Decidable Fragments of Many-Sorted Logic

Aharon Abadi

IBM Haifa Research Lab
Haifa University Campus, Haifa, 31905 Israel

Alexander Rabinovich

The Blavatnik School of Computer Science,
Tel-Aviv University, Israel

Mooly Sagiv

The Blavatnik School of Computer Science,
Tel-Aviv University, Israel

Abstract

Many natural specifications use types. We investigate the decidability of fragments of many-sorted first-order logic. We identified some decidable fragments and illustrated their usefulness by formalizing specifications considered in the literature. Often the intended interpretations of specifications are finite. We prove that the formulas in these fragments are valid iff they are valid over the finite structures. We extend these results to logics that allow a restricted form of transitive closure.

We tried to extend the classical classification of the quantifier prefixes into decidable/undecidable classes to the many sorted logic. However, our results indicate that a naive extension fails and more subtle classification is needed.

Key words: decidable logic, many-sorted logic, the classical decision problem, verification, transitive closure.

⋆ An extended abstract of this paper was published in [1].

Email addresses: aharona@il.ibm.com (Aharon Abadi), rabinoa@post.tau.ac.il (Alexander Rabinovich), msagiv@post.tau.ac.il (Mooly Sagiv).

1. Introduction

Systems with unbounded resources such as dynamically allocated objects and threads are heavily used in data structure implementations, web servers, and other areas. This paper develops new methods for proving properties of such systems. Our method is based on two principles: (i) formalizing the system and the required properties in fragments of many-sorted first-order logic and (ii) developing algorithms that decide whether a formula in these fragments holds over the finite models. Deciding whether a formula holds over the finite models is actually harder than deciding whether a formula is valid (holds over all models).

This paper was inspired by the Alloy Analyzer - a tool for analyzing models written in Alloy, a simple structural modeling language based on first-order logic [11, 12]. We illustrate the usefulness of our method by formalizing examples from Alloy. However, our results are more generally applicable and can be used to prove obligations obtained by translation into many-sorted logic from other specification and modeling languages.

1.1. Main Results

The main results in this paper are decidable fragments of many-sorted first-order logic. Our methods can generate finite counter-examples and finite models satisfying a given specification, which is hard for a resolution-based theorem prover. The rest of this subsection elaborates on these results.

1.2. Adding Types

We are looking for decidable and expressive fragments of first-order-logic. Many natural specifications use types. Hence, we consider fragments of many-sorted first-order-logic. In this paper we consider only cases where the interpretation of sorts (types) are disjoint.

The problem of classifying fragments of first-order logic with respect to the decidability and complexity of the satisfiability problem has long been a major topic in the study of classical logic. E. Borger et al. [6] provide the complete classification of fragments with a decidable validity problem and fragments with the finite model property. This classification is based on the quantifier prefixes and vocabulary of the formulas. However, this classification deals only with one-sorted logics and usually does not apply to specifications of practical problems, many of which are many-sorted.

For example, the finite model property fails for formulas with the quantifier prefix $\forall\forall\exists$ and equality [9]. Information about sorts (types) can reduce the complexity of this prefix class. For example, consider the formula $\forall x, y : A\ \exists z : B\ \psi(x, y, z)$, where ψ is a quantifier-free formula with equality and without function symbols. Each model M of the formula contains a sub-model M' that satisfies the formula and has only two elements. Indeed, let M be a model of the formula; we can pick two arbitrary elements $a_1 \epsilon A^M, b_1 \epsilon B^M$ such that $M \models \psi(a_1, a_1, b_1)$, and define M' to be M restricted to the universe $\{a_1, b_1\}$. Hence, many-sorted sentences with the quantifier prefix $\forall x : A \forall y : A \exists z : B$ have the finite-model property. Usually, as in the above example, the inclusion of sorts simplifies the verification task.

1.3. Our Contribution

The main technical contribution of this paper is the identification of a fragment of many-sorted logic that is (1) decidable; (2) useful - can formalize many examples; and (3) has the finite counter-model property. The finite counter-model property guarantees that a formula has a counter-model iff it has a finite counter-model, or equivalently, a formula is valid iff it is valid over the finite models.

Our second contribution is an attempt to classify decidable prefix classes of many-sorted logic. We show that a naive extension of one-sorted prefix classes to a many-sorted case inherits neither decidability nor the finite model property.

We extended our results to a logic that allows restricted use of the transitive closure.

The rest of this paper is organized as follows. In Section 2, we describe three fragments of many-sorted logic and formalize some Alloy examples using formulas in these fragments. In Section 3, we prove that the validity over the finite models problems are decidable for our fragments. In Section 4, we investigate ways of generalizing classification of decidable fragments from first-order logic to many-sorted logic. In Section 5, we discuss related works. Section 6 contains our conclusions.

2. Three Fragments of Many-Sorted First-Order Logic

The task of verifying that a program P satisfies a property θ can be reduced to the validity problem for sentences of the form $\psi \Rightarrow \theta$, where the sentence ψ formalizes the behavior of P.

We introduce three fragments St_0, St_1 and St_2 of many-sorted logic to describe the behavior of programs and systems. Safety properties of programs/systems can usually be formalized by universal sentences. We show that the validity and validity over the finite models problems are decidable for formulas of the form $\psi \Rightarrow \theta$, where $\psi \in St_i$ $(i = 0, 1, 2)$ and θ is universal. This allows automatic verification that a given program/system satisfies a property expressed as a universal formula.

St_0 is a natural fragment of the universal formulas that have the following *finite model property*: if $\psi \in St_0$, then it has a model iff it has a finite model.

St_0 has an even stronger *satisfiability with a finite extension* property that we introduce in Section 3. This property implies that the validity problem over the finite models is decidable for the sentences of the form $\psi \Rightarrow \theta$, where $\psi \in St_0$ and θ is universal. In Section 2.3, we formalize a birthday book example [16] in St_0.

Motivated by examples from Alloy, in Section 2.1 we introduce a more expressive (though less natural) set of formulas St_1. The St_1 formulas also have the satisfiability with a finite extension property, and might therefore be suitable for automatic verification of safety properties. The behavior of many specifications from [11, 12] can be formalized in St_1. We also describe the Railway safety example, which cannot be formalized in St_1. Our attempts to formalize the Railway safety example led us to a fragment St_2, which is defined in Section 2.4. This fragment has the satisfiability with a finite extension property. Almost all the specifications from Alloy that we examined and that do not use the transitive closure can be formalized by formulas of the form $\psi \Rightarrow \theta$, where $\psi \in St_2$ and θ is universal.

2.1. St_0 Class

In this subsection we describe a simple class of formulas denoted as St_0.

Definition 1 (Stratified Vocabulary)**.** A vocabulary Σ for many-sorted logic is *stratified* if there is a function $level$ from sorts (types) into Nat such that for every function symbol $f : A_1 \times \ldots \times A_m \rightarrow B$, $level(B) < level(A_i)$ for all $i = 1, \ldots, m$.

It is clear that for a finite stratified vocabulary Σ and a finite set V of variables, there are only a finite number of terms over Σ with the variables in V.

2.1.1. St_0 Syntax

The formulas in St_0 are universal formulas, over a stratified vocabulary.

It is easy to show that St_0 has the finite model property, due to the finiteness of the Herbrand model over St_0 vocabulary. We extend this class to the class St_1.

2.2. St_1 Class

St_1 is an extension of St_0 with a restricted use of the new atomic formula $x \in Im[f]$, where f is a function symbol. The formula $x \in Im[f]$ is shorthand for $\exists y_1 : A_1 \ldots \exists y_n : A_n \ (x = f(y_1, \ldots, y_n))$.

