L} \rightarrow \mathbb{L}$	$leaf : \mathbb{N} \rightarrow \mathbb{BT}$ $bt : \mathbb{BT} \times \mathbb{N} \times \mathbb{BT} \rightarrow \mathbb{BT}$
Destructors	$p : \mathbb{N} \rightarrow \mathbb{N}$ $\forall x \in \mathbb{N}. p(s(x)) \simeq x$	(omitted)	(omitted)
Symbols	$+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ $\cdot : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ $< : \mathbb{N} \times \mathbb{N} \rightarrow \{\top, \perp\}$	$++ : \mathbb{L} \times \mathbb{L} \rightarrow \mathbb{L}$ $len : \mathbb{L} \rightarrow \mathbb{N}$ $suff : \mathbb{L} \times \mathbb{L} \rightarrow \{\top, \perp\}$ $in_{\mathbb{L}} : \mathbb{N} \times \mathbb{L} \rightarrow \{\top, \perp\}$	$in_{\mathbb{BT}} : \mathbb{N} \times \mathbb{BT} \rightarrow \{\top, \perp\}$

Fig. 6. Definitions for the theories of natural numbers \mathbb{N} , lists \mathbb{L} , and binary trees \mathbb{BT} . The axioms defining function and predicate symbols can be found in Figure (7). When it is clear from the context we will just write in instead of $in_{\mathbb{L}}/in_{\mathbb{BT}}$. We omit the destructors for \mathbb{L} and \mathbb{BT} , since we do not use them in our derivations.

To shorten the derivations, the factoring rule is not stated explicitly, we also merge selected steps and indicate all applied rules in the inference tag. Notably, we merge the application of **MagInd** and the following BR (e.g., compare steps 2-5 from Example 2 with steps 2-3 from Example 4 below). We also merge the applications of BR or Sup from the right-hand column of Figure 4 with a following ER on $r \not\approx r'$ in case the answer literal arguments r, r' from the premises are computably unifiable.

For some of the examples, we use magic axioms different from (6) and (15). In those cases we state these adapted magic axioms above the derivation. For readability reasons only the axioms that are needed for deriving the empty clause will be written down in the derivation.

Finally, to highlight the important derivation steps, some literals and terms are typeset in bold. This is done either when the **MagInd** is applied on the literals, or when an inference rule applied on the literal causes a **rec**-term to change.

D.1 Theory of Natural Numbers

Example 4 (Associativity of addition).

Specification. We recall the specification:

$$\forall x_1, x_2, x_3 \in \mathbb{N}. \exists y \in \mathbb{N}. (x_1 + x_2) + x_3 \simeq x_1 + y$$

Details of the magic. We use magic axiom (6) instantiated with $G[t, x] := (\sigma_1 + \sigma_2) + t \simeq \sigma_1 + x$. The clasified magic formula used in the proof then corresponds to the following two clauses:

$$\begin{aligned} (\sigma_1 + \sigma_2) + 0 \not\approx \sigma_1 + u_0 \vee (\sigma_1 + \sigma_2) + \sigma_y \simeq \sigma_1 + \sigma_w \\ \vee (\sigma_1 + \sigma_2) + z \not\approx \sigma_1 + \mathbf{rec}(u_0, u_s, z) \end{aligned} \quad (37)$$

$$\begin{aligned} (\sigma_1 + \sigma_2) + 0 \not\approx \sigma_1 + u_0 \vee (\sigma_1 + \sigma_2) + \mathbf{s}(\sigma_y) \not\approx \sigma_1 + u_s \\ \vee (\sigma_1 + \sigma_2) + z \not\approx \sigma_1 + \mathbf{rec}(u_0, u_s, z) \end{aligned} \quad (38)$$

Derivation and program.

$\forall x \in \mathbb{N}. s(x) \neq 0$	(A1 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. x \simeq y \rightarrow s(x) \simeq s(y)$	(A2 $_{\mathbb{N}}$)
$\forall x \in \mathbb{N}. x + 0 \simeq x$	(A3 $_{\mathbb{N}}$)
$\forall x, k \in \mathbb{N}. x + s(k) \simeq s(x + k)$	(A4 $_{\mathbb{N}}$)
$\forall x \in \mathbb{N}. x \cdot 0 \simeq 0$	(A5 $_{\mathbb{N}}$)
$\forall x, n \in \mathbb{N}. x \cdot s(n) \simeq x \cdot n + x$	(A6 $_{\mathbb{N}}$)
$\forall x \in \mathbb{N}. \neg(x < 0)$	(A7 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. x < s(y) \leftrightarrow x < y \vee x \simeq y$	(A8 $_{\mathbb{N}}$)
$\forall x, y, z \in \mathbb{N}. (x < y \wedge y < z) \rightarrow x < z$	(A9 $_{\mathbb{N}}$)
$\forall x \in \mathbb{L}. x \simeq x \mathbin{++} \text{nil}$	(A1 $_{\mathbb{L}}$)
$\forall x, l \in \mathbb{L}. \forall n \in \mathbb{N}. \text{cons}(n, l) \mathbin{++} x \simeq \text{cons}(n, l \mathbin{++} x)$	(A2 $_{\mathbb{L}}$)
$\text{len}(\text{nil}) \simeq 0$	(A3 $_{\mathbb{L}}$)
$\forall l \in \mathbb{L}. \forall n \in \mathbb{N}. \text{len}(\text{cons}(n, l)) \simeq s(\text{len}(l))$	(A4 $_{\mathbb{L}}$)
$\forall l \in \mathbb{L}. \text{suff}(\text{nil}, l)$	(A5 $_{\mathbb{L}}$)
$\forall l \in \mathbb{L}. \forall n \in \mathbb{N}. \neg(\text{suff}(\text{cons}(n, l), \text{nil}))$	(A6 $_{\mathbb{L}}$)
$\forall l, l' \in \mathbb{L}. \forall n \in \mathbb{N}. \text{suff}(l', l) \rightarrow \text{suff}(l', \text{cons}(n, l))$	(A7 $_{\mathbb{L}}$)
$\forall l, l' \in \mathbb{L}. \forall n' \in \mathbb{N}. \text{suff}(\text{cons}(n', l'), l) \rightarrow \text{suff}(l', l)$	(A8 $_{\mathbb{L}}$)
$\forall l, l' \in \mathbb{L}. \forall n, n' \in \mathbb{N}. \text{cons}(n, l) = \text{cons}(n', l') \rightarrow (n = n' \wedge l = l')$	(A9 $_{\mathbb{L}}$)
$\forall n \in \mathbb{N}. \neg \text{in}_{\mathbb{L}}(n, \text{nil})$	(A10 $_{\mathbb{L}}$)
$\forall l \in \mathbb{L}. \forall n, k \in \mathbb{N}. \text{in}_{\mathbb{L}}(n, \text{cons}(k, l)) \leftrightarrow (\text{in}_{\mathbb{L}}(n, l) \vee n \simeq k)$	(A11 $_{\mathbb{L}}$)
$\forall n, k \in \mathbb{N}. \text{in}_{\mathbb{BT}}(n, \text{leaf}(k)) \leftrightarrow n \simeq k$	(A1 $_{\mathbb{BT}}$)
$\forall l, r \in \mathbb{BT}. \forall n, k \in \mathbb{N}. \text{in}_{\mathbb{BT}}(n, \text{bt}(l, k, r)) \leftrightarrow (\text{in}_{\mathbb{BT}}(n, l) \vee \text{in}_{\mathbb{BT}}(n, r) \vee n \simeq k)$	(A2 $_{\mathbb{BT}}$)

Fig. 7. List of all axioms defining function and predicate symbols and/or are used for the derivations.

1. $(\sigma_1 + \sigma_2) + \sigma_3 \neq \sigma_1 + y \vee \text{ans}(y)$ [input]
2. $(\sigma_1 + \sigma_2) + 0 \neq \sigma_1 + u_0 \vee (\sigma_1 + \sigma_2) + \sigma_y \simeq \sigma_1 + \sigma_w \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_3))$ [MagInd, BR 1]
3. $(\sigma_1 + \sigma_2) + 0 \neq \sigma_1 + u_0 \vee (\sigma_1 + \sigma_2) + s(\sigma_y) \neq \sigma_1 + u_s \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_3))$ [MagInd, BR 1]
4. $\sigma_1 + \sigma_2 \neq \sigma_1 + u_0 \vee (\sigma_1 + \sigma_2) + \sigma_y \simeq \sigma_1 + \sigma_w \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_3))$ [Sup (A3 $_{\mathbb{N}}$), 2]
5. $\sigma_1 + \sigma_2 \neq \sigma_1 + u_0 \vee (\sigma_1 + \sigma_2) + s(\sigma_y) \neq \sigma_1 + u_s \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_3))$ [Sup (A3 $_{\mathbb{N}}$), 3]
6. $(\sigma_1 + \sigma_2) + \sigma_y \simeq \sigma_1 + \sigma_w \vee \text{ans}(\text{rec}(\sigma_2, u_s, \sigma_3))$ [ER 4]
7. $(\sigma_1 + \sigma_2) + s(\sigma_y) \neq \sigma_1 + u_s \vee \text{ans}(\text{rec}(\sigma_2, u_s, \sigma_3))$ [ER 5]
8. $s((\sigma_1 + \sigma_2) + \sigma_y) \neq \sigma_1 + u_s \vee \text{ans}(\text{rec}(\sigma_2, u_s, \sigma_3))$ [Sup (A4 $_{\mathbb{N}}$), 7]

$\forall x \in \mathbb{N}. x \simeq \mathbf{0} + x$	(L1 $_{\mathbb{N}}$)
$\forall x, n \in \mathbb{N}. \mathbf{s}(x) + n \simeq x + \mathbf{s}(n)$	(L2 $_{\mathbb{N}}$)
$\forall x \in \mathbb{N}. \mathbf{0} \cdot x \simeq \mathbf{0}$	(L3 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. \mathbf{s}(x) \cdot y \simeq x \cdot y + y$	(L4 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. \mathbf{s}(x) < y \rightarrow x < y$	(L5 $_{\mathbb{N}}$)
$\forall x, y, z \in \mathbb{N}. (y + x \simeq z \wedge y < z) \rightarrow x \not\leq \mathbf{0}$	(L6 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. x < y \rightarrow x \not\leq y$	(L7 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. x < y \rightarrow (\mathbf{s}(x) < y \vee \mathbf{s}(x) \simeq y)$	(L8 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. y \leq x \rightarrow y \leq \mathbf{s}(x)$	(L9 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. y \not\leq \mathbf{0} \rightarrow x < x + y$	(L10 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. x < y \rightarrow x \leq y$	(L11 $_{\mathbb{N}}$)
$\forall x, y \in \mathbb{N}. x < y \rightarrow x \not\leq y$	(L12 $_{\mathbb{N}}$)
$\forall x, y, z \in \mathbb{N}. ((x \leq y \vee x \leq z) \wedge y \leq z) \rightarrow x \leq z$	(L13 $_{\mathbb{N}}$)
$\forall n, n', m \in \mathbb{N}. n \simeq n' \rightarrow m + n \simeq m + n'$	(L14 $_{\mathbb{N}}$)

$\forall x \in \mathbb{L}. \mathbf{nil}++x \simeq x$	(L1 $_{\mathbb{L}}$)
$\forall l, l' \in \mathbb{L}. \forall n \in \mathbb{N}. l' \not\leq \mathbf{cons}(n, l) \rightarrow (\mathbf{suff}(l', \mathbf{cons}(n, l)) \rightarrow \mathbf{suff}(l', l))$	(L2 $_{\mathbb{L}}$)
$\forall x, y, z \in \mathbb{L}. \forall n \in \mathbb{N}. (x \simeq y++z \wedge \mathbf{suff}(\mathbf{cons}(n, z), x)) \rightarrow \mathbf{suff}(\mathbf{cons}(n, \mathbf{nil}), y)$	(L3 $_{\mathbb{L}}$)
$\forall x, y \in \mathbb{L}. n \in \mathbb{N}. \mathbf{suff}(\mathbf{cons}(n, x), y) \rightarrow x \not\leq y$	(L4 $_{\mathbb{L}}$)
$\forall x, y, z \in \mathbb{L}. (x \simeq y++z \wedge x \not\leq z) \rightarrow y \not\leq \mathbf{nil}$	(L5 $_{\mathbb{L}}$)
$\forall x, y, z \in \mathbb{L}. x \simeq y \rightarrow x++z \simeq y++z$	(L6 $_{\mathbb{L}}$)

Fig. 8. List of all lemmas used for the derivations.

