jar manuscript No.
(will be inserted by the editor)

Automatic Synthesis of Logical Models for Order-Sorted First-Order Theories

Salvador Lucas · Raúl Gutiérrez

Abstract In program analysis, the *synthesis of models* of logical theories representing the program semantics is often useful to prove program properties. We use *Order-Sorted First-Order Logic* as an appropriate framework to describe the semantics and properties of programs as given theories. Then we investigate the *automatic synthesis* of models for such theories. We use *convex polytopic domains* as a flexible approach to associate different domains to different *sorts*. We introduce a framework for the *piecewise definition* of functions and predicates. We develop its use with *linear expressions* (in a wide sense, including linear transformations represented as matrices) and inequalities to specify functions and predicates. In this way, algorithms and tools from linear algebra and arithmetic constraint solving (e.g., SMT) can be used as a backend for an efficient implementation.

Keywords Logical models · Order-sorted first-order logic · Program analysis

1 Introduction

The interplay between logic and program analysis and verification is central in the development of reliable software. Starting from the landmark papers by Floyd, Hoare, and Naur [36,62,98] many researchers have used *(first-order) logic* as a *universal language* to describe the semantics of languages and specify program properties [22]. Reasoning methods have been developed for the (automatic) verification of such properties thus leading to what is known as the *formal methods approach* to reliable software development (see [95] for instance).

Many-sorted logic [109,73,122] often improves one-sorted (first-order) logic regarding expressivity[1], knowledge representation [59], simplicity[2] [59,73,109], im-

Partially supported by the EU (FEDER), projects TIN2015-69175-C4-1-R, and GV PROMETEOII/2015/013. R. Gutiérrez also supported by Juan de la Cierva Fellowship JCI-2012-13528.

DSIC, Universitat Politècnica de València, Spain

[1] "Quantification in first-order logic always involves all elements of the universe. However, it is often more natural to just quantify over those elements of the universe which satisfy a certain condition" [19,52].

[2] "The unsorted theory will in general be much *larger* (more symbols and more axioms) than the sorted theory" [59, page 500].

(Rf) $\dfrac{}{x \to^* x}$ (T) $\dfrac{x \to y \quad y \to^* z}{x \to^* z}$

(C) $\dfrac{x_i \to y_i}{f(x_1,\ldots,x_i,\ldots,x_k) \to f(x_1,\ldots,y_i,\ldots,x_k)}$ where $f \in \Sigma_{w,s}$, $w = s_1,\ldots,s_k$, and $1 \leq i \leq k$ (Re) $\dfrac{}{\ell \to r}$ where $\ell \to r \in \mathcal{R}$

Fig. 1 Inference rules for Order-Sorted TRSs $\mathcal{R}$

proved deductive efficiency[3] [24,110,121], etc. The main idea is distinguishing different kinds of objects sharing some common properties by associating them a given *sort*. Variables have sorts and are bound to objects of this sort. Similarly, the arguments of function and predicate symbols are typed with sorts and only objects of those sorts are allowed in the corresponding argument. The outcome of a function also has a sort. The introduction of an *ordering* on sorts [46,24] provides an increased expressiveness over many-sorted logic as *Order-Sorted First-Order Logic* (OS-FOL). Indeed, most programming languages enable the use of types or sorts in program components. For instance, the specification and programming languages OBJ [53], CafeOBJ [40], and Maude [23] support the specification of sorts with a *subsort* ordering among them by means of the subsort relation <. They also allow the definition of *sorted* function symbols (in the aforementioned sense) when writing programs as *Order-Sorted Term Rewriting Systems* (OS-TRS).

Remark 1 OS-TRSs are OS-FOL theories where only two predicate symbols $\to$ and $\to^*$ are used. Programs are *specified* by means of rules $\ell \to r$ for sorted terms ℓ and r. Computations are described by means of the one-step rewrite relation $\to$ and the zero-or-more-steps relation $\to^*$, see Figure 1. All rules in the inference system in Figure 1 are *schematic* in that each inference rule $\frac{B_1 \cdots B_n}{A}$ can actually be used under any *instance* $\frac{\sigma(B_1) \cdots \sigma(B_n)}{\sigma(A)}$ of the rule by a substitution σ [113].

Programmers often use a *sort discipline* to ensure a ‘good behavior’ of programs.

Example 1 The following OS-TRS is based on the famous Toyama's example [115]. It shows that *the use of sorts can reinforce termination* (see also [123]).

```
mod ToyamaOS is
  sorts S S1 S2 .
  subsorts S2 < S1 .
  op a : -> S2 .   op f : S1 S1 S1 -> S .
  op b : -> S1 .   op g : S1 S1 -> S1 .
  var x : S2 .     vars  y z : S1 .
  rl f(a,b,x) => f(x,x,x) .
  rl g(y,z) => y .
  rl g(y,z) => z .
endm
```

The unsorted version of this module is nonterminating [115]. Actually, if `S1` and `S2` are merged into a single sort (thus becoming *many-sorted*, with only two *unrelated* sorts), it is also nonterminating:

$$\underline{\mathtt{f(a,b,g(a,b))}} \to \mathtt{f(}\underline{\mathtt{g(a,b)}}\mathtt{,g(a,b),g(a,b))} \to \mathtt{f(a,}\underline{\mathtt{g(a,b)}}\mathtt{,g(a,b))} \to \mathtt{f(a,b,g(a,b))} \to \cdots$$

[3] “Many sorted logics can allow an increase in deductive efficiency by eliminating useless branches of the search space” [24, Abstract].

But the subsort hierarchy makes `ToyamaOS` *terminating*. For instance, variable `x` (of sort `S2`) cannot be bound to terms of sort `S1`, which is supersort of `S2`. Since $\mathtt{g}(\mathtt{a},\mathtt{b})$ is of sort `S1`, the second step, which requires a binding $\mathtt{x} \mapsto \mathtt{g}(\mathtt{a},\mathtt{b})$, is not possible.

In program analysis and verification, the *synthesis of models* $\mathcal{A}$ for theories $\mathcal{S}$ representing programs or program properties is useful to:

1. *Approximate* computational relations associated to programs by means of a computational logic: one-step transitions $\to$, big-step transitions $\Downarrow$, etc. By *soundness* of the corresponding logic we can over-approximate provability of formulas φ in $\mathcal{S}$ (denoted $\mathcal{S} \vdash \varphi$) as satisfaction in $\mathcal{A}$ (denoted $\mathcal{A} \models \varphi$) for any model $\mathcal{A}$ of $\mathcal{S}$. This provides a logic-based abstraction mechanism that can be used in program analysis and verification (see also [29]).
2. Solve *verification conditions* [69,75], i.e., "logical formulas whose satisfiability implies program correctness" [5] and other safety properties [12,13,54,56,107].
3. *Bound* the *derivational complexity* of rewrite systems [67]. The association of a numerical measure to the computational relation $\to$ is often used to obtain such bounds, see [65,66,119], and also [94] and the references therein. Furthermore, similar techniques have also been used to bound the so-called *runtime complexity* where the focus is on rewrite sequences starting from terms where a defined function symbol is applied to arguments in normal form [61], in closer correspondence to the so-called *initial expressions* in functional programming.
4. Similarly, the analysis of resource consumption (time, space) of functional programs is also based on the appropriate generation of interpretations that can be used to measure the property of interest [3,41,89,96,103].
5. *Implement* proof obligations in program analysis by means of appropriate *relations*; for instance, in proofs of termination of declarative programs it is often necessary to *compare* expressions by means of special (well-founded) relations, so that a witness of termination can be obtained [80,82].

Since most programs use types or sorts, it is natural to include them in the logic we use to reason about them [49]. Section 2 summarizes the basics of OS-FOL.

When giving semantics to function or predicate symbols without any *intended interpretation* (e.g., those in `ToyamaOS`) it is useful to *map* them into (combinations of) symbols which are better understood (e.g., Presburger's arithmetic, [26,27]).

Remark 2 The interpretation of a binary function symbol `g` as an *arithmetic expression*, displayed as $\mathtt{g}^{\mathcal{A}}(x,y) \overset{\text{def}}{=} x+y+1$ in the description of an algebra $\mathcal{A}$, is *another symbolic device* to avoid the *extensional* association of the *mapping* $\{(0,0) \mapsto 1, (1,0) \mapsto 2, (0,1) \mapsto 2, \ldots\}$ to `g`. The point is that the new symbols x, y, $+$, and 1 occurring in the right-hand side of the equation have an *intended interpretation.*

Following this general approach, the first part of the paper develops the notion of derived structures and models. Burstall and Goguen [20,48] introduced the notion of *derivor*[4] to *transform* terms of a given signature Σ into terms of another (already familiar) signature Σ'. For instance, the interpretation of `g` in Remark 2 is a *derived* interpretation via *arithmetic expressions*. We generalize them to deal with OS-FOL theories (Section 3). *Derivors* d are used in [50] to induce a *derived* Σ-algebra $d\mathcal{A}'$ from a *given* Σ'-algebra $\mathcal{A}'$. We generalize this to *derive* OS-FOL *structures* (Section

[4] Implemented as part of the mechanisms available in the programming language Clear [20] to synthesize operations out from a set of basic (and available) ones [47,48].

4). In Section 5 we discuss a few usual requirements in program analysis and the possibility of representing them as formulas which are added to the original theory representing our specific problem. Some of them (e.g., well-foundedness) cannot be expressed at this syntactic (first-order!) level. Thus, we need to guarantee them at the *semantic* level by an appropriate selection of the structure which is used to derive the model. This first part of the paper (Sections 3 to 5) provides a *general methodology* to translate a *given* OS-FOL theory into a *target* theory for which the generation of a model can be (efficiently) automated. The derivor can then be used 'backwards' to provide a model for the original theory.

In the second part of the paper, we discuss the use of *linear algebra* for this automation purpose. Nevertheless, other possibilities could be explored in this second stage *without* changing anything essential in the first part of the treatment (e.g., *algebraic* or *rational* functions as sketched in Section 3). In Section 6 we introduce a generic notion of *piecewise definition* of function and predicate interpretations, which is similar to McCarthy's *conditional forms* [91] (see also [117, 118] for a recent approach concerning the synthesis of *ranking functions*), and use it as a powerful method to describe functions and predicates in algebras and structures. Our current approach to the (automatic) synthesis of models for OS-FOL theories relies on the use of *linear expressions* to define the derivors, which are used to describe functions and predicates using the previously introduced piecewise schema. This is explained in Section 7. The main purpose of using linear expressions is being able to map the final problem into the range of available techniques and tools to deal with linear arithmetic expressions (e.g., SMT techniques [97]). The simultaneous use of different (unbounded, e.g., $\mathbb{R}$; semi-bounded, e.g., $[0, +\infty)$; and bounded, e.g. $[0, 1]$) domains for different sorts can be essential to obtain a model which can be used to solve the problem at stake. The *convex domains* introduced in[5] [80] provide an appropriate framework to generate them, since only linear expressions and formulas are required. Thus, the mechanisms of linear algebra and efficient constraint solving techniques based on linear arithmetics (using SMT) provide a suitable basis for an efficient generation of the models. Section 8 explains how to obtain order-sorted first-order structures based on *convex domains*. Section 9 explains their integration in the linear piecewise approach presented in Section 7.

In program analysis, a standard practice is the use of expressions made up of *parameters* and variables, so that a concrete interpretation (witnessing, e.g., termination) is obtained by instantiating the parameters with numbers by means of a *constraint solving* process[6]. For instance, continuing Remark 2, there is no special reason to prefer $x + y + 1$ over 1 or $2x + 1$ as the interpretation of g. The choice will depend on the specific application described by the theory at stake. Thus, a 'winning' approach is assigning a *parametric* expression $g_1x + g_2y + g_0$ to symbol g, so that parameters g_1, g_2 and g_0 can be appropriately instantiated (to natural numbers) depending on other constraints in the problem. The choice of *linear expressions* is essential to achieve an efficient *mechanization* of the computation of derived models. Section 10 briefly discusses these issues. This methodology is the basis for the implementation of our tool AGES for the automatic generation of models for order-sorted first-order theories [57]. It has also been used in the last

[5] The idea of using domains defined by means of linear inequations in program analysis actually goes back to earlier works like [30], for instance.

[6] In the realm of program termination, see, e.g., [87] for an early reference and [25] for a more systematic approach focusing on termination of rewrite systems.

version of our tool MU-TERM for proving termination properties of (variants of) rewrite systems [1]. Section 11 discusses related work. Section 12 concludes.

This paper is a revised and extended version of [77] and [78, Sections 4 and 5.3]. The main improvements with respect to these previous papers are the following:

1. The definition of function symbols is *relational* (or implicit) [63, pages 71–72] rather than *algebraic* (or explicit).
2. The notion of derivor is generalized to primarily deal with order-sorted signatures with predicates.
3. The notion of derived structure is also new.
4. The notion of *piecewise definition* of functions and predicates is here the most basic one for automation.
5. We provide sufficient conditions to prove well-foundedness of piecewise binary relations.
6. The systematic definition of derivors to transform OS-FOL theories into an OS-FOL theory amenable for automation is made explicit.
7. The whole approach to the synthesis of structures through parametric expressions and constraint solving has been reworked.

2 Order-Sorted First-Order Logic

The material in this section follows [49,52]. Given a set of *sorts* S, a *many-sorted signature* is an $S^* \times S$-indexed family of sets $\Sigma = \{\Sigma_{w,s}\}_{(w,s)\in S^*\times S}$ containing *function symbols* with a given string of argument sorts and a result sort. If $f \in \Sigma_{s_1\cdots s_k,s}$, we often write $f : s_1 \cdots s_k \to s$ (a *rank* declaration for symbol f). Constant symbols c (having no argument) have rank declaration $c : \lambda \to s$ for some sort s (where λ denotes the *empty* sequence). An *order-sorted signature* $(S, \leq, \Sigma)$ consists of a poset of sorts $(S, \leq)$ together with a many-sorted signature (S, Σ). The *connected components* of $(S, \leq)$ are the equivalence classes $[s]$ corresponding to the least equivalence relation $\equiv_\leq$ containing $\leq$. We extend the order $\leq$ on S to strings of equal length in S^* by $s_1 \cdots s_n \leq s'_1 \cdots s'_n$ iff $s_i \leq s'_i$ for all i, $1 \leq i \leq n$. Symbols f can be *subsort-overloaded*, i.e., they can have several rank declarations related in the $\leq$ ordering. Constant symbols, however, have only one rank declaration. Furthermore, the following *monotonicity condition* must be satisfied: $f \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ and $w_1 \leq w_2$ imply $s_1 \leq s_2$. An order-sorted signature Σ is *regular* iff given $w_0 \leq w_1$ in S^* and $f \in \Sigma_{w_1,s_1}$, there is a least $(w, s) \in S^* \times S$ such that $f \in \Sigma_{w,s}$ and $w_0 \leq w$. If, in addition, each connected component $[s]$ of the sort poset has a top element $\top_{[s]} \in [s]$, then the regular signature is called *coherent*.

Given an S-sorted set $\mathcal{X} = \{\mathcal{X}_s \mid s \in S\}$ of *mutually disjoint* sets of variables (which are also disjoint from the signature Σ), the set $\mathcal{T}_\Sigma(\mathcal{X})_s$ of terms of sort s is the least set such that (i) $\mathcal{X}_s \subseteq \mathcal{T}_\Sigma(\mathcal{X})_s$; (ii) if $s' \leq s$, then $\mathcal{T}_\Sigma(\mathcal{X})_{s'} \subseteq \mathcal{T}_\Sigma(\mathcal{X})_s$; and (iii) for each $f : s_1 \ldots s_k \to s$ and $t_i \in \mathcal{T}_\Sigma(\mathcal{X})_{s_i}$, $1 \leq i \leq k$, $f(t_1, \ldots, t_k) \in \mathcal{T}_\Sigma(\mathcal{X})_s$. If $\mathcal{X} = \varnothing$, we write $\mathcal{T}_\Sigma$ rather than $\mathcal{T}_\Sigma(\varnothing)$ for the set of *ground* terms. The set $\mathcal{T}_\Sigma(\mathcal{X})$ of *order-sorted terms* is $\mathcal{T}_\Sigma(\mathcal{X}) = \bigcup_{s\in S} \mathcal{T}_\Sigma(\mathcal{X})_s$. The assumption that Σ is regular yields the very useful property that for any Σ-term t there is a *least sort* $ls_\Sigma(t)$ such that $t \in \mathcal{T}_\Sigma(\mathcal{X})_{ls_\Sigma(t)}$ [52, Proposition 2.10]. Furthermore, if $[s] \neq [s']$, then $\mathcal{T}_\Sigma(\mathcal{X})_{[s]} \cap \mathcal{T}_\Sigma(\mathcal{X})_{[s']} = \varnothing$. In the following, Σ will always be assumed *regular*.

Example 2 The order-sorted signature for `ToyamaOS` is $(S, \leq, \Sigma)$ where $S = \{\mathtt{S}, \mathtt{S1}, \mathtt{S2}\}$ and $\leq$ is the least ordering on S satisfying $\mathtt{S2} \leq \mathtt{S1}$. Thus, $(S, \leq)$ consists of *connected components* $[\mathtt{S}] = \{\mathtt{S}\}$ and $[\mathtt{S1}] = \{\mathtt{S2}, \mathtt{S1}\}$ with `S` (resp. `S1`) the *top sort* of $[\mathtt{S}]$ (resp. $[\mathtt{S1}]$). The signature $\Sigma = \Sigma_{\mathtt{S1}} \cup \Sigma_{\mathtt{S2}} \cup \Sigma_{\mathtt{S1}\,\mathtt{S1},\mathtt{S1}} \cup \Sigma_{\mathtt{S1}\,\mathtt{S1}\,\mathtt{S1},\mathtt{S}}$, with $\Sigma_{\mathtt{S1}} = \{\mathtt{b}\}$, $\Sigma_{\mathtt{S2}} = \{\mathtt{a}\}$, $\Sigma_{\mathtt{S1}\,\mathtt{S1},\mathtt{S1}} = \{\mathtt{g}\}$, and $\Sigma_{\mathtt{S1}\,\mathtt{S1}\,\mathtt{S1},\mathtt{S}} = \{\mathtt{f}\}$ is *regular* and *coherent*.

An order-sorted signature *with predicates* is a quadruple $\Omega = (S, \leq, \Sigma, \Pi)$ such that $(S, \leq, \Sigma)$ is a coherent order-sorted signature, and $\Pi = \{\Pi_w \mid w \in S^*\}$ is a family of *predicate symbols* P, Q, ... We write $P : w$ for $P \in \Pi_w$. Overloading is also allowed on predicates with the following regularity condition[7] [49, Definition 11]: for each w_0 such that there is $P \in \Pi_{w_1}$ with $w_0 \leq w_1$, there is a least w such that $P \in \Pi_w$ and $w_0 \leq w$. Furthermore, if the equality symbol is used (as usual as a *logical symbol*, as in first-order logic), then there is an equality predicate symbol $= \in \Pi_{ss}$ iff s is the top of a connected component of the sort poset S. We often write Σ, Π instead of $(S, \leq, \Sigma, \Pi)$ if S and $\leq$ are clear from the context.

Remark 3 Order-sorted signatures with predicates for OS-TRSs contain (at least) as many overloads for the computational relation $\to^*$ as connected components $[s]$ in $S/_{\equiv_\leq}$: due to axiom (Rf), terms in a class $\mathcal{T}_\Sigma(\mathcal{X})_{[s]}$ rewrite with $\to^*$. By coherence of the signature, we can just let $\to^* \in \Pi_{\top_{[s]}\,\top_{[s]}}$ for all $s \in S$. Then, rule (T) requires a corresponding overload for $\to$ as well, i.e., $\Pi_{\top_{[s]}\,\top_{[s]}} = \{\to, \to^*\}$ for all $s \in S$. This is compatible with any possible instance of rule (Re) because terms ℓ and r in rewrite rules $\ell \to r$ of OS-TRSs belong to $\mathcal{T}_\Sigma(\mathcal{X})_{[s]}$ for some $s \in S$. By coherence, $\ell, r \in \mathcal{T}_\Sigma(\mathcal{X})_{\top_{[s]}}$ for some $s \in S$.

Example 3 The signature $(S, \leq, \Sigma)$ in Example 2 is extended into $(S, \leq, \Sigma, \Pi)$ where

$$\Pi_{\mathtt{S}\,\mathtt{S}} = \Pi_{\mathtt{S1}\,\mathtt{S1}} = \{\to, \to^*\}$$

are the only nonempty sets of predicate symbols.

The formulas φ of an order-sorted signature with predicates Σ, Π are built up from atoms $P(t_1, \ldots, t_n)$ with $P \in \Pi_w$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{X})_w$, logic connectives (e.g., $\wedge$, $\neg$) and quantifiers ($\forall$) as follows: (i) if $P \in \Pi_w$, $w = s_1 \cdots s_n$, and $t_i \in \mathcal{T}_\Sigma(\mathcal{X})_{s_i}$ for all i, $1 \leq i \leq n$, then $P(t_1, \ldots, t_n) \in \mathit{Form}_{\Sigma,\Pi}$; (ii) if $\varphi \in \mathit{Form}_{\Sigma,\Pi}$, then $\neg\varphi \in \mathit{Form}_{\Sigma,\Pi}$; (iii) if $\varphi, \varphi' \in \mathit{Form}_{\Sigma,\Pi}$, then $\varphi \wedge \varphi' \in \mathit{Form}_{\Sigma,\Pi}$; (iv) if $s \in S$, $x \in \mathcal{X}_s$, and $\varphi \in \mathit{Form}_{\Sigma,\Pi}$, then $(\forall x : s)\, \varphi \in \mathit{Form}_{\Sigma,\Pi}$. As usual, we can consider formulas involving other logic connectives and quantifiers (e.g., $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\exists$,...) by using their standard definitions in terms of $\wedge$, $\neg$, $\forall$. A formula without any occurrence of a quantifier is said to be *quantifier-free*. A closed formula, i.e., one whose variables are all universally or existentially quantified, is called a *sentence*.