This is formalized below.

2.2.1. St_1 Vocabulary

St_1 vocabulary contains predicates, function symbols, the equality symbol, and the atomic formula $x \in Im[f]$, where f is a function symbol.

2.2.2. St_1 Syntax

The formulas in St_1 are universal formulas, over a stratified vocabulary, and for every function symbol $f : A_1 \times \ldots \times A_n \rightarrow B$ that participates in a subformula $x \epsilon Im[f]$, f is the only function symbol in the vocabulary with the range B.

The semantics is the same as for many-sorted logic. For the new atomic formula, the semantics is as for the formula $\exists y_1 : A_1 \ldots \exists y_n : A_n \ (x = f(y_1, \ldots, y_n))$.

In Section 3, we prove that St_1 has satisfiability with a finite extension property that generalizes the finite model property.

The requirement that f is the only function with range B is essential. For example, consider the conjunction of the following formulas containing two functions:
g **is onto:** $\forall y : T' \ y \in Im[g]$
f **is not onto:** $\forall x : T \ g(a) \neq f(x)$
f **is one-to-one:** $\forall x_1, x_2 : T \ f(x_1) = f(x_2) \Rightarrow x_1 = x_2$
It is clear that this formula has only infinite models.

2.3. Examples

Most of our examples come from Alloy [11, 12]. The vast majority of Alloy examples include transitive closure and thus cannot be formalized in our logic [1]. We examined eight Alloy specifications without transitive closure and seven of them can be formalized in our logics. Our first example is the birthday book [16], which can be formalized in St_0. The second example is a Railway Safety specification, which cannot be formalized by formulas in St_1. However, it can be formalized in St_2, which is an extension of St_1; the fragment St_2 is described in Section 2.4.

The Alloy specifications are composed of three parts: (1) *Facts*, (2) *Formulas* and (3) *Assert*. Facts is a set of many-sorted formulas that describe intended models and constrain the values of the functions and relations. *Formulas* is a set of parameterizable formulas intended to be used as abbreviations (macros) in other parts. *Assert* is a set of many-sorted formulas that formalize properties. The verification task is to check whether the formulas in Assert are logical consequences of the formulas from Facts.

Some specifications do not contain Assert part. For such a specification we want to check whether the specification is consistent, i.e., whether the set of formulas in Facts is satisfiable.

2.3.1. Birthday Book

Table 1 is used to model a simple Birthday book program [2] [16]. A birthday book has two relations: $known$, is a mapping between birthday book to people who are known by this birthday book, and $date$, set of triples (birthday book, the person, and the birth date of this person). The operation $getDate$ gets the birth date for a given birthday book and person. The operation $AddBirthday$ adds an association between a name and a date. The constant b_1 represents the current book, b_2 represents a new book obtained from b_1 by adding to it the new elements p_1 and d_1. The assertion $Assert$ states that if you add an entry (p_1, d_1) to book b_1 and then look it up (at the new book b_2), you get back what you just entered.

The specification assertion has the form $\psi \Rightarrow \theta$, where ψ is in St_0 and θ is universal.

The specification contains only one function $getDate : \ BirthdayBook \times Person \rightarrow Date$. We can define *level* as follows: *level(BirthdayBook)* $= 1$, *level(Person)* $= 1$ and *level(Date)* $= 0$.

2.3.2. Railway Safety Example

This example analyzes a policy for controlling the motion of trains in a railway system. Gates are placed on track segments to prevent trains from colliding. We need a criterion to determine when gates should be closed. There are many formalizations of the railway crossing problem; some of them consider discrete time; others, consider continuous time. Our formulation is from Alloy [11, 12] and it uses a discrete time.

The type *Movers* and the relation *moving* represent sets of moving trains. Some of the relation and function symbols have the suffix *_current* or *_next* to represent an interpretation for the current and next periods. For example, instead of $P(t) \Rightarrow P(t+1)$ we write $P\_current \Rightarrow P\_next$. Here $P(t) \Rightarrow P(t+1)$ means that if P holds at time t then P holds at time $t+1$ and $P\_current \Rightarrow P\_next$ means that if P holds at *current* time then P holds at *next* time.

Tables 2 and 3 contain the specification of the train example.

Formulas Description.

[1] In Section 3.5, we extend this results to support a limited use of transitive closure. However, this extension is still not powerful enough to cover many of the Alloy examples.

[2] The translation to Alloy is given as an example in the Alloy distribution found at http://alloy.mit.edu

Types	$\mathit{Person}, \mathit{Date}, \mathit{BirthdayBook}$
Relations	$\mathit{known} \subseteq \mathit{BirthdayBook} \times \mathit{Person}$
	$\mathit{date} \subseteq \mathit{BirthdayBook} \times \mathit{Person} \times \mathit{Date}$
Functions	$\mathit{getDate} : \mathit{BirthdayBook} \times \mathit{Person} \rightarrow \mathit{Date}$
Constants	$b1, b2 : \mathit{BirthdayBook}$
	$d1, d2 : \mathit{Date}$
	$p1 : \mathit{Person}$
Facts	$\forall b : \mathit{BirthdayBook}\ \forall p : \mathit{Person}\ \forall d : \mathit{Date}\ \mathit{date}(b, p, d) \Rightarrow \mathit{known}(b, p)$
	$\forall b : \mathit{BirthdayBook}\ \forall p : \mathit{Person}\ \mathit{known}(b, p) \Rightarrow \mathit{date}(b, p, \mathit{getDate}(b, p))$
	$\forall b' : \mathit{BirthdayBook}\ \forall p' : \mathit{Person}\ \forall d', d'' : \mathit{Date}$
	$\mathit{date}(b', p', d') \wedge \mathit{date}(b', p', d'') \Rightarrow d' = d''$
Formulas	$\mathit{AddBirthday} : (bb, bb' : \mathit{BirthdayBook}, p : \mathit{Person}, d : \mathit{Date})$
	$\neg \mathit{known}(bb, p) \wedge \forall p' : \mathit{Person}\ \forall d' : \mathit{Date}\ \mathit{date}(bb', p', d') \Leftrightarrow$
	$(p' = p \wedge d' = d) \vee \mathit{date}(bb, p', d')$
Assert	$\mathit{Facts} \wedge \mathit{AddBirthday}(b1, b2, p1, d1) \wedge \mathit{date}(b2, p1, d2) \Rightarrow d1 = d2$

Table 1. Constants, Facts, and Formulas used in the Birthday Book example.

- $safe\_current$ and $safe\_next$ operations express that for any pair of distinct trains t_1 and t_2, the segment occupied by t_1 does not overlap with the segment occupied by t_2.
- $moveOk$ describes under which gate conditions it is legal for a set of trains to move.
- $trainMove$ is a physical constraint: a driver cannot choose to cross from one segment into another segment to which it is not connected. The constraint has two parts. The first ensures that every train that moves ends up in the $next$ time on a segment that is a successor of the segment it was in during the previous $current$ time. The second ensures that the trains that do not move actually stay on the same segments.
- $gatePolicy$ describes the safety mechanism, enforced as a policy on a gate state. It comprises two constraints. The first is concerned with trains and gates; it ensures that the segments that are predecessors of those segments that are occupied by trains should have closed gates. In other words, a gate should be down when there is a train ahead. This is an unnecessarily stringent policy, since it does not permit a train to move to any successor of a segment when one successor is occupied. The second constraint is concerned with gates alone; it ensures that between any pair of segments that have an overlapping successor, at most, one gate can not be closed.