9. $\mathbf{s}(\sigma_1 + \sigma_w) \not\leq \sigma_1 + u_s \vee \mathbf{ans}(\mathbf{rec}(\sigma_2, u_s, \sigma_3))$ [Sup 6, 8]
10. $\sigma_1 + \mathbf{s}(\sigma_w) \not\leq \sigma_1 + u_s \vee \mathbf{ans}(\mathbf{rec}(\sigma_2, u_s, \sigma_3))$ [Sup (A4 $_{\mathbb{N}}$), 9]
11. $\mathbf{ans}(\mathbf{rec}(\sigma_2, \mathbf{s}(\sigma_w), \sigma_3))$ [ER 10]
12. \square [answer literal removal 11]

Note that while a possible program would be $x_2 + x_3$, using our framework we synthesize a *syntactically* different program. Induction is applied on the literal in clause 1, producing the clauses (37) and (38), which are resolved with clause 1 to obtain the clauses 2 and 3. Now (A3 $_{\mathbb{N}}$) can be used to rewrite the first literal in clauses 2 and 3, and then we resolve the first literal of 4 and 5 with the substitution $\{u_0 \mapsto \sigma_2\}$, resulting in clause 6 and 7. Using (A4 $_{\mathbb{N}}$) twice and the substitution $\{u_s \mapsto \mathbf{s}(\sigma_w)\}$ the clause containing only the answer literal is derived. The program recorded in step 12 is $\mathbf{rec}^R(x_2, \mathbf{s}(\sigma_w), x_3) = R(x_2, \lambda\sigma_w.\mathbf{s}(\sigma_w))(x_3)$. The program is therefore

$$f(x_3),$$

where f is the recursive function defined as:

$$\begin{aligned} f(0) &\simeq x_2 \\ f(\mathbf{s}(n)) &\simeq \mathbf{s}(f(n)). \end{aligned}$$

Note that $f(n)$ computes $x_2 + n$, and therefore the program is semantically equivalent to $x_2 + x_3$. \square

Example 5 (Subtraction with condition).

Specification.

$$\forall x_1, x_2 \in \mathbb{N}. \exists y \in \mathbb{N}. (x_2 < x_1 \rightarrow x_2 + y \simeq x_1)$$

To showcase synthesis of nested recursive functions, we mark the destructor \mathbf{p} as uncomputable, effectively disallowing the solution from Table 1 on a syntactic level. As a result, we derive a recursive function \mathbf{pre} which is semantically equivalent to \mathbf{p} for all arguments other than 0.

Details of the magic. For synthesizing two linked recursive functions, induction is applied two times. Firstly, for proving $\neg(\sigma_y < \sigma_x) \vee \sigma_y + z \simeq \sigma_x$ induction needs to be applied on both literals. Therefore we use magic axiom (6) with $G[t, x] := L_1[t, x] \vee L_2[t, x]$. The CNF of the corresponding magic formula consists of the following clauses:

$$\overline{L_1}[0, u_0] \vee L_1[\sigma_y, \sigma_w] \vee L_2[\sigma_y, \sigma_w] \vee L_1[z, \mathbf{rec}(u_0, u_s, z)] \vee L_2[z, \mathbf{rec}(u_0, u_s, z)] \quad (39)$$

$$\overline{L_2}[0, u_0] \vee L_1[\sigma_y, \sigma_w] \vee L_2[\sigma_y, \sigma_w] \vee L_1[z, \mathbf{rec}(u_0, u_s, z)] \vee L_2[z, \mathbf{rec}(u_0, u_s, z)] \quad (40)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_1}[\mathbf{s}(\sigma_y), u_s] \vee L_1[z, \mathbf{rec}(u_0, u_s, z)] \vee L_2[z, \mathbf{rec}(u_0, u_s, z)] \quad (41)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[\mathbf{s}(\sigma_y), u_s] \vee L_1[z, \mathbf{rec}(u_0, u_s, z)] \vee L_2[z, \mathbf{rec}(u_0, u_s, z)] \quad (42)$$

$$\overline{L_2}[0, u_0] \vee \overline{L_1}[\mathbf{s}(\sigma_y), u_s] \vee L_1[z, \mathbf{rec}(u_0, u_s, z)] \vee L_2[z, \mathbf{rec}(u_0, u_s, z)] \quad (43)$$

$$\overline{L_2}[0, u_0] \vee \overline{L_2}[\mathbf{s}(\sigma_y), u_s] \vee L_1[z, \mathbf{rec}(u_0, u_s, z)] \vee L_2[z, \mathbf{rec}(u_0, u_s, z)] \quad (44)$$

We will denote the application of induction with this magic formula by $\text{MagInd}'_{\mathbb{N}}$.

Induction is applied a second time on the literal $L[t, x] : t \not\approx \mathbf{s}(x)$. However, this time we need to apply induction only starting with $\mathbf{s}(0)$ as the base case, using the following magic axiom:

$$\left(\exists u_0. L[\mathbf{s}(0), u_0] \wedge \forall y. (y \not\approx 0 \wedge \exists w. L[y, w] \rightarrow \exists u_s. L[\mathbf{s}(y), u_s]) \right) \rightarrow \forall z. \exists x. (z \not\approx 0 \rightarrow L[z, x])$$

Note that this magic axiom results in a different scheme for the synthesized recursive function: the base case of the function will be $\mathbf{s}(0)$, and the recursive case will be $\mathbf{s}(n)$ conditioned on $n \not\approx 0$. The CNF of the corresponding magic formula is:

$$\overline{L}[\mathbf{s}(0), u_0] \vee \sigma_y \not\approx 0 \vee z \simeq 0 \vee L[z, \mathbf{rec}(u_0, u_s, z)] \quad (45)$$

$$\overline{L}[\mathbf{s}(0), u_0] \vee L[\sigma_y, \sigma_w] \vee z \simeq 0 \vee L[z, \mathbf{rec}(u_0, u_s, z)] \quad (46)$$

$$\overline{L}[\mathbf{s}(0), u_0] \vee \overline{L}[\mathbf{s}(\sigma_y), u_s] \vee z \simeq 0 \vee L[z, \mathbf{rec}(u_0, u_s, z)] \quad (47)$$

We will denote the application of induction with this magic formula by $\text{MagInd}''_{\mathbb{N}}$.

Derivation and program. For readability, we denote the two rec -symbols used in the two magic formulas by rec_{sub} and rec_{pre} .

1. $\sigma_2 < \sigma_1 \vee \text{ans}(y)$ [input]
2. $\sigma_2 + y \not\leq \sigma_1 \vee \text{ans}(y)$ [input]
3. $\mathbf{0} + \mathbf{u}_0 \not\leq \sigma_1 \vee \neg(\sigma_y < \sigma_1) \vee \sigma_y + \sigma_w \simeq \sigma_1 \vee$
 $\vee \text{ans}(\text{rec}_{\text{sub}}(u_0, u_s, \sigma_2))$ [MagInd' $_{\mathbb{N}}$, BR 1, 2]
4. $\mathbf{0} + \mathbf{u}_0 \not\leq \sigma_1 \vee s(\sigma_y) < \sigma_1 \vee \text{ans}(\text{rec}_{\text{sub}}(u_0, u_s, \sigma_2))$ [MagInd' $_{\mathbb{N}}$, BR 1, 2]
5. $\mathbf{0} + \mathbf{u}_0 \not\leq \sigma_1 \vee s(\sigma_y) + u_s \not\leq \sigma_1 \vee \text{ans}(\text{rec}_{\text{sub}}(u_0, u_s, \sigma_2))$ [MagInd' $_{\mathbb{N}}$, BR 1, 2]
6. $\neg(\sigma_y < \sigma_1) \vee \sigma_y + \sigma_w \simeq \sigma_1 \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [Sup, ER (L1 $_{\mathbb{N}}$), 3]
7. $s(\sigma_y) < \sigma_1 \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [Sup, ER (L1 $_{\mathbb{N}}$), 4]
8. $s(\sigma_y) + u_s \not\leq \sigma_1 \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [Sup, ER (L1 $_{\mathbb{N}}$), 5]
9. $\sigma_y + s(u_s) \not\leq \sigma_1 \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [Sup (L2 $_{\mathbb{N}}$), 8]
10. $\sigma_y < \sigma_1 \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [BR (L5 $_{\mathbb{N}}$), 7]
11. $\sigma_y + \sigma_w \simeq \sigma_1 \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [BR 6, 10]
12. $\sigma_y + s(u_s) \not\leq \sigma_y + \sigma_w \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [Sup 9, 11]
13. $\sigma_w \not\leq s(u_s) \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [BR (L14 $_{\mathbb{N}}$), 12]
14. $\mathbf{s}(\mathbf{0}) \not\leq \mathbf{s}(\mathbf{u}_0) \vee \sigma_y \simeq s(\sigma'_w) \vee \sigma_w \simeq \mathbf{0} \vee$
 $\vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, \text{rec}_{\text{pre}}(u_0, u_s, \sigma_w), \sigma_2))$ [MagInd'' $_{\mathbb{N}}$, BR 13]
15. $\mathbf{s}(\mathbf{0}) \not\leq \mathbf{s}(\mathbf{u}_0) \vee s(\sigma_y) \not\leq s(u_s) \vee \sigma_w \simeq \mathbf{0} \vee$
 $\vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, \text{rec}_{\text{pre}}(u_0, u_s, \sigma_w), \sigma_2))$ [MagInd'' $_{\mathbb{N}}$, BR 13]
16. $\sigma_y \simeq s(\sigma'_w) \vee \sigma_w \simeq \mathbf{0} \vee$
 $\vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, \text{rec}_{\text{pre}}(\mathbf{0}, u_s, \sigma_w), \sigma_2))$ [ER 14]
17. $\mathbf{s}(\sigma_y) \not\leq \mathbf{s}(u_s) \vee \sigma_w \simeq \mathbf{0} \vee$
 $\vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, \text{rec}_{\text{pre}}(\mathbf{0}, u_s, \sigma_w), \sigma_2))$ [ER 15]
18. $\mathbf{s}(\sigma_y) \simeq \mathbf{s}(s(\sigma'_w)) \vee \sigma_w \simeq \mathbf{0} \vee$
 $\vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, \text{rec}_{\text{pre}}(\mathbf{0}, u_s, \sigma_w), \sigma_2))$ [BR (A2 $_{\mathbb{N}}$), 16]
19. $\sigma_w \simeq \mathbf{0} \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, \text{rec}_{\text{pre}}(\mathbf{0}, \mathbf{s}(\sigma'_w), \sigma_w), \sigma_2))$ [BR 17, 18, ER]
20. $\sigma_w \not\leq \mathbf{0} \vee \text{ans}(\text{rec}_{\text{sub}}(\sigma_1, u_s, \sigma_2))$ [BR (L6 $_{\mathbb{N}}$), 10, 11]
21. $\text{ans}(\text{rec}_{\text{sub}}(\sigma_1, \text{rec}_{\text{pre}}(\mathbf{0}, \mathbf{s}(\sigma'_w), \sigma_w), \sigma_2))$ [BR 19, 20, ER]
22. \square [answer literal removal 21]