2.1 Semantics

Given a many-sorted signature (S, Σ), an (S, Σ)-algebra $\mathcal{A}$ (or just a Σ-algebra, if S is clear from the context) is a family $\{\mathcal{A}_s \mid s \in S\}$ of sets called the *carriers* or

[7] As for terms, the regularity requirement guarantees the existence of a *least rank* w for overloaded predicates, see [49, Footnote 7].

domains[8] of $\mathcal{A}$ together with a function $f^{\mathcal{A}}_{w,s} \in \mathcal{A}_w \to \mathcal{A}_s$ for each $f \in \Sigma_{w,s}$ where $\mathcal{A}_w = \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_k}$ if $w = s_1 \cdots s_k$, and $\mathcal{A}_w$ is a one point set when $w = \lambda$. Given an order-sorted signature $(S, \leq, \Sigma)$, an $(S, \leq, \Sigma)$-algebra (or Σ-algebra if $(S, \leq)$ is clear from the context) is an (S, Σ)-algebra such that

1. If $s, s' \in S$ are such that $s \leq s'$, then $\mathcal{A}_s \subseteq \mathcal{A}_{s'}$, and
2. If $f \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ and $w_1 \leq w_2$, then $f^{\mathcal{A}}_{w_1,s_1} \in \mathcal{A}_{w_1} \to \mathcal{A}_{s_1}$ equals $f^{\mathcal{A}}_{w_2,s_2} \in \mathcal{A}_{w_2} \to \mathcal{A}_{s_2}$ on $\mathcal{A}_{w_1}$.

An (S, Σ)-homomorphism between (S, Σ)-algebras $\mathcal{A}$ and $\mathcal{A}'$ is an S-sorted function $h = \{h_s : \mathcal{A}_s \to \mathcal{A}'_s \mid s \in S\}$ such that for each $f \in \Sigma_{w,s}$ with $w = s_1, \ldots, s_k$, $h_s(f^{\mathcal{A}}_{w,s}(a_1, \ldots, a_k)) = f^{\mathcal{A}'}_{w,s}(h_{s_1}(a_1), \ldots, h_{s_k}(a_k))$. If $w = \lambda$, we have $h_s(f^{\mathcal{A}}) = f^{\mathcal{A}'}$. An $(S, \leq, \Sigma)$-homomorphism $h : \mathcal{A} \to \mathcal{A}'$ between $(S, \leq, \Sigma)$-algebras $\mathcal{A}$ and $\mathcal{A}'$ is an (S, Σ)-homomorphism that also satisfies the following condition: if $s \leq s'$ and $a \in \mathcal{A}_s$, then $h_s(a) = h_{s'}(a)$. The family of *domains* $\{\mathcal{T}_\Sigma(\mathcal{X})_s\}_{s \in S}$ together with $f : (t_1, \ldots, t_k) \mapsto f(t_1, \ldots, t_k)$ define an order-sorted Σ-algebra called the *free* algebra on $\mathcal{X}$ and denoted $\mathcal{T}_\Sigma(\mathcal{X})$.

Let $\Omega = (S, \leq, \Sigma, \Pi)$ be an order-sorted signature with predicates. An *Ω-structure*[9] is an order-sorted $(S, \leq, \Sigma)$-algebra $\mathcal{A}$ together with an assignment to each $P \in \Pi_w$ of a subset $P^{\mathcal{A}}_w \subseteq \mathcal{A}_w$ such that [49]: (i) for P the identity predicate $\_ = \_ : ss$, the assignment is the identity relation, i.e., $(=)^{\mathcal{A}}_{s\,s} = \{(a, a) \mid a \in \mathcal{A}_s\}$; and (ii) whenever $P : w_1$ and $P : w_2$ and $w_1 \leq w_2$, then $P^{\mathcal{A}}_{w_1} = \mathcal{A}_{w_1} \cap P^{\mathcal{A}}_{w_2}$.

Let $\mathcal{A}, \mathcal{A}'$ be Ω-structures. Then, an *Ω-homomorphism* $h : \mathcal{A} \to \mathcal{A}'$ is an $(S, \leq, \Sigma)$-homomorphism such that, for each $P \in \Pi_w$ with $w = s_1, \ldots, s_n$, if and only if $(a_1, \ldots, a_n) \in P^{\mathcal{A}}_w$, then $h(a_1, \ldots, a_n) \in P^{\mathcal{A}'}_w$. Given an S-sorted *valuation mapping* $\alpha : \mathcal{X} \to \mathcal{A}$, the evaluation mapping $[\_]^\alpha_{\mathcal{A}} : \mathcal{T}_\Sigma(\mathcal{X}) \to \mathcal{A}$ is the unique $(S, \leq, \Sigma)$-homomorphism extending α [52]. Finally, $[\_]^\alpha_{\mathcal{A}} : \mathit{Form}_{\Sigma,\Pi} \to \mathit{Bool}$ is given by:

1. $[P(t_1, \ldots, t_n)]^\alpha_{\mathcal{A}} = \mathit{true}$ (with $P \in \Pi_w$) if and only if $([t_1]^\alpha_{\mathcal{A}}, \ldots, [t_n]^\alpha_{\mathcal{A}}) \in P^{\mathcal{A}}_w$;
2. $[\neg\varphi]^\alpha_{\mathcal{A}} = \mathit{true}$ if and only if $[\varphi]^\alpha_{\mathcal{A}} = \mathit{false}$;
3. $[\varphi \wedge \psi]^\alpha_{\mathcal{A}} = \mathit{true}$ if and only if $[\varphi]^\alpha_{\mathcal{A}} = \mathit{true}$ and $[\psi]^\alpha_{\mathcal{A}} = \mathit{true}$; and
4. $[(\forall x : s)\, \varphi]^\alpha_{\mathcal{A}} = \mathit{true}$ if and only if for all $a \in \mathcal{A}_s$, $[\varphi]^{\alpha[x \mapsto a]}_{\mathcal{A}} = \mathit{true}$.

A valuation $\alpha \in \mathcal{X} \to \mathcal{A}$ *satisfies* $\varphi \in \mathit{Form}_{\Sigma,\Pi}$ in $\mathcal{A}$ (written $\mathcal{A} \models \varphi\,[\alpha]$) if $[\varphi]^\alpha_{\mathcal{A}} = \mathit{true}$. We then say that φ is *satisfiable*; otherwise, i.e., there is no valuation α such that $\mathcal{A} \models \varphi\,[\alpha]$, we say that φ is *unsatisfiable* or *inconsistent*. If $\mathcal{A} \models \varphi\,[\alpha]$ for *all* valuations α, we write $\mathcal{A} \models \varphi$ and say that $\mathcal{A}$ is a *model* of φ or that φ is *true* in $\mathcal{A}$ [64, page 12]. Initial valuations are not relevant for establishing the satisfiability of *sentences*; thus, both notions coincide on them. Thus, we indistinctly say that $\mathcal{A}$ *satisfies* a sentence φ or that $\mathcal{A}$ is a *model* of φ if $\mathcal{A} \models \varphi$ [7, page 22]. We say that $\mathcal{A}$ is *a model of a set of sentences* $\mathcal{S} \subseteq \mathit{Form}_{\Sigma,\Pi}$ (written $\mathcal{A} \models \mathcal{S}$) if for all $\varphi \in \mathcal{S}$, $\mathcal{A} \models \varphi$. Given a sentence φ, we write $\mathcal{S} \models \varphi$ iff $\mathcal{A} \models \varphi$ holds for *all models* $\mathcal{A}$ of $\mathcal{S}$.

2.1.1 Clauses, normalization

A *literal* is an atom or the negation of an atom. A *clause* is a disjunction of literals. A *set of clauses* S is regarded as a conjunction of all clauses in S, where

[8] Following [64, Section 1.1], these sets can be *empty*.

[9] As in [64], we use 'structure' and reserve the word 'model' to refer those structures satisfying a given set of sentences (theory).

$$\begin{array}{rll}
(\forall x : \mathtt{S}) & x \to^* x & (1)\\
(\forall x : \mathtt{S1}) & x \to^* x & (2)\\
(\forall x, y, z : \mathtt{S}) & x \to y \wedge y \to^* z \Rightarrow x \to^* z & (3)\\
(\forall x, y, z : \mathtt{S1}) & x \to y \wedge y \to^* z \Rightarrow x \to^* z & (4)\\
(\forall x_1, y_1, x_2, x_3 : \mathtt{S1}) & x_1 \to y_1 \Rightarrow \mathtt{f}(x_1, x_2, x_3) \to \mathtt{f}(y_1, x_2, x_3) & (5)\\
(\forall x_1, x_2, y_2, x_3 : \mathtt{S1}) & x_2 \to y_2 \Rightarrow \mathtt{f}(x_1, x_2, x_3) \to \mathtt{f}(x_1, y_2, x_3) & (6)\\
(\forall x_1, x_2, x_3, y_3 : \mathtt{S1}) & x_3 \to y_3 \Rightarrow \mathtt{f}(x_1, x_2, x_3) \to \mathtt{f}(x_1, x_2, y_3) & (7)\\
(\forall x_1, y_1, x_2 : \mathtt{S1}) & x_1 \to y_1 \Rightarrow \mathtt{g}(x_1, x_2) \to \mathtt{g}(y_1, x_2) & (8)\\
(\forall x_1, x_2, y_2 : \mathtt{S1}) & x_2 \to y_2 \Rightarrow \mathtt{g}(x_1, x_2) \to \mathtt{g}(x_1, y_2) & (9)\\
(\forall x : \mathtt{S2}) & \mathtt{f}(\mathtt{a}, \mathtt{b}, x) \to \mathtt{f}(x, x, x) & (10)\\
(\forall x, y : \mathtt{S1}) & \mathtt{g}(x, y) \to x & (11)\\
(\forall x, y : \mathtt{S1}) & \mathtt{g}(x, y) \to y & (12)
\end{array}$$

Fig. 2 Order-Sorted First-Order Theory for `ToyamaOS`

every variable in S is *universally quantified* [22]. For every sentence $\varphi \in Form_{\Sigma,\Pi}$ there is a sentence φ' in *clausal form* (i.e., that can be seen as a *set of clauses* in the above sense) which is inconsistent if and only if φ is [22, Section 4.2] (see [110] for the OS-FOL setting). Thus, "*all questions concerning the satisfiability of sentences in predicate logic can be addressed to sentences in clausal form*" [33, Section 2]. A *Horn clause* is a clause $\neg A_1 \vee \cdots \vee \neg A_n \vee B$ with at most one non-negated atom; in implicative form: $A_1 \wedge \cdots \wedge A_n \Rightarrow B$.

2.2 Theories and programs

A *theory* is a set of sentences. Given a logic $\mathcal{L}$ describing computations in a (declarative) programming language, programs are viewed as *theories* of $\mathcal{L}$ [92, Section 6]. For instance, in the logic of OS-TRSs, the theory for an OS-TRS $\mathcal{R} = (S, \leq, \Sigma, R)$ with set of rules R (for instance, our running example) is obtained from the *schematic* inference rules in Figure 1 after *specializing* them as $(C)_{f,i}$ for each $f \in \Sigma_{s_1 \cdots s_k, s}$ and i, $1 \leq i \leq k$ and $(Re)_\rho$ for all $\rho : \ell \to r \in R$. Then, inference rules $\frac{B_1, \ldots, B_n}{A}$ become *implications* $B_1 \wedge \cdots \wedge B_n \Rightarrow A$.

Example 4 The *theory* for `ToyamaOS` is shown in Figure 2: (1) and (2) specialize (Rf) in Figure 1 for the overloads of $\to^*$; (3) and (4) specialize (T) for the overloads of $\to$ and $\to^*$; (5), (6), and (7) specialize (C) for symbol `f` (using the appropriate overloads of $\to$ according to the rank of `f`) and (8) and (9) specialize (C) for symbol `g`. Finally, (10), (11), and (12) specialize (Re) for each rewrite rule in `ToyamaOS`.

3 Derivors for order-sorted signatures with predicates

The notion of *derivor* [50] generalizes signature morphisms.[10] Derivors d can be used to define theory transformations (also denoted d):

[10] Goguen and Burstall consider the notion of a derivor as *a more general kind of morphism between signatures* [48, Section 6]. Martí-Oliet, Meseguer, and Palomino develop a similar notion called *generalized signature morphism* [90, Definition 3].

1. Each sort $s \in S$ is given a corresponding sort $\tau(s) \in S'$ by some $\tau : S \to S'$.
2. Each function symbol $f \in \Sigma_{s_1 \cdots s_k, s}$ is given a *derived term* $d_{s_1 \cdots s_k, s}(f) \in \mathcal{T}_{\Sigma'}(\mathcal{X}')$ of sort $\tau(s)$ with variables $x_1, \ldots, x_k$ such that, for all i, $1 \leq i \leq k$, $x_i \in \mathcal{X}_{s_i} \cap \mathcal{X}_{\tau(s_i)}$. Here, we assume that each variable $x \in \mathcal{X}_s$ of sort s *remains* as a variable $x \in \mathcal{X}'_{\tau(s)}$ of sort $\tau(s)$ in the S'-indexed set of variables $\mathcal{X}'$. Consequently, for all $s \in S$, such 'imported' variables $x \in \mathcal{X}_s \cap \mathcal{X}'_{\tau(s)}$ do not belong to any other set of variables $\mathcal{X}'_{s'}$ if $s' \neq \tau(s)$, i.e., for all $s \in S$ and $s' \in S'$, if $s' \neq \tau(s)$, then $\mathcal{X}_s \cap \mathcal{X}'_{s'} = \varnothing$.

This is then *extended* into a mapping $d : \mathcal{T}_\Sigma(\mathcal{X}) \to \mathcal{T}_{\Sigma'}(\mathcal{X}')$: each term $t \in \mathcal{T}_\Sigma(\mathcal{X})_s$ of sort $s \in S$ is transformed into a term $d(t)$ of sort $\tau(s)$:

1. If $t = x$ (of sort s), then $d(t) = x$ (of sort $\tau(s)$).
2. If $t = f(t_1, \ldots, t_k)$ with $f \in \Sigma_{w,s}$, then $d(t) = \sigma(d_{w,s}(f))$ where, for all i, $1 \leq i \leq k$, $\sigma(x_i) = d(t_i)$.

We generalize derivors to order-sorted signatures with predicates.

Remark 4 Some presentations of first-order logic do not use function symbols f of arity $k > 0$ (i.e., nonconstant symbols) with the proviso that, by using the *equality symbol*, they can be introduced by means of appropriate predicate symbols $R_f(x_1, \ldots, x_k, y)$ with $k + 1$ arguments (free variables) with the last one playing the role of *output argument*, so that $R_f(x_1, \ldots, x_k, y)$ means $f(x_1, \ldots, x_k) = y$. With some additional conditions we guarantee *totality* and *uniqueness* properties of functions as relations (see [63, pages 71–72] and also Section 4).

This *relational* approach to define functions can be used advantageously. The use of *algebraic* or *rational functions* [15], which are ultimately defined by means of polynomials, can be implemented in this way.

Example 5 The (non-negative) square root $sqrt(x)$ of $x \geq 0$ (an algebraic function) can be defined as follows:

$$R_{sqrt}(x, y) \Leftrightarrow x = y^2 \wedge y \geq 0 \tag{13}$$

And a rational function like $f(x, y, z) = \frac{x+z+xy+yz}{y}$, for real values $x, y, z \geq 1$ (see [76, Example 8]) can be defined as follows:

$$R_f(x, y, z, t) \Leftrightarrow x + z + xy + yz = ty \tag{14}$$

In both cases, we obtain (decidable!) sentences of the *First-Order Logic of the Real Closed Fields* [9, 114]. This provides a basis for their implementation.

The systematic use of this relational approach (Remark 4) leads to a more general notion of derivor which is used in the following (in particular, to deal with the piecewise definition of functions and predicates, see Section 6). In the following definition, we call $\tau : S \to S'$ *monotone* iff $\forall s, s' \in S, s \leq s' \Rightarrow \tau(s) \leq' \tau(s')$ holds.

Definition 1 Let $\Omega = (S, \leq, \Sigma, \Pi)$ and $\Omega' = (S', \leq', \Sigma', \Pi')$ be order-sorted signatures with predicates. A *general derivor* from Ω to Ω' consists of a monotone mapping $\tau : S \to S'$, a mapping $d : S \to Form_{\Sigma', \Pi'}$, a family of mappings $d_{w,s} : \Sigma_{w,s} \to Form_{\Sigma', \Pi'}$, and a family $d_w : \Pi \to Form_{\Sigma', \Pi'}$ such that

1. $d(s)$ is a formula $\Delta_s(x)$ with at most a single free variable $x \in \mathcal{X}_s \cap \mathcal{X}_{\tau(s)}$.
2. for all $f \in \Sigma_{w,s}$, with $w = s_1 \cdots s_k$, $d_{w,s}(f)$ is a formula $\Phi^f_{w,s}(x_1, \ldots, x_k, y) \in$ *Form*$_{\Sigma',\Pi'}$ with free variables $x_i \in \mathcal{X}_{s_i} \cap \mathcal{X}_{\tau(s_i)}$, $1 \leq i \leq k$, and $y \in \mathcal{X}_s \cap \mathcal{X}_{\tau(s)}$.
3. for all $P \in \Pi_w$, with $w = s_1 \cdots s_n$, $d_w(P)$ is a formula $\Phi^P_w(x_1, \ldots, x_n) \in$ *Form*$_{\Sigma',\Pi'}$ with free variables $x_i \in \mathcal{X}_{s_i} \cap \mathcal{X}_{\tau(s_i)}$, $1 \leq i \leq n$.

In the following, when no confusion arises, we often write s' instead of $\tau(s)$.

Remark 5 In Definition 1, $\Delta_s(x)$ is intended to provide an explicit description of sort $s \in S$ in the derived structure $\mathcal{A} = d\mathcal{A}'$ as follows: $\mathcal{A}_s = \{x \in \mathcal{A}'_{\tau(s)} \mid \Delta_s(x)\}$.

Remark 6 A derivor (as in [50], see above) can be seen as a general derivor by using, for each $f \in \Sigma_{w,s}$, an equation $y = d_{w,s}(f)$, where $y \in \mathcal{X}'_s$ is a fresh variable, instead of the *term* $d_{w,s}(f) \in \mathcal{T}_{\Sigma'}(\mathcal{X}')_{\tau(s)}$. We often use the term-based notation for derivors of function symbols and assume the previous translation when necessary.