The *Assert* implies that if a move is permitted according to the rules of *moveOK*, and if the trains move according to the physical constraints of *trainMove*, and if the safety mechanism described by *gatePolicy* is enforced, then a transition from a safe state results in a state that is also safe. In other words, safety is preserved.

The specification is not in our fragment St_1, because it contains the functions

$$getSegment\_current : Train \rightarrow Segment$$

Types	*Train*, *Segment*, *GateState*, *Movers*
Relations	*next* $\subseteq$ *Segment* $\times$ *Segment*
	Overlaps $\subseteq$ *Segment* $\times$ *Segment*
	on_current $\subseteq$ *Train* $\times$ *Segment*
	on_next $\subseteq$ *Train* $\times$ *Segment*
	Occupied_current $\subseteq$ *Segment*
	Occupied_next $\subseteq$ *Segment*
	moving $\subseteq$ *Movers* $\times$ *Train*
	closed $\subseteq$ *GateState* $\times$ *Segment*
Functions	*getSegment_current* : *Train* $\rightarrow$ *Segment*
	getSegment_next : *Train* $\rightarrow$ *Segment*
Constants	g : *GateState* m : *Movers*
Facts	*–At any moment every train is on some segment*
	$\forall t$: *Train* *on_current*$(t,$ *getSegment_current*$(t))$
	$\forall t$: *Train* *on_next*$(t,$ *getSegment_next*$(t))$
	–At any moment every train is at most on one segment
	$\forall t$: *Train* $\forall s_1, s_2$: *Segment*
	$($*on_current*$(t, s_1) \wedge$ *on_current*$(t, s_2)) \Rightarrow s_1 = s_2$
	$\forall t$: *Train* $\forall s_1, s_2$: *Segment*
	$($*on_next*$(t, s_1) \wedge$ *on_next*$(t, s_2)) \Rightarrow s_1 = s_2$
	–Occupied gives the set of segments occupied by trains
	$\forall s$: *Segment* *Occupied_current*$(s) \Rightarrow s \in$ *Im*[*getSegment_current*]
	$\forall s$: *Segment* *Occupied_next*$(s) \Rightarrow s \in$ *Im*[*getSegment_next*]
	$\forall t$: *Train* $\forall s$: *Segment* *on_current*$(t, s) \Rightarrow$ *Occupied_current*(s)
	$\forall t$: *Train* $\forall s$: *Segment* *on_next*$(t, s) \Rightarrow$ *Occupied_next*(s)
	–Overlaps is symmetric and reflexive
	$\forall s_1, s_2$: *Segment* *Overlaps*$(s_1, s_2) \Leftrightarrow$ *Overlaps*(s_2, s_1)
	$\forall s$: *Segment* *Overlaps*(s, s)

Table 2. Types, relations, functions, constants and facts used in the train example

and

$$getSegment\_next : Train \rightarrow Segment$$

and both these functions appear in formula $x \epsilon Im[\ldots]$ violating our requirements for St_1 formulas.

Formulas	*safe_current :* $\forall t_1, t_2$: *Train* $\forall s_1, s_2$: *Segment* $(t_1 \neq t_2 \wedge$ *on_current*$(t_1, s_1) \wedge$ *on_current*$(t_2, s_2)) \Rightarrow$ $\neg$*Overlaps*(s_1, s_2) *safe_next:* $\forall t_1, t_1$: *Train* $\forall s_1, s_2$: *Segment* $(t_1 \neq t_2 \wedge$ *on_next*$(t_1, s_1) \wedge$ *on_next*$(t_2, s_2)) \Rightarrow$ $\neg Overlaps(s_1, s_2)$ *moveOk*$(g$: *GateState* ,m : *Movers*$)$: $\forall s$: *Segment* $\forall t$: *Train* $(moving(m, t) \wedge$ *on_current*$(t, s)) \Rightarrow$ $\neg$*closed*(g, s) *trainMove*$(m$: *Movers*$)$ $\forall\ t$: *Train*$\forall s_1, s_2$: *Segment* $($*moving*$(m, t) \wedge$ *on_next*$(t, s_2) \wedge on\_current(t, s_1)) \Rightarrow$ *next*(s_1, s_2) $\wedge$ $\forall t$: *Train* $\forall s$: *Segment* $\neg$*moving*$(m, t) \Rightarrow ($*on_next*$(t, s) \Leftrightarrow$ *on_current*$(t, s))$ *gatePolicy*$(g$: *GateState*$)$ $\forall s_1, s_2, s_3$: *Segment* $next(s_1, s_2) \wedge$ *Occupied_current*$(s_3) \wedge overlaps(s_2, s_3)) \Rightarrow$ $closed(g, s_1)$ $\wedge$ $\forall s_1, s_2, s_3, s_4$: *Segment* $(s_1 \neq s_2 \wedge next(s_1, s_3) \wedge next(s_2, s_4) \wedge overlaps(s_3, s_4)) \Rightarrow$ $(closed(g, s_1) \vee closed(g, s_2))$
Assert	(*Facts* $\wedge$ *safe_current* $\wedge$ *moveOk(g,m)* $\wedge$ *trainMove(m)* $\wedge$ *gatePolicy*$(g)) \Rightarrow$ *safe_next*

Table 3. Formulas and assert used in the train example

2.4. St_2 Class

2.4.1. St_2 Vocabulary

St_2 Vocabulary contains predicates, function symbols, the equality symbol, and atomic formulas $x \in Im[f]$, where f is a function symbol.

2.4.2. St_2 Syntax

The formulas in St_2 are universal formulas over a stratified vocabulary. For every function $f : A_1 \times \ldots \times A_k \to B$ that participates in a subformula $x \epsilon Im[f]$ the following condition holds:

For every function symbol $g : \bar{A}_1 \times \ldots \times \bar{A}_{\bar{k}} \to B$ different from f:

$$(*)\begin{cases} f \text{ and } g \text{ have the same type } A_1 \times \ldots \times A_k \to B \text{ and} \\ \forall b : B \forall a_1 : A_1, \ldots, \forall a_k : A_k \; [g(a_1, \ldots, a_k) = b] \Rightarrow [b \notin \textit{Im}(f) \vee f(a_1, \ldots, a_k) = b] \end{cases}$$

Note that (*) is a semantical requirement. When we say that a Str_2 formula ψ is "satisfiable", we mean that it is satisfiable in a structure that fulfills this semantical requirement (*).

In many cases formalized by us the requirement (*) above immediately follows from the intended interpretation of functions. In the Railway safety example, some work needs to be done to derive this requirement from the specification.

First we notice that the specification contains functions $getSegment\_current : Train \to Segment$ and $getSegment\_next : Train \to Segment$. We can define *level* as follows: *level(Train)* $= 1$, *level(Segment)* $= 0$, *level(GateState)* $= 0$ and *level(Movers)* $= 0$.

It remains to prove that the semantic requirement holds. In the Train specification there are *getSegment_current*, *getSegment_next* functions such that $x \in Im[$*getSegment_current*$]$ and $x \in Im[$*getSegment_next*$]$ participates in the formula. Therefore, it remains to show that

$$\forall b : \textit{Segment} \forall a : \textit{Train} \; [\textit{getSegment\_current}(a) = b] \Rightarrow$$

$$[b \notin \textit{Im}(\textit{getSegment\_next}) \vee \textit{getSegment\_next}(a) = b]$$

and

$$\forall b : \textit{Segment} \forall a : \textit{Train} \; [\textit{getSegment\_next}(a) = b] \Rightarrow$$

$$[b \notin \textit{Im}(\textit{getSegment\_current}) \vee \textit{getSegment\_current}(a) = b]$$

We prove a stronger requirement

$$\forall t_1, t_2 : \textit{Train} \;\; (t_1 \neq t_2) \Rightarrow \textit{getSegment\_current}(t_1) \neq \textit{getSegment\_next}(t_2)$$

It is clear that the first two requirements above follow from the last requirement. Therefore it is suffices to show that $\forall t_1, t_2 :$ *Train* $(t_1 \neq t_2) \Rightarrow$ *getSegment_current*$(t_1) \neq$ *getSegment_next*(t_2).