Induction is applied on both clauses 1 and 2, where $L_1[t, x] := \neg(t < \sigma_x)$ and $L_2[t, x] := t + z \simeq \sigma_x$. This results in six clauses, three of which we omit in the derivation, because they are not used further. Using lemmas from Figure 8, we derive 13, on which we then again apply induction. After termination the rec_{sub} -term inside the answer literal is translated into

$$\text{rec}_{\text{sub}}(x_1, \text{rec}_{\text{pre}}(\mathbf{0}, \mathbf{s}(\sigma'_w), \sigma_w), x_2)^{\text{R}} = \text{R}(x_1, \lambda\sigma_w. \text{R}'(\mathbf{0}, \lambda\sigma'_w. \mathbf{s}(\sigma'_w))(\sigma_w))(x_2).$$

The program we end up with is therefore

$$\text{sub}(x_2),$$

where sub is the recursive function defined as

$$\begin{aligned} \text{sub}(0) &\simeq x_1 \\ \text{sub}(s(n)) &\simeq \text{pre}(\text{sub}(n)), \end{aligned}$$

and pre is the recursive function defined as

$$\begin{aligned} \text{pre}(s(0)) &\simeq 0, \\ n \neq 0 &\rightarrow \text{pre}(s(n)) \simeq s(\text{pre}(n)). \end{aligned}$$

□

Example 6 (Floored square root).

Specification.

$$\forall x \exists y. (y \cdot y \leq x \wedge x < s(y) \cdot s(y))$$

Details of the magic. For this example and for Example 7, the specification contains two unit clauses. In order to derive the synthesized program we use a magic axiom with $G[t, x] := L_1[t, x] \wedge L_2[t, x]$. The CNF of its magic formula is:

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee L_1[\sigma_y, \sigma_w] \vee L_1[z, \text{rec}(u_0, u_s, z)] \quad (48)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee L_1[\sigma_y, \sigma_w] \vee L_2[z, \text{rec}(u_0, u_s, z)] \quad (49)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee L_2[\sigma_y, \sigma_w] \vee L_1[z, \text{rec}(u_0, u_s, z)] \quad (50)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee L_2[\sigma_y, \sigma_w] \vee L_2[z, \text{rec}(u_0, u_s, z)] \quad (51)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee \overline{L_1}[s(\sigma_y), u_s] \vee \overline{L_2}[s(\sigma_y), u_s] \vee L_1[z, \text{rec}(u_0, u_s, z)] \quad (52)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee \overline{L_1}[s(\sigma_y), u_s] \vee \overline{L_2}[s(\sigma_y), u_s] \vee L_2[z, \text{rec}(u_0, u_s, z)] \quad (53)$$

Note that after we resolve the formulas above with a premise $\overline{L_1}[t, x] \vee \overline{L_2}[t, x] \vee C \vee \text{ans}(r[x])$, we obtain only three clauses:

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee L_1[\sigma_y, \sigma_w] \vee C \vee \text{ans}(r[\text{rec}(u_0, u_s, t)]) \quad (54)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee L_2[\sigma_y, \sigma_w] \vee C \vee \text{ans}(r[\text{rec}(u_0, u_s, t)]) \quad (55)$$

$$\overline{L_1}[0, u_0] \vee \overline{L_2}[0, u_0] \vee \overline{L_1}[s(\sigma_y), u_s] \vee \overline{L_2}[s(\sigma_y), u_s] \vee C \vee \text{ans}(r[\text{rec}(u_0, u_s, t)]) \quad (56)$$

We will denote the application of induction with this magic formula by $\text{MagInd}_{\mathbb{N}}'''$.

For the derivation, we instantiate the magic formula with $G[t, y] := y \cdot y \leq t \wedge t < s(y) \cdot s(y)$.

Derivation and program.

1. $\sigma_x < \mathbf{y} \cdot \mathbf{y} \vee \neg(\sigma_x < \mathbf{s}(\mathbf{y}) \cdot \mathbf{s}(\mathbf{y})) \vee \text{ans}(y)$ [input]
2. $0 < u_0 \cdot u_0 \vee \neg(0 < s(u_0) \cdot s(u_0)) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_w) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_x))$ [MagInd $'''_{\mathbb{N}}$, BR 1]
3. $0 < u_0 \cdot u_0 \vee \neg(0 < s(u_0) \cdot s(u_0)) \vee \sigma_y < s(\sigma_w) \cdot s(\sigma_w) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_x))$ [MagInd $'''_{\mathbb{N}}$, BR 1]

4. $0 < u_0 \cdot u_0 \vee \neg(0 < s(u_0) \cdot s(u_0)) \vee s(\sigma_y) < u_s \cdot u_s \vee$
 $\vee \neg(s(\sigma_y) < s(u_s) \cdot s(u_s)) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_x))$ [MagInd $'''_{\mathbb{N}}$, BR 1]
5. $0 \not\leq u_0 \cdot u_0 \vee \neg(0 < s(u_0) \cdot s(u_0)) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_w) \vee$
 $\vee \text{ans}(\text{rec}(u_0, u_s, \sigma_x))$ [BR (L7 $_{\mathbb{N}}$), 2]
6. $0 \not\leq u_0 \cdot u_0 \vee \neg(0 < s(u_0) \cdot s(u_0)) \vee \sigma_y < s(\sigma_w) \cdot s(\sigma_w) \vee$
 $\vee \text{ans}(\text{rec}(u_0, u_s, \sigma_x))$ [BR (L7 $_{\mathbb{N}}$), 3]
7. $0 \not\leq u_0 \cdot u_0 \vee \neg(0 < s(u_0) \cdot s(u_0)) \vee s(\sigma_y) < u_s \cdot u_s \vee$
 $\vee \neg(s(\sigma_y) < s(u_s) \cdot s(u_s)) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_x))$ [BR (L7 $_{\mathbb{N}}$), 4]
8. $\neg(0 < s(0) \cdot s(0)) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_w) \vee$
 $\vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A5 $_{\mathbb{N}}$), 5]
9. $\neg(0 < s(0) \cdot s(0)) \vee \sigma_y < s(\sigma_w) \cdot s(\sigma_w) \vee$
 $\vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A5 $_{\mathbb{N}}$), 6]
10. $\neg(0 < s(0) \cdot s(0)) \vee s(\sigma_y) < u_s \cdot u_s \vee$
 $\vee \neg(s(\sigma_y) < s(u_s) \cdot s(u_s)) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A5 $_{\mathbb{N}}$), 7]
11. $\neg(0 < s(0) \cdot 0 + s(0)) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_w) \vee$
 $\vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A6 $_{\mathbb{N}}$), 8]
12. $\neg(0 < s(0) \cdot 0 + s(0)) \vee \sigma_y < s(\sigma_w) \cdot s(\sigma_w) \vee$
 $\vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A6 $_{\mathbb{N}}$), 9]
13. $\neg(0 < s(0) \cdot 0 + s(0)) \vee s(\sigma_y) < u_s \cdot u_s \vee$
 $\vee \neg(s(\sigma_y) < s(u_s) \cdot s(u_s)) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A6 $_{\mathbb{N}}$), 10]
14. $\neg(0 < 0 + s(0)) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_w) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A5 $_{\mathbb{N}}$), 11]
15. $\neg(0 < 0 + s(0)) \vee \sigma_y < s(\sigma_w) \cdot s(\sigma_w) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A5 $_{\mathbb{N}}$), 12]
16. $\neg(0 < 0 + s(0)) \vee s(\sigma_y) < u_s \cdot u_s \vee \neg(s(\sigma_y) < s(u_s) \cdot s(u_s)) \vee$
 $\vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [ER (A5 $_{\mathbb{N}}$), 13]
17. $\neg(0 < s(0)) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_w) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [Sup (L1 $_{\mathbb{N}}$), 14]
18. $\neg(0 < s(0)) \vee \sigma_y < s(\sigma_w) \cdot s(\sigma_w) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [Sup (L1 $_{\mathbb{N}}$), 15]
19. $\neg(0 < s(0)) \vee s(\sigma_y) < u_s \cdot u_s \vee \neg(s(\sigma_y) < s(u_s) \cdot s(u_s)) \vee$
 $\vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [Sup (L1 $_{\mathbb{N}}$), 16]
20. $\neg(\sigma_y < \sigma_w \cdot \sigma_w) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [BR (A8 $_{\mathbb{N}}$), 17]
21. $\sigma_y < s(\sigma_w) \cdot s(\sigma_w) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [BR (A8 $_{\mathbb{N}}$), 18]
22. $s(\sigma_y) < u_s \cdot u_s \vee \neg(s(\sigma_y) < s(u_s) \cdot s(u_s)) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [BR (A8 $_{\mathbb{N}}$), 19]
23. $s(\sigma_y) < s(\sigma_w) \cdot s(\sigma_w) \vee s(\sigma_y) \simeq s(\sigma_w) \cdot s(\sigma_w) \vee$
 $\vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [BR (L8 $_{\mathbb{N}}$), 21]
24. $s(\sigma_y) < \sigma_w \cdot \sigma_w \vee s(\sigma_y) \simeq s(\sigma_w) \cdot s(\sigma_w) \vee \text{ans}(\text{rec}(0, \sigma_w, \sigma_x))$ [BR 22, 23, ER]
25. $\neg(s(\sigma_y) < \sigma_w \cdot \sigma_w) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [BR (L9 $_{\mathbb{N}}$), 20]
26. $s(\sigma_y) \simeq s(\sigma_w) \cdot s(\sigma_w) \vee \text{ans}(\text{rec}(0, \sigma_w, \sigma_x))$ [BR 24, 25]
27. $s(\sigma_y) \not\leq u_s \cdot u_s \vee \neg(s(\sigma_y) < s(u_s) \cdot s(u_s)) \vee \text{ans}(\text{rec}(0, u_s, \sigma_x))$ [BR (L7 $_{\mathbb{N}}$), 22]
28. $s(\sigma_y) < s(\sigma_w) \cdot s(\sigma_w) \vee \neg(s(\sigma_y) < s(s(\sigma_w)) \cdot s(s(\sigma_w))) \vee$
 $\vee \text{ans}(\text{rec}(0, s(\sigma_w), \sigma_x))$ [BR 23, 27, ER]
29. $s(\sigma_y) < s(\sigma_w) \cdot s(\sigma_w) \vee$
 $\vee \neg(s(\sigma_w) \cdot s(\sigma_w) < s(s(\sigma_w)) \cdot s(s(\sigma_w))) \vee$
 $\vee \text{ans}(\text{rec}(0, s(\sigma_w), \sigma_x))$ [Sup 23, 28]
30. $s(\sigma_y) < s(\sigma_w) \cdot s(\sigma_w) \vee$
 $\vee \neg(s(\sigma_w) \cdot s(\sigma_w) < s(s(\sigma_w)) \cdot s(\sigma_w) + s(s(\sigma_w))) \vee$
 $\vee \text{ans}(\text{rec}(0, s(\sigma_w), \sigma_x))$ [Sup (A6 $_{\mathbb{N}}$), 29]