Every quantifier-free formula $\varphi \in$ *Form*$_{\Sigma,\Pi}$ containing an occurrence of a function symbol $f : w \to s$ (written $\varphi = C[f(t_1, \ldots, t_k)]$) is logically equivalent to the formula $(\forall y : s)\, f(t_1, \ldots, t_k) = y \Rightarrow C[y]$ where y is a fresh variable of sort s not occurring in φ. Proceeding in this way we can *flatten* every formula φ to obtain an equivalent formula where function calls are replaced by new variables holding the value of the call. Conversely, we can use this trick to define functions whose return value depends on the satisfaction of appropriate logical conditions. In this way, a general derivor is extended to a mapping $d :$ *Form*$_{\Sigma,\Pi} \to$ *Form*$_{\Sigma',\Pi'}$ as follows:

1. $d(P(t_1, \ldots, t_n))$, where $P \in \Pi_w$ for some $w \in S^+$, is
$$(\forall y^P_1 : s'_1, \ldots, y^P_n : s'_n)\, \omega(t_1, y^P_1) \wedge \cdots \wedge \omega(t_n, y^P_n) \Rightarrow \sigma(\varphi')$$
where σ is a substitution and, if we let $s' = \tau(s)$ for any $s \in S$ in the following:
   (a) $y^P_1, \ldots, y^P_n$ are new variables of sorts $s'_1, \ldots, s'_n$, respectively.
   (b) φ' with free variables $x_1, \ldots, x_n$ of sorts $s'_1, \ldots, s'_n$, respectively, is obtained from $d_w(P)$ by renaming its bound variables so that no bound variable in φ' occurs free in the antecedent of the implication.
   (c) $\sigma(x_i) = y^P_i$ for all $1 \leq i \leq n$ and $\sigma(x) = x$ for any other variable x.
   (d) $\omega(t, z)$, where t is a term of sort s and z is a variable of sort s', is a formula in *Form*$_{\Sigma',\Pi'}$ defined as follows:
      – If t is a variable $x \in \mathcal{X}_s$, then $\omega(t, z) \overset{\text{def}}{=} x = z$ (note that $x \in \mathcal{X}_{s'}$) and we assume $= \in \Pi'_{s's'}$.
      – If t is $f(t_1, \ldots, t_k)$ with $f \in \Sigma_{w,s}$, $w = s_1, \ldots, s_k$, and $t_i \in \mathcal{T}_\Sigma(\mathcal{X})_{s_i}$ for $1 \leq i \leq k$, then $\omega(t, z)$ is
$$(\forall y^f_1 : s'_1, \ldots, y^f_k : s'_k)\, \omega(t_1, y^f_1) \wedge \cdots \wedge \omega(t_k, y^f_k) \wedge \theta(d_{w,s}(f))$$
      where, if $d_{w,s}(f)$ has free variables $x_1, \ldots, x_k, y$,
         i. $y^f_1, \ldots, y^f_k$ are new variables of sorts $s'_1, \ldots, s'_k$, respectively.
         ii. $\theta(x_i) = y^f_i$ for all $1 \leq i \leq k$, $\theta(y) = z$, and $\theta(x) = x$ for any other variable x.
2. For the logical connectives, we have:
$$d(\neg\varphi) = \neg d(\varphi) \tag{15}$$
$$d(\varphi \wedge \varphi') = d(\varphi) \wedge d(\varphi') \tag{16}$$
$$d((\forall x : s)\varphi) = (\forall x : s')(\Delta_s(x) \Rightarrow d(\varphi)) \tag{17}$$

$$(\forall x : \mathsf{nat}) \quad x \geq x \tag{18}$$
$$(\forall x : \mathsf{nat}) \quad x \geq x \tag{19}$$
$$(\forall x, y, z : \mathsf{nat}) \quad x > y \wedge y \geq z \Rightarrow x \geq z \tag{20}$$
$$(\forall x, y, z : \mathsf{nat}) \quad x > y \wedge y \geq z \Rightarrow x \geq z \tag{21}$$
$$(\forall x_1, y_1, x_2, x_3 : \mathsf{nat}) \quad x_1 > y_1 \Rightarrow x_1 + x_2 + x_3 > y_1 + x_2 + x_3 \tag{22}$$
$$(\forall x_1, x_2, y_2, x_3 : \mathsf{nat}) \quad x_2 > y_2 \Rightarrow x_1 + x_2 + x_3 > x_1 + y_2 + x_3 \tag{23}$$
$$(\forall x_1, x_2, x_3, y_3 : \mathsf{nat}) \quad x_3 > y_3 \Rightarrow x_1 + x_2 + x_3 > x_1 + x_2 + y_3 \tag{24}$$
$$(\forall x_1, y_1, x_2 : \mathsf{nat}) \quad x_1 > y_1 \Rightarrow x_1 + x_2 + 1 > y_1 + x_2 + 1 \tag{25}$$
$$(\forall x_1, x_2, y_2 : \mathsf{nat}) \quad x_2 > y_2 \Rightarrow x_1 + x_2 + 1 > x_1 + y_2 + 1 \tag{26}$$
$$(\forall x : \mathsf{zero}) \quad 0 + 1 + x > x + x + x \tag{27}$$
$$(\forall x, y : \mathsf{nat}) \quad x + y + 1 > x \tag{28}$$
$$(\forall x, y : \mathsf{nat}) \quad x + y + 1 > y \tag{29}$$

Fig. 3 Derived sentences for the sentences in Figure 2

Example 6 Consider $\Omega' = (S', \leq', \Sigma', \Pi')$ with $S' = \{\mathsf{zero}, \mathsf{nat}\}$, $\mathsf{zero} \leq' \mathsf{nat}$, and $\Sigma' = \Sigma'_{\lambda,\mathsf{zero}} \cup \Sigma'_{\lambda,\mathsf{nat}} \cup \Sigma'_{\mathsf{nat}^2\mathsf{nat}}$ where $\Sigma'_{\lambda,\mathsf{zero}} = \{0\}$, $\Sigma'_{\lambda,\mathsf{nat}} = \{1\}$, and $\Sigma'_{\mathsf{nat}^2\mathsf{nat}} = \{+\}$. We define a derivor as follows: $\tau(\mathtt{S}) = \tau(\mathtt{S1}) = \mathsf{nat}$ and $\tau(\mathtt{S2}) = \mathsf{zero}$, $d(\mathtt{S}) = d(\mathtt{S1}) = d(\mathtt{S2}) = \mathit{true}$, $d_{\lambda,\mathtt{S2}}(\mathtt{a}) = 0$, $d_{\lambda,\mathtt{S1}}(\mathtt{b}) = 1$, $d_{\mathtt{S1}\,\mathtt{S1}\,\mathtt{S1},\mathtt{S1}}(\mathtt{f}) = x + y + z$, and $d_{\mathtt{S1}\,\mathtt{S1},\mathtt{S1}}(\mathtt{g}) = x + y + 1$ (see Remark 6). For the overloaded predicates $\to, \to^* \in \Pi_{\mathtt{S}\,\mathtt{S}} \cup \Pi_{\mathtt{S1}\,\mathtt{S1}}$ as follows: $d_{\mathtt{S}\,\mathtt{S}}(\to) \overset{\text{def}}{=} x > y$ and $d_{\mathtt{S}\,\mathtt{S}}(\to^*) \overset{\text{def}}{=} x \geq y$ with $x, y \in \mathcal{X}_{\mathsf{nat}}$ (similarly for $d_{\mathtt{S1}\,\mathtt{S1}}(\to)$ and $d_{\mathtt{S1}\,\mathtt{S1}}(\to)$; note that $\tau(\mathtt{S}) = \tau(\mathtt{S1}) = \mathsf{nat}$). Sentences (1)–(12) are then translated into the *derived* sentences (18)–(29) in Figure 3.

4 Derived structures and models

Since $\Phi^f_{w,s}(x_1, \ldots, x_k, y)$ (see Definition 1) must provide a *functional* interpretation $f^{\mathcal{A}}_{w,s}$ for $f : w \to s$ (with $w = s_1 \cdots s_k$) in any *derived* algebra (or structure) $\mathcal{A} = d\mathcal{A}'$, we need to impose some requirements to such formulas. If $s'_i = \tau(s_i)$ for $1 \leq i \leq k$, and $s' = \tau(s)$, then the following conditions must be satisfied:

1. (Totality/Algebraicity) The outcome y of the function is of sort s':

$$\mathcal{A}' \models (\forall x_1 : s'_1, \ldots, x_k : s'_k \; \exists y : s') \left(\bigwedge_{i=1}^{k} \Delta_{s_i}(x_i) \right) \Rightarrow \Delta_s(y) \wedge \Phi^f_{w,s}(x_1, \ldots, x_k, y) \tag{30}$$

2. (Uniqueness) The outcome of the function is determined by the arguments:

$$\mathcal{A}' \models (\forall x_1 : s'_1, \ldots, x_k : s'_k, y, z : s') \left(\bigwedge_{i=1}^{k} \Delta_{s_i}(x_i) \right) \wedge \Phi^f_{w,s}(x_1, \ldots, x_k, y) \wedge \Phi^f_{w,s}(x_1, \ldots, x_k, z) \Rightarrow y = z \tag{31}$$

The following definition establishes the conditions for a *target* structure to guarantee that a general derivor provides a *sound* description of (i) the subsort relation, (ii) function symbols as *mathematical functions*, and (iii) overloaded symbols.

Definition 2 Given order-sorted signatures with predicates $\Omega = (S, \leq, \Sigma, \Pi)$ and $\Omega' = (S', \leq', \Sigma', \Pi')$, a general derivor $\langle \tau, d\rangle$ from Ω to Ω', and an Ω'-structure $\mathcal{A}' = (\mathcal{A}', \Sigma'_{\mathcal{A}'}, \Pi'_{\mathcal{A}'})$, we say that d is $\mathcal{A}'$-*sound* if the following conditions hold:

1. for all $s_1, s_2 \in S$, if $s_1 \leq s_2$, then $\mathcal{A}' \models (\forall x : s'_1)\, \Delta_{s_1}(x) \Rightarrow \Delta_{s_2}(x)$.
2. for all $f \in \Sigma_{w,s}$, conditions (30) and (31) hold.
3. If $f \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$, $w_1 \leq w_2$, and $w_1 = s_{11} \cdots s_{1k}$, the following holds:

$$\mathcal{A}' \models (\forall x_1 : s'_{11}, \ldots, x_k : s'_{1k}, y : s'_1, z : s'_2)$$
$$\bigwedge_{i=1}^{k} \Delta_{s_{1i}}(x_i) \wedge \Phi^f_{w_1,s_1}(x_1, \ldots, x_k, y) \wedge \Phi^f_{w_2,s_2}(x_1, \ldots, x_k, z) \Rightarrow y = z \quad (32)$$

4. If $P \in \Pi_{w_1} \cap \Pi_{w_2}$, $w_1 \leq w_2$, and $w_1 = s_{11} \cdots s_{1n}$, the following holds

$$\mathcal{A}' \models (\forall x_1 : s'_{11}, \ldots, x_n : s'_{1n})$$
$$\bigwedge_{i=1}^{k} \Delta_{s_{1i}}(x_i) \Rightarrow (\Phi^P_{w_1}(x_1, \ldots, x_n) \Leftrightarrow \Phi^P_{w_2}(x_1, \ldots, x_n)) \quad (33)$$

Definition 3 (Derived structure) Let $\Omega = (S, \leq, \Sigma, \Pi)$ and $\Omega' = (S', \leq', \Sigma', \Pi')$ be order-sorted signatures with predicates and $\langle \tau, d\rangle$ be a general derivor from Ω to Ω'. Let $\mathcal{A}' = (\mathcal{A}', \Sigma'_{\mathcal{A}'}, \Pi'_{\mathcal{A}'})$ be an Ω'-structure such that d is $\mathcal{A}'$-*sound*. The Ω-structure $d\mathcal{A}'$ (denoted $\mathcal{A}$ for short) derived from $\mathcal{A}'$ by $\langle \tau, d\rangle$ consists of:

1. The S-sorted set of domains $\mathcal{A} = \{\mathcal{A}_s \mid s \in S\}$ where, for each sort $s \in S$, $\mathcal{A}_s = \{a \in \mathcal{A}'_{s'} \mid [\Delta_s(x)]^{\{x \mapsto a\}}_{\mathcal{A}'}\}$.
2. For each $f \in \Sigma_{w,s}$ such that $d_{w,s}(f)$ has free variables in $\{x_1, \ldots, x_k, y\}$ where for all i, $1 \leq i \leq k$, $x_i \in \mathcal{X}_{s_i} \cap \mathcal{X}_{s'_i}$ and $y \in \mathcal{X}_s \cap \mathcal{X}_{s'}$, a mapping $f^{\mathcal{A}}_{w,s}$ as follows: for all $a_1 \in \mathcal{A}_{s_1}, \ldots, a_k \in \mathcal{A}_{s_k}$ and $b \in \mathcal{A}_s$, $f^{\mathcal{A}}_{w,s}(a_1, \ldots, a_k) = b$ iff $[d_{w,s}(f)]^{\alpha}_{\mathcal{A}'}$ holds for α given by $\alpha(x_i) = a_i$ for all $1 \leq i \leq k$ and $\alpha(y) = b$.
3. For each $P \in \Pi_w$ such that $d_w(P)$ has free variables in $\{x_1, \ldots, x_n\}$ where for all i, $1 \leq i \leq n$, $x_i \in \mathcal{X}_{s_i} \cap \mathcal{X}_{s'_i}$, interpretations $P^{\mathcal{A}}_w$ defined to be set of tuples in $\mathcal{A}_w$ that satisfy $d_w(P)$, i.e.,

$$P^{\mathcal{A}}_w = \{(a_1, \ldots, a_n) \in \mathcal{A}_w \mid [d_w(P)]^{\alpha}_{\mathcal{A}'}, \alpha(x_i) = a_i, 1 \leq i \leq n\}$$

The following obvious result formalizes the use of the previous construction.

Theorem 1 *Let $\Omega = (S, \leq, \Sigma, \Pi)$ and $\Omega' = (S', \leq', \Sigma', \Pi')$ be order-sorted signatures with predicates and $\langle \tau, d\rangle$ be a derivor from Ω to Ω'. Let $\mathcal{A}'$ be an Ω'-structure such that d is $\mathcal{A}'$-*sound*, and $\mathcal{S} \subseteq Form_{\Sigma,\Pi}$ be a theory. If $\mathcal{A}' \models d(\mathcal{S})$, then $d\mathcal{A}' \models \mathcal{S}$.*

5 Additional requirements in logic and non-logic form

Derived models for a theory $\mathcal{S}$ representing a program analysis or verification problem can be expected to meet some requirements which sometimes can be guaranteed by just adding further OS-FOL sentences. In other cases this is not possible, but we can still add specific requirements on the *interpretation* to achieve the goal. We consider some of them.

5.1 Well-founded relations

Well-foundedness of (binary) relations is required in many important applications (in particular, in termination analysis).

Definition 4 (Well-founded relation) Consider a binary relation R on a set A, i.e., $R \subseteq A \times A$. We say that R is *well-founded* if there is no infinite sequence $a_1, a_2, \ldots$ such that for all $i \geq 1$, $a_i \in A$ and $a_i \, R \, a_{i+1}$.

Well-foundedness can be expressed in *second-order logic* [111, Section 5.1.4], where a new kind of variables (called *relation* and *function* variables) is introduced with an *arity* distinction for them (so that there are n-place predicate and function variables for $n > 0$). Then, a new kind of sentences can be written where such predicate and function variables may occur in the same places where predicate and function symbols (respectively) are allowed in first-order logic; furthermore, they can be quantified using $\forall$ and $\exists$ as well [17, Section 22]. A relation R is well-founded iff the following second-order formula φ holds [111]:

$$\forall X[\exists x(x \in X) \Rightarrow \exists x(x \in X \wedge \forall y(y \in X \Rightarrow \neg(x \; R \; y)))] \tag{34}$$

Here, X is a *monadic* predicate variable and we write $x \in X$ rather than $X(x)$. Unfortunately, the well-foundedness of a relation $P_{ss}^{\mathcal{A}}$ interpreting a binary predicate $P \in \Pi_{ss}$ can *not* be characterized in first-order logic [111, Section 5.1.4].

Remark 7 According to [63, Section 20], φ, i.e., (34), can be expressed in a *two-sorted* FOL with sorts s_1, s_2 by just adding a new predicate symbol $\epsilon : s_1 s_2$ (and giving any other predicate or function symbol a rank using s_1 only) to obtain $\varphi^{\downarrow}$:

$$\forall z^2[\exists x^1(x^1 \epsilon \, z^2) \Rightarrow \exists x^1(x^1 \epsilon \, z^2 \wedge \forall y^1(y^1 \epsilon \, z^2 \Rightarrow \neg(x^1 \;\; R \; y^1)))] \tag{35}$$

so that, for all (second-order) models[11] $\mathcal{A}$ of φ there is a two-sorted *first-order* model $\mathcal{A}^{\downarrow}$ of $\varphi^{\downarrow}$, where $\mathcal{A}^{\downarrow}_{s_1}$ is $\mathcal{A}$ and $\mathcal{A}^{\downarrow}_{s_2}$ is $\mathcal{P}(\mathcal{A})$, the collection of subsets of $\mathcal{A}$. Then, ϵ is interpreted as the membership relation of elements in $\mathcal{A}$ in some set in $\mathcal{P}(\mathcal{A})$. Therefore, the second-order satisfaction $\mathcal{A} \models \varphi$ of φ by $\mathcal{A}$ implies the two-sorted, first-order satisfaction $\mathcal{A}^{\downarrow} \models \varphi^{\downarrow}$ of $\varphi^{\downarrow}$ by $\mathcal{A}^{\downarrow}$. However, if $\mathcal{A}$ is a two-sorted first-order structure satisfying $\varphi^{\downarrow}$, i.e., $\mathcal{A} \models \varphi^{\downarrow}$ holds, this does *not*, in general, imply that the *second-order structure* $\mathcal{A}^{\uparrow}$ (obtained from $\mathcal{A}$ by just disregarding the interpretation of ϵ and all sort information), satisfies φ, i.e., we cannot guarantee that $\mathcal{A}^{\uparrow} \models \varphi$ holds. Hence, finding a model $\mathcal{A}$ of $\varphi^{\downarrow}$ does *not* guarantee that $\mathcal{A}^{\uparrow}$ is a model of φ. Therefore, finding a model of (35) does not guarantee that the 'synthesized' relation $R^{\mathcal{A}}$ is well-founded.

Hence, we guarantee well-foundedness of the relation $P_{ss}^{\mathcal{A}}$ interpreting a predicate $P \in \Pi_{ss}$ at the *semantic* level by an appropriate choice of $P_{ss}^{\mathcal{A}}$ (see Section 8.3.1).

Proposition 1 *Let Ω and Ω' be order-sorted signatures with predicates, $\langle \tau, d \rangle$ be a general derivor from Ω to Ω', and $P \in \Pi_{ss}$ (for some $s \in S$) be such that $d_{ss}(P) = \Phi_{ss}^{P}(x, y)$. Let $\mathcal{A}'$ be an Ω'-structure and $\mathcal{A} = d\mathcal{A}'$. If $R = \{(a, b) \in \mathcal{A}'_{\tau(s)\tau(s)} \mid [\Phi_{ss}^{P}]_{\mathcal{A}'}^{\{x \mapsto a, y \mapsto b\}}\}$ is well-founded, then $P_{ss}^{\mathcal{A}}$ is well-founded.*

[11] Second-order structures are defined as for (unsorted) FOL; the difference with second-order logic is in the treatment of second-order variables that brings a different notion of *satisfaction* [17, page 280].

Proof By contradiction. If $P^{\mathcal{A}}_{ss}$ is not well-founded, then there is an infinite sequence $(a_i)_{i\geq 1}$ with $a_i \in \mathcal{A}_s$ such that for all $i \geq 1$ $(a_i, a_{i+1}) \in P^{\mathcal{A}}_{ss}$. By Definition 3(3), $[\Phi^P_{ss}]^{\{x\mapsto a_i, y\mapsto a_{i+1}\}}_{\mathcal{A}'}$ holds for all $i \geq 1$. Since, by Definition 3(1), $\mathcal{A}_s \subseteq \mathcal{A}'_{\tau(s)}$, it follows that for all $i \geq 1$ $(a_i, a_{i+1}) \in R$, i.e., R is not well-founded, a contradiction.

5.2 Non-empty domains.

An important requirement in termination analysis is that the domain $\mathcal{A}_s$ where a well-founded relation R is defined is *non-empty*.

Remark 8 Termination of (unsorted) rewriting can be proved by using *well-founded monotone algebras* [123, Section 2.1], i.e., algebras $\mathcal{A}$ whose domain is given a *well-founded ordering* $\succ$ such that the following *monotonicity requirement* is satisfied: for all k-ary symbols f, $1 \leq i \leq k$, and $a_1, \ldots, a_k, a, b \in \mathcal{A}$,

$$a \succ b \Rightarrow f^{\mathcal{A}}(a_1, \ldots, a_{i-1}, a, \ldots, a_k) \succ f^{\mathcal{A}}(a_1, \ldots, a_{i-1}, a, \ldots, a_k)$$

Then, an ordering $\succ_{\mathcal{A}}$ on *terms* is defined as follows: for all terms s, t,

$$s \succ_{\mathcal{A}} t \iff (\forall \alpha \in \mathcal{X} \to \mathcal{A})\ [s]^{\alpha}_{\mathcal{A}} \succ [t]^{\alpha}_{\mathcal{A}} \qquad (36)$$

A TRS $\mathcal{R}$ is terminating iff there is a monotone algebra $\mathcal{A}$ with a non-empty domain such that for all rules $\ell \to r$ in $\mathcal{R}$, $\ell \succ_{\mathcal{A}} r$, [123, Proposition 1]. This is because well-foundedness of $\succ$ on $\mathcal{A}$ together with monotonicity induces a *well-founded and monotonic ordering* on terms which can then be used to prove termination of $\mathcal{R}$, according to the well-known Lankford's Theorem [74, page 11]. Indeed, if the domain $\mathcal{A}$ of the algebra is *empty*, then the rightmost 'sentence' in (36) is *vacuously true*, disregarding the terms s and t. Thus, for all terms t, we would have $t \succ_{\mathcal{A}} t \succ_{\mathcal{A}} \cdots$ contradicting the necessary well-foundedness of $\succ_{\mathcal{A}}$.

In a many-sorted or order-sorted setting, the requirement of non-empty domains in algebras or structures could be relaxed as there can be good reasons to do so (see [51] and the references therein). If the signature contains no constant of sort s, we can add a sentence $(\exists x : s)\, x = x$ to our theory $\mathcal{S}$ to guarantee that $\mathcal{A}_s \neq \varnothing$ in any possible interpretation of sort s. By *skolemization*, this is equivalent to adding a *fresh* constant `k` of sort s to the signature.

Example 7 Consider Ω' in Example 6 and the Ω'-structure $\mathcal{A}'$ with $\mathcal{A}'_{\mathtt{nat}} = \mathbb{N}$ and $\mathcal{A}'_{\mathtt{zero}} = \{0\}$. Note that $\mathcal{A}'_{\mathtt{zero}} \subseteq \mathcal{A}'_{\mathtt{nat}}$. Symbols 0, 1, $+$, $\geq$ and $>$ are given the *intended* interpretations over the natural numbers. Then, $\mathcal{A}'$ satisfies the sentences in Figure 3: (18)–(26) and (28)–(29) hold by standard properties of the arithmetic operations and comparison operators (reflexivity and transitivity of $\geq_{\mathbb{N}}$, etc.). And (27) holds due to our *choice* for $\mathcal{A}_{\mathsf{zero}}$: since $\mathcal{A}_{\mathsf{zero}} = \{0\}$, x is restricted to take value 0; thus, $\forall x \in \{0\}\, 0 + 1 + x >_{\mathbb{N}} x + x + x$ becomes $1 >_{\mathbb{N}} 0$, which is true. Since $>_{\mathbb{N}}$ is a well-founded relation over $\mathcal{A}_{\mathsf{nat}} \neq \varnothing$, termination of `ToyamaOS` is proved. The *choice* of a well-founded ordering $>_{\mathbb{N}}$ to interpret $\to$ is essential to conclude termination of `ToyamaOS` from the fact that $\mathcal{A}$ is a model of sentences (18)–(29).