Let *M* be model such that

$$M \vDash (\textit{Facts} \wedge \textit{safe\_current} \wedge \textit{moveOk(g,m)} \wedge \textit{trainMove(m)} \wedge \textit{gatePolicy}(g))$$

Let $t_1 \neq t_2$ and suppose that *getSegment_current*$(t_1) = s$. From the Train $Facts$ immediately follows that *Occupied_current*(s). Hence from *gatePolicy* follows that all previous *Segments* of s have a closed gate. Thus, according to *moveOk*, no train comes to s at the next time. But $M \vDash$ *safe_current* so $s \neq$ *getSegment_current*(t_2). From this and from the fact that no train comes to s at the next time, it follows that $s \neq$ *getSegment_next*(t_2).

3. Decidability of Validity Problem

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be sets of formulas. We denote by $\mathcal{F}_1 \Rightarrow \mathcal{F}_2$ the set $\{\psi \Rightarrow \varphi \;:\; \psi \in \mathcal{F}_1$ and $\varphi \in \mathcal{F}_2\}$. The set of universal sentences is denoted by UN. The main results of this section are stated in the following theorem.

Theorem 2. *The validity problem for $St_2 \Rightarrow UN$ is decidable.*

We also prove that every sentence in $St_2 \Rightarrow UN$ is valid iff it holds over the class of finite models.

The section is organized as follows. First, we introduce basic definitions. Next, following Beauquier and Slissenko [2, 3] we provide sufficient semantical conditions for the decidability of the validity problem. Unfortunately, these semantical conditions are undecidable. However, we show that the formulas in $St_2 \Rightarrow UN$ satisfy these semantical conditions.

3.1. Basic Definitions

Definition 3 (Partial Model)**.** Let L be a many-sorted first-order language. A partial Model M' of L consists of the following ingredients:

- For every sort s a non-empty set D'_s, called the $domain$ of M'.
- For every predicate symbol p^i_s of L with argument types $s_1, \ldots, s_n$ an assignment of an n-place relation $(p^i_s)^{M'}$ in $D'_{s_1} \times \ldots \times D'_{s_n}$.
- For every function symbol f^i_s of L with type $f^i_s : s_1 \times s_2 \times \ldots s_n \rightarrow s$ an assignment of a partial n-place operation $(f^i_s)^{M'}$ in $D'_{s_1} \times \ldots \times D'_{s_n} \rightarrow D'_s$.
- For every individual constant c^i_s of L an assignment of an element $(c^i_s)^{M'}$ of D'_s.

We say that a partial model is finite if every D'_s is finite. A partial model M' is a model if every function $(f^i_s)^{M'} : D'_{s_1} \times \ldots \times D'_{s_n} \rightarrow D'_s$ is total.

The following definition strengthens the notion of the finite model property.

Definition 4 (Satisfiability with Finite Extension)**.** A formula ψ is satisfiable with a finite extension iff for every finite partial model M': if M' can be extended to a model M of ψ, then M' can be extended to a finite model $\bar{M}$ of ψ.

The satisfiability with a finite extension definition was inspired by (but is quite different from) the definition of *C-satisfiable* with augmentation for complexity (k, n) in [2, 3].

Definition 5 (k-Refutability)**.** A formula ψ is k-refutable iff for every counter-model M of ψ there exists a finite partial model M' such that:

- For each sort $s : |D'_s| \leq k$
- M is an extension of M'
- Any extension of M' to a model is a counter-model of ψ.

We say that a formula is finitely refutable if it is k-refutable for some $k \in Nat$.

Example 6 (k-Refutability)**.** Recall the formula *safe_current* of Railway Safety system:
safe_current :

$$\begin{aligned}&\forall t_1, t_1 : \textit{Train}\ \forall s_1, s_2 : \textit{Segment}\\ &\quad (t_1 \neq t_2 \wedge \textit{on\_current}(t_1, s_1) \wedge \textit{on\_current}(t_2, s_2))\\ &\quad \Rightarrow \neg \textit{Overlaps}(s_1, s_2)\end{aligned}$$

The constraint ensures that at *current* moment for any pair of distinct trains t_1 and t_2, the segment that t_1 occupies is not a member of the set of segments that overlap with the segment occupied by t_2. Let us show that *safe_current* is 2-refutable. Suppose that *safe_current* has a counter-model M then there are: t_1, t_2 : $\textit{Train}^M$, s_1, s_2 : $\textit{Segment}^M$ such that $M \models \neg(\textit{on\_current}(t_1, s_2) \wedge \textit{on\_current}(t_2, s_2) \wedge t_1 \neq t_2 \Rightarrow \neg \textit{Overlaps}(s_1, s_2))$. Take M' as the submodel of M with the domains $\textit{Train}^{M'} = \{t_1, t_2\}$, $\textit{Segment}^{M'} = \{s_1, s_2\}$. For any extension of

M' to model $\bar{M}$ it still holds that $\bar{M} \models \neg(\textit{on\_current}(t_1, s_2) \wedge \textit{on\_current}(t_2, s_2) \wedge t_1 \neq t_2 \Rightarrow \neg\textit{Overlaps}(s_1, s_2))$, so $\bar{M}$ is a counter-model of *safe_current*.

From the above example we can learn that if M is a counter-model for a k-refutable formula, then M contains k elements in the domain that cause a contradiction. If we take the partial model obtained by the restriction of M to these elements, then any extension of it still contains these elements and therefore it still is a counter-model.

In the rest of this section we prove the decidability of formulas of the form $\theta \Rightarrow \vartheta$, where θ is *satisfiable with finite extension* and ϑ is *k-refutable* for some k. In addition we prove that:

- Every formula in St_2 is *satisfiable with finite extension*.
- A formula is equivalent to a formula from UN iff the formula is *k-refutable* for some k.

This completes the proof of decidability of formulas of the form $St_2 \rightarrow UN$.

3.2. Sufficient Semantical Conditions for Decidability

The next lemma is a consequence of the Definitions 4 and 5.

Lemma 7 (Finite Counter-Model Property)**.** *Let ψ be a formula of the form $\theta \Rightarrow \varphi$, where θ is satisfiable with a finite extension and φ is finitely refutable. Then $\neg\psi$ has the finite model property.*

Proof. Suppose that $\neg\psi$ has model M, hence $M \models \theta \wedge \neg\varphi$. Hence, $M \models \neg\varphi$. However, *φ is k-refutable*, therefore M has a finite partial submodel M' as in the definition of *k-refutability*. M' can be extended to M and $M \models \theta$. *θ is satisfiable with a finite extension*, hence M' can be extended to $\bar{M}$ such that $\bar{M}$ is finite and $\bar{M} \models \theta$. From *k-refutability* $\bar{M} \models \neg\varphi$.
Therefore, if $\neg\psi$ has a model then it has a finite model. □

Note that the lemma does not give a bound to the size of the model.