31. $s(\sigma_y) < s(\sigma_w) \cdot s(\sigma_w) \vee$
 $\vee \neg(s(\sigma_w) \cdot s(\sigma_w) < s(\sigma_w) \cdot s(\sigma_w) + s(\sigma_w) + s(s(\sigma_w))) \vee$
 $\vee \text{ans}(\text{rec}(0, s(\sigma_w), \sigma_x))$ [Sup (L4 $_{\mathbb{N}}$), 30]
32. $s(\sigma_w) \cdot s(\sigma_w) < s(\sigma_w) \cdot s(\sigma_w) + s(\sigma_w)$ [BR (A1 $_{\mathbb{N}}$), (L10 $_{\mathbb{N}}$)]
33. $s(\sigma_w) \cdot s(\sigma_w) + s(\sigma_w) < s(\sigma_w) \cdot s(\sigma_w) + s(\sigma_w) + s(s(\sigma_w))$ [BR (A1 $_{\mathbb{N}}$), (L10 $_{\mathbb{N}}$)]
34. $s(\sigma_w) \cdot s(\sigma_w) < s(\sigma_w) \cdot s(\sigma_w) + s(\sigma_w) + s(s(\sigma_w))$ [BR (A9 $_{\mathbb{N}}$), 32, 33]
35. $s(\sigma_y) < s(\sigma_w) \cdot s(\sigma_w) \vee \text{ans}(\text{rec}(0, s(\sigma_w), \sigma_x))$ [BR 31, 34]
36. $s(\sigma_y) \not\approx s(\sigma_w) \cdot s(\sigma_w) \vee \text{ans}(\text{rec}(0, s(\sigma_w), \sigma_x))$ [BR (L7 $_{\mathbb{N}}$), 35]
37. $\text{ans}(\text{rec}(0, \text{if } s(\sigma_y) \simeq s(\sigma_w) \cdot s(\sigma_w) \text{ then } s(\sigma_w) \text{ else } \sigma_w, \sigma_x))$ [BR' 26, 36]
38. \square [answer literal removal 37]

Note that in step 37 we use BR' from Figure 5, which introduces an if–then–else into the rec-term. The synthesized program is therefore

$$f(x),$$

where f is the recursive function defined as

$$\begin{aligned} f(0) &\simeq 0 \\ f(s(n)) &\simeq \text{if } s(n) \simeq s(f(n)) \cdot s(f(n)) \text{ then } s(f(n)) \text{ else } f(n). \end{aligned}$$

□

Example 7 (Floored division).

Specification. Note that the specification is very similar to the previous one, with an additional condition.

$$\forall x_1, x_2 \exists y. (x_2 \neq 0 \rightarrow (y \cdot x_2 \leq x_1 \wedge x_1 < s(y) \cdot x_2))$$

Details of the magic. We use the same rule $\text{MagInd}'_{\mathbb{N}}$ as in the previous example, this time with a magic formula instantiated with $G[t, y] := y \cdot \sigma_2 \leq t \wedge t < s(y) \cdot \sigma_2$.

Derivation and program.

1. $\sigma_2 \neq 0 \vee \text{ans}(y)$ [input]
2. $\sigma_1 < y \cdot \sigma_2 \vee \neg(\sigma_1 < s(y) \cdot \sigma_2) \vee \text{ans}(y)$ [input]
3. $0 < u_0 \cdot \sigma_2 \vee \neg(0 < s(u_0) \cdot \sigma_2) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_2) \vee$
 $\vee \text{ans}(\text{rec}(u_0, u_s, \sigma_1))$ [MagInd''' $_{\mathbb{N}}$, BR 2]
4. $0 < u_0 \cdot \sigma_2 \vee \neg(0 < s(u_0) \cdot \sigma_2) \vee \sigma_y < s(\sigma_w) \cdot \sigma_2 \vee$
 $\vee \text{ans}(\text{rec}(u_0, u_s, \sigma_1))$ [MagInd''' $_{\mathbb{N}}$, BR 2]
5. $0 < u_0 \cdot \sigma_2 \vee \neg(0 < s(u_0) \cdot \sigma_2) \vee s(\sigma_y) < u_s \cdot \sigma_2 \vee$
 $\vee \neg(s(\sigma_y) < s(u_s) \cdot \sigma_2) \vee \text{ans}(\text{rec}(u_0, u_s, \sigma_1))$ [MagInd''' $_{\mathbb{N}}$, BR 2]
6. $0 \not\approx u_0 \cdot \sigma_2 \vee \neg(0 < s(u_0) \cdot \sigma_2) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_2) \vee$
 $\vee \text{ans}(\text{rec}(u_0, u_s, \sigma_1))$ [BR (L7 $_{\mathbb{N}}$), 3]
7. $0 \not\approx u_0 \cdot \sigma_2 \vee \neg(0 < s(u_0) \cdot \sigma_2) \vee \sigma_y < s(\sigma_w) \cdot \sigma_2 \vee$
 $\vee \text{ans}(\text{rec}(u_0, u_s, \sigma_1))$ [BR (L7 $_{\mathbb{N}}$), 4]

8. $\mathbf{0} \not\leq \mathbf{u}_0 \cdot \sigma_2 \vee \neg(\mathbf{0} < \mathbf{s}(u_0) \cdot \sigma_2) \vee \mathbf{s}(\sigma_y) < u_s \cdot \sigma_2 \vee$
 $\vee \neg(\mathbf{s}(\sigma_y) < \mathbf{s}(u_s) \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(u_0, u_s, \sigma_1))$ [BR (L7 \mathbb{N}), 5]
9. $\neg(\mathbf{0} < \mathbf{s}(\mathbf{0}) \cdot \sigma_2) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [ER (L3 \mathbb{N}), 6]
10. $\neg(\mathbf{0} < \mathbf{s}(\mathbf{0}) \cdot \sigma_2) \vee \sigma_y < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [ER (L3 \mathbb{N}), 7]
11. $\neg(\mathbf{0} < \mathbf{s}(\mathbf{0}) \cdot \sigma_2) \vee \mathbf{s}(\sigma_y) < u_s \cdot \sigma_2 \vee$
 $\vee \neg(\mathbf{s}(\sigma_y) < \mathbf{s}(u_s) \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [ER (L3 \mathbb{N}), 8]
12. $\neg(\mathbf{0} < \mathbf{0} \cdot \sigma_2 + \sigma_2) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_2) \vee$
 $\vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [ER (L4 \mathbb{N}), 9]
13. $\neg(\mathbf{0} < \mathbf{0} \cdot \sigma_2 + \sigma_2) \vee \sigma_y < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee$
 $\vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [ER (L4 \mathbb{N}), 10]
14. $\neg(\mathbf{0} < \mathbf{0} \cdot \sigma_2 + \sigma_2) \vee \mathbf{s}(\sigma_y) < u_s \cdot \sigma_2 \vee$
 $\vee \neg(\mathbf{s}(\sigma_y) < \mathbf{s}(u_s) \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [ER (L4 \mathbb{N}), 11]
15. $\neg(\mathbf{0} < \mathbf{0} + \sigma_2) \vee \neg(\sigma_y < \sigma_w \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [Sup (L3 \mathbb{N}), 12]
16. $\neg(\mathbf{0} < \mathbf{0} + \sigma_2) \vee \sigma_y < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [Sup (L3 \mathbb{N}), 13]
17. $\neg(\mathbf{0} < \mathbf{0} + \sigma_2) \vee \mathbf{s}(\sigma_y) < u_s \cdot \sigma_2 \vee \neg(\mathbf{s}(\sigma_y) < \mathbf{s}(u_s) \cdot \sigma_2) \vee$
 $\vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [Sup (L3 \mathbb{N}), 14]
18. $x < x + \sigma_2 \vee \mathbf{ans}(y)$ [BR (L10 \mathbb{N}), 1]
19. $\neg(\sigma_y < \sigma_w \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [BR 15, 18]
20. $\sigma_y < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [BR 16, 18]
21. $\mathbf{s}(\sigma_y) < u_s \cdot \sigma_2 \vee \neg(\mathbf{s}(\sigma_y) < \mathbf{s}(u_s) \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [BR 17, 18]
22. $\mathbf{s}(\sigma_y) < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \mathbf{s}(\sigma_y) \simeq \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee$
 $\vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [BR (L8 \mathbb{N}), 20]
23. $\mathbf{s}(\sigma_y) < \sigma_w \cdot \sigma_2 \vee \mathbf{s}(\sigma_y) \simeq \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee$
 $\vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, \sigma_w, \sigma_1))$ [BR 21, 22]
24. $\neg(\mathbf{s}(\sigma_y) < \sigma_w \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [BR (L9 \mathbb{N}), 19]
25. $\mathbf{s}(\sigma_y) \simeq \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, \sigma_w, \sigma_1))$ [BR 23, 24]
26. $\mathbf{s}(\sigma_y) \not\leq u_s \cdot \sigma_2 \vee \neg(\mathbf{s}(\sigma_y) < \mathbf{s}(u_s) \cdot \sigma_2) \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, u_s, \sigma_1))$ [BR (L7 \mathbb{N}), 21]
27. $\mathbf{s}(\sigma_y) < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \neg(\mathbf{s}(\sigma_y) < \mathbf{s}(\mathbf{s}(\sigma_w)) \cdot \sigma_2) \vee$
 $\vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, \mathbf{s}(\sigma_w), \sigma_1))$ [BR 22, 26, ER]
28. $\mathbf{s}(\sigma_y) < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \neg(\mathbf{s}(\sigma_w) \cdot \sigma_2 < \mathbf{s}(\mathbf{s}(\sigma_w)) \cdot \sigma_2) \vee$
 $\vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, \mathbf{s}(\sigma_w), \sigma_1))$ [Sup 22, 27]
29. $\mathbf{s}(\sigma_y) < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \neg(\mathbf{s}(\sigma_w) \cdot \sigma_2 < \mathbf{s}(\sigma_w) \cdot \sigma_2 + \sigma_2) \vee$
 $\vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, \mathbf{s}(\sigma_w), \sigma_1))$ [Sup (L4 \mathbb{N}), 28]
30. $\mathbf{s}(\sigma_y) < \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, \mathbf{s}(\sigma_w), \sigma_1))$ [BR 18, 29]
31. $\mathbf{s}(\sigma_y) \not\leq \mathbf{s}(\sigma_w) \cdot \sigma_2 \vee \mathbf{ans}(\mathbf{rec}(\mathbf{0}, \mathbf{s}(\sigma_w), \sigma_1))$ [BR (L7 \mathbb{N}), 30]
32. $\mathbf{ans}(\mathbf{rec}(\mathbf{0}, \mathbf{if} \mathbf{s}(\sigma_y) \simeq \mathbf{s}(\sigma_w) \cdot \sigma_2 \mathbf{then} \mathbf{s}(\sigma_w) \mathbf{else} \sigma_w, \sigma_1))$ [BR' 25, 31]
33. \square [answer literal removal 32]