5.3 Specification of requirements in *target* logic form

We assume that the source theory $\mathcal{S} \subseteq \mathit{Form}_{\Sigma,\Pi}$ contains all 'basic' information about the problem at stake (e.g., the semantics of the program as given by the OS-FOL theory `ToyamaOS` in Figure 2) together with any other requirement in *source* logic form (i.e., sentences $\varphi \in \mathit{Form}_{\Sigma,\Pi}$). Requirements that cannot be expressed in this way (e.g., well-foundedness, see Section 5.1), must be guaranteed at the *derived level* by sentences in $\mathit{Form}_{\Sigma',\Pi'}$ interpreted over specific structures $\mathcal{A}'$ so that the requirement is *propagated backwards* (e.g., Proposition 1, regarding well-foundedness). Then, we actually start with a pair $\langle \mathcal{S} \mid \rho \rangle$ where ρ is a *list* of pairs where the requirements are associated to syntactic components of Ω. For instance, $(\to : \mathsf{SS}, \mathsf{wellfounded})$ says that predicate $\to : \mathsf{SS}$ in `ToyamaOS` language should be interpreted as a well-founded relation in the derived structure.

6 Piecewise function and predicate definitions

Derived interpretations $d_{w,s}(f)$ for function symbols $f : w \to s$ (with $w = s_1 \cdots s_k$) can be given by using a *sequence* of $N_{f:w\to s}$ (or just N_f if no confusion arises) terms $t^{f,i} \in \mathcal{T}_{\Sigma'}(\mathcal{X}')_{s'}$, for $1 \leq i \leq N_f$, with variables $x_1, \ldots, x_k$ of sorts $s_1, \ldots, s_k$ (used in $\mathcal{X}'$ with sorts $s'_1, \ldots, s'_k$, see Section 3). The use of terms $t^{f,i}$ is controlled by *qualifiers* $\psi^{f,i} \in \mathit{Form}_{\Sigma',\Pi'}$ which are formulas with free variables $x_1, \ldots, x_k$:

$$d_{w,s}(f) = \begin{cases} t^{f,1} & \text{if } \psi^{f,1} \\ & \vdots \\ t^{f,N_f} & \text{if } \psi^{f,N_f} \end{cases} \tag{37}$$

Eventually, the last formula ψ^{f,N_f} can be *true* (often written "*otherwise*") to *accept* any combination of arguments to f not allowed by qualifiers $\psi^{f,1}, \ldots, \psi^{f,N_f-1}$.

Example 8 Functions max and min are defined as follows:

$$\max(x,y) = \begin{cases} x & \text{if } x \geq y \\ y & \text{otherwise} \end{cases} \qquad \min(x,y) = \begin{cases} x & \text{if } x \leq y \\ y & \text{otherwise} \end{cases}$$

A piecewise function definition $d_{w,s}(f)$ as in (37) is interpreted by the following *characteristic formula*:

$$\Phi^f_{w,s}(x_1, \ldots, x_k, y) \overset{\text{def}}{=} \bigvee_{i=1}^{N_f} \left(\Psi^{f,i}(x_1, \ldots, x_k) \wedge t^{f,i} = y \right) \tag{38}$$

where $x_1, \ldots, x_k$ and y are free variables and the formulas

$$\Psi^{f,i}(x_1, \ldots, x_k) \overset{\text{def}}{=} \bigwedge_{j=1}^{i-1} \neg\psi^{f,j}(x_1, \ldots, x_k) \wedge \psi^{f,i}(x_1, \ldots, x_k) \tag{39}$$

for $1 \leq i \leq N_f$ characterize the *pieces* of the domain of $f^{\mathcal{A}}_{w,s}$ (as in Definition 3) defined by the qualifiers. Note that formulas $\Psi^{f,i}$ *exclude each other*. The domain of the entire function is characterized by the disjunction $\bigvee_{i=1}^{N_f} \Psi^{f,i}$, or just by the disjunction $\bigvee_{i=1}^{N_f} \psi^{f,i}$ of the qualifiers. We say that (37) is a *piecewise function definition* of f and that $\Phi^f_{w,s}$ is its *characteristic formula*.

Remark 9 (Totality/algebraicity) The following *condition*

$$\mathcal{A}' \models (\forall x_1 : s_1, \ldots, x_k : s_k) \bigvee_{i=1}^{N_f} \psi^{f,i} \tag{40}$$

guaranteeing that $d_{w,s}(f)$ denotes a *total function* $f^{\mathcal{A}}_{w,s}$ is easily fulfilled by just letting $\psi^{f,N_f} \overset{\text{def}}{=} \mathit{true}$ (or *otherwise*). Then, (30) becomes

$$\mathcal{A}' \models (\forall x_1 : s'_1, \ldots, x_k : s'_k, y : s') \left(\bigwedge_{i=1}^{k} \Delta_{s_i}(x_i) \right) \wedge \Phi^f_{w,s}(x_1, \ldots, x_k, y) \Rightarrow \Delta_s(y) \tag{41}$$

Note that y is *universally* quantified now.

Remark 10 (Uniqueness) Note that $f^{\mathcal{A}}_{w,s}$ (as in Definition 3) is well-defined: given $a_1 \in \mathcal{A}_{s_1}, \ldots, a_k \in \mathcal{A}_{s_k}, b \in \mathcal{A}_s$, there is at most one $i \in \{1, \ldots, N_f\}$ such that $[\Psi^{f,i}]^{\alpha}_{\mathcal{A}}$ holds with $\alpha = \{x_1 \mapsto a_1, \ldots, x_k \mapsto a_k, y \mapsto b\}$. Therefore, $f^{\mathcal{A}}_{w,s}(a_1, \ldots, a_k)$ is uniquely defined as $[t^{f,i}]^{\alpha}_{\mathcal{A}}$ (which is equal to b, see (38)) because $t^{f,i}$ is a term.

Example 9 Assume that max : `Int Int` $\to$ `Int` is a function symbol from a signature Σ. Consider the *target* signature with predicates $\mathcal{P} = (S^{\mathcal{P}}, \leq^{\mathcal{P}}, \Sigma^{\mathcal{P}}, \Pi^{\mathcal{P}})$ of (a fragment of) Presburger's arithmetic with $S^{\mathcal{P}} = \{\mathsf{int}\}$ and $\leq^{\mathcal{P}}$ being the equality. Only predicate symbols $\geq, = \in \Pi^{\mathcal{P}}_{\mathsf{int}\,\mathsf{int}}$ are used. Let $\tau(\mathtt{Int}) = \mathsf{int}$ and $\Delta_{\mathtt{Int}}(x) \overset{\text{def}}{=} \mathit{true}$ with x of sort int. The characteristic formula for $d_{\mathtt{Int}\ \mathtt{Int},\mathtt{Int}}(\max)$ as in Example 8 is:

$$\Phi^{\max}_{\mathtt{Int}\ \mathtt{Int},\mathtt{Int}}(x, y, z) \overset{\text{def}}{=} (x \geq y \wedge x = z) \vee (\neg(x \geq y) \wedge \mathit{true} \wedge y = z)$$

Interpretations for *predicate symbols* $P \in \Pi_w$ (with $w = s_1 \cdots s_n$) can also be given by using a *sequence* of $N_{P:w}$ (or just N_P) *test pieces* $\varphi^{P,i} \in \mathit{Form}_{\Sigma',\Pi'}$ for $1 \leq i \leq N_P$ which are formulas with free variables $x_1, \ldots, x_n$ such that, for all i, $1 \leq i \leq n$, $x_i \in \mathcal{X}_{s_i} \cap \mathcal{X}_{s'_i}$ whose use is controlled by *qualifiers* $\psi^{P,i} \in \mathit{Form}_{\Sigma',\Pi'}$ with (the same) free variables $x_1, \ldots, x_n$, written

$$d_w(P) \Leftrightarrow \begin{cases} \varphi^{P,1} & \text{if } \psi^{P,1} \\ & \vdots \\ \varphi^{P,N_P} & \text{if } \psi^{P,N_P} \end{cases} \tag{42}$$

We think of $d_w(P)$ as decomposed into N_P *pieces* characterized by the formulas

$$\Psi^{P,i}(x_1, \ldots, x_n) \overset{\text{def}}{=} \bigwedge_{j=1}^{i-1} \neg\psi^{P,j}(x_1, \ldots, x_n) \wedge \psi^{P,i}(x_1, \ldots, x_n) \tag{43}$$

for $1 \leq i \leq N_P$, which *exclude each other*. Then, for all $a_1 \in \mathcal{A}_{s_1}, \ldots, a_n \in \mathcal{A}_{s_n}$, $P^{\mathcal{A}}_w(a_1, \ldots, a_n)$ is equivalent to $[\varphi^{P,i}]^{\alpha}_{\mathcal{A}}$ with $\alpha = \{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$, provided that $[\Psi^{P,i}]^{\alpha}_{\mathcal{A}}$ holds. This corresponds to the following *characteristic formula*:

$$\Phi^P_w(x_1, \ldots, x_n) \overset{\text{def}}{=} \bigvee_{i=1}^{N_f} \left(\Psi^{P,i}(x_1, \ldots, x_n) \wedge \varphi^{P,i}(x_1, \ldots, x_n) \right) \tag{44}$$

with free variables $x_1, \ldots, x_n$. We say that $d_w(P)$ given as in (42) is a *piecewise predicate definition* with characteristic formula Φ^P_w.

Remark 11 A single-row definition, i.e., $N_P = 1$, is a 'degenerate' case which is equivalent to defining $d_w(P)$ as $\varphi^{P,1} \wedge \psi^{P,1}$, thus making notation (42) quite useless.

Example 10 The *lexicographic product* $>^{lex}$ of strict orderings $>_i$ on sets A_i for $1 \leq i \leq n$ is a relation on tuples $\mathbf{x}, \mathbf{y} \in A_1 \times \cdots \times A_n$ defined as follows:

$$\mathbf{x} = (x_1, \ldots, x_n)^T >^{lex} (y_1, \ldots, y_n)^T = \mathbf{y} \Leftrightarrow \begin{cases} x_1 >_1 y_1 & \textit{if } x_1 \neq_1 y_1 \\ \vdots & \\ x_{n-1} >_{n-1} y_{n-1} & \textit{if } x_{n-1} \neq_{n-1} y_{n-1} \\ x_n >_n y_n & \textit{otherwise} \end{cases}$$

6.1 Well-foundedness of piecewise binary relations

A binary relation R on a set A is *disjunctively well-founded* if it is the union of a finite set of well-founded relations, i.e., $R = \bigcup_{i=1}^{n} R_i$ for well-founded relations $R_1, \ldots, R_n \subseteq A \times A$ [104]. A transitive and disjunctively well-founded relation R is *well-founded.* In the following, we say that a formula $\overline{\varphi}(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$ of sorts $s_1, \ldots, s_n$, respectively, is an *overapproximation* of a formula $\varphi(x_1, \ldots, x_n)$ (with the same free variables) if the following sentence holds:

$$(\forall x_1 : s_1, \ldots, x_n : s_n)\, \varphi(x_1, \ldots, x_n) \Rightarrow \overline{\varphi}(x_1, \ldots, x_n)$$

Theorem 2 *Let $\Omega = (S, \leq, \Sigma, \Pi)$ be an order-sorted signature with predicates, $s \in S$, $x, y \in \mathcal{X}_s$, $N > 0$, and $\mathcal{A}$ be an Ω-structure. Let*

$$R(x, y) \Leftrightarrow \begin{cases} \varphi^1 & \textit{if } \psi^1 \\ \vdots & \\ \varphi^N & \textit{if } \psi^N \end{cases} \tag{45}$$

with $\varphi^1, \ldots, \varphi^N, \psi^1, \ldots, \psi^N \in Form_{\Sigma,\Pi}$ having free variables x, y and $\Phi_{ss}(x, y) \overset{def}{=} \bigvee_{i=1}^{N} \left(\Psi^i(x, y) \wedge \varphi^i(x, y)\right)$, where Ψ^i is as in (43), using φ^i instead of $\varphi^{P,i}$ and ψ^i instead of $\psi^{P,i}$. For all $1 \leq i \leq N$, let $\Theta_i(x, y)$ be an overapproximation of $\Psi^i(x, y) \wedge \varphi^i(x, y)$ and $\overline{\Phi}_{ss}(x, y) \overset{def}{=} \bigvee_{i=1}^{N} \Theta_i(x, y)$. If

1. *for all i, $1 \leq i \leq N$, $R_i = \{(a, b) \in \mathcal{A}_{ss} \mid [\Theta_i]_{\mathcal{A}}^{\{x \mapsto a, y \mapsto b\}}\}$ is well-founded, and*
2. *$\mathcal{A} \models (\forall x, y, z : s)(\overline{\Phi}_{ss}(x, y) \wedge \overline{\Phi}_{ss}(y, z) \Rightarrow \overline{\Phi}_{ss}(x, z))$,*

then $R_{ss}^{\mathcal{A}} = \{(a, b) \in \mathcal{A}_{ss} \mid [\Phi_{ss}]_{\mathcal{A}}^{\{x \mapsto a, y \mapsto b\}}\}$ is well-founded.

Example 11 Using the notation in (43) and (44) regarding the piecewise definition of a predicate $P \in \Pi_{ss}$, let $\Theta_i(x, y) \overset{\text{def}}{=} \psi^{P,i}(x, y) \wedge \varphi^{P,i}(x, y)$. Note that $\Theta_i(x, y)$ overapproximates $\Psi^{P,i}(x, y) \wedge \varphi^{P,i}(x, y)$ because $\Theta_i(x, y)$ is obtained from it by removing the 'negative' conjuncts $\bigwedge_{j=1}^{i-1} \neg\psi^{P,j}(x, y)$ from $\Psi^{P,i}(x, y)$, see (43).

We use Theorem 2 and the overapproximation in Example 11 in Theorem 4 below.

7 Piecewise definitions based on linear expressions

In the following, we use a logic based on linear expressions with intended numerical interpretation. Let $\Lambda = (\mathsf{N}, \leq_{\mathsf{N}}, \mathsf{LExp}, \mathsf{BCmp})$ be a signature with predicates where:

1. N consists of sorts $\nu_1, \nu_2, \ldots$ that will be interpreted as *numerical structures* (essentially sets of numerical vectors).
2. $\leq_{\mathsf{N}}$ is an ordering on N.
3. The signature LExp is the union of $\mathsf{LExp}_{\lambda,\nu_i}$, $\mathsf{LExp}_{\nu_i,\nu_j}$ and $\mathsf{LExp}_{\nu_i\nu_i,\nu_i}$, where, for each $i, j \in \mathbb{N}$,
   - $\mathsf{LExp}_{\lambda,\nu_i}$ consists of *constant symbols* (we call them *constant coefficients*),
   - Symbols $c \in \mathsf{LExp}_{\nu_i,\nu_j}$ (called *linear coefficients*) permit the definition of *linear monomials* cx of sort ν_j for each variable x of sort ν_i, and
   - $\mathsf{LExp}_{\nu_i\nu_i,\nu_i} = \{+\}$ contains overloaded versions of the addition operator.
4. BCmp is the union of $\mathsf{BCmp}_{\nu_i'\nu_i'} = \{>, \geq, \leq, <, =\}$ for each $i \in \mathbb{N}$. We do *not* assume any specific relationship among them. For instance, we do not assume $x \geq y$ as equivalent to $x > y \vee x = y$.

We define a (generic) derivor for a given signature with predicates Ω where function and predicate symbols are given piecewise definitions based on linear expressions in Λ. We also need a set of formulas Ξ to be satisfied by the considered structure $\mathcal{A}'$ in order to guarantee that the obtained derivor is *safe* (Definition 2).

1. *Sorts.* Define an *injective* mapping $\tau : S \to \mathsf{N}$; in the following, $\tau(s) \in \mathsf{N}$ is denoted ν_s. Each sort s is given a *domain inequality* $\Delta_s(x)$ as follows:
$$s_1 x \geq s_0 \tag{46}$$
with $s_1 \in \mathsf{LExp}_{\nu_s,\nu}$ and $s_0 \in \mathsf{LExp}_{\lambda,\nu}$ for some $\nu \in \mathsf{N}$.
2. *Subsorts.* The subsort relation $\leq_{\mathsf{N}}$ among sorts in N is the least one satisfying: if $s \leq s'$ for $s, s' \in S$, then $\nu_s \leq_{\mathsf{N}} \nu_{s'}$. We add the following formulas to Ξ:
$$\{(\forall x : \nu_s)\, s_1 x \geq s_0 \Rightarrow s_1' x \geq s_0' \mid s, s' \in S, s \leq s'\} \tag{47}$$
3. *Constants.* For each $f \in \Sigma_{\lambda,s}$, $d_{\lambda,s}(f) = f_0 \in \mathsf{LExp}_{\lambda,\nu_s}$. We add a sentence $s_1 f_0 \geq s_0$ (algebraicity, see (41)) to Ξ.
4. *Non-constant function symbols.* For each $f \in \Sigma_{w,s}$, with $w = s_1 \cdots s_k$, $k > 0$, define N_f, the number of rows of the piecewise function definition for f. Then:
$$d_{w,s}(f) = \begin{cases} f^1(x_1, \ldots, x_k) & \text{if } \overline{f}^1(x_1, \ldots, x_k) \geq \overline{f}_0^1 \\ \vdots & \\ f^{N_f - 1}(x_1, \ldots, x_k) & \text{if } \overline{f}^{N_f-1}(x_1, \ldots, x_k) \geq \overline{f}_0^{N_f-1} \\ f^{N_f}(x_1, \ldots, x_k) & \text{otherwise} \end{cases} \tag{48}$$
where for all $1 \leq j \leq N_f$,
   - $f^j(x_1, \ldots, x_k) \overset{\text{def}}{=} \sum_{i=1}^{k} f_i^j x_i + f_0^j$ with $f_i^j \in \mathsf{LExp}_{\nu_{s_i},\nu_s}$ for all $1 \leq i \leq k$ $f_0^j \in \mathsf{LExp}_{\lambda,\nu_s}$ and $+ \in \mathsf{LExp}_{\nu_s\nu_s,\nu_s}$, and
   - $\overline{f}^j(x_1, \ldots, x_k) \overset{\text{def}}{=} \sum_{i=1}^{k} \overline{f}_i^j x_i$ with $\overline{f}_i^j \in \mathsf{LExp}_{\nu_{s_i},\nu_j}$ for all $1 \leq i \leq k$, and $\overline{f}_0^j \in \mathsf{LExp}_{\lambda,\nu_j}$ for some $\nu_j \in \mathsf{N}$ and $+ \in \mathsf{LExp}_{\nu_j\nu_j,\nu_j}$.

   For each characteristic formula (38), we add a sentence (41) to Ξ.

5. *Overloaded functions.* For each $f \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ with $w \leq w'$, add (32) to Ξ.
6. *Equality.* For each $= \in \Pi_{ss}$, let $d_{ss}(=) \stackrel{\text{def}}{=} x_1 = x_2$ where the equality symbol in the right-hand side of the definition is $= \in \mathsf{BCmp}_{\nu_s \nu_s}$.
7. *Predicate symbols.* For each $P \in \Pi_w$ with $w = s_1, \ldots, s_n$, let

$$d_w(P) = \begin{cases} P^1(x_1, \ldots, x_n) \geq P_0^1 & \text{if } \overline{P}^1(x_1, \ldots, x_n) \geq \overline{P}_0^1 \\ \vdots & \\ P^{N_P}(x_1, \ldots, x_n) \geq P_0^{N_P} & \text{if } \overline{P}^{N_P}(x_1, \ldots, x_n) \geq \overline{P}_0^{N_P} \end{cases} \tag{49}$$

   for some $N_P > 1$, where for all $1 \leq j \leq N_P$, $P^j(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \sum_{i=1}^n P_i^j x_i$ and there are $\nu_j, \nu_j' \in \mathsf{N}$ such that for all $1 \leq i \leq n$, $P_i^j \in \mathsf{LExp}_{\nu_{s_i}, \nu_j}$, $P_0^j \in \mathsf{LExp}_{\lambda, \nu_j}$, $\overline{P}_i^j \in \mathsf{LExp}_{\nu_{s_i}, \nu_j'}$, and $\overline{P}_0^j \in \mathsf{LExp}_{\lambda, \nu_j'}$.
8. *Overloaded predicates.* For each $P \in \Pi_w \cap \Pi_{w'}$ with $w \leq w'$, we add (33) to Ξ.