Theorem 8 (Sufficient Conditions for Decidability)**.** *Let $\mathcal{F}_{fin-ref}$ be a set of sentences in many-sorted first-order logic that are finitely refutable and let $\mathcal{F}_{sat-fin-ext}$ be a set of sentences in many-sorted first-order logic that are satisfiable with a finite extension. Then the validity problem for $\mathcal{F}_{sat-fin-ext} \Rightarrow \mathcal{F}_{fin-ref}$ is decidable. Moreover, if $\psi \in \mathcal{F}_{sat-fin-ext} \Rightarrow \mathcal{F}_{fin-ref}$, then ψ is valid iff it is valid over the finite models.*

Proof. The validity problem for many-sorted first-order logic is recursively enumerable. According to Lemma 7, if a sentence in this class is not valid then it has a finite counter-model. Hence, to check whether a sentence φ in this class is valid we can start (1) to enumerate proofs while looking for a proof of φ and (2) to enumerate all finite models while looking for a counter-model for φ. Either (1) or (2) succeed. If (1) succeeds, then φ is valid; if (2) succeeds, then φ is not valid. □

Since Lemma 7 does not provide a bound of the size of the model, we cannot provide a concrete complexity bound on the algorithm in Theorem 8.

Theorem 8 provides semantical conditions on a class of formulas that ensure the decidability of the validity problem for this class. Unfortunately, these semantical conditions are undecidable.

Theorem 9. *The following semantical properties of sentences are undecidable:*

(1) For every $k \in Nat$:
Input: *A formula ψ.*
Question: *Is ψ k-refutable?*
(2) ***Input:*** *A formula ψ.*
Question: *Is ψ finitely refutable ?*
(3) ***Input:*** *A formula ψ.*
Question: *Is ψ* satisfiable with a finite extension*?*

Proof. (1) and (2) follow from Trakhtenbrot's theorem [17, 4]. The Trakhtenbrot theorem states that the set of sentences over a relational vocabulary that are valid over the finite models cannot be separated by a recursive set from the set of unsatisfiable sentences.

(1) Let Ref_k be the set of k refutable sentences over a relational vocabulary. We will show that Ref_k contains the set of unsatisfiable sentences and is disjoint from the sentences valid over all finite models. Hence, it is not recursive.

By definition, Ref_k contains the set of unsatisfiable sentences.

Assume that ψ is valid over the finite models. Let M be a counter-model of ψ and let ***M'*** be a partial submodel of M of size $\leq k$. Let $\bar{M}$ be an extension of M' to a model over the same domain as M'. Since ψ is valid over finite models, $\bar{M}$ cannot be a counter-model of ψ. Therefore, ψ cannot be k refutable.

(2) The set of finitely refutable sentences is equal to $\cup_k Ref_k$. Since for every k the set Ref_k contains the set of unsatisfiable sentences and is disjoint from the set of sentences valid over the finite models, the set $\cup_k Ref_k$ separates between the sentences valid over the finite models and the set of unsatisfiable sentences, and therefore it is not recursive.

(3) The Halting Problem is the problem of deciding whether a given Turing machine accepts the empty word. It is well known that the Halting Problem is unsolvable.

We are going to reduce the Halting Problem to the satisfiability with a finite extension problem.

Standard proofs of the undecidability of the first-order predicate logic (see e.g., [5]) provide an algorithm that for any Turing machine m constructs a formula ***Run[m]*** that encodes a computation of Turing machine m on the empty word. This formula has the following properties:

(1) If *m* accepts the empty word, then ***Run[m]*** is satisfiable and all its models are finite.
(2) If *m* does not accepts the empty word, then ***Run[m]*** is satisfiable and all its models are infinite.

Therefore, ***Run[m]*** is satisfiable with finite extension iff m accepts the empty word.

Indeed, assume m does not accept the empty word. Let M be a model that satisfies ***Run[m]***. Such an M exists and it is infinite, moreover, no partial model of M can be extended to a finite model of ***Run[m]***. Hence, ***Run[m]*** is not satisfiable with a finite extension.

On the other hand, if m accepts the empty word and M satisfies ***Run[m]***, then M is finite and every finite partial submodel of M can be extended to a finite model of ***Run[m]***, namely to M. Hence, in this case ***Run[m]*** is satisfiable with a finite extension.

This accomplishes our reduction of the Halting Problem to

satisfiability with the finite extension problem. Therefore, the satisfiability with a finite extension problem is undecidable. □

In the next two subsections we describe syntactical conditions that ensure

(1) finite refutability property.
(2) satisfiability with a finite extension property.

3.3. Syntactical Conditions for Decidability

The proof of the following lemma uses the preservation theorem [7] from first-order logic, which says that a sentence ψ is equivalent to a universal formula iff any submodel of a model of ψ is a model of ψ. The preservation theorem is valid also for the many-sorted first-order logics.

Lemma 10 (Syntactical Conditions for Finite Refutability)**.** *A sentence ψ is k-refutable for some k iff ψ is equivalent to a universal sentence.*

Proof. $\Rightarrow$
Let ψ be *k-refutable* sentence for some k. Suppose, by contradiction, that ψ is not equivalent to a universal sentence.

By the preservation theorem, ψ is not equivalent to a universal sentence iff there is a model M and sub-model M' of M such that:

- $M \models \psi$
- $M' \not\models \psi$

Let M and M' be such models.

ψ is *k-refutable* and $M' \not\models \psi$, Hence, M' contains a partial model M'' of size at most k such that for any extension $\bar{M}$ of M'' we have $\bar{M} \not\models \psi$. However, M is an extension of M'' and $M \models \psi$ - a contradiction.

$\Leftarrow$

Let $\psi = \forall x_1 \ldots \forall x_n \ \phi(x_1, \ldots, x_n)$, where ϕ is quantifier-free. We will show that ψ is k-refutable, where k is the number of terms in ψ.

Let M be a counter-model of ψ. Hence $M \models \neg\phi(a_1, \ldots, a_n)$ for some $a_1, \ldots, a_n$ from $Dom(M)$. Let M' be the partial submodel of M over the following set

$$D = \{ \text{ the values of the terms from } \psi \text{ under the assignment of } a_i \text{ to } x_i \ (i = 1, \ldots, n)\}$$

Let $\bar{M}$ be any extension of M' to a model. $\bar{M}$ and M have the same interpretation for the terms and sub-formulas from ϕ under the assignment of a_i to x_i $(i = 1, \ldots, n)$. Hence, using the fact that $M \models \neg\phi(a_1, \ldots, a_n)$, we obtain that $\bar{M} \models \neg\phi(a_1, \ldots, a_n)$. Hence, $\bar{M}$ is a counter-model of ψ. Therefore for any extension $\bar{M}$ of M' to a model $\bar{M}$ is a counter-model of ψ. $\square$

Usually safety properties are easily formalized by universal formulas. Hence, the class $\mathcal{F}_{sat-fin-ext} \Rightarrow UN$ is appropriate for verification of safety properties and has a decidable validity problem.

The next theorem is our main technical theorem.

Theorem 11. *Every St_2 formula is satisfiable with a finite extension.*

Proof. Assume that a formula $\psi \in St_2$ is satisfiable in M and that M' is a finite partial sub-model of M.

First, we extend M' to a finite partial sub-model M'' of M such that $Im[f]$ has a "correct" interpretation. Assume that the levels of types in Σ are in the set $\{0, \ldots, m\}$.

Let $M_0 = M'$. For $i = 0, \ldots, m$ we define D_i ,N_i and M_{i+1} as follows. Let D_i be the set of elements in M_i of the types at level i such that $b \in D_i$ iff $M \models b \in Im[f]$ and there is no tuple $\bar{a} \in M_i$ with $M \models f(\bar{a}) = b$.

Now, for every $b \in D_i$ choose $\bar{a} \in M$ such that $M \models f(\bar{a}) = b$. Observe that each element in $\bar{a}$ has a type at level $> i$. Let N_i be the set of all chosen elements (for all elements in D_i and all function symbols in Σ). Let M_{i+1} be the partial sub-model of M over $Dom(M_i) \cup N_i$.