The program we obtain is

$$f(x_1),$$

where f is the recursive function defined as

$$\begin{aligned} f(\mathbf{0}) &\simeq \mathbf{0} \\ f(\mathbf{s}(n)) &\simeq \mathbf{if} \mathbf{s}(n) \simeq \mathbf{s}(f(n)) \cdot x_2 \mathbf{then} \mathbf{s}(f(n)) \mathbf{else} f(n). \end{aligned}$$

\square

D.2 Theory of Lists

Example 8 (Length of two concatenated lists).

Specification.

$$\forall x_1, x_2 \in \mathbb{L}. \exists y \in \mathbb{N}. y \simeq \text{len}(x_1 \mathbin{++} x_2)$$

To avoid the trivial solution of $\text{len}(x_1 \mathbin{++} x_2)$, we mark the symbol $\mathbin{++}$ as uncomputable.

Details of the magic. This is the first example over the theory of lists. The structure of the magic axiom we use is similar to the one for natural numbers, just with different constructors. The 0 -constructor changes to nil and the s -constructor to cons , with an additional argument n , which is in the magic formula skolemized as σ_n .

The CNF of the magic formula is:

$$\bar{L}[\text{nil}, u_{\text{nil}}] \vee L[\sigma_l, \sigma_w] \vee L[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (57)$$

$$\bar{L}[\text{nil}, u_{\text{nil}}] \vee \bar{L}[\text{cons}(\sigma_n, \sigma_l), u_{\text{cons}}] \vee L[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (58)$$

We will denote the application of induction with this magic formula by $\text{MagInd}_{\mathbb{L}}$.

Derivation and program.

1. $y \not\approx \text{len}(\sigma_1 \mathbin{++} \sigma_2) \vee \text{ans}(y)$ [input]
2. $u_{\text{nil}} \not\approx \text{len}(\text{nil} \mathbin{++} \sigma_2) \vee \sigma_w \simeq \text{len}(\sigma_l \mathbin{++} \sigma_2) \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_1))$ [MagInd $_{\mathbb{L}}$, BR 1]
3. $u_{\text{nil}} \not\approx \text{len}(\text{nil} \mathbin{++} \sigma_2) \vee u_{\text{cons}} \not\approx \text{len}(\text{cons}(\sigma_n, \sigma_l) \mathbin{++} \sigma_2) \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_1))$ [MagInd $_{\mathbb{L}}$, BR 1]
4. $u_{\text{nil}} \not\approx \text{len}(\sigma_2) \vee \sigma_w \simeq \text{len}(\sigma_l \mathbin{++} \sigma_2) \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_1))$ [Sup (L1 $_{\mathbb{L}}$), 2]
5. $\sigma_w \simeq \text{len}(\sigma_l \mathbin{++} \sigma_2) \vee \text{ans}(\text{rec}(\text{len}(\sigma_2), u_{\text{cons}}, \sigma_1))$ [ER 4]
6. $u_{\text{nil}} \not\approx \text{len}(\sigma_2) \vee u_{\text{cons}} \not\approx \text{len}(\text{cons}(\sigma_n, \sigma_l) \mathbin{++} \sigma_2) \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_1))$ [Sup (L1 $_{\mathbb{L}}$), 3]
7. $u_{\text{cons}} \not\approx \text{len}(\text{cons}(\sigma_n, \sigma_l) \mathbin{++} \sigma_2) \vee \text{ans}(\text{rec}(\text{len}(\sigma_2), u_{\text{cons}}, \sigma_1))$ [ER 6]
8. $u_{\text{cons}} \not\approx \text{len}(\text{cons}(\sigma_n, \sigma_l) \mathbin{++} \sigma_2) \vee \text{ans}(\text{rec}(\text{len}(\sigma_2), u_{\text{cons}}, \sigma_1))$ [Sup (A2 $_{\mathbb{L}}$), 7]
9. $u_{\text{cons}} \not\approx s(\text{len}(\sigma_l \mathbin{++} \sigma_2)) \vee \text{ans}(\text{rec}(\text{len}(\sigma_2), u_{\text{cons}}, \sigma_1))$ [Sup (A4 $_{\mathbb{L}}$), 8]
10. $u_{\text{cons}} \not\approx s(\sigma_w) \vee \text{ans}(\text{rec}(\text{len}(\sigma_2), u_{\text{cons}}, \sigma_1))$ [Sup 5, 9]
11. $\text{ans}(\text{rec}(\text{len}(\sigma_2), s(\sigma_w), \sigma_1))$ [ER 10]
12. \square [answer literal removal 11]

The program constructed from $\text{rec}(\text{len}(x_2), s(f(l)), x_1)$ is

$$f(x_1)$$

where

$$\begin{aligned} f(\text{nil}) &\simeq \text{len}(x_2) \\ f(\text{cons}(n, l)) &\simeq s(f(l)). \end{aligned}$$

\square

Example 9 (Last element of a list).

Specification. The specification is

$$\forall x \in \mathbb{L} \exists y \in \mathbb{N}. (x \neq \text{nil} \rightarrow \exists z \in \mathbb{L}. x \simeq z++\text{cons}(y, \text{nil}))$$

Details of the magic. We apply induction with the base $\text{cons}(a, \text{nil})$, similarly how we did with $\text{MagInd}'_{\mathbb{N}}$ in Example 5. The CNF of the magic formula is:

$$\overline{L}[\text{cons}(\sigma_a, \text{nil}), u_{\text{nil}}] \vee \sigma_l \neq \text{nil} \vee z \simeq \text{nil} \vee L[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (59)$$

$$\overline{L}[\text{cons}(\sigma_a, \text{nil}), u_{\text{nil}}] \vee L[\sigma_l, \sigma_w] \vee z \simeq \text{nil} \vee L[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (60)$$

$$\overline{L}[\text{cons}(\sigma_a, \text{nil}), u_{\text{nil}}] \vee \overline{L}[s(\sigma_l), u_{\text{cons}}] \vee z \simeq \text{nil} \vee L[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (61)$$

We will denote the application of induction with this magic formula by $\text{MagInd}'_{\mathbb{L}}$.

For the derivation we instantiate the magic formula with $L[t, x] := \exists z \in \mathbb{L}. t \simeq z++\text{cons}(x, \text{nil})$.

Derivation and program.

1. $\sigma_x \neq \text{nil} \vee \text{ans}(y)$ [input]
2. $\sigma_x \neq z++\text{cons}(y, \text{nil}) \vee \text{ans}(y)$ [input]
3. $\text{cons}(\sigma_a, \text{nil}) \neq z_1++\text{cons}(u_{\text{nil}}, \text{nil}) \vee \sigma_l \simeq \sigma_z++\text{cons}(\sigma_w, \text{nil}) \vee$
 $\vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [MagInd}'_{\mathbb{L}}, \text{BR } 2]
4. $\text{cons}(\sigma_a, \text{nil}) \neq z_1++\text{cons}(u_{\text{nil}}, \text{nil}) \vee$
 $\vee \text{cons}(\sigma_n, \sigma_l) \neq z_2++\text{cons}(u_{\text{cons}}, \text{nil}) \vee$
 $\vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [MagInd}'_{\mathbb{L}}, \text{BR } 2]
5. $\text{cons}(\sigma_a, \text{nil}) \neq z_1++\text{cons}(u_{\text{nil}}, \text{nil}) \vee \sigma_l \simeq \sigma_z++\text{cons}(\sigma_w, \text{nil}) \vee$
 $\vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [BR 1, 3, ER]
6. $\text{cons}(\sigma_a, \text{nil}) \neq z_1++\text{cons}(u_{\text{nil}}, \text{nil}) \vee$
 $\vee \text{cons}(\sigma_n, \sigma_l) \neq z_2++\text{cons}(u_{\text{cons}}, \text{nil}) \vee$
 $\vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [BR 1, 4, ER]
7. $\sigma_l \simeq \sigma_z++\text{cons}(\sigma_w, \text{nil}) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{cons}}, \sigma_x))$ [ER (L1_{\mathbb{L}}), 5]
8. $\text{cons}(\sigma_n, \sigma_l) \neq z_2++\text{cons}(u_{\text{cons}}, \text{nil}) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{cons}}, \sigma_x))$ [ER (L1_{\mathbb{L}}), 6]
9. $\text{cons}(\sigma_n, \sigma_z++\text{cons}(\sigma_w, \text{nil})) \neq z_2++\text{cons}(u_{\text{cons}}, \text{nil}) \vee$
 $\vee \text{ans}(\text{rec}(\sigma_a, u_{\text{cons}}, \sigma_x))$ [Sup 7, 8]
10. $\text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [Sup (A2_{\mathbb{L}}), 9, ER]
11. \square [answer literal removal 10]

We derive the program

$$f(x),$$

where f is the recursive function defined as

$$\begin{aligned} f(\text{cons}(a, \text{nil})) &\simeq a \\ f(\text{cons}(n, l)) &\simeq f(l). \end{aligned}$$

Note. Similarly to this example, we could synthesize the function that returns the first element of a list. However, in practice such a negated skolemized specification would be instantly resolved using the destructor for cons . \square

Example 10 (Prefix of a List Given Its Suffix).