Example 12 Let $\Xi = \varnothing$. For `ToyamaOS` we obtain the following derivor:

1. Let $\Delta_{\mathsf{S}}(x) \stackrel{\text{def}}{=} \mathtt{S}_1 x \geq \mathtt{S}_0$, $\Delta_{\mathsf{S1}}(x) \stackrel{\text{def}}{=} \mathtt{S1}_1 x \geq \mathtt{S1}_0$, and $\Delta_{\mathsf{S2}}(x) \stackrel{\text{def}}{=} \mathtt{S2}_1 x \geq \mathtt{S2}_0$ where $\mathtt{S}_1 \in \mathsf{LExp}_{\nu_{\mathsf{S}}, \nu}$, $\mathtt{S1}_1 \in \mathsf{LExp}_{\nu_{\mathsf{S1}}, \nu_1}$, $\mathtt{S2}_1 \in \mathsf{LExp}_{\nu_{\mathsf{S2}}, \nu_2}$, $\mathtt{S}_0 \in \mathsf{LExp}_{\lambda, \nu}$, $\mathtt{S1}_0 \in \mathsf{LExp}_{\lambda, \nu_1}$, and $\mathtt{S2}_0 \in \mathsf{LExp}_{\lambda, \nu_2}$ for some $\nu, \nu_1, \nu_2 \in \mathsf{N}$.
2. $\nu_{\mathsf{S2}} \leq_{\mathsf{N}} \nu_{\mathsf{S1}}$. We add $(\forall x : \nu_{\mathsf{S2}})\, \mathtt{S2}_1 x \geq \mathtt{S2}_0 \Rightarrow \mathtt{S1}_1 x \geq \mathtt{S1}_0$ to the set Ξ:
3. Let $N_{\mathtt{a}} = N_{\mathtt{b}} = 1$ and $N_{\mathtt{f}} = N_{\mathtt{g}} = 2$. Then, $d_{\lambda, \mathsf{S2}}(\mathtt{a}) = \mathtt{a}_0$, $d_{\lambda, \mathsf{S1}}(\mathtt{b}) = \mathtt{b}_0$, and

$$d_{\mathsf{S1\,S1\,S1,S}}(\mathtt{f}) = \begin{cases} \mathtt{f}_1^1 x_1 + \mathtt{f}_2^1 x_2 + \mathtt{f}_3^1 x_3 + \mathtt{f}_0^1 & \text{if } \overline{\mathtt{f}}_1^1 x_1 + \overline{\mathtt{f}}_2^1 x_2 + \overline{\mathtt{f}}_3^1 x_3 \geq \overline{\mathtt{f}}_0^1 \\ \mathtt{f}_1^2 x_1 + \mathtt{f}_2^2 x_2 + \mathtt{f}_3^2 x_3 + \mathtt{f}_0^2 & \text{otherwise} \end{cases}$$

$$d_{\mathsf{S1\,S1,S1}}(\mathtt{g}) = \begin{cases} \mathtt{g}_1^1 x_1 + \mathtt{g}_2^1 x_2 + \mathtt{g}_0^1 & \text{if } \overline{\mathtt{g}}_1^1 x_1 + \overline{\mathtt{g}}_2^1 x_2 \geq \overline{\mathtt{g}}_0^1 \\ \mathtt{g}_1^2 x_1 + \mathtt{g}_2^2 x_2 + \mathtt{g}_0^2 & \text{otherwise} \end{cases}$$

   Accordingly, we add algebraicity conditions (41) for `a`, `b`, `f`, and `g` to Ξ.
4. Let $N_{\to:\mathsf{SS}} = N_{\to:\mathsf{S1\,S1}} = N_{\to^*:\mathsf{S\,S}} = N_{\to^*:\mathsf{S1\,S1}} = 1$. Then,

$$\begin{array}{ll} d_{\mathsf{S\,S}}(\to) = r_1 x_1 + r_2 x_2 \geq r_0 & d_{\mathsf{S1\,S1}}(\to) = r_1' x_1 + r_2' x_2 \geq r_0' \\ d_{\mathsf{S\,S}}(\to^*) = s_1 x_1 + s_2 x_2 \geq s_0 & d_{\mathsf{S1\,S1}}(\to^*) = s_1' x_1 + s_2' x_2 \geq s_0' \end{array}$$

7.1 Normal form of derived formulas

In the following, we assume $\mathcal{S}' = d(\mathcal{S}) \cup \Xi$ normalized as a set of universally quantified clauses (Section 5.3) consisting of (possibly negated) *atoms* of the form

$$A(x_1, \ldots, x_m) = b \text{ or } A(x_1, \ldots, x_m) \geq b \tag{50}$$

for variables $x_1, \ldots, x_m$ of sorts $\nu_1, \ldots, \nu_m$, where $A(x_1, \ldots, x_m) = \sum_{i=1}^m A_i x_i$ is a *linear expression* with $A_i \in \mathsf{LExp}_{\nu_i, \nu}$ for some sort ν, $b \in \mathsf{LExp}_{\lambda, \nu}$, and $=, \geq \in \mathsf{BCmp}_{\nu, \nu}$. That is, $\varphi' \in \mathcal{S}'$ has the following (implicative) form, for $M, P, Q \in \mathbb{N}$:

$$(\forall x_1 : \nu_1, \ldots, x_M : \nu_M) \bigwedge_{i=1}^{P} B_i^-(\mathbf{x}) \Rightarrow \bigvee_{j=1}^{Q} B_j^+(\mathbf{x}) \tag{51}$$

where both B_i^- and B_j^+ are *atoms* of one of the forms (50).

As remarked above, we consider linear expressions in a broad sense, including those with *matrices* as multiplicative factors. The following section shows how to define a sufficiently flexible class of structures which can be used for our purposes.

Sort	C^s	$\mathbf{b}^s$	$D(\mathsf{C}^s,\mathbf{b}^s)$	N	$\mathcal{A}_s = D_N(\mathsf{C}^s,\mathbf{b}^s)$
$\varnothing$	(0)	(1)	$\varnothing$	–	$\varnothing$
Nat	(1)	(0)	$[0,+\infty)$	$\mathbb{Z}$	$\mathbb{N}$
NzNat	(1)	(1)	$[1,+\infty)$	$\mathbb{Z}$	$\{1,2,\ldots\}$
Zero	$(1,-1)^T$	$(0,0)^T$	$\{0\}$	–	$\{0\}$
Bool	$(1,-1)^T$	$(0,-1)^T$	$[0,1]$	$\mathbb{Z}$	$\{0,1\}$
Char	$(1,-1)^T$	$(0,-255)^T$	$[0,255]$	$\mathbb{Z}$	$\{0,1,\ldots,255\}$
Int	(0)	(0)	$\mathbb{R}$	$\mathbb{Z}$	$\mathbb{Z}$

Fig. 4 Convex domains for some usual sorts

8 Order-Sorted Structures with Convex Domains

In this section we introduce a class of structures which can be systematically used in the last step of the synthesis of models through the definition of *piecewise and linear derivors* as in the previous section. Our starting point are the convex polytopic domains introduced for termination analysis in [80].

Definition 5 [80, Definition 1] Given a matrix $\mathsf{C} \in \mathbb{R}^{m\times n}$, and $\mathbf{b} \in \mathbb{R}^m$, the set $D(\mathsf{C},\mathbf{b}) = \{\mathbf{x} \in \mathbb{R}^n \mid \mathsf{C}\mathbf{x} \geq \mathbf{b}\}$ is called a *convex polytopic domain.*

In Definition 5, vectors $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$ are *compared* using the *coordinate-wise* extension of the ordering $\geq$ among *numbers* which, by abuse, we denote using $\geq$ as well:

$$\mathbf{x} = (x_1,\ldots,x_n)^T \geq (y_1,\ldots,y_n)^T = \mathbf{y} \text{ iff } x_1 \geq y_1 \wedge \cdots \wedge x_n \geq y_n \tag{52}$$

In the following, we introduce a simple approach to define structures based on convex polytopic domains including functions and predicates. Section 9 explores its combination with the piecewise scheme discussed before.

8.1 Domains

Sorts $s \in S$ are given convex domains $\mathcal{A}_s = D(\mathsf{C}^s,\mathbf{b}^s)$, where $\mathsf{C}^s \in \mathbb{R}^{m_s\times n_s}$ is an $m_s \times n_s$-matrix and $\mathbf{b}^s \in \mathbb{R}^{m_s}$. Thus, $\mathcal{A}_s \subseteq \mathbb{R}^{n_s}$. Given $s \in S$, we have to *fix* m_s and n_s according to some criterion.

Remark 12 (Bounded domains) In order to generate a *bounded* domain $\mathcal{A}_s \subseteq [\alpha,\beta]^{n_s}$ for some $\alpha,\beta \in \mathbb{R}$, we need to impose $m_s \geq n_s+1$. Indeed, convex polytopic domains are intersections of hyperplanes defined by $\mathsf{C}^s_{i.}\mathbf{x} \geq \mathbf{b}_i$ for all $1 \leq i \leq m_s$, where $\mathsf{C}^s_{i.}$ is the i-th row of matrix C^s. Thus, we need to intersect at least $n_s + 1$ such hyperplanes to enclose $D(\mathsf{C}^s,\mathbf{b}^s)$ into $[\alpha,\beta]^{n_s}$. For intervals ($n_s = 1$), fixing $m_s = 2$ suffices because more than 2 rows in C^s adds nothing.

Convex domains can be parameterized by a subset $N \subseteq \mathbb{R}$ with $\mathsf{C} \in N^{m\times n}$, and $\mathbf{b} \in N^m$ and defining $D_N(\mathsf{C},\mathbf{b}) = \{\mathbf{x} \in N^n \mid \mathsf{C}\mathbf{x} \geq \mathbf{b}\}$. Figure 4 shows *intended* interpretations as convex domains for some usual sorts.

Regarding the *subsort* relation, if $s \leq s'$, then $\mathcal{A}_s = D(\mathsf{C}^s,\mathbf{b}^s) \subseteq D(\mathsf{C}^{s'},\mathbf{b}^{s'}) = \mathcal{A}_{s'}$ must hold. This is achieved by the following sentence:

$$(\forall x \in \mathbb{R}^{n_s})\, \mathsf{C}^s x \geq \mathbf{b}^s \Rightarrow \mathsf{C}^{s'} x \geq \mathbf{b}^{s'} \tag{53}$$

We need $n_s = n_{s'}$ so that the objects in both domains have the same dimension and the aforementioned inclusion makes sense.

8.2 Functions

By a *many-sorted convex matrix intepretation* for $f : w \to s$ where $w = s_1 \cdots s_k$, we mean a linear expression $F_1x_1 + \cdots + F_kx_k + F_0$ such that

1. $F_0 \in \mathbb{R}^{n_s}$; for all i, $1 \leq i \leq k$, $F_i \in \mathbb{R}^{n_s \times n_{s_i}}$ are $n_s \times n_{s_i}$-matrices and x_i are variables ranging on $\mathcal{A}_{s_i}$, and
2. the following *algebraicity condition* is satisfied:

$$\forall x_1 \in \mathbb{R}^{n_{s_1}}, \ldots \forall x_k \in \mathbb{R}^{n_{s_k}} \left(\bigwedge_{i=1}^{k} \mathsf{C}^{s_i} x_i \geq \mathbf{b}^{s_i} \Rightarrow \mathsf{C}^{s}(\sum_{i=1}^{k} F_i x_i + F_0) \geq \mathbf{b}^{s} \right) \quad (54)$$

If $k = 0$ (f is a constant symbol $f : \lambda \to s$), then condition (54) becomes $\mathsf{C}^s F_0 \geq \mathbf{b}^s$.

8.3 Predicates

Each predicate symbol $P \in \Pi_w$ with $w = s_1 \cdots s_k$ (we use 'k' here to avoid confusions with the use of 'n' for the dimension of the domains) is given an inequality

$$R_1x_1 + \cdots + R_kx_k \geq R_0 \qquad \text{or } \textstyle\sum_{i=1}^{k} R_ix_i \geq R_0 \text{ for short} \quad (55)$$

where (i) $R_0 \in \mathbb{R}^{m_P}$ for some $m_P > 0$ and for all i, $1 \leq i \leq k$, (ii) $R_i \in \mathbb{R}^{m_P \times n_{s_i}}$ are $m_P \times n_{s_i}$-matrices, and (iii) x_i are variables ranging on $\mathcal{A}_{s_i}$. Then,

$$P_w^{\mathcal{A}} = \{(x_1, \ldots, x_k) \in \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_k} \mid \sum_{i=1}^{k} R_ix_i \geq R_0\}$$

or, in our specific setting,

$$P_w^{\mathcal{A}} = \{(x_1, \ldots, x_k) \in \mathbb{R}^{n_{s_1}} \times \cdots \times \mathbb{R}^{n_{s_k}} \mid \textstyle\bigwedge_{i=1}^{k} \mathsf{C}^{s_i} x_i \geq \mathbf{b}^{s_i} \wedge \sum_{i=1}^{k} R_ix_i \geq R_0\}$$

Note that $P_w^{\mathcal{A}} \subseteq \mathcal{A}_w$, as required.

Remark 13 If $w = \lambda$ ($k = 0$), then $P_w^{\mathcal{A}} = \{() \mid \mathbf{0} \geq R_0\}$ is a singleton $\{()\}$ if $\mathbf{0} \geq R_0$ (P interpreted as *true*) and an empty set $\varnothing$ if $\mathbf{0} \not\geq R_0$ (P interpreted as *false*).

Example 13 (Equality) The interpretation of an equality predicate $= \in \Pi_{s\,s}$ must be the *equality* relation $\{(x, x) \mid x \in \mathcal{A}_s\}$ on $\mathcal{A}_s$. With $m_P = 2n_s$, $R_1, R_2 \in \mathbb{R}^{m_P \times n_s}$ given by $R_1 = \begin{bmatrix} I_{n_s} \\ -I_{n_s} \end{bmatrix}$ (for I_{n_s} the *identity* matrix of $n_s \times n_s$ entries), $R_2 = -R_1$, and $R_0 = (0, \ldots, 0)^T \in \mathbb{R}^{m_P}$, we obtain the equality on $\mathbb{R}^{n_s}$.

Example 14 (Orderings) The ordering $\geq$ on n-tuples $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ (52) is obtained if $R_1 = I_n$, $R_2 = -I_n$ and $R_0 = \mathbf{0}$.

Remark 14 Let $P, P' \in \Pi_w$ with $w = s_1 \cdots s_k$ be given inequalities $\sum_{i=1}^{k} R_ix_i \geq R_0$ and $\sum_{i=1}^{k} R'_ix_i \geq R'_0$, respectively, where for all $i, 1 \leq i \leq k$, R_i is an $m \times n_{s_i}$ matrix and R'_i is an $m' \times n_{s_i}$ matrix for some positive integers m and m'. Then, the *conjunction* $P(x_1, \ldots, x_k) \wedge P'(x_1, \ldots, x_k)$ can be seen as a *single* linear inequality

$$\sum_{i=1}^{k} \begin{bmatrix} R_i \\ R'_i \end{bmatrix} x_i \geq \begin{bmatrix} R_0 \\ R'_0 \end{bmatrix}$$

where the i-th matrix coefficient is an $(m + m') \times n_{s_i}$-matrix and the constant coefficient a vector in $\mathbb{R}^{m+m'}$.

8.3.1 Well-foundedness

The following result provides a sufficient condition to guarantee *well-foundedness* of a binary relation R on $\mathbb{R}^n$ defined as explained in Section 8.3.

Theorem 3 *Let $R_1, R_2 \in \mathbb{R}^{m\times n}$ and $R_0 \in \mathbb{R}^m$ for some $m, n > 0$, and R be a binary relation on $A \subseteq \mathbb{R}^n$ as follows: for all $\mathbf{x}, \mathbf{y} \in A$, $\mathbf{x}\, R\, \mathbf{y}$ if and only if $R_1\mathbf{x} + R_2\mathbf{y} \geq R_0$. If there is $i \in \{1, \dots, n\}$ such that (a) $(R_2)_{i\cdot} = -(R_1)_{i\cdot}$, i.e., the i-th row of R_2 is obtained from the i-th row of R_1 by negating all components, (b) there is $\alpha \in \mathbb{R}$ such that for all $\mathbf{x} \in A$, $(R_1)_{i\cdot}\mathbf{x} \geq \alpha$, and (c) $(R_0)_i > 0$, then R is well-founded.*

Proof By contradiction. If R is not well-founded, then there is an infinite sequence $\mathbf{x}_1, \dots, \mathbf{x}_n, \dots$ of vectors in A such that, for all $j \geq 1$, $x_j\, R\, x_{j+1}$. By (a) we have that, for all $j \geq 1$, $(R_1)_{i\cdot}\mathbf{x}_j - (R_1)_{i\cdot}\mathbf{x}_{j+1} \geq (R_0)_i$. For all $p > 0$,

$$\sum_{j=1}^{p} (R_1)_{i\cdot}\mathbf{x}_j - (R_1)_{i\cdot}\mathbf{x}_{j+1} = (R_1)_{i\cdot}\mathbf{x}_1 - (R_1)_{i\cdot}\mathbf{x}_{p+1} \geq p(R_0)_i$$

By (b), there is $\alpha \in \mathbb{R}$ such that for all $p > 0$, $(R_1)_{i\cdot}\mathbf{x}_p \geq \alpha$. Therefore, for all $p > 0$, $(R_1)_{i\cdot}\mathbf{x}_1 - \alpha \geq (R_1)_{i\cdot}\mathbf{x}_1 - (R_1)_{i\cdot}\mathbf{x}_{p+1}$, and then $(R_1)_{i\cdot}\mathbf{x}_1 - \alpha \geq p(R_0)_i$. Let $r = (R_0)_i$. By (c), $r > 0$. Then, for all $p > 0$, $(R_1)_{i\cdot}\mathbf{x}_1 \geq \alpha + pr$, leading to a contradiction because $\alpha + pr$ tends to infinity as p grows to infinity, but $(R_1)_{i\cdot}\mathbf{x}_1 \in \mathbb{R}$ is fixed.

Example 15 Borrowing [2], the following *strict* ordering on vectors in $\mathbb{R}^n$:

$$(x_1, \dots, x_n)^T >_\delta (y_1, \dots, y_n)^T \text{ iff } x_1 >_\delta y_1 \wedge (x_2, \dots, x_n)^T \geq (y_2, \dots, y_n)^T$$

is obtained if $R_1 = I_n$, $R_2 = -I_n$ and $R_0 = (\delta, 0, \dots, 0)^T$. Here, given $\delta > 0$, for all $x, y \in \mathbb{R}$, $x >_\delta y$ iff $x - y \geq \delta$, see [76]. Theorem 3 guarantees the well-foundedness of the restriction of $>_\delta$ to any $A \subseteq \mathbb{R}^n$ such that $A \subseteq [\alpha, \infty)^n$ for some $\alpha \in \mathbb{R}$.

Example 16 The following *strict* ordering on vectors in $\mathbb{R}^n$: $\mathbf{x} >^w_\Sigma \mathbf{y}$ iff $\mathbf{x} \geq \mathbf{y} \wedge \sum_{i=1}^n x_i >_1 \sum_{i=1}^n y_i$, borrowing the "*weak decrease + strict decrease in sum of components*" ordering over tuples of natural numbers in [99, Definition 3.1], is obtained if $m_P = n + 1$ (i.e., R_1, R_2 are $(n + 1) \times n$-matrices and $R_0 \in \mathbb{R}^{n+1}$) and

$$R_1 = \begin{bmatrix} \mathbf{1}^T \\ I_n \end{bmatrix} \qquad R_2 = -R_1 \qquad R_0 = (\delta, 0, \dots, 0)^T$$

for some $\delta > 0$, where $\mathbf{1}$ is the constant vector $(1, \dots, 1)^T \in \mathbb{R}^n$. Take $A \subseteq [\alpha, +\infty)^n$, for some $\alpha \geq 0$ and $i = 1$ with the corresponding R_1, R_2, and R_0 to prove $>^w_\Sigma$ well-founded on A. Theorem 3 guarantees well-foundedness of $>^w_\Sigma$.

The following result is a simple consequence of Theorem 2 when the overapproximation in Example 11 is considered, and taking into account Remark 14.

Theorem 4 *Let R be a binary relation on $A \subseteq \mathbb{R}^n$,* piecewise *defined as follows*

$$R(x, y) = \begin{cases} R^1(x, y) \geq R^1_0 & \textit{if } \widehat{R}^1(x, y) \geq \widehat{R}^1_0 \\ \vdots & \\ R^N(x, y) \geq R^N_0 & \textit{if } \widehat{R}^N(x, y) \geq \widehat{R}^N_0 \end{cases} \tag{56}$$

for some $N > 0$, where for all $1 \leq i \leq N$, $R^i(x,y) \overset{def}{=} R^i_1 x + R^i_2 y$ and $\widehat{R}^i(x,y) \overset{def}{=} \widehat{R}^j_1 x + \widehat{R}^i_2 y$, with $R^i_1, R^i_2 \in \mathbb{R}^{m_i \times n}$, $\widehat{R}^i_1, \widehat{R}^i_1 \in \mathbb{R}^{m'_i \times n}$, $R^i_0 \in \mathbb{R}^{m_i}$ and $\widehat{R}^i_0 \in \mathbb{R}^{m'_i}$ for some $m_i, m'_i \in \mathbb{N}$. Let $\Phi(x,y) \overset{def}{=} \bigvee_{i=1}^N \left(\Psi^i(x,y) \wedge \varphi^i(x,y)\right)$, where Ψ^i is as in (43), using $R^i(x,y) \geq R^i_0$ instead of $\varphi^{P,i}$ and $\widehat{R}^i(x,y) \geq \widehat{R}^i_0$ instead of $\psi^{P,i}$. Let $\overline{\Phi}(x,y) \overset{def}{=} \bigvee_{i=1}^N \Theta_i(x,y)$ where for all $1 \leq i \leq N$

$$\Theta_i(x,y) \overset{def}{=} \begin{bmatrix} R^i_1 \\ \widehat{R}^i_1 \end{bmatrix} x + \begin{bmatrix} R^i_2 \\ \widehat{R}^i_2 \end{bmatrix} y \geq \begin{bmatrix} R^i_0 \\ \widehat{R}^i_0 \end{bmatrix} \tag{57}$$

If the relations defined by $\Theta_i(x,y)$ on A are well-founded for all $1 \leq i \leq N$, and

$$(\forall x, y, z \in A)\ \overline{\Phi}(x,y) \wedge \overline{\Phi}(y,z) \Rightarrow \overline{\Phi}(x,z) \tag{58}$$

holds, then $R = \{(\mathbf{a},\mathbf{b}) \in A^2 \mid \Phi(\mathbf{a},\mathbf{b})$ holds$\}$ is well-founded (on A).