It is not difficult to show that M_{i+1} is a finite partial submodel of M and that for every $b \in Dom(M_i)$ if B is the type of b and the level of B is at most i, then there is $\bar{a} \in M_{i+1}$ such that $M \models f(\bar{a}) = b$ iff there is $\bar{a'} \in M_{i+1}$ such that $M_{i+1} \models f(\bar{a'}) = b$.

In particular, for every $b \in Dom(M_{m+1})$ if $M \models b \in Im[f]$, then there is a tuple $\bar{a} \in Dom(M_{m+1})$ such that $M_{m+1} \models f(\bar{a}) = b$.

Next, define M'' as M_{m+1} and let Ass be the set of assignments to the variables with values in $Dom(M'')$ and let $\bar{D}$ be the set of values (in M) of all terms over Σ under these assignments. The set $\bar{D}$ is finite, because our vocabulary is stratified and M'' is finite. Let $\bar{M}$ be the partial submodel of M over the domain $\bar{D}$. From the definition of $\bar{M}$, it follows that $\bar{M}$ is a submodel of M, i.e., all functional symbols are interpreted by total functions.

It remains to be shown that the interpretations of $Im[f]$ in M and $\bar{M}$ agree. For this, we need the semantic requirement of St_2. Let $b \in Dom(\bar{M})$ and suppose that $M \models b \in Im[f]$. We need to show that there is a tuple $\bar{a} \in Dom(\bar{M})$ such that $\bar{M} \models f(\bar{a}) = b$. If $b \in Dom(M_{m+1})$, it follows from the previous assumption. If $b \notin Dom(M_{m+1})$, then $M \models b = g(\bar{a})$ for some g and $\bar{a} \in Dom(\bar{M})$. From the semantic requirement, g and f have the same type, therefore $f(\bar{a})$ is defined. Hence, from the semantic requirement and from the fact that $M \models g(\bar{a}) = b$ and the fact that $M \models b \in Im[f]$ it follows that $M \models f(\bar{a}) = b$. Hence, $\bar{M} \models f(\bar{a}) = b$. □

Finally, Theorem 2 is an immediate consequence of Theorem 8, Lemma 10 and Theorem 11.

3.4. Strong Satisfiability with Finite Extension

From the proof of Theorem 11 we learn that $\bar{M}$ is a submodel of ***M***. We formalize this property in the following definition.

Definition 12 (Strong Satisfiability with Finite Extension)**.** A formula ψ is strongly satisfiable with a finite extension iff for every finite partial model M': if M' can be extended to a model M of ψ, then M' can be extended to a finite model $\bar{M}$ of ψ such that $\bar{M}$ is a submodel of ***M***.

Conclusion 13. The formulas from St_0, St_1 and St_2 are strongly satisfiable with a finite extension.

3.5. Transitive Closure

Most of the Alloy specifications contain transitive closure. We partially treat transitive closure and succeed in covering some of the Alloy specifications with transitive closure.

Definition 14 (Transitive Closure Model)**.** Let ψ be a formula containing two binary predicates $p, tc_p \subset T \times T$ for some type T. A tc-model M_{tc} of ψ is a model such that for each assignment z:
$M_{tc}, z \models tc_p(t, s)$ iff there are $e_1, e_2, \ldots, e_n$ in the domain of M_{tc} such that $e_1 = z(t)$ and $e_n = z(s)$ and $M_{tc} z[x := e_i, y := e_{i+1}] \models p(x, y)$ for $i = 1 \ldots n - 1$.

Theorem 15. *Let ψ be strongly satisfiable with a finite extension. Suppose:*
- ψ contains two binary predicates $p, tc_p \subset T \times T$ for some type T.
- tc_p does not appear positive in ψ.
Then ψ has the satisfiability with a finite extension property for tc-models, i.e., if M' is any finite partial model of a tc-model that satisfies ψ, then M' can be extended to a finite tc-model that satisfies ψ.

Proof. Let $M \models \psi$ be a tc-model and let M' be a finite partial submodel of M. ψ is strongly satisfiable with a finite extension, therefore there is a finite extension $\bar{M}$ of M' such that $\bar{M} \models \psi$ and $\bar{M}$ is a submodel of M. If $\bar{M}$ is a tc-model, the proof is finished.

Suppose $\bar{M}$ is not a tc-model. $\bar{M}$ is a submodel of M and M is a tc-model. Let us build $\bar{M}'$ which is the same model as $\bar{M}$ except that we redeclare tc_p to be the transitive closure of relation p. From the fact that $\bar{M}$ is a submodel of M follows that $p^{\bar{M}'}$ is a subset of p^M; therefore, for any elements $e_1,\ e_2$ $M \not\models tc_p(e_1, e_2)$, then $\bar{M}' \not\models tc_p(e_1, e_2)$. From this and from the fact that tc_p can appear only negatively, it holds that $\bar{M}' \models \psi$, and $\bar{M}'$ is a tc-model. □

We succeeded in formalizing the Alloy Grandpa example [11, 12] using this theorem. We noticed that some examples use transitive closure only for negating cycles. For example, suppose there is a relation *next* and a property that says there is no sequence of elements $e_1, \ldots, e_n$, such that $e_1 = e_n$ and $next(e_i, e_{i+1})$ $i = 0, \ldots, n-1$. The formula $\neg tc_{next}(e_1, e_1)$ enforce the absence of cycles and it contains only the negative occurrence of the transitive closure.

In [10] the small model property was proven for a fragment of first-order logic with deterministic transitive closure. Here, deterministic transitive closure is a restriction of transitive closure to paths that have no choices. For a binary relation $E(x, y)$, define $E_d(x, y)$ as follows:

$$E_d(x, y) := E(x, y) \wedge \forall z\ (E(x, z) \Rightarrow z = y)$$

That is, if vertex v has more than one outgoing *E-edge*, then it has no outgoing E_d edges. Define the deterministic transitive closure of E (notation - dtc_E) as the transitive closure of E_d.

Note that if a binary relation E is a graph of a partial function, i.e., for every x there is at most one y such that $E(x, y)$ holds, then the deterministic transitive closure of E coincides with the transitive closure of E.

Theorem 16. *Let T be a type and let $F \subseteq St_2$ be the set of formulas that fulfils the following conditions:*
(1) There are two predicates $p, tc_p \in T \times T$.
(2) Every predicate q different from p or tc_p contains at most one argument of type T.
(3) The interpretation of p is the graph of a partial function (hence, the deterministic transitive closure of p is the same as the transitive closure of p).
(4) There is no function with an argument in T.
(5) If T is a range of a function f then $x \in Im[f]$ does not occur in formulas from F.
Then, the satisfiability problem for F is decidable.

Proof. (Sketch) The proof is similar to the proof of Theorem 4 from [10] combined with the proof of Theorem 11. This proof does not contain any new ideas. Here is a short sketch. Let M_{tc} be a tc-model of $\psi \in F$. Like in the proof of theorem Theorem 11, we can build a finite submodel M' of M_{tc} that satisfies ψ. All types have only a finite number of elements. Hence, we can treat the formula like a formula over type T only. Then as for the proof of theorem Theorem 4 from [10], we can prove that M' can be extended to a tc-model M'_{tc} that satisfies ψ (by adding required new elements to type T). □

Types	*Level*(*represents years*), *Module*(*represents courses*)
Relations	*Level is linear ordered*
	next $\subseteq$ *Level* $\times$ *Level*
	Prerequisite on Modules
	prereq $\subseteq$ *Module* $\times$ *Module*
	Modules to be taken concurrently
	coreq $\subseteq$ *Module* $\times$ *Module*
	Modules that are incompatible
	excluded $\subseteq$ *Module* $\times$ *Module*
	coreq_ext $\subseteq$ *Module* $\times$ *Module*
Functions	*For each module assigns its level*
	level : *Module* $\rightarrow$ *Level*
Constants	*First and last years and some modules*
	$first,\ last \subseteq$ *Level*, $m_1, \ldots, m_6 \subseteq$ *Module*

Table 4. Relation, Functions, and Constants used in the university example.