Specification. Recall the specification:

$$\forall x_1, x_2 \in \mathbb{L}. \exists y \in \mathbb{L}. (\text{suff}(x_2, x_1) \rightarrow x_1 \simeq y \mathbin{++} x_2)$$

Details of the magic. We need to apply induction on $\text{suff}(\sigma_2, \sigma_1)$ and $\sigma_1 \not\simeq y \mathbin{++} \sigma_2$. Therefore we instantiate the magic axiom for lists with a disjunction $G[t, x] := L_1[t] \vee L_2[t, x]$. We obtain a formula analogous to the one from Example 5. The CNF of the corresponding magic formula is:

$$\overline{L_1}[\text{nil}, u_{\text{nil}}] \vee L_1[\sigma_l, \sigma_w] \vee L_2[\sigma_l, \sigma_w] \vee L_1[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \vee L_2[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (62)$$

$$\overline{L_2}[\text{nil}, u_{\text{nil}}] \vee L_1[\sigma_l, \sigma_w] \vee L_2[\sigma_l, \sigma_w] \vee L_1[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \vee L_2[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (63)$$

$$\overline{L_1}[\text{nil}, u_{\text{nil}}] \vee \overline{L_1}[\text{cons}(\sigma_n, \sigma_l), u_{\text{cons}}] \vee L_1[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \vee L_2[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (64)$$

$$\overline{L_1}[\text{nil}, u_{\text{nil}}] \vee \overline{L_2}[\text{cons}(\sigma_n, \sigma_l), u_{\text{cons}}] \vee L_1[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \vee L_2[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (65)$$

$$\overline{L_2}[\text{nil}, u_{\text{nil}}] \vee \overline{L_1}[\text{cons}(\sigma_n, \sigma_l), u_{\text{cons}}] \vee L_1[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \vee L_2[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (66)$$

$$\overline{L_2}[\text{nil}, u_{\text{nil}}] \vee \overline{L_2}[\text{cons}(\sigma_n, \sigma_l), u_{\text{cons}}] \vee L_1[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \vee L_2[z, \text{rec}(u_{\text{nil}}, u_{\text{cons}}, z)] \quad (67)$$

We will denote the application of induction with this magic formula by $\text{MagInd}''_{\mathbb{L}}$.

After applying $\text{MagInd}''_{\mathbb{L}}$ with $G[t, y] := \neg(\text{suff}(t, \sigma_1)) \vee \sigma_1 \simeq y \mathbin{++} t$ in the derivation, we need to apply induction again. Similarly to Example 5, the second time we need a non-standard base case: instead of nil , we use $\text{cons}(a, \text{nil})$. The magic axiom is:

$$\begin{aligned} & (\forall a. \exists u_{\text{nil}}. G[\text{cons}(a, \text{nil}), u_{\text{nil}}] \wedge \forall n, l. (l \not\simeq \text{nil} \wedge \exists w. G[l, w] \rightarrow \exists u_{\text{cons}}. G[\text{cons}(n, l), u_{\text{cons}}])) \\ & \rightarrow \forall z. \exists x. (z \not\simeq \text{nil} \rightarrow G[z, x]) \end{aligned} \quad (68)$$

We will denote the application of induction with this magic formula by $\text{MagInd}'''_{\mathbb{L}}$.

We instantiate it with $G[t, x] := \neg(\text{suff}(\text{cons}(\sigma_n, \text{nil}), t)) \vee t \simeq x \mathbin{++} \text{cons}(\sigma_n, \text{nil})$.

Derivation and program.

1. $\text{suff}(\sigma_2, \sigma_1) \vee \text{ans}(y)$ [input]
2. $\sigma_1 \not\simeq y \mathbin{++} \sigma_2 \vee \text{ans}(y)$ [input]
3. $\sigma_1 \not\simeq u_{\text{nil}} \mathbin{++} \text{nil} \vee \neg \text{suff}(\sigma_l, \sigma_1) \vee \sigma_1 \simeq \sigma_w \mathbin{++} \sigma_l \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(u_{\text{nil}}, u_{\text{cons}}, \sigma_2))$ [MagInd $''_{\mathbb{L}}$, BR 1, 2]
4. $\sigma_1 \not\simeq u_{\text{nil}} \mathbin{++} \text{nil} \vee \text{suff}(\text{cons}(\sigma_n, \sigma_l), \sigma_1) \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(u_{\text{nil}}, u_{\text{cons}}, \sigma_2))$ [MagInd $''_{\mathbb{L}}$, BR 1, 2]
5. $\sigma_1 \not\simeq u_{\text{nil}} \mathbin{++} \text{nil} \vee \sigma_1 \not\simeq u_{\text{cons}} \mathbin{++} \text{cons}(\sigma_n, \sigma_l) \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(u_{\text{nil}}, u_{\text{cons}}, \sigma_2))$ [MagInd $''_{\mathbb{L}}$, BR 1, 2]
6. $\neg \text{suff}(\sigma_l, \sigma_1) \vee \sigma_1 \simeq \sigma_w \mathbin{++} \sigma_l \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [Sup, ER (A1 $_{\mathbb{L}}$), 3]

7. $\text{suff}(\text{cons}(\sigma_n, \sigma_l), \sigma_1) \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [Sup, ER (A1 $_{\mathbb{L}}$), 4]
8. $\sigma_1 \not\approx u_{\text{cons}}++\text{cons}(\sigma_n, \sigma_l) \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [Sup, ER (A1 $_{\mathbb{L}}$), 5]
9. $\text{suff}(\sigma_l, \sigma_1) \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [BR (A8 $_{\mathbb{L}}$), 7]
10. $\sigma_1 \simeq \sigma_w++\sigma_l \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [BR 6, 9]
11. $\sigma_w++\sigma_l \not\approx u_{\text{cons}}++\text{cons}(\sigma_n, \sigma_l) \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [Sup 8, 10]
12. $\sigma_w++\sigma_l \not\approx u_{\text{cons}}++\text{cons}(\sigma_n, \text{nil}++\sigma_l) \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [Sup (A1 $_{\mathbb{L}}$), 11]
13. $\sigma_w++\sigma_l \not\approx u_{\text{cons}}++\text{cons}(\sigma_n, \text{nil})++\sigma_l \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [Sup (A2 $_{\mathbb{L}}$), 12]
14. $\sigma_w \not\approx u_{\text{cons}}++\text{cons}(\sigma_n, \text{nil}) \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [BR (L6 $_{\mathbb{L}}$), 13]
15. $\text{suff}(\text{cons}(\sigma_n, \text{nil}), \sigma_w) \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [BR (L3 $_{\mathbb{L}}$), 7, 10]
16. $\text{cons}(a, \text{nil}) \not\approx u_{\text{nil}}++\text{cons}(\sigma_n, \text{nil}) \vee \neg \text{suff}(\text{cons}(\sigma_n, \text{nil}), \sigma_l) \vee$
 $\vee \sigma_l \simeq \sigma'_w++\text{cons}(\sigma_n, \text{nil}) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(u_{\text{nil}}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [MagInd $_{\mathbb{L}}$ ^{'''}, BR 14, 15]
17. $\text{cons}(a, \text{nil}) \not\approx u_{\text{nil}}++\text{cons}(\sigma_n, \text{nil}) \vee$
 $\vee \text{suff}(\text{cons}(\sigma_n, \text{nil}), \text{cons}(\sigma'_n, \sigma_l)) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(u_{\text{nil}}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [MagInd $_{\mathbb{L}}$ ^{'''}, BR 14, 15]
18. $\text{cons}(a, \text{nil}) \not\approx u_{\text{nil}}++\text{cons}(\sigma_n, \text{nil}) \vee$
 $\vee \text{cons}(\sigma'_n, \sigma_l) \not\approx u_{\text{cons}}++\text{cons}(\sigma_n, \text{nil}) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(u_{\text{nil}}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [MagInd $_{\mathbb{L}}$ ^{'''}, BR 14, 15]
19. $\text{cons}(a, \text{nil}) \not\approx u_{\text{nil}}++\text{cons}(\sigma_n, \text{nil}) \vee \sigma_l \not\approx \text{nil} \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(u_{\text{nil}}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [MagInd $_{\mathbb{L}}$ ^{'''}, BR 14, 15]
20. $\neg \text{suff}(\text{cons}(\sigma_n, \text{nil}), \sigma_l) \vee \sigma_l \simeq \sigma'_w++\text{cons}(\sigma_n, \text{nil}) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [BR (A1 $_{\mathbb{L}}$), 16]
21. $\text{suff}(\text{cons}(\sigma_n, \text{nil}), \text{cons}(\sigma'_n, \sigma_l)) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [BR (A1 $_{\mathbb{L}}$), 17]
22. $\text{cons}(\sigma'_n, \sigma_l) \not\approx u_{\text{cons}}++\text{cons}(\sigma_n, \text{nil}) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [BR (A1 $_{\mathbb{L}}$), 18]
23. $\sigma_l \not\approx \text{nil} \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [BR (A1 $_{\mathbb{L}}$), 19]
24. $\text{cons}(\sigma_n, \text{nil}) = \text{cons}(\sigma'_n, \sigma_l) \vee \text{suff}(\text{cons}(\sigma_n, \text{nil}), \sigma_l) \vee$
 $\vee \sigma_w \simeq \text{nil} \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [BR (L2 $_{\mathbb{L}}$), 21]
25. $\sigma_l \simeq \text{nil} \vee \text{suff}(\text{cons}(\sigma_n, \text{nil}), \sigma_l) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [BR (A9 $_{\mathbb{L}}$), 24]
26. $\text{suff}(\text{cons}(\sigma_n, \text{nil}), \sigma_l) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [BR 23, 25]
27. $\sigma_l \simeq \sigma'_w++\text{cons}(\sigma_n, \text{nil}) \vee \sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [BR 20, 26]
28. $\text{cons}(\sigma'_n, \sigma'_w++\text{cons}(\sigma_n, \text{nil})) \not\approx u_{\text{cons}}++\text{cons}(\sigma_n, \text{nil}) \vee$
 $\vee \sigma_w \simeq \text{nil} \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, u_{\text{cons}}, \sigma_w), \sigma_2))$ [Sup 22, 27]
29. $\sigma_w \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, \text{cons}(\sigma'_n, \sigma'_w), \sigma_w), \sigma_2))$ [Sup (A2 $_{\mathbb{L}}$), 28]
30. $\sigma_1 \not\approx \sigma_l \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [BR (L4 $_{\mathbb{L}}$), 7]
31. $\sigma_1 \simeq \sigma_l \vee \sigma_w \not\approx \text{nil} \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [BR (L5 $_{\mathbb{L}}$), 10]
32. $\sigma_w \not\approx \text{nil} \vee \text{ans}(\text{rec}_{\text{pref}}(\sigma_1, u_{\text{cons}}, \sigma_2))$ [BR 30, 31]
33. $\text{ans}(\text{rec}_{\text{pref}}(\sigma_1, \text{rec}_{\text{remove}}(\text{nil}, \text{cons}(\sigma'_n, \sigma'_w), \sigma_w), \sigma_2))$ [BR, ER 29, 32]
34. \square [answer literal removal 33]

The program is constructed as

$$\text{pref}(x_2),$$

where pref is the recursive function defined as

$$\begin{aligned} \text{pref}(\text{nil}) &\simeq x_1 \\ \text{pref}(\text{cons}(n, l)) &\simeq \text{remove}(\text{pref}(l)), \end{aligned}$$

and remove is the recursive function defined as

$$\begin{aligned} \text{remove}(\text{cons}(a, \text{nil})) &\simeq \text{nil} \\ \text{remove}(\text{cons}(n, l)) &\simeq \text{cons}(n, \text{remove}(l)). \end{aligned}$$

□

Example 11 (Maximum Element of a List).