Well-foundedness of relations Θ_i in Theorem 4 can be proved using Theorem 3.

Example 17 For pairs (x_1, x_2) of numbers, the lexicographic ordering admits a compact definition as a piecewise predicate:

$$\mathbf{x} = (x_1,x_2)^T >^{lex} (y_1,y_2)^T = \mathbf{y} \Leftrightarrow \begin{cases} x_2 > y_2 & \text{if } x_1 = y_1 \\ x_1 > y_1 & \text{otherwise} \end{cases}$$

which is written in the *linear format* with *convex-domain* interpretation as follows:

$$\mathbf{x} >^{lex} \mathbf{y} \Leftrightarrow \begin{cases} \begin{bmatrix} 0\ 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0\ {-1} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \geq \begin{bmatrix} 1 \end{bmatrix} & \text{if } \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 1\ 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -1\ 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \geq \begin{bmatrix} 1 \end{bmatrix} & \text{otherwise} \end{cases}$$

In order to prove $>^{lex}$ well-founded, we use Theorem 4 as follows:

1. The two components of the piecewise relation for $>^{lex}$ are:

$$\Theta_1(\mathbf{x},\mathbf{y}) \overset{\text{def}}{=} \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ -1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{59}$$

$$\Theta_2(\mathbf{x},\mathbf{y}) \overset{\text{def}}{=} \begin{bmatrix} 1\ 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -1\ 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \geq \begin{bmatrix} 1 \end{bmatrix} \tag{60}$$

   Note that $\Theta_1(\mathbf{x},\mathbf{y}) \Leftrightarrow x_2 \geq y_2 + 1 \wedge x_1 = y_1$ and $\Theta_2(\mathbf{x},\mathbf{y}) \Leftrightarrow x_1 \geq y_1 + 1$.
2. Well-foundedness of Θ_1 and Θ_2 on $[0, +\infty)^2$ can be proved using Theorem 3.
3. Regarding (58), we have to prove:

$$(\Theta_1(x,y) \vee \Theta_2(x,y)) \wedge (\Theta_1(y,z) \vee \Theta_2(y,z)) \Rightarrow (\Theta_1(x,z) \vee \Theta_2(x,z))$$

   that is:

$$((x_2 \geq y_2 + 1 \wedge x_1 = y_1) \vee x_1 \geq y_1 + 1) \wedge ((y_2 \geq z_2 + 1 \wedge y_1 = z_1) \vee y_1 \geq z_1 + 1) \\ \Rightarrow ((x_2 \geq z_2 + 1 \wedge x_1 = z_1) \vee x_1 \geq z_1 + 1)$$

   which can be proved true by considering the different combinations of cases.

Thus, we conclude well-foundedness of $>^{lex}$ using Theorem 4.

9 Structures with convex domains and piecewise definitions

The piecewise linear schema to define derivors introduced in Section 7 is used together with Λ-structures $\mathcal{A}'$ based on convex polytopic domains to derive an Ω-structure $\mathcal{A} \overset{\text{def}}{=} d\mathcal{A}'$ as explained in Definition 3:

- *Sorts.* Sorts $\nu \in \mathsf{N}$ are interpreted as $\mathcal{A}'_\nu = N^{n_\nu}$ where N is a set of numbers (e.g., $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, etc.) and $n_\nu > 0$.

  Remark 15 Actually, N should be a *ring with identity*[12] (e.g., $\mathbb{Z}$, $\mathbb{Q}$, etc.) so that we can use matrix algebra to deal with linear applications for a *vector space* over N [106, Section 1.3]. Although this excludes $\mathbb{N}$, we can still use it as the domain of a sort $s \in S$ by means of the domain constraints (see Figure 4).

  The choice of N (typically $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{R}$) essentially depends on the availability of techniques to prove satisfiability of the formulas that are obtained.
- *Subsorts.* If ν, ν' are such that $\nu \leq_{\mathsf{N}} \nu'$, then $n_\nu = n_{\nu'}$.
- *Function symbols.*
  1. Each constant $c \in \mathsf{LExp}_{\lambda,\nu}$ for $\nu \in \mathsf{N}$ is interpreted as a *vector* $c^{\mathcal{A}}_{\lambda,\nu} \in N^{n_\nu}$.
  2. Each function $c\cdot \in \mathsf{LExp}_{\nu,\nu'}$ is interpreted as a linear mapping from n_ν-dimensional vectors into $n_{\nu'}$-dimensional vectors given by a *matrix* $c\cdot^{\mathcal{A}}_{\nu,\nu'} \in N^{n_{\nu'} \times n_\nu}$ as usual (e.g., [106, Section 6.2]).
  3. Each operator $+ \in \mathsf{LExp}_{\nu\ \nu,\nu}$ is interpreted as the (componentwise) addition $+^{\mathcal{A}}_{\nu\ \nu,\nu}$ of n_ν-dimensional vectors.
- *Predicate symbols.* We only use $= \in \mathsf{BCmp}_{\nu\ \nu}$ (interpreted as in Example 13) and $\geq \in \mathsf{BCmp}_{\nu\ \nu}$ (interpreted as in Example 14), for each $\nu \in \mathsf{N}$.

  Remark 16 N is assumed to be ordered by a partial order (i.e., a reflexive, antisymmetric, and transitive relation [31, Definition 1.2]).

Example 18 A Λ-structure to be used with the derivor in Example 12 is as follows. Sorts $\nu_{\mathtt{S}}$, $\nu_{\mathtt{S1}}$, and $\nu_{\mathtt{S2}}$ and auxiliary sorts ν and ν_1 are all interpreted as $\mathbb{Z}$. Sort ν_2 is interpreted as $\mathbb{Z}^2$. The coefficients for the domain inequalities are:

$$\mathtt{S}_1 = 1 \qquad \mathtt{S}_0 = 0 \qquad \mathtt{S1}_1 = 1 \qquad \mathtt{S1}_0 = 0 \qquad \mathtt{S2}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad \mathtt{S2}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Therefore, the derived domains for $\mathtt{S}$, $\mathtt{S1}$, and $\mathtt{S2}$ are:

$$\begin{aligned}
\mathcal{A}_{\mathtt{S}} &= \{x \in \mathbb{Z} \mid \mathtt{S}_1 x \geq \mathtt{S}_0\} = \{x \in \mathbb{Z} \mid x \geq 0\} = \mathbb{N} \\
\mathcal{A}_{\mathtt{S1}} &= \{x \in \mathbb{Z} \mid \mathtt{S1}_1 x \geq \mathtt{S1}_0\} = \{x \in \mathbb{Z} \mid x \geq 0\} = \mathbb{N} \\
\mathcal{A}_{\mathtt{S2}} &= \{x \in \mathbb{Z} \mid \mathtt{S2}_1 x \geq \mathtt{S2}_0\} = \{x \in \mathbb{Z} \mid \begin{bmatrix} 1 \\ -1 \end{bmatrix} x \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix}\} = \{0\}
\end{aligned}$$

With the following assignment for the function symbols:

$$\begin{array}{llll}
\mathtt{a}_0 = 0 & \mathtt{f}^1_1 = \mathtt{f}^1_2 = \mathtt{f}^1_3 = 1 & \mathtt{f}^1_0 = 0 & \overline{\mathtt{f}}^1_1 = \overline{\mathtt{f}}^1_2 = \overline{\mathtt{f}}^1_3 = \overline{\mathtt{f}}^1_0 = 0 \\
\mathtt{b}_0 = 1 & \mathtt{f}^2_1 = \mathtt{f}^2_2 = \mathtt{f}^2_3 = 0 & \mathtt{f}^2_0 = 0 & \\
 & \mathtt{g}^1_1 = \mathtt{g}^1_2 = 1 & \mathtt{g}^1_0 = 1 & \overline{\mathtt{g}}^1_1 = \overline{\mathtt{g}}^1_2 = \overline{\mathtt{g}}^1_0 = 0 \\
 & \mathtt{g}^2_1 = \mathtt{g}^2_2 = 0 & \mathtt{g}^2_0 = 0 &
\end{array}$$

[12] A *ring with identity* is a set with a rule of addition and a rule of multiplication satisfying the commutative, associative, zero element, inverse and distributive rules [106, Section 1.3].

and for the predicate symbols:

$$\begin{array}{llllll} r_1 = 1 & r_2 = -1 & r_0 = 1 & r'_1 = 1 & r'_2 = -1 & r'_0 = 1 \\ s_1 = 1 & s_2 = -1 & s_0 = 0 & s'_1 = 1 & s'_2 = -1 & s'_0 = 0 \end{array}$$

we obtain the structure in Example 6 as a particular case of Example 12.

9.1 Checking satisfiability of sentences

Now, we have to *check* whether $\mathcal{A}'$ satisfies $\mathcal{S}'$. Since variables $x : \nu$ actually represent *tuples* $(a_1, \ldots, a_{n_\nu}) \in N^{n_\nu}$, we think of such a variable as a sequence $x_1, \ldots, x_{n_\nu}$ of n_ν variables ranging on N. For instance, consider

$$(\forall x : \nu_s)\, s_1 x \geq s_0 \Rightarrow s'_1 x \geq s'_0 \tag{61}$$

as in (47) for sorts $\nu_s, \nu_{s'}$ such that $\nu_s \leq_{\mathsf{N}} \nu_{s'}$. Provided that

1. $\nu_s \in \mathsf{N}$ is interpreted as the set $N^{n_{\nu_s}}$ of tuples of n_{ν_s} numbers in N,
2. $(s_1)^{\mathcal{A}'}_{\nu_s,\nu}$ is a matrix $A^1 \in N^{n_\nu \times n_{\nu_s}}$ for some sort ν, $(s_0)^{\mathcal{A}'}_{\lambda,\nu}$ is a vector $A^0 \in N^{n_\nu}$, $(s'_1)^{\mathcal{A}'}_{\nu_{s'},\nu'}$ is a matrix $B^1 \in N^{n_{\nu'} \times n_{\nu_s}}$ for some sort ν' (remember that $n_{\nu_s} = n_{\nu_{s'}}$ due to $\nu_s \leq_{\mathsf{N}} \nu_{s'}$), and $(s'_0)^{\mathcal{A}'}_{\lambda,\nu}$ is a vector $B^0 \in N^{n_{\nu'}}$,

formula (61) is treated as the following sentence involving a conjunction of affine arithmetic inequalities (recall that $n_s = n_{s'}$)

$$(\forall x_1, \ldots, x_{n_{\nu_s}} \in N) \left(\bigwedge_{i=1}^{m_s} \sum_{j=1}^{n_s} A^1_{ij} x_j \geq A^0_i \right) \Rightarrow \left(\bigwedge_{i=1}^{m_{s'}} \sum_{j=1}^{n_s} B^1_{ij} x_j \geq B^0_i \right) \tag{62}$$

which corresponds to the matrix-vector product (the $\sum$ expressions) together with the *pointwise* comparison of components of tuples (the $\bigwedge$ connectives).

Once $\mathcal{A}'$ is fixed as explained above, the standard definition of satisfaction is used to check whether $\mathcal{A}'$ satisfies $\mathcal{S}'$. However, for the sake of the *automation*, it is worth to make it *explicit* as we do *not* provide the matrices and vectors in the definition of the structure. This is addressed in the next section.

10 Parametric structures and constraint-solving

The *automatic generation* of models for a theory (e.g., $\mathcal{S}' = d(\mathcal{S}) \cup \Xi$) is a *bottom-up* process where things remain 'unspecified' until an attempt to *solve* some constraints obtained from $\mathcal{S}'$ succeeds. The *solution* is then used to synthesize a structure which yields (by construction) a model of $\mathcal{S}'$ and then of $\mathcal{S}$ (Theorem 1). This is accomplished by interpreting the function and predicate symbols *without an intended interpretation* as parametric objects: symbols $b \in \mathsf{LExp}_{\lambda,\nu}$ and $C \in \mathsf{LExp}_{\nu,\nu'}$ for $\nu, \nu' \in \mathsf{N}$ are given parametric vectors and matrices, respectively:

$$\mathbf{b}^\nu = \begin{bmatrix} \mathbf{b}^\nu_1 \\ \vdots \\ \mathbf{b}^\nu_{n_\nu} \end{bmatrix} \qquad \mathsf{C}^{\nu,\nu'} = \begin{bmatrix} \mathsf{C}^{\nu,\nu'}_{11} & \cdots & \mathsf{C}^{\nu,\nu'}_{1n_\nu} \\ \vdots & \ddots & \vdots \\ \mathsf{C}^{\nu,\nu'}_{n_{\nu'}1} & \cdots & \mathsf{C}^{\nu,\nu'}_{n_{\nu'}n_\nu} \end{bmatrix}$$

where $\mathsf{C}^{\nu,\nu'}_{ij}$ and $\mathbf{b}^\nu_i$ are *parameters*, i.e., *variables* assumed to be *existentially quantified* in any formula during the generation process.

10.1 Parametric sentences from linear sentences

Clauses $\varphi \in \mathcal{S}'$ of the form (51) are translated into *parametric clauses* $\varpi(\varphi)$:

1. If φ is $A(x^1,\ldots,x^m) \bowtie b$ for variables $x^1,\ldots,x^m$ of sort $\nu_1,\ldots,\nu_m$, respectively, $A(x^1,\ldots,x^m) = \sum_{j=1}^m A^j x^j$ with $A^j \in \mathsf{LExp}_{\nu_j,\nu}$ for some sort ν, $b \in \mathsf{LExp}_{\lambda,\nu}$, and $\bowtie \in \{=,\geq\} \subseteq \mathsf{BCmp}_{\nu,\nu}$, then:
$$\varpi(A(x^1,\ldots,x^m) \bowtie b) = \bigwedge_{i=1}^{n_\nu} \sum_{j=1}^{m} \sum_{k=1}^{n_{\nu_j}} A^i_{jk} \cdot x^j_k \bowtie b_i \tag{63}$$
where the A^i_{jk} and b_i are parameters. The *multiplication rule* of N (see Remark 15) is represented by '$\cdot$'. And $\bowtie$ is the equality or inequality relation *on* N.
2. $\varpi(\varphi \wedge \varphi') = \varpi(\varphi) \wedge \varpi(\varphi')$ (similarly $\varpi(\varphi \vee \varphi') = \varpi(\varphi) \vee \varpi(\varphi')$).
3. $\varpi(\varphi \Rightarrow \varphi') = \varpi(\varphi) \Rightarrow \varpi(\varphi')$.
4. $\varpi((\forall x : \nu)\varphi) = (\forall x_1 \in N,\ldots,x_{n_\nu} \in N)\varpi(\varphi)$

In this way, we obtain a set $\mathcal{S}^\sharp = \varpi(\mathcal{S}')$ of *parametric sentences.*

10.2 Fulfilling well-foundedness requirements

For binary predicates $P \in \Pi_{ss}$ which are required to be *well-founded*, we use Theorems 3 and 4 to guarantee that the synthesized interpretation $P^{d\mathcal{A}'}_{ss}$ is well-founded. For instance, assume that (according to Section 7),

1. the domain inequality $\Delta_s(x)$ for sort s is $s^1 x \geq s^0$, with $s_1 \in \mathsf{LExp}_{\nu_s,\nu}$ and $s_0 \in \mathsf{LExp}_{\lambda,\nu}$ for some $\nu \in \mathsf{N}$.
2. $d_{ss}(P) = P^1 y_1 + P^2 y_2 \geq P^0$, with $P^1, P^2 \in \mathsf{LExp}_{\nu_s,\nu'}$ for some $\nu' \in \mathsf{N}$ and $P^0 \in \mathsf{LExp}_{\lambda,\nu'}$ (we use y to avoid confusions with the x's in the formulas below).

The application of Theorem 3 amounts at adding the following formula to $\mathcal{S}^\sharp$:
$$\bigvee_{i=1}^{n_{\nu'}} \left[\bigwedge_{j=1}^{n_{\nu_s}} P^1_{ij} = -P^2_{ij} \wedge P^0_i > 0 \wedge \left(\left(\bigwedge_{j=1}^{n_\nu} \sum_{k=1}^{n_{\nu_s}} s^1_{jk} x_k \geq s^0_j \right) \Rightarrow \sum_{j=1}^{n_{\nu_s}} P^1_{ij} \cdot x_j \geq \alpha \right) \right] \tag{64}$$
with $x_1,\ldots,x_{n_{\nu_s}}$ universally quantified on N and α a *parameter* (existentially quantified at the outermost level of the sentence). Theorem 4 would be used likewise.

10.3 Normal form of parametric sentences

After normalization, $\mathcal{S}^\sharp$ is a set of *clauses* of the following shape, for $M,P,Q \in \mathbb{N}$:
$$(\forall x_1,\ldots,x_M) \bigwedge_{i=1}^{P} e^-_i(\boldsymbol{\pi},\mathbf{x}) \bowtie d^-_i \Rightarrow \bigvee_{i=1}^{Q} e^+_i(\boldsymbol{\pi},\mathbf{x}) \bowtie d^+_i \tag{65}$$

where (after applying some arithmetic rules to relocate some components)

1. $\boldsymbol{\pi}$ is a vector of parameters taken from $\pi_1,\ldots,\pi_K$ and ranging on appropriate (search) domains of parameters, included in N,

2. $\mathbf{x}$ is a vector of variables taken from $x_1, \ldots, x_M$ and ranging on N,
3. $e_i^-(\boldsymbol{\pi}, \mathbf{x})$ and $e_i^+(\boldsymbol{\pi}, \mathbf{x})$ are expressions $\sum_k \pi_k \cdot x_k$ for parameters π_k and variables x_k (note that they are *linear* regarding variables x_k),
4. d_i^- and d_i^+ are parameters (or 0),
5. $\bowtie \in \{=, \geq, >\}$ are the usual comparison operators on numbers.

10.4 Quantifier elimination using Farkas' lemma

If each clause (65) can be written as a *set* of clauses in the following *affine form*:

$$(\forall x_1, \ldots, x_M) \bigwedge_{i=1}^{P'} e_i(\boldsymbol{\pi}, \mathbf{x}) \geq d_i \Rightarrow e(\boldsymbol{\pi}, \mathbf{x}) \geq d \tag{66}$$

for some $P' \in \mathbb{N}$, then the following Affine form of Farkas' Lemma [108, cf. Corollary 7.1h] considered in [80, Section 5.1] is useful.

Theorem 5 (Affine form of Farkas' Lemma) *Let $A\mathbf{x} \geq \mathbf{b}$ be a linear system of k inequalities and n unknowns over the real numbers with non-empty solution set S and let $\mathbf{c} \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$. Then, the following statements are equivalent:*

1. $\mathbf{c}^T\mathbf{x} \geq \beta$ for all $\mathbf{x} \in S$,
2. $\exists \boldsymbol{\lambda} \in \mathbb{R}_0^k$ such that $\mathbf{c} = A^T\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^T\mathbf{b} \geq \beta$.

We use (2) in Theorem 5 as a *sufficient condition* for (1): proving $\forall \mathbf{x}\ (A\mathbf{x} \geq \mathbf{b} \Rightarrow c^T\mathbf{x} \geq \beta)$ recasts into the *constraint solving problem* of finding a nonnegative vector $\boldsymbol{\lambda}$ such that $\mathbf{c}$ is a linear nonnegative combination of the *rows* of A and β is smaller than the corresponding linear combination of the components of $\mathbf{b}$.[13] For this reason, Theorem 5 can be used with matrices $A \in N^{m\times n}$, vectors $\mathbf{x}, \mathbf{b}, \mathbf{c} \in N^n$, and $\beta \in N$, with $N \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}\}$. Since $N \subseteq \mathbb{R}$, whenever (2) holds, we have that $\mathbf{c}^T\mathbf{x} \geq \beta$ holds for all $\mathbf{x} \in S = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \geq \mathbf{b}\}$. Thus, in particular it is true for the subset $S \cap N^n = \{\mathbf{x} \in N^n \mid A\mathbf{x} \geq \mathbf{b}\}$ we are interested in.

In order for sentences (65) to fit format (66), we (repeatedly) do the following:

1. Use (from left to right) the tautologies:

$$A \Rightarrow B \vee C \Leftrightarrow A \wedge \neg B \Rightarrow C \tag{67}$$

$$A \Rightarrow B \wedge C \Leftrightarrow A \Rightarrow B \wedge A \Rightarrow C \tag{68}$$

$$A \wedge (B \vee B') \Rightarrow C \Leftrightarrow (A \wedge B \Rightarrow C) \wedge (A \wedge B' \Rightarrow C) \tag{69}$$

   In particular, (67) is used to move positive constraints $e_i^+(\boldsymbol{\pi}, \mathbf{x}) \bowtie d_i^+$ in (65) to the antecedent of the implications. It also can be used to move negated atoms to the antecedent thus removing the negation: $A \Rightarrow \neg B \vee C \Leftrightarrow A \wedge B \Rightarrow C$.
2. Atoms $e(\boldsymbol{\pi}, \mathbf{x}) = d$ are replaced by[14] $e(\boldsymbol{\pi}, \mathbf{x}) \geq d \wedge -e(\boldsymbol{\pi}, \mathbf{x}) \geq -d$. If the atom is 'positive' (of type $e^+(\boldsymbol{\pi}, \mathbf{x})$), use (68) afterwards.