3.5.1. University Example

The University example is formalized in Tables 4 and 5. At the university, the processes of student enrollment, assessment, course transfer, and completion, as well as the slower processes of course modification take place against a background of modules. Tables 4 and 5 show our formalization of the University example taken from Alloy. It is clear that this example satisfies conditions of Theorem 16. *next* is equal to tc_{next} and there is no function with an argument of type *Level* and *Im* is not used in the formula. Hence, we can verify whether the specification is consistent (satisfiable).

4. Some Fragments of Many-Sorted Logic

In the previous section we introduced decidable fragments of many-sorted logic. In this section, we consider classes from first-order logic that have the finite-model property. We try to find a way to extend these classes to many-sorted logic.

We use the notation from [6] only as the names for the five classes below. According to [6], the following classes have the finite model property:

- $[\exists^*\forall^*, all]_=$ (Ramsey 1930) the class of all sentences with quantifier prefix [3] $\exists^*\forall^*$ over arbitrary relational vocabulary with equality.
- $[\exists^*\forall\exists^*, all]_=$ (Ackermann 1928) the class of all sentences with quantifier prefix $\exists^*\forall\exists^*$ over an arbitrary relational vocabulary with equality.
- $[\exists^*, all, all]_=$ (Gurevich 1976) the class of all sentences with quantifier prefix $\exists^*$ over an arbitrary vocabulary with equality.

[3] $\exists^*\forall^*$ stands for the quantifier prefix that begins with any number of $\exists$ quantifiers that follows any number of $\forall$ quantifiers. We use regular expressions for other quantifier prefixes.

facts	*Next is linear order*
	$\forall l$: *Level* $\neg next(last, l) \wedge$
	$[\,\forall l_1, l_2$: *Level* $l_1 = l_2 \vee \forall\, elem$: *Level* $[\forall\, l_1, l_2$: *Level* $(next(elem, l_1) \wedge$
	$next(elem, l_2)) \Rightarrow l_1 = l_2 \wedge \neg tc_{next}(elem, elem)]] \wedge$
	$\forall\, l$: *Level* $tc_{next}(first, l) \vee l = first$
	Module cannot exclude itself
	$\forall m$: *Module* $\neg tc_{next}(m, m)$
	Module cannot coreq itself
	$\forall m$: *Module* $\neg$*coreq*(m, m)
	No cycles in prerequisites
	$\forall m$: *Module* $\neg$*prereq*(m, m)
	coreq is transitive and symmetric
	$\forall m_1, m_2, m_3$: *Module coreq*$(m_1, m_2) \wedge$ *coreq*$(m_2, m_3) \Rightarrow$ *coreq*(m_1, m_3)
	$\forall m_1, m_2$: *Module coreq*$(m_1, m_2) \Leftrightarrow$ *coreq*(m_2, m_1)
	prereq must be at lower level
	$\forall p_1, p_2$: *Module* $prereq(p_1, p_2) \Rightarrow tc_{next}(level(p_1), level(p_2))$
	coreqs must be at the same level
	$\forall m_1, m_2$: *Module coreq*$(m_1, m_2) \Rightarrow$ *level*$(m_1) =$ *level*(m_2)
	there are some coreq , prereq and excluded
	coreq$(m_1, m_2) \wedge$ *prereq*$(m_3, m_4) \wedge$ *excluded*(m_5, m_6)

Table 5. Facts used in the university example

- $[\exists^*\forall, all, (1)]_=$ (Grädel 1996) the class of all sentences with quantifier prefix $\exists^*\forall$ over a vocabulary that contains unary function and arbitrary predicate symbols with equality.
- FO^2 (Mortimer 1975) [15] the class of all sentences over a relational vocabulary with equality, that contain at most two distinct variables.

Below we describe a generic naive way to generalize a class of first-order formulas to many-sorted logic. Unfortunately, finite model property and decidability are not preserved under this generalization.

Let $Q_1 \ldots Q_m$ be a quantifier prefix in many-sorted logic. Its projection on a type A is obtained by erasing all quantifiers over the variables of types distinct from A. One can hope that if for every type A the projection of the quantifier prefix on A is in a decidable class of one-sorted logic, then this prefix is in a decidable class of many-sorted logic. However, we show that neither the decidability nor the finite model property for a prefix of many-sorted logic is inherited from the corresponding properties of projections.

When we take a projection of a formula to a type, in addition to removing the quantifiers over other types, we should also modify the quantifier-free part of the formula. Here is a definition:

Definition 17 (Projection of a Formula on Type A)**.** Let ψ be a formula of many-sorted logic in

the prenex normal form. Its projection on type A is denoted by $\bar{\psi}^A$ and is obtained as follows:

(1) For each type T different from A:

   (a) Eliminate all quantifiers of type T.

   (b) Replace every term of type T by constant C^T.

(2) Let $R(t_1, \dots t_k)$ be an atomic sub-formula that contains new constants C^{T_j} $(1 \leqslant j \leqslant m)$ at positions $i_1, i_2, \dots i_m$.
Introduce a new predicate name $P_{i_1,i_2,\dots,i_m}$ of arity $k - m$
and replace $R(t_1, \dots t_k)$
by $P_{i_1,i_2,\dots,i_m}(t_1, \dots t_{i_1-1}, t_{i_1+1}, \dots t_{i_2-1}, t_{i_2+1} \dots t_{i_m-1}, t_{i_m+1} \dots t_k)$.

(3) Let $f(t_1, \dots t_k)$ be a term which contains new constants C^{T_j} $(1 \leqslant j \leqslant m)$ at positions $i_1, i_2, \dots i_m$.
Introduce a new function name $f_{i_1,i_2,\dots,i_m}$ of arity $k - m$
and replace $f(t_1, \dots t_k)$
by $f_{i_1,i_2,\dots,i_m}(t_1, \dots t_{i_1-1}, t_{i_1+1}, \dots t_{i_2-1}, t_{i_2+1} \dots t_{i_m-1}, t_{i_m+1} \dots t_k)$.

For a formula ψ its projection on A is the formula $\bar{\psi}^A$ with one type; hence, it can be considered as the first-order logic formula.

Definition 18 (Naive Extension)**.** A set of many-sorted first-order formulas D^{ext} is a naive extension of a set of first-order formulas D if for every $\psi \in D^{ext}$ and for every type A, it holds that $\bar{\psi}^A \in D$.

Examples:

(1) Let ψ be $\forall x_1 : A\, \forall x_2 : B\, \exists y_1 : A\, \forall y_2 : B\, p(x_1, y_1, x_2) \vee q(y_1, y_2)$.
Let us look at its projections on A and B. After the first two steps we obtain the formulas $\forall x_1 : A\ \exists y_1 : A\, p(x_1, y_1, c^B) \vee q(y_1, c^B)$ and $\forall x_2 : B\, \forall y_2 : B\, p(c^A, c^A, x_2) \vee q(c^A, y_2)$. After replacing predicates, we obtain:

$$\forall x_1 : A\ \exists y_1 : A\, p_3(x_1, y_1) \vee q_2(y_1)$$

and

$$\forall x_2 : B\, \forall y_2 : B\, p_{1,2}(x_2) \vee q_1(y_2).$$

Both formulas are in FO^2. Hence, ψ is in $[FO^2]^{ext}$.