Specification.

$$\forall x \in \mathbb{L} \exists y \in \mathbb{N}. (x \neq \text{nil} \rightarrow (\text{in}(y, x) \wedge \forall k \in \mathbb{N} (\text{in}(k, x) \rightarrow k \leq y)))$$

Details of the magic. We instantiate the magic axiom with a formula of the form $G[t, x] := L_1[t, x] \wedge (L_2[t] \vee L_3[x])$. Similarly as in previous example, we choose the base case $\text{cons}(a, \text{nil})$. We will denote the application of induction with this magic axiom by $\text{MagInd}_{\perp}''''$, and we apply it with $G[t, x] := \text{in}(x, t) \wedge (\neg \text{in}(\sigma_k, t) \vee \sigma_k \leq x)$.

Derivation and program.

1. $\sigma_x \neq \text{nil} \vee \text{ans}(y)$ [input]
2. $\text{in}(\sigma_k, \sigma_x) \vee \neg \text{in}(y, \sigma_x) \vee \text{ans}(y)$ [input]
3. $y < \sigma_k \vee \neg \text{in}(y, \sigma_x) \vee \text{ans}(y)$ [input]
4. $\neg \text{in}(u_{\text{nil}}, \text{cons}(\sigma_a, \text{nil})) \vee \text{in}(\sigma_k, \text{cons}(\sigma_a, \text{nil})) \vee$
 $\vee \text{in}(\sigma_w, \sigma_l) \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [MagInd $_{\perp}$ '''' , BR 2, 3]
5. $\neg \text{in}(u_{\text{nil}}, \text{cons}(\sigma_a, \text{nil})) \vee \text{in}(\sigma_k, \text{cons}(\sigma_a, \text{nil})) \vee$
 $\vee \neg \text{in}(\sigma_k, \sigma_l) \vee \sigma_k \leq \sigma_w \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [MagInd $_{\perp}$ '''' , BR 2, 3]
6. $\neg \text{in}(u_{\text{nil}}, \text{cons}(\sigma_a, \text{nil})) \vee u_{\text{nil}} < \sigma_k \vee \text{in}(\sigma_w, \sigma_l) \vee$
 $\vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [MagInd $_{\perp}$ '''' , BR 2, 3]
7. $\neg \text{in}(u_{\text{nil}}, \text{cons}(\sigma_a, \text{nil})) \vee u_{\text{nil}} < \sigma_k \vee \neg \text{in}(\sigma_k, \sigma_l) \vee$
 $\vee \sigma_k \leq \sigma_w \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [MagInd $_{\perp}$ '''' , BR 2, 3]
8. $\neg \text{in}(u_{\text{nil}}, \text{cons}(\sigma_a, \text{nil})) \vee \text{in}(\sigma_k, \text{cons}(\sigma_a, \text{nil})) \vee$
 $\vee \neg \text{in}(u_{\text{cons}}, \text{cons}(\sigma_n, \sigma_l)) \vee \text{in}(\sigma_k, \text{cons}(\sigma_n, \sigma_l)) \vee$
 $\vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [MagInd $_{\perp}$ '''' , BR 2, 3]
9. $\neg \text{in}(u_{\text{nil}}, \text{cons}(\sigma_a, \text{nil})) \vee \text{in}(\sigma_k, \text{cons}(\sigma_a, \text{nil})) \vee$
 $\vee \neg \text{in}(u_{\text{cons}}, \text{cons}(\sigma_n, \sigma_l)) \vee u_{\text{cons}} < \sigma_k \vee \sigma_x \simeq \text{nil} \vee$
 $\vee \text{ans}(\text{rec}(u_{\text{nil}}, u_{\text{cons}}, \sigma_x))$ [MagInd $_{\perp}$ '''' , BR 2, 3]

34. $u_{\text{cons}} \not\leq \sigma_n \vee u_{\text{cons}} < \sigma_k \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{cons}}, \sigma_x))$ [BR (A11 $_{\mathbb{L}}$), 31]
35. $\neg \text{in}(u_{\text{cons}}, \sigma_l) \vee u_{\text{cons}} < \sigma_k \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{cons}}, \sigma_x))$ [BR (A11 $_{\mathbb{L}}$), 30]
36. $\text{in}(\sigma_k, \text{cons}(\sigma_n, \sigma_l)) \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 28, 33, ER]
37. $\sigma_k \simeq \sigma_n \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR (A11 $_{\mathbb{L}}$), 36]
38. $\sigma_w < \sigma_k \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 28, 35]
39. $\neg \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 29, 38]
40. $\sigma_k \simeq \sigma_n \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 37, 39]
41. $\sigma_w < \sigma_n \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [Sup 38, 40]
42. $\text{in}(\sigma_k, \text{cons}(\sigma_n, \sigma_l)) \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [ER 32]
43. $\sigma_n < \sigma_k \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [ER 34]
44. $\sigma_k \simeq \sigma_n \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR (A11 $_{\mathbb{L}}$), 42]
45. $\sigma_k \simeq \sigma_n \vee \sigma_k \leq \sigma_w \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR 29, 44]
46. $\neg(\sigma_n < \sigma_k) \vee \sigma_k \leq \sigma_w \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR (L7 $_{\mathbb{N}}$), 45]
47. $\sigma_k \leq \sigma_w \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR 43, 46]
48. $\sigma_n \leq \sigma_k \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR (L11 $_{\mathbb{N}}$), 43]
49. $\sigma_n \leq \sigma_w \vee \sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR (A9 $_{\mathbb{N}}$), 47, 48]
50. $\sigma_x \simeq \text{nil} \vee \text{ans}(\text{rec}(\sigma_a, \text{if } \sigma_w < \sigma_n \text{ then } \sigma_n \text{ else } \sigma_w, \sigma_x))$ [BR' 41, 49]
51. $\text{ans}(\text{rec}(\sigma_a, \text{if } \sigma_w < \sigma_n \text{ then } \sigma_n \text{ else } \sigma_w, \sigma_x))$ [BR, ER 1, 50]
52. \square [answer literal removal 51]

The program is then

$$f(x)$$

where f is the recursive function defined as

$$\begin{aligned} f(\text{cons}(a, \text{nil})) &\simeq a \\ f(\text{cons}(n, l)) &\simeq \text{if } f(l) < n \text{ then } n \text{ else } f(l). \end{aligned}$$

\square

D.3 Theory of Binary Trees

Example 12 (Maximum Element of a Tree).

Specification.

$$\forall x \in \mathbb{BT} \exists y \in \mathbb{N}. (\text{in}(y, x) \wedge \forall k \in \mathbb{N} (\text{in}(k, x) \rightarrow k \leq y))$$

Details of the magic. This example is analogous to the previous one, but since we are working with binary trees, the axiom we use as well as the resulting derivation are more complex. The structure of the formula used to instantiate the magic axiom is the same as in Example 11, $G[t, x] := L_1[t, x] \wedge (L_2[t] \vee L_3[x])$, but we use the standard base case $\text{leaf}(a)$. We will denote the application of induction with this magic axiom by $\text{MagInd}'_{\mathbb{BT}}$, and we apply it with $G[t, x] := \text{in}(x, t) \wedge (\neg \text{in}(\sigma_k, t) \vee \sigma_k \leq x)$.

Derivation and program.

1. $\text{in}(\sigma_k, \sigma_x) \vee \neg \text{in}(y, \sigma_x) \vee \text{ans}(y)$ [input]

27. $\sigma_a \simeq \sigma_k \vee \text{in}(\sigma_w, \sigma_l) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A1_{BT}), 15]
28. $\sigma_a \simeq \sigma_k \vee \neg \text{in}(k, \sigma_l) \vee \neg(\sigma_w < k) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A1_{BT}), 16]
29. $\sigma_a \not\simeq \sigma_k \vee \text{in}(\sigma_w, \sigma_l) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (L12_N), 17]
30. $\sigma_a \not\simeq \sigma_k \vee \neg \text{in}(k, \sigma_l) \vee \neg(\sigma_w < k) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (L12_N), 18]
31. $\sigma_a \simeq \sigma_k \vee \text{in}(\sigma'_w, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A1_{BT}), 19]
32. $\sigma_a \simeq \sigma_k \vee \neg \text{in}(k, \sigma_r) \vee \neg(\sigma'_w < k) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A1_{BT}), 20]
33. $\sigma_a \not\simeq \sigma_k \vee \text{in}(\sigma'_w, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (L12_N), 21]
34. $\sigma_a \not\simeq \sigma_k \vee \neg \text{in}(k, \sigma_r) \vee \neg(\sigma'_w < k) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (L12_N), 22]
35. $\sigma_a \simeq \sigma_k \vee \neg \text{in}(u_{\text{bt}}, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee$
 $\vee \text{in}(\sigma_k, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A1_{BT}), 23]
36. $\sigma_a \simeq \sigma_k \vee \neg \text{in}(u_{\text{bt}}, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee u_{\text{bt}} < \sigma_k \vee$
 $\vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A1_{BT}), 24]
37. $\sigma_a \not\simeq \sigma_k \vee \neg \text{in}(u_{\text{bt}}, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee$
 $\vee \text{in}(\sigma_k, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (L12_N), 25]
38. $\sigma_a \not\simeq \sigma_k \vee \neg \text{in}(u_{\text{bt}}, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee u_{\text{bt}} < \sigma_k \vee$
 $\vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (L12_N), 26]
39. $\text{in}(\sigma_w, \sigma_l) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR 27, 29]
40. $\neg \text{in}(k, \sigma_l) \vee \neg(\sigma_w < k) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR 28, 30]
41. $\text{in}(\sigma'_w, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR 31, 33]
42. $\neg \text{in}(k, \sigma_r) \vee \neg(\sigma'_w < k) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR 32, 34]
43. $\neg \text{in}(u_{\text{bt}}, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee \text{in}(\sigma_k, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee$
 $\vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR 35, 37]
44. $\neg \text{in}(u_{\text{bt}}, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee u_{\text{bt}} < \sigma_k \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR 36, 38]
45. $u_{\text{bt}} \not\simeq \sigma_n \vee \text{in}(\sigma_k, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 43]
46. $\neg \text{in}(u_{\text{bt}}, \sigma_l) \vee \text{in}(\sigma_k, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 43]
47. $\neg \text{in}(u_{\text{bt}}, \sigma_r) \vee \text{in}(\sigma_k, \text{bt}(\sigma_l, \sigma_n, \sigma_r)) \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 43]
48. $u_{\text{bt}} \not\simeq \sigma_n \vee \sigma_k \simeq \sigma_n \vee \text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee$
 $\vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 45]
49. $\neg \text{in}(u_{\text{bt}}, \sigma_l) \vee \sigma_k \simeq \sigma_n \vee \text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee$
 $\vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 46]
50. $\neg \text{in}(u_{\text{bt}}, \sigma_r) \vee \sigma_k \simeq \sigma_n \vee \text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee$
 $\vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 47]
51. $u_{\text{bt}} \not\simeq \sigma_n \vee u_{\text{bt}} < \sigma_k \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 44]
52. $\neg \text{in}(u_{\text{bt}}, \sigma_l) \vee u_{\text{bt}} < \sigma_k \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 44]
53. $\neg \text{in}(u_{\text{bt}}, \sigma_r) \vee u_{\text{bt}} < \sigma_k \vee \text{ans}(\text{rec}(\sigma_a, u_{\text{bt}}, \sigma_x))$ [BR (A2_{BT}), 44]
54. $\sigma_k \simeq \sigma_n \vee \text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [ER 48]
55. $\sigma_n < \sigma_k \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [ER 51]
56. $\sigma_n \not\simeq \sigma_k \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR (L12_N), 55]
57. $\text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR 54, 56]
58. $\neg(\sigma_w < \sigma_k) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR 40, 57]
59. $\neg(\sigma_w < \sigma_n) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR (A9_N), 55, 58]
60. $\neg(\sigma_w < \sigma_n) \vee \neg(\sigma'_w < \sigma_k) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR 42, 59]
61. $\neg(\sigma_w < \sigma_n) \vee \neg(\sigma'_w < \sigma_n) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_n, \sigma_x))$ [BR (A9_N), 55, 60]
62. $\sigma_k \simeq \sigma_n \vee \text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 39, 49, ER]
63. $\sigma_w < \sigma_k \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 39, 52, ER]
64. $\sigma_w < \sigma_n \vee \text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [Sup 62, 63]