[13] If $A\mathbf{x} \geq \mathbf{b}$ has no solution, i.e., S in Theorem 5 is *empty*, the conditional sentence (1) trivially holds. Thus, we do not need to check S for emptiness when using Farkas' result, although the systematic use of (2) to check (1) in this case may fail, thus eventually leading to loose some positive answers for (1).

[14] We rely on the existence of inverse additive elements of the ring structure of N for this, see Remark 15; also on the antisymmetry of $\geq$ (Remark 16).

3. Negated atoms in the antecedent of an implication, (like those eventually obtained after applying (67)) which are based on predicates $=$ and $\geq$ yield atoms with $\neq$ and $<$ which are not allowed in (66). We also may have atoms with $>$ (see Section 10.2). If $N = \mathbb{Z}$, we can easily deal with these situations:
   (a) Replace $e(\boldsymbol{\pi}, \mathbf{x}) > d$ by $e(\boldsymbol{\pi}, \mathbf{x}) \geq d + 1$.
   (b) Replace $e(\boldsymbol{\pi}, \mathbf{x}) < d$ by $-e(\boldsymbol{\pi}, \mathbf{x}) \geq 1 - d$.
   (c) Replace $e(\boldsymbol{\pi}, \mathbf{x}) \neq d$ by $e(\boldsymbol{\pi}, \mathbf{x}) \geq d + 1 \vee -e(\boldsymbol{\pi}, \mathbf{x}) \geq 1 - d$; then apply (69).

Example 19 Let φ be *algebraicity* sentence (41) for constant $\mathtt{a}$ in Example 12:

$$(\forall y : \nu_{\mathtt{S2}})\ \mathtt{a}^0 = y \Rightarrow \mathtt{S2}^1 y \geq \mathtt{S2}^0 \tag{70}$$

where $\mathtt{a}^0 \in \mathsf{LExp}_{\lambda,\nu_{\mathtt{S2}}}$, $\mathtt{S2}^1 \in \mathsf{LExp}_{\nu_{\mathtt{S2}},\nu_2}$, and $\mathtt{S2}^0 \in \mathsf{LExp}_{\lambda,\nu_2}$ for some $\nu_2 \in \mathsf{N}$. We use superindices instead of subscripts to use the matrix/vector notation below. Let $n_{\nu_{\mathtt{S2}}} = 1$ and $n_{\nu_{\nu_2}} = 2$ (see Remark 12). Then, $\varpi(\varphi)$ is

$$(\forall y \in \mathbb{Z})\ \mathtt{a}^0 = y \Rightarrow \mathtt{S2}^1_1 y \geq \mathtt{S2}^0_1 \wedge \mathtt{S2}^1_2 y \geq \mathtt{S2}^0_2 \tag{71}$$

Note that (71) is *not* in affine form. We apply the previous steps to transform it into the following set of affine forms (the quantification remain equal):

$$\{\ \ y \geq \mathtt{a}^0 \wedge -y \geq -\mathtt{a}^0 \Rightarrow \mathtt{S2}^1_1 y \geq \mathtt{S2}^0_1,\ \ \ y \geq \mathtt{a}^0 \wedge -y \geq -\mathtt{a}^0 \Rightarrow \mathtt{S2}^1_2 y \geq \mathtt{S2}^0_2\ \ \} \tag{72}$$

Although it is a very simple example, we use it to exemplify how Farkas' Lemma works. The application to a larger set of formulas is analogous. Each of the affine forms in (72) is treated separately. First, we write them in matrix form as follows:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} y \geq \begin{bmatrix} \mathtt{a}^0 \\ -\mathtt{a}^0 \end{bmatrix} \Rightarrow \mathtt{S2}^1_1 y \geq \mathtt{S2}^0_1 \tag{73}$$

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} y \geq \begin{bmatrix} \mathtt{a}^0 \\ -\mathtt{a}^0 \end{bmatrix} \Rightarrow \mathtt{S2}^1_2 y \geq \mathtt{S2}^0_2 \tag{74}$$

Now we apply Theorem 5 to *each* of them *simultaneously*, i.e., the constraint solving problem must be solved at once. The reason is that the external existential quantification on the parameters concerns both affine forms. Therefore, we need to find two vectors $\boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2 \in \mathbb{R}^2$ of non-negative numbers such that

$$\mathtt{S2}^1_1 = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} \lambda^1_1 \\ \lambda^1_2 \end{bmatrix} \text{ and } \begin{bmatrix} \lambda^1_1 & \lambda^1_2 \end{bmatrix} \begin{bmatrix} \mathtt{a}^0 \\ -\mathtt{a}^0 \end{bmatrix} \geq \mathtt{S2}^0_1, \qquad \text{or equivalently:}$$

$$\mathtt{S2}^1_1 = \lambda^1_1 - \lambda^1_2 \text{ and } \lambda^1_1 \mathtt{a}^0 - \lambda^1_2 \mathtt{a}^0 \geq \mathtt{S2}^0_1 \tag{75}$$

for the first affine form (73) and

$$\mathtt{S2}^1_2 = \lambda^2_1 - \lambda^2_2 \text{ and } \lambda^2_1 \mathtt{a}^0 - \lambda^2_2 \mathtt{a}^0 \geq \mathtt{S2}^0_2 \tag{76}$$

for the second one (74). Note that no universally quantified variables remain. Actually, these *constraint solving problems* can be formulated as a satisfiability problem for a single sentence as follows:

$$\begin{aligned}(\exists \mathtt{a}^0, \mathtt{S2}^1_1, \mathtt{S2}^1_2, \mathtt{S2}^0_1, \mathtt{S2}^0_2, \lambda^1_1, \lambda^1_2, \lambda^2_1, \lambda^2_2)\ & \lambda^1_1 \geq 0 \wedge \lambda^1_2 \geq 0 \wedge \lambda^2_1 \geq 0 \wedge \lambda^2_2 \geq 0 \\ \wedge\ \mathtt{S2}^1_1 = \lambda^1_1 - \lambda^1_2 \wedge \lambda^1_1 \mathtt{a}^0 - \lambda^1_2 \mathtt{a}^0 \geq \mathtt{S2}^0_1 \wedge\ & \mathtt{S2}^1_2 = \lambda^2_1 - \lambda^2_2 \wedge \lambda^2_1 \mathtt{a}^0 - \lambda^2_2 \mathtt{a}^0 \geq \mathtt{S2}^0_2\end{aligned} \tag{77}$$

A *solution* for these *non-linear* arithmetic constraints can be obtained by using standard methods, see [11,18].

11 Related work

11.1 Termination of declarative programs

The generation of (homogeneous)[15] *algebras* using parametric interpretations followed by a constraint solving process is standard in termination analysis of term rewriting, with a *built-in* requirement of *monotonicity* for (some of) the synthesized functions [25] (see also Remark 8). In this setting, starting from [34], matrix interpretations have been successfully used in the last decade to prove termination of term rewriting [2,28,35,99] and also in complexity analysis of rewrite systems, see [94] for a summary of research including relevant references. In a recent paper, Waldmann discusses the use of a subclass of convex polytopic domains to define *algebras* which can be used in proofs of termination of rewriting and for other purposes, like the analysis of derivational complexity [119]. He also provides an implementation of the automatic generation of such algebras as part of his tool matchbox [120]. However, his domains are by default *bounded from below* (subsets of vectors of non-negative rational numbers); also the matrices F_i which are used in the defininition of functions (see Section 8.2) are restricted to contain natural numbers only. In contrast to this situation, and after 25 years of research on termination of order-sorted and many-sorted rewrite systems [6,45,81,101,123], no systematic treatment of the generation of *heterogeneous* algebras [10], including the generation of different *domains for sorts* and interpretations of *ranked functions* in many-sorted or order-sorted algebras has been attempted to date.[16]

Our tool AGES (*Automatic GEneration of logical modelS*) implements the techniques described in this paper to generate a model $\mathcal{A}$ for an OS-FOL *theory*. We also have integrated the methods developed in this paper as part of the termination tool MU-TERM. Convex domains and interpretations (for function symbols) were successfully used to prove operational termination (i.e., the absence of infinite proof trees when a computation is attempted [79]) of *Conditional Term Rewriting Systems* (CTRSs, see [100, Chapter 7] for a survey) in the 2015 and 2016 editions of the International Termination Competition [42]. The 2017 version of MU-TERM incorporates piecewise functions and automatically generated relations. In this setting, a simple example illustrates the impact of the research in this paper.

Example 20 Consider the following CTRS $\mathcal{R}$ [100, Example 7.2.45]:

$$a \to a \Leftarrow b \to x, c \to x \qquad (78) \qquad\qquad c \to d \Leftarrow d \to x, e \to x \qquad (80)$$

$$b \to d \Leftarrow d \to x, e \to x \qquad (79)$$

By using the results in [83], and considering the *conditional dependency pair* $A \to A \Leftarrow b \to x, c \to x$ for $\mathcal{R}$ (where A is a new constant symbol), we can prove $\mathcal{R}$ operationally terminating if the following sentence (with $\sqsupset$ a new *predicate symbol*)

$$(\forall x)\ b \to^* x \wedge c \to^* x \Rightarrow A \sqsupset A \qquad (81)$$

holds in a model $\mathcal{A}$ of the theory associated to $\mathcal{R}$ where $\sqsupset^{\mathcal{A}}$ is well-founded.

[15] i.e., with a *single* domain [10, page 116], as a particular case of *heterogeneous* algebras, consisting of an indexed set of domains and functions over such domains.

[16] In [34,35], the automated generation of monotone many-sorted algebras is considered. However, only two-sorted algebras are considered for generating interpretations of function symbols, on a previously *fixed* interpretation of sorts, see [35, Section 5].

The only possibility for (81) to hold is that $b \to^* x \wedge c \to^* x$ is *unsatisfiable* in $\mathcal{A}$. Otherwise, $A^{\mathcal{A}} \sqsupset^{\mathcal{A}} A^{\mathcal{A}}$ should hold, which is *not* possible if $\sqsupset^{\mathcal{A}}$ is well-founded. Thus, (81) is *not* satisfied by a 'typical' interpretation with domain $\mathbb{N}$, with $\to^*$ interpreted as $\geq_{\mathbb{N}}$ and $\sqsupset$ as the usual well-founded ordering $>_{\mathbb{N}}$: for all $b^{\mathcal{A}}, c^{\mathcal{A}} \in \mathbb{N}$, the interpreted formula $b^{\mathcal{A}} \geq x \wedge c^{\mathcal{A}} \geq x$ is satisfied by $x = 0$! The problem stands if $\mathcal{A} = \mathbb{N}^n$ and the usual extensions of $\geq_{\mathbb{N}}$ and $>_{\mathbb{N}}$ to tuples are used.

In contrast, we can provide at least *two* quite different solutions to this problem. First, we encode the OS-FOL theory $\mathcal{S}$ for $\mathcal{R}$ as follows:

$$(\forall x : S) \;\; x \to^* x \tag{82}$$

$$(\forall x, y, z : S) \;\; x \to y \wedge y \to^* z \Rightarrow x \to^* z \tag{83}$$

$$(\forall x : S) \;\; b \to^* x \wedge c \to^* x \Rightarrow a \to a \tag{84}$$

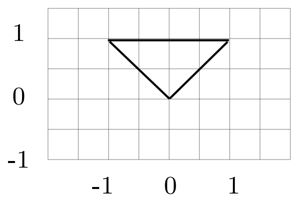
$$(\forall x : S) \;\; d \to^* x \wedge e \to^* x \Rightarrow b \to d \tag{85}$$

$$(\forall x : S) \;\; d \to^* x \wedge e \to^* x \Rightarrow c \to d \tag{86}$$

We add (81) to $\mathcal{S}$ and the requirement of $\sqsupset$ being *well-founded* so that operational termination of $\mathcal{R}$ can be concluded from the existence of a model for $\mathcal{S}$. Now,

1. We can use a domain $\mathcal{A}_S$ of *pairs* of numbers for sort S (i.e., $n_S = 2$) and, according to Remark 8.1, we let $m_S = n_S + 1 = 3$ so that a bounded domain is obtained if necessary. With $\mathcal{A}_S$ given by the following matrix and vector:

$$\mathsf{C}^S = \begin{bmatrix} -1 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix} \qquad \mathbf{b}^S = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$



   we have the values contained in the area of the displayed inverted triangle. The (constant) function symbols are interpreted as follows:

$$b^{\mathcal{A}} = d^{\mathcal{A}} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad A^{\mathcal{A}} = a^{\mathcal{A}} = c^{\mathcal{A}} = e^{\mathcal{A}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

   The interpretation of $\to^*$ is the pointwise extension $\geq$ of $\geq_{\mathbb{N}}$ (Example 14) and both $\to$ and $\sqsupset$ are interpreted as the (well-founded) relation $>$ on vectors (Example 15). Sentences (81) – (86) are all satisfied by $\mathcal{A}$. For instance, due to the use of a *bounded* domain $\mathcal{A}_S$ as above, no $x \in \mathcal{A}_S$ satisfies $b^{\mathcal{A}} \geq x$ and $c^{\mathcal{A}} \geq x$. Therefore, the antecedent of the implication in (81) does *not* hold for any $x \in \mathcal{A}_S$ and the sentence trivially holds.

2. We can choose $\mathcal{A}_S = \mathbb{N}$ provided that *alternative* interpretations of $\to$ and $\to^*$ are used. With $b^{\mathcal{A}} = e^{\mathcal{A}} = 2$, $A^{\mathcal{A}} = a^{\mathcal{A}} = c^{\mathcal{A}} = d^{\mathcal{A}} = 0$, and

$$x \to^{\mathcal{A}} y \Leftrightarrow x =_{\mathbb{N}} y \qquad x(\to^*)^{\mathcal{A}} y \Leftrightarrow y \leq_{\mathbb{N}} x \leq_{\mathbb{N}} y + 1 \qquad x \sqsupset^{\mathcal{A}} y \Leftrightarrow x >_{\mathbb{N}} y$$

   we also obtain a model of (81) – (86). The point now is that the antecedent of the implications is always false (for every natural number) due to the special interpretation of $\to^*$. For instance, (81) becomes

$$x \leq 2 \leq x + 1 \wedge x \leq 0 \leq x + 1 \Rightarrow 0 > 0$$

   which holds because there is no $x \in \mathbb{N}$ such that $[x, x + 1]$ includes 0 and 2.

The piecewise approach is flexible enough to represent a good number of functions (see Example 8) and predicates (Example 10). Indeed, these classes of functions (e.g., max/min) and predicates (lexicographic orderings) have already been used in the literature since long time ago, see [16,36,37,60,87,89,116], for instance. Our approach permits their use in a *common framework*, as particular cases of a *single format*. The good point is that all these interpretations can be obtained *automatically* using the methodology presented in the previous sections.

11.2 Program analysis and verification

In proofs of program correctness, inferring interpretations for 'unknown' predicate symbols introduced to formalize the *verification conditions* associated to the verification problem is also important [69,75]. Indeed, characterizations of program properties (correctness, partial correctness, equivalence, termination,...) as satisfiability problems in first-order logic can be found in early papers in program analysis by Zohar Manna and his collaborators [84–86] and also [72].

Example 21 The following *summation* program P [88, page 557]:

$$\mathsf{sum}(n) \Leftarrow \mathsf{if}\ n=0\ \mathsf{then}\ 0\ \mathsf{else}\ n+\mathsf{sum}(n-1) \tag{87}$$

is intended to compute the addition of the first n natural numbers. Manna and Pnueli describe the meaning of the program using the following formula $W_P(Q)$:

$$(\forall n)\,((n=0 \Rightarrow Q(n,0)) \wedge (n>0 \Rightarrow (\forall p)\ (Q(n-1,p) \Rightarrow Q(n,n+p)))) \tag{88}$$

where Q is intended to *simulate* the function sum computed by the program: if $f(n)$ returns a number s, then $Q(n,s)$ holds. Actually, Q is the only *uninterpreted* symbol; all other symbols in (88) receive the usual interpretation (integer constants, arithmetic operators, or arithmetic comparison predicates). According to [88, Theorem 1], the partial correctness of P is equivalent to the *satisfiability* of

$$W_P(Q) \wedge (\forall n,s)\ (\varphi(n) \wedge Q(n,s)) \Rightarrow \psi(n,s) \tag{89}$$

where $\varphi(n)$ is the *precondition* for the summation program (for instance, $n \geq 0$); and $\psi(n,s)$ is the postcondition (for instance, $s = n(n+1) \div 2$). However, other pre- and postconditions can be given to investigate other *program properties.* For instance, with $\varphi(n)$ being $n > 0$ and $\psi(s,n)$ being $s > 0$ we say that the outcome of the program is positive whenever the input is positive. We can use AGES to check that (89) holds when $\varphi(n)$ is $n > 0$ and $\psi(n,s)$ is $s > 0$, see [57, Example 3].

After the seminal contributions in the late sixties [36,62,98] and Naur's dramatic call to write programs on *more solid principles* ("We cannot indefinitely continue to build on sand" [98, page 310]), attempts to use theorem proving techniques in *automated* program analysis and verification date back to the early seventies, as reported in [32,58,69,72,75]. And the application of linear arithmetic and linear algebra techniques started almost immediately [21,26,27,30,68,112].

Indeed, this logic-based approach is alive and healthy, see [5,14,29,54–56] and the references therein, thanks to the generalized use of SMT techniques as a *backend* where different kinds of program analysis and verification problems and approaches can be mapped to [97]. In particular, the use of *Horn clauses* or *constrained Horn Clauses*[17] as a basis for program verification has recently deserved a lot of attention and a number of associated tools have been developed so far [4,12,13,54,56,107].

For instance, in [13] a generic Horn solver is developed and used, in particular, to prove termination of an imperative program dealing with arrays [13, Section 5.3]. The main approach is similar to ours (defining a generic approach to 'solve' unknown relations occurring in a logic description of a given problem, thus applying it to a variety of analysis and verification problems), but the proof of termination finally requires a *postprocessing* where disjunctive well-foundedness of a number of components is proved by a different tool. In our approach we could specify a disjunction of atoms using OS-FOL sentences as part of the original theory with a requirement of well-foundedness for the corresponding predicates (as sketched in Section 5) and then reinforce well-foundedness of the associated relations as explained in Section 10.2.

12 Conclusions and future work

The main contribution of this paper to the effort of applying logic-based techniques in program analysis and verification is the development of a generic *frontend* to map purely symbolic components (functions and predicates) of a first-order logic with sorts into arithmetic constraints. Our starting point is the piecewise description of functions and predicates. The advantage of this approach is its flexibility to represent different, well-known, and widely used abstractions like linear functions, max/min functions, lexicographic orderings, etc. We obtain them all as particular cases of a single framework. We have extended the notion of *derivor* (Sections 3 and 4) to map a theory in the source language to another theory in a language of linear expressions (Section 7). The class of OS-FOL structures based on the *convex polytopic domains* described in Section 8 fits this logic very well (Section 9) and permits a flexible translation into *arithmetic constraints* by using parametric interpretations as explained in Section 10. A summary of our contributions follows:

1. The systematic generation of *many-sorted structures* with function and predicate symbols interpreted as *relations*. This yields a powerful framework to define a variety of functions and relations based on polynomial constraints which are still amenable to automation as they rely on decidable theories (Example 5).
2. The notion of derivor and derived structure (and model) generalizes [50] to order-sorted signatures with predicates. Our generalization is twofold: we handle functions as (special) relations and also apply the transformation to formulas instead of just terms (much in the style of [8,90,93]).
3. The *piecewise definition* of predicates and the sufficient condition for well-foundedness based on a combination of disjunctive well-foundedness and abstraction is, as far as we know, new in the literature. The systematic treatment of piecewise function definitions in a relational style is also new.
4. We provide a systematic scheme to *derive* models for a source OS-FOL theory $\mathcal{S}$ enriched with specific *requirements* that cannot be expressed using OS-FOL sentences (e.g., well-foundedness).

[17] Constrained Horn clauses are essentially Horn clauses where some atoms have a predefined structure and interpretation established by a well-known theory (e.g., linear arithmetic).

```
fmod PATH is
 sorts Node Edge Path .
 subsorts Edge < Path .
 ops source target : Edge -> Node .
 ops source target : Path -> Node .
 op _;_ : [Path] [Path] -> [Path] .
 var E : Edge .
 vars P Q R S : Path .
 cmb E ; P : Path if target(E) = source(P) .
 ceq (P ; Q) ; R = P ; (Q ; R) if target(P) = source(Q) /\ target(Q) = source(R) .
 ceq source(P) = source(E) if E ; S := P .
 ceq target(P) = target(S) if E ; S := P .
endfm
```

Fig. 5 PATH program for graph specification

5. We explain the definition of OS-FOL structures based on convex domains and also provide sufficient conditions guaranteeing the well-foundedness of relations defined on convex domains by piecewise definitions.
6. We explain the synthesis of models based on such structures by using well-known techniques from linear algebra and constraint solving (SMT).
7. A system implementing the techniques described in this paper to automatically generate models is available (AGES). The techniques described in this paper have also been used in our termination tool MU-TERM.