65. $\sigma_w < \sigma_n \vee \neg(\sigma_w < \sigma_k) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 40, 64]
66. $\sigma_w < \sigma_n \vee \neg(\sigma_w < \sigma_k) \vee \neg(\sigma'_w < \sigma_k) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 42, 65]
67. $\sigma_w < \sigma'_w \vee \neg(\sigma_w < \sigma'_w)$ [Taut.]
68. $\sigma_w < \sigma_n \vee \neg(\sigma_w < \sigma_k) \vee \sigma_w < \sigma'_w \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR (L13 \mathbb{N}), 66, 67]
69. $\sigma_w < \sigma_n \vee \sigma_w < \sigma'_w \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR 63, 68]
70. $\sigma'_w < \sigma_n \vee \sigma_w < \sigma'_w \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR (L13 \mathbb{N}), 69]
71. $\sigma_k \simeq \sigma_n \vee \text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma'_w, \sigma_x))$ [BR 41, 50, ER]
72. $\sigma'_w < \sigma_k \vee \text{ans}(\text{rec}(\sigma_a, \sigma'_w, \sigma_x))$ [BR 41, 53, ER]
73. $\sigma'_w < \sigma_n \vee \text{in}(\sigma_k, \sigma_l) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma'_w, \sigma_x))$ [Sup 71, 72]
74. $\sigma'_w < \sigma_n \vee \neg(\sigma_w < \sigma_k) \vee \text{in}(\sigma_k, \sigma_r) \vee \text{ans}(\text{rec}(\sigma_a, \sigma'_w, \sigma_x))$ [BR 40, 73]
75. $\sigma'_w < \sigma_n \vee \neg(\sigma_w < \sigma_k) \vee \neg(\sigma'_w < \sigma_k) \vee \text{ans}(\text{rec}(\sigma_a, \sigma'_w, \sigma_x))$ [BR 42, 74]
76. $\sigma'_w < \sigma_n \vee \neg(\sigma'_w < \sigma_k) \vee \neg(\sigma_w < \sigma'_w) \vee \text{ans}(\text{rec}(\sigma_a, \sigma_w, \sigma_x))$ [BR (L13 \mathbb{N}), 67, 75]
77. $\sigma'_w < \sigma_n \vee \neg(\sigma_w < \sigma'_w) \vee \text{ans}(\text{rec}(\sigma_a, \sigma'_w, \sigma_x))$ [BR 72, 76]
78. $\sigma_w < \sigma_n \vee \neg(\sigma_w < \sigma'_w) \vee \text{ans}(\text{rec}(\sigma_a, \sigma'_w, \sigma_x))$ [BR (L13 \mathbb{N}), 77]
79. $\neg(\sigma'_w < \sigma_n) \vee \sigma_w < \sigma'_w \vee \text{ans}(\text{rec}(\sigma_a, \text{if } \sigma_w < \sigma_n \text{ then } \sigma_n \text{ else } \sigma_w, \sigma_x))$ [BR' 61, 69]
80. $\sigma_w < \sigma'_w \vee \text{ans}(\text{rec}(\sigma_a, \text{if } \sigma'_w < \sigma_n \text{ then if } \sigma_w < \sigma_n \text{ then } \sigma_n \text{ else } \sigma_w \text{ else } \sigma_w, \sigma_x))$ [BR' 70, 79]
81. $\neg(\sigma_w < \sigma_n) \vee \neg(\sigma_w < \sigma'_w) \vee \text{ans}(\text{rec}(\sigma_a, \text{if } \sigma'_w < \sigma_n \text{ then } \sigma_n \text{ else } \sigma'_w, \sigma_x))$ [BR' 61, 77]
82. $\neg(\sigma_w < \sigma'_w) \vee \text{ans}(\text{rec}(\sigma_a, \text{if } \sigma_w < \sigma_n \text{ then if } \sigma'_w < \sigma_n \text{ then } \sigma_n \text{ else } \sigma'_w \text{ else } \sigma'_w, \sigma_x))$ [BR' 78, 81]
83. $\text{ans}(\text{rec}(\sigma_a, \text{if } \sigma_w < \sigma'_w \text{ then if } \sigma_w < \sigma_n \text{ then if } \sigma'_w < \sigma_n \text{ then } \sigma_n \text{ else } \sigma'_w \text{ else } \sigma'_w \text{ else if } \sigma'_w < \sigma_n \text{ then if } \sigma_w < \sigma_n \text{ then } \sigma_n \text{ else } \sigma_w \text{ else } \sigma_w, \sigma_x))$ [BR' 80, 82]
84. \square [answer literal removal 83]

The program we end up with is

$$f(x),$$

where

$$\begin{aligned}
f(\text{leaf}(a)) &\simeq a \\
f(\text{bt}(l, n, r)) &\simeq \text{if } f(l) < f(r) \text{ then} \\
&\quad \text{if } f(l) < n \text{ then} \\
&\quad \quad \text{if } f(r) < n \text{ then } n \text{ else } f(r) \\
&\quad \quad \text{else } f(r) \\
&\quad \text{else if } f(r) < n \text{ then} \\
&\quad \quad \text{if } f(l) < n \text{ then } n \text{ else } f(l) \\
&\quad \text{else } f(l).
\end{aligned}$$

\square

E Vampire Outputs

In this appendix we show the full derivation of the recursive program for (3) produced automatically by VAMPIRE. The runtime was 0.021s. The derivation uses the TPTP syntax [26]. We underline the final derived program and highlight it in red. The function `rf85` corresponds to f from Example 2, and the variable `X0` to the input variable x .

Note that the derivations produced by VAMPIRE might differ from those presented in the paper and Appendix D due to VAMPIRE using specific ordering and selection constraints, and a limited subset of `MagInd` instances.

All our examples as well as the instructions to run VAMPIRE are available online at https://github.com/vprover/vampire_benchmarks/tree/master/synthesis/recursive.

Configuration used for Example 2:

```
--forced_options ind=struct:indu=off:qa=synthesis
```

Output:

```
% Inputs for synthesis:
5. ~! [X0 : 'nat()'] : ? [X1 : 'nat()'] : half(X1) = X0 [
    negated conjecture 4]
% Recursive function definitions:
rf85(zero) = s(zero)
rf85(s(X5)) = s(s(rf85(X5)))
% SZS answers Tuple [[rf85(X0)]|_]
% SZS output start Proof
2. zero = half(s(zero)) [input]
3. ! [X0 : 'nat()'] : s(half(X0)) = half(s(s(X0))) [input]
4. ! [X0 : 'nat()'] : ? [X1 : 'nat()'] : half(X1) = X0 [input
    ]
5. ~! [X0 : 'nat()'] : ? [X1 : 'nat()'] : half(X1) = X0 [
    negated conjecture 4]
9. ! [X1 : 'nat()'] : ~(half(X1) = sK1_in & ans0(X1)) [answer
    literal with input var skolemisation 5]
10. ! [X0 : 'nat()'] : ~(half(X0) = sK1_in & ans0(X0)) [
    rectify 9]
11. ! [X0 : 'nat()'] : (half(X0) != sK1_in | ~ans0(X0)) [ennf
    transformation 10]
12. half(X0) != sK1_in | ~ans0(X0) [cnf transformation 11]
13. s(half(X0)) = half(s(s(X0))) [cnf transformation 3]
14. zero = half(s(zero)) [cnf transformation 2]
20. ? [X5 : 'nat()'] : ? [X6 : 'nat()'] : ! [X7 : 'nat()', X3
    : 'nat()'] : ! [X8 : 'nat()'] : ((zero = half(X3) & (half
    (X6) = X5 => s(X5) = half(X7))) => half(rec2(X3, X7, X8)) =
    X8) [structural induction hypothesis]
21. ? [X5 : 'nat()'] : ? [X6 : 'nat()'] : ! [X7 : 'nat()', X3
    : 'nat()'] : ! [X8 : 'nat()'] : (half(rec2(X3, X7, X8)) =
    X8 | (zero != half(X3) | (s(X5) != half(X7) & half(X6) =
    X5))) [ennf transformation 20]
```



```

22. sK3 = half(sK4) | zero != half(X3) | half(rec2(X3,X7,X8))
   = X8 [cnf transformation 21]
23. half(X7) != s(sK3) | zero != half(X3) | half(rec2(X3,X7,
   X8)) = X8 [cnf transformation 21]
24. half(X1) != s(sK3) | zero != half(X0) | ~ans0(rec2(X0,X1,
   sK1_in)) [resolution 23,12]
25. zero != half(X0) | sK3 = half(sK4) | ~ans0(rec2(X0,X1,
   sK1_in)) [resolution 22,12]
123. zero != zero | sK3 = half(sK4) | ~ans0(rec2(s(zero),X0,
   sK1_in)) [superposition 25,14]
127. sK3 = half(sK4) | ~ans0(rec2(s(zero),X0,sK1_in)) [
   trivial inequality removal 123]
160. s(half(X0)) != s(sK3) | zero != half(X1) | ~ans0(rec2(X1
   ,s(s(X0)),sK1_in)) [superposition 24,13]
724. s(sK3) != s(sK3) | zero != half(s(zero)) | ~ans0(rec2(s(
   zero),s(s(sK4)),sK1_in)) [superposition 160,127]
753. zero != half(s(zero)) | ~ans0(rec2(s(zero),s(s(sK4))),
   sK1_in)) [trivial inequality removal 724]
758. ~ans0(rec2(s(zero),s(s(sK4)),sK1_in)) [subsumption
   resolution 753,14]
759. ans0(X0) [answer literal]
760. $false [unit resulting resolution 759,758]
% SZS output end Proof
% -----
% Version: Vampire 4.8 (commit 3cddf8311 on 2024-01-28
   09:37:47 +0100)
% Linked with Z3 4.12.2.0
   e417f7d78509b2d0c9ebc911fee7632e6ef546b6 z3-4.8.4-7517-
   ge417f7d78
% Termination reason: Refutation

% Memory used [KB]: 718
% Time elapsed: 0.021 s
% -----
% -----

```