An important motivation to develop this paper was to provide a flexible and mechanizable framework for the definition of appropriate abstractions to be used in automated proofs of *operational termination* of declarative programs [82], where *structures* rather than just algebras are required due to the logic-based definition of operational termination [78,80,82]. For instance, in [78], the operational termination of the Maude program PATH in Figure 5 has been *semi-automatically* proved by using the tool AGES. In PATH, sort Node represents the nodes in a graph and sorts Edge and Path are intended to classify paths consisting of a single edge or many of them, respectively [23, pages 561–562]. Note the *overloaded* syntax for operators source and target. The essential aspect is that the computational description of PATH cannot be given in terms of reduction relations only. There are also *memberships*, *pattern matching* operations, *conditions* in rules, and everything is combined in an inference system which describes the computations (see [78, Figure 1]). As shown in [78], the role of logical models defining well-founded relations in proofs of operational termination of a program like PATH is analogous to the role of *well-founded algebras* in proofs of termination of rewriting (see Remark 8). For this reason, the research in this paper is an essential step towards the implementation of a tool for automatically proving operational termination of declarative programs based on the OT Framework [82].

12.1 Future work

As suggested in Example 5, our approach can also adapted to define other kind of structures based on domains and function and predicate interpretations which benefit from existing algorithms and techniques from *Real Algebraic Geometry* [9, 105] or matrix polynomials [28]. This is a subject for future work.

The ability to generate (well-founded) relations interpreting binary predicates with rank is also important in termination analysis of OS-TRSs using the dependency pair framework [81]. In this way, better techniques to prove termination of OS-TRSs become available for proving other termination properties which are *persistent* and remain unchanged after *sort introduction* [123]. This is the case of termination of TRSs when some syntactical restrictions are required on their rules [6, 70] and of innermost termination of TRS [38, 71], which has been recently proved useful to prove termination of programs with pre-defined data structures and operations like integer arithmetic [43, 44, 102]. In other settings, like higher-order rewriting, type information has also been proved important to prove termination [39] and the techniques developed in this paper could be useful as well. Thus, this is also an important subject of future work. Also, we plan to investigate the practical use and impact of the techniques developed in this paper in the more general field of program analysis and verification.

Acknowledgments. We thank José Meseguer for his comments and suggestions after carefully reading a preliminary version of this paper. We also thank him for drawing our attention to several relevant references. Salvador Lucas is most grateful to Prof. Manfred Schmidt-Schauss for sending him a hardcopy of his PhD Thesis [110]. We also thank the anonymous referees for many remarks and suggestions that led to improve the paper.

References

1. B. Alarcón, R. Gutiérrez, S. Lucas, and R. Navarro-Marset. Proving Termination Properties with MU-TERM. In *Proc. of AMAST'10*, LNCS 6486:201-208, 2011.
2. B. Alarcón, S. Lucas, and R. Navarro-Marset. Using Matrix Interpretations over the Reals in Proofs of Termination. In *Proc. of PROLE'09*. pages 255-264, September 2009.
3. E. Albert, S. Genaim, and R. Gutiérrez. A Transformational Approach to Resource Analysis with Typed-Norms. In *Revised Selected Papers from LOPSTR'13*, LNCS 8901:38-53, 2013.
4. E. de Angelis, F. Fioravante, A. Pettorossi, and M. Proietti. Proving correctness of imperative programs by linearizing constrained Horn clauses. *Theory and Practice of Logic Programming* 15(4-5):635-650, 2015.
5. E. de Angelis, F. Fioravante, A. Pettorossi, and M. Proietti. Semantics-based generation of verification conditions by program specialization. In *Proc. of PPDP'15*, pages 91-102, ACM Press, 2015.
6. T. Aoto. Solution to the Problem of Zantema on a Persistent Property of Term Rewriting Systems. *Journal of Functional and Logic Programming*, 2001(11):1-20, 2001.
7. J. Barwise. An Introduction to First-Order Logic. In J. Barwise, editor, Handbook of Mathematical Logic, North-Holland, 1977.
8. J. Barwise. Axioms for Abstract Model Theory. *Annals of Mathematical Logic* 7:221–265, 1974.
9. J. Bochnak, M. Coste, and M.-F. Roy. Real Algebraic Geometry. Springer-Verlag, Berlin, 1998.
10. G. Birkhoff and J.D. Lipson. Heterogeneous Algebras. *Journal of Combinatorial Theory* 8:115-133, 1970.
11. M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. The Barcelogic SMT Solver. In *Proc. of CAV'08*, LNCS 5123:294-298, 2008.
12. N. Bjørner, A. Gurfinkel, K. McMillan, and A. Rybalchenko. Horn-Clause Solvers for Program Verification. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, LNCS 9300:24-51, 2015
13. N. Bjørner, K. McMillan, and A. Rybalchenko. On Solving Universally Quantified Horn-Clauses. In *Proc. of SAS'13*, LNCS 7935:105-125, 2013.

14. N. Bjørner, K. McMillan, and A. Rybalchenko. Program Verification as Satisfiability Modulo Theories. In *Proc. of SMT'12*, EPiC Series in Computing 20:3-11, 2013.
15. G.A. Bliss. Algebraic Functions. Dover, 2004.
16. G. Bonfante, J-Y. Marion, and J.-Y. Moyen. On Lexicographic Termination Ordering With Space Bound Certifications. In *Revised Papers from PSI 2001*, LNCS 2244:482-393, 2001.
17. G.S. Boolos, J.P. Burgess, and R.C. Jeffrey. Computability and Logic, fourth edition. Cambridge University Press, 2002.
18. C. Borralleras, S. Lucas, A. Oliveras, E. Rodríguez, and A. Rubio. SAT Modulo Linear Arithmetic for Solving Polynomial Constraints. *Journal of Automated Reasoning* 48:107-131, 2012.
19. H.-J. Bürckert, B. Hollunder, and A. Laux. On Skolemization in constrained logics. *Annals of Mathematics and Artificial Intelligence* 18:95-131, 1996.
20. R.M. Burstall and J.A. Goguen. Putting Theories together to make specifications. In *Proc. of IJCAI'77*, pages 1045-1058, William Kaufmann, 1977.
21. M. Caplain. Finding invariant assertions for proving programs. In *Proc. of the International Conference on Reliable Software*, pages 165-171, ACM Press, 1975.
22. C.L. Chang and R.C. Lee. Symbolic Logic and Mechanical Theorem Proving. Academic Press, 1973.
23. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. All About Maude – A High-Performance Logical Framework. LNCS 4350, 2007.
24. A.G. Cohn. Improving the expressiveness of many sorted logic. In *Proc. of the National Conference on Artificial Intelligence*, pp. 84-87, AAAI Press, 1983.
25. E. Contejean, C. Marché, A.-P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325-363, 2006.
26. D.C. Cooper. Programs for Mechanical Program Verification. *Machine Intelligence* 6:43-59, Edinburgh University Press, 1971.
27. D.C. Cooper. Theorem Proving in Arithmetic without Multiplication. *Machine Intelligence* 7:91-99, Edinburgh University Press, 1972.
28. P. Courtieu, G. Gbedo, and O. Pons. Improved Matrix Interpretations. In *Proc. of SOFSEM'10*, LNCS 5901:283-295, 2010.
29. P. Cousot, R. Cousot, and L. Mauborgne. Logical Abstract Domains and Interpretations. In *The Future of Sofware Engineering*, pages 48-71, Springer-Verlag, 2011.
30. P. Cousot and N. Halbwachs. Automatic Discovery of linear restraints among variables of a program. In *Conference Record of POPL'78*, pages 84-96, ACM Press, 1978.
31. B.A. Davey and H.A. Priestley. Introduction to Lattices and Order. Cambridge University Press, 1990.
32. B. Elspas, K.N. Levitt, R.J. Waldinger, and A. Waksman. An Assessment of Techniques for Proving Program Correctness. *Computing Surveys* 4(2):97-147, 1972.
33. M.H. van Emdem and R.A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM* 23(4):733-742, 1976.
34. J. Endrullis, J. Waldmann, and H. Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. In *Proc. of IJCAR'06*, LNCS 4130:574-588, 2006.
35. J. Endrullis, J. Waldmann, and H. Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. *Journal of Automated Reasoning* 40(2-3):195-220, 2008.
36. R.W. Floyd. Assigning meanings to programs. *Mathematical aspects of computer science* 19:19-32, 1967.
37. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Maximal Termination. In *Proc. of RTA'08*, LNCS 5117:110-125, 2008.
38. C. Fuhs, J. Giesl, M. Parting, P. Schneider-Kamp and S. Swiderski. Proving Termination by Dependency Pairs and Inductive Theorem Proving. *Journal of Automatic Reasoning*, 47:133–160, 2011.
39. C. Fuhs and C. Kop. Polynomial Interpretations for Higher-Order Rewriting. In *Proc. of RTA'12*, LIPIcs 15:176-192, 2012.
40. K. Futatsugi and R. Diaconescu. *CafeOBJ Report.* World Scientific, AMAST Series, 1998.
41. M. Gaboardi and R. Péchoux. On bounding space usage of streams using interpretation analysis. *Science of Computer Programming* 111:395-425, 2015.
42. J. Giesl, F. Mesnard, A. Rubio, R. Thiemann, and J. Waldmann. Termination Competition (termCOMP 2015). In *Proc. of CADE'15*, LNCS 9195:105-108, 2015.

43. J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, and C. Fuhs. Symbolic Evaluation Graphs and Term Rewriting – A General Methodology for Analyzing Logic Programs. In *Proc. of the PPDP'12*, pages 1-12, ACM Press, 2012.
44. J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated Termination Proofs for Haskell by Term Rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2), Article 7, January 2011.
45. I. Gnaedig. Termination of Order-sorted Rewriting. In *Proc. of ALP'92*, LNCS 632:37-52, 1992.
46. J.A. Goguen. Order-Sorted Algebra. Semantics and Theory of Computation Report 14, UCLA, 1978.
47. J.A. Goguen and R.M. Burstall. Some Fundamental Algebraic Tools for the Semantics of Computation. Part 1: Comma Categories, Colimits, Signatures and Theories. *Theoretical Computer Science* 31:175-209, 1984.
48. J.A. Goguen and R.M. Burstall. Some fundamental algebraic tools for the semantics of computation. Part 2: Signed and Abstract Theories. *Theoretical Computer Science* 31:263-295, 1984.
49. J. Goguen and J. Meseguer. Models and Equality for Logical Programming. In *Proc. of TAPSOFT'87*, LNCS 250:1-22, 1987.
50. J.A. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In *Current trends in Programming Methodology*, pages 80-149, Prentice Hall, 1978.
51. J.A. Goguen and J. Meseguer. Remarks on Remarks on Many-Sorted Equational Logic. *Sigplan Notices* 22(4):41-48, 1987.
52. J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.
53. J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In J. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: algebraic specification in action*, Kluwer, 2000.
54. S. Grebenshikov, N.P. Lopes, C. Popeea, and A. Rybalchenko. Synthesizing Software Verifiers from Proof Rules. In *Proc. of PLDI'12*, pages 405-416, ACM Press, 2012.
55. S. Gulwani and A. Tiwari. Combining Abstract Interpreters. In *Proc. of PLDI'06*, pages 376-386, ACM Press, 2006.
56. A. Gurfinkel, T. Kahsai, A. Komuravelli, and J.A. Navas. The SeaHorn Verification Framework. In *Proc. of CAV'15, Part I*, LNCS 9206:343-361, 2015.
57. R. Gutiérrez, S. Lucas, and P. Reinoso. A tool for the automatic generation of logical models of order-sorted first-order theories. In *Proc. of PROLE'16*, pages 215-230, 2016. Tool available at `http://zenon.dsic.upv.es/ages/`.
58. S.L. Hantler and J.C. King. An Introduction to Proving the Correctness of Programs. *ACM Computing Surveys* 8(3):331-353, 1976.
59. P. Hayes. A Logic of Actions. *Machine Intelligence* 6:495-520, Edinburgh University Press, Edinburgh, 1971.
60. B. Heidergott, G.J. Olsder, and J. van der Woude. Max Plus at Work. Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications. Princeton University Press, 2006.
61. N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. In *Proc. of IJCAR 2008*, LNCS 5195:364?379, 2008.
62. C.A.R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM* 12(10):576-583, 1969.
63. W. Hodges. Elementary Predicate Logic. *Handbook of Philosophical Logic* Volume 1, pages 1-131. Reidel Publishing Company, 1983.
64. W. Hodges. A shorter model theory. Cambridge University Press, 1997.
65. D. Hofbauer. Termination Proofs by Context-Dependent Interpretation. In *Proc. of RTA'01*, LNCS 2051:108-121, 2001.
66. D. Hofbauer. Termination Proofs for Ground Rewrite Systems. Interpretations and Derivational Complexity. *Applicable Algebra in Engineering, Communication and Computing*, 12:21-38, 2001.
67. D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. of RTA'89*, LNCS 355:167-177, 1989.
68. T.E. Hull, W.H. Enright, and A.E. Sedgwick. The correctness of numerical algorithms. In *Proc. of PAAP'72*, pages 66-73, 1972.

69. S. Igarashi, R.L. London, and D. Luckham. Automatic Program Verification I: A Logical Basis and its Implementation. *Acta Informatica* 4:145-182, 1975.
70. M. Iwami. Persistence of Termination of Term Rewriting Systems with Ordered Sorts. In *In Proc. of 5th JSSST Workshop on Programming and Programming Languages*, Shizuoka, Japan, pp.47-56, 2003.
71. M. Iwami. Persistence of Termination for Non-Overlapping Term Rewriting Systems. In *Proc of Algebraic Systems, Formal Languages and Conventional and Unconventional Computation Theory,* Kokyuroku RIMS, University of Kyoto, 1366:91-99, 2004
72. S. Katz and Z. Manna. Logical Analysis of Programs. *Communications of the ACM* 19(4):188-206, 1976.
73. C.H. Langford. Review: *Über deduktive Theorien mit mehreren Sorten von Grunddingen.* Journal of Symbolic Logic 4(2):98, June 1939.
74. D.S. Lankford. Some approaches to equality for computational logic: A survey and assessment. Memo ATP-36, Automatic Theorem Proving Project, University of Texas, Austin, TX.
75. R.L. London. The Current State of Proving Programs Correct. In *Proc. of ACM'72*, volume 1, pages 39-46, ACM 1972.
76. S. Lucas. Polynomials over the Reals in Proofs of Termination: from Theory to Practice. *RAIRO Theoretical Informatics and Applications*, 39(3):547–586, 2005.
77. S. Lucas. Synthesis of models for order-sorted first-order theories using linear algebra and constraint solving. *Electronic Proceedings in Theoretical Computer Science* 200:32-47, 2015.
78. S. Lucas. Use Of Logical Models For Proving Operational Termination In General Logics. In *Selected papers from WRLA'16*, LNCS 9942:1-21, 2016.
79. S. Lucas, C. Marché, and J. Meseguer. Operational termination of conditional term rewriting systems. *Information Processing Letters*, 95:446–453, 2005.
80. S. Lucas and J. Meseguer. Models for Logics and Conditional Constraints in Automated Proofs of Termination. In *Proc. of AISC'14*, LNAI 8884:7-18, 2014.
81. S. Lucas and J. Meseguer. Order-Sorted Dependency Pairs. In *Proc. of PPDP'08* , pages 108-119, ACM Press, 2008.
82. S. Lucas and J. Meseguer. Proving Operational Termination Of Declarative Programs In General Logics. In *Proc. of PPDP'14*, pages 111-122, ACM Digital Library, 2014.
83. S. Lucas and J. Meseguer. Dependency pairs for proving termination properties of conditional term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 86:236-268, 2017.
84. Z. Manna. The Correctness of Programs. *Journal of Computer and System Sciences* 3:119-127, 1969.
85. Z. Manna. Properties of programs and the First-Order Predicate Calculus. *Journal of the ACM* 16(2):244-255, 1969.
86. Z. Manna. Termination of programs represented as interpreted graphs. In *Proc. of AFIPS'70*, pages 83-89, 1970.
87. Z. Manna and S. Ness. On the termination of Markov algorithms. In *Proc. of the Third Hawaii International Conference on System Science*, pages 789-792, 1970.
88. Z. Manna and A. Pnueli. Formalization of Properties of Functional Programs. *Journal of the ACM* 17(3):555-569, 1970.
89. Y.-I. Marion and R. Péchoux. Sup-Interpretations, a Semantic Method for Static Analysis of Program Resources. *ACM Transactions on Computational Logic* 10(4), Article 27 (31 pages), 2009.
90. N. Martí-Oliet, J. Meseguer, and M. Palomino. Theoroidal Maps as Algebraic Simulations. In *Revised Selected Papers from WADT'04*, LNCS 3423:126-143, 2005.
91. J. McCarthy. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. *Communications of the ACM* 3(4):184-195, 1960.
92. J. Meseguer. General Logics. In H.-D. Ebbinghaus et al., editors, *Logic Colloquium'87*, pages 275-329, North-Holland, 1989.
93. J. Meseguer and S. Skeirik. Equational Formulas and Pattern Operations in Initial Order-Sorted Algebras. In *Revised Selected Papers from LOPSTR'15*, LNCS 9527:36-53, 2015.
94. A. Middeldorp. Matrix Interpretations for Polynomial Derivational Complexity of Rewrite Systems. In *Proc. of LPAR'12*, LNCS 7180:12, 2012.
95. J.-F. Monin. Understanding Formal Methods. Springer-Verlag, London, 2003.
96. M. Montenegro, R. Peña, and C. Segura. Space consumption analysis by abstract interpretation: Inference of recursive functions. *Science of Computer Programming* 111:426-457, 2015.

97. L. de Moura and N. Bjørner. Satisfiability Modulo Theories: Introduction and Applications. *Communications of the ACM* 54(9):69-77, 2011.
98. P. Naur. Proof of algorithms by general snapshots. Bit 6:310-316, 1966.
99. F. Neurauter and A. Middeldorp. Revisiting Matrix Interpretations for Proving Termination of Term Rewriting. In *Proc. of RTA'11*, LIPICS 10:251-266, 2011.
100. E. Ohlebusch. *Advanced Topics in Term Rewriting.* Springer-Verlag, Apr. 2002.
101. P.C. Ölveczky and O. Lysne. Order-Sorted Termination: The Unsorted Way. In *Proc. of ALP'96*, LNCS 1139:92-106, 1996.
102. C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. Automated Termination Analysis of Java Bytecode by Term Rewriting. In *Proc. of RTA'10*, LIPICS 6:259–276, 2010.
103. R. Péchoux. Synthesis of sup-interpretations: A survey. *Theoretical Computer Science* 467:30-52, 2013.
104. A. Podelski and A. Rybalchenko. Transition Invariants. In *Proc. of LICS'04*, pages 32-41, IEEE Computer Society 2004.
105. A. Prestel and C.N. Delzell. Positive Polynomials. From Hilbert's 17th Problem to Real Algebra. Springer-Verlag, Berlin, 2001.
106. D.J.S. Robinson. A Course in Linear Algebra with Applications (2^{nd} edition). World Scientific Publishing, Co., 2006.
107. P. Rümmer, H. Hojjat, and V. Kuncak. Disjunctive Interpolants for Horn-Clause Verification. In *Proc. of CAV'13*, LNCS 8044:347-363, 2013.
108. A. Schrijver. Theory of linear and integer programming. John Wiley & sons, 1986.
109. A. Schmidt. Über deduktive Theorien mit mehreren Sorten von Grunddingen. *Mathematische Annalen* 115(4):485-506, 1938.
110. M. Schmidt-Schauss. Computational Aspects Of An Order-Sorted Logic With Term Declarations. PhD Thesis, Fachbereich Informatik der Universität Kaiserslautern, April 1988.
111. S. Shapiro. Foundations without Foundationalism: A Case for Second-Order Logic. Clarendon Press, 1991.
112. R.E. Shostak. A Practical Decision Procedure for Arithmetic with Function Symbols. *Journal of the ACM* 26(2):351-360, 1979.
113. R.M. Smullyan. Theory of Formal Systems. Princeton University Press, 1961.
114. A. Tarski. A Decision Method for Elementary Algebra and Geometry. Second Edition. University of California Press, Berkeley, 1951.
115. Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters* 25:141-143, 1987.
116. A.M. Turing, Checking a Large Routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, Univ. Math. Lab., Cambridge, pp. 67-69, 1949.
117. C. Urban. The Abstract Domain of Segmented Ranking Functions. In *Proc. of SAS'13*, LNCS 7935:43-62, 2013.
118. C. Urban, A. Gurfinkel, and T. Kahsai. Synthesizing Ranking Functions From Bits and Pieces. In *Proc. of TACAS'16*, LNCS 9636:54-70, 2016.
119. J. Waldmann. Matrix Interpretations on Polyhedral Domains. In *Proc. of RTA'15* LIPICS 26:318-333, 2015.
120. J. Waldmann, A. Bau, and E. Noeth. Matchbox termination prover. `http://github.com/jwaldmann/matchbox/`, 2014.
121. C. Walther. A Mechanical Solution of Schubert's Steamroller by Many-Sorted Resolution. Aritificial Intelligence 26:217-224, 1985.
122. H. Wang. Logic of many-sorted theories. *Journal of Symbolic Logic* 17(2):105-116, 1952.
123. H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17:23-50, 1994.