(2) Let ψ be $\forall x_1 : A\, \forall x_2 : B\, \exists y_1 : A\, \exists y_2 : B\, p(x_1, y_1, x_2) \vee p(x_1, x_1, y_2) \vee q(y_1, x_1)$. Its projections on A and B are $\forall x_1 : A\, \exists y_1 : A\, p_3(x_1, y_1) \vee p_3(x_1, x_1) \vee q(y_1, x_1)$ and $\forall x_2 : A\, \exists y_2 : A\, p_{1,2}(x_2) \vee p_{1,2}(y_2) \vee q_{12}$. Since the projections are in Ackermann class, ψ is in the extension of Ackermann class.

Note that the extension of the Ramsey class to many-sorted logic is a subclass of St_0, consisting of St_0 formulae not containing function symbols of arity ≥ 1. Therefore it has the finite model property and is decidable. It is easy to prove that the naive extension of Gurevich's class is decidable. The next two theorems state that the naive extensions of the Ackermann, Grädel, and Mortimer classes are undecidable and therefore do not have the finite model property.

Let us recall the tilling problem.

Definition 19. Define a *tiling problem*, $\mathcal{T} = \langle T, Adj_h, Adj_v \rangle$, to consist of a finite list of tile types, $T = [t_0, \ldots t_k]$, together with horizontal and vertical adjacency relations, $Adj_h, Adj_v \subseteq T^2$. Here $Adj_h(a, b)$ means that tiles of type b fit immediately to the right of tiles of type a, and $Adj_v(a, b)$ means that tiles of type b fit one step down from those of type a. A *solution* to a tiling problem is an arrangement of instances of the tiles in the grid $Nat \times Nat$ where all adjacency relationships are respected.

It is well known that the tiling problem is undecidable. (See [4] for a thorough treatment of tiling problems, as well as discussions of many relevant decidable and undecidable logics.).

Theorem 20 (Undecidability)**.** *The satisfiability problem is undecidable for each of the following fragments:* $[\exists^*\forall, all, (1)]^{ext}_{=}$, $[FO^2]^{ext}$, *and* $[\exists^*\forall\exists^*, all]^{ext}_{=}$.

Proof. The proof of the next theorem shows that the tiling problem is reducible to the satisfiability problem of each of the fragments $[\exists^*\forall, all, (1)]^{ext}_{=}$, $[FO^2]^{ext}$, and $[\exists^*\forall\exists^*, all]^{ext}_{=}$.

Let $\mathcal{T}$ be a tiling problem. We are going to define a formula Ψ such that Ψ is satisfiable iff $\mathcal{T}$ has a solution. Then for each of these three fragments we construct from Ψ an equisatisfiable formula in this fragment.

The formula has two types: h (for horizontal) and v (for vertical). The vocabulary contains a binary relational symbol T_i of type $h \times v$ for every tile type t_i, and it has two functions $H : h \to h$ and $V : v \to v$.

Let Ψ be

$$\forall x : h \forall y : v \Psi_1 \wedge \forall x : h \forall y : v \Psi_2 \wedge \forall x : h \forall y : v \Psi_3$$

where

(1) $\Psi_1(x, y)$ is $\big(\bigwedge_i \big(T_i(x, y) \Leftrightarrow \neg \bigvee_{j \neq i} T_j(x, y)\big)$ - the pair (x, y) is tiled by exactly one tile.

(2) $\Psi_2(x, y)$ is $\bigwedge_i \big[\big(T_i(x, y) \to \vee_{\{j \,:\, Adj_h(t_i, t_j)\}} T_j(H(x), y)\big)\big]$ - tiling respects the horizontal adjacency relation.

(3) Ψ_3 is $\bigwedge_i \big[\big(T_i(x, y) \to \vee_{\{j \,:\, Adj_v(t_j, t_i)\}} T_j(x, V(y))\big)\big]$ - tiling respects the vertical adjacency relation.

Note that Ψ is a universal formula. It is satisfiable iff it is satisfiable in a Herbrand model. It is clear that Ψ is satisfiable in a Herbrand model iff $\mathcal{T}$ has a solution.

This proves that the satisfiability problem for $[\forall, all, (1)]^{ext}$ formulas is undecidable. The class $[\forall, all, (1)]^{ext}$ is a subclass of $[\exists^*\forall, all, (1)]$. Hence, the satisfiability problem for $[\exists^*\forall, all, (1)]$ formulas is undecidable.

Let ψ_2 be obtained from Ψ_2 when $H(x)$ is replaced by a variable z and let ψ_3 be obtained from Ψ_3 when $V(x)$ is replaced by a variable w. Let Φ_2 be

$$\forall x : h \forall y : v \Psi_1 \wedge \forall x : h \exists z : h \forall y : v \psi_2 \wedge \forall y : v \exists w : h \forall x : h \psi_3.$$

Note that Ψ is the Skolem normal form of Φ_2. Hence, Φ_2 is satisfiable iff Ψ is satisfiable iff $\mathcal{T}$ has a solution. Observe that Φ_2 is in $[FO^2]^{ext}$. Therefore, the satisfiability problem for $[FO^2]^{ext}$ formulas is undecidable.

Finally, we show that the naive extension of the Ackermann class $[\exists^*\forall\exists^*, all]$ is undecidable. This proof is more subtle and we start from the following formula Θ:

$$\Theta ::= \forall x : h \exists x' : h \forall y : v \exists y' : v \big[\big(R(x', y) \to R(x, y)\big) \wedge R(x, y') \wedge \neg R(x', y')\big]$$

We first show that the finite model property fails for Θ.

The formula expresses that for every x there is x' such that the set $\{y \ : \ R(x',y)\}$ is a proper subset of $\{y \ : \ R(x,y)\}$. It is almost clear that for every model M of Θ and $a \in M$ the set $\{b : R(a,b)$ holds in $M\}$ cannot be finite. Hence, the finite model property fails for Θ. Below is a formal proof.

Assume $M \models \Theta$. Then there is an expansion M' of M where the Skolem normal form Θ' of Θ is satisfiable, where

Θ' is $\forall x : h \forall y : v\big(R(H(x),y) \rightarrow R(x,y)\big) \wedge R(x,U(x,y)) \wedge \neg R(H(x),U(x,y))$

Let x_0 be an element of type h. Define $x_{i+1} = H(x_i)$. Let y_0 be an element of type v such that $R(x_0,y_0)$ holds (such an element exists). Define $y_{i+1} = U(x_i,y_i)$.

We will show that all x_i are different and all y_i are

different. Let D_i be $\{y \ : \ R(x_i,y)\}$. From the definition of x_i, y_i D_i and Θ' it follows that

(1) $D_i \supsetneq D_{i+1}$

(2) $y_{i+1} \in D_i \setminus D_{i+1}$

From (1) and (2) and the definition of D_i it follows that all x_i are different and all y_i are different. Hence, Θ' and Θ have no finite model. Note that Θ is satisfiable where h and v are interpreted as the set of naturals and $R(i,j)$ holds if $i \leq j$.

Now we define Φ_3 in the extension of the Ackermann class, which is satisfiable if $\mathcal{T}$ has a solution. Φ_3 is $\forall x : h \exists x' : h \forall y : v \exists y' : v \varphi_3$, where φ_3 is the conjunction of the following quantifier-free formulas:

(1) $\big(R(x',y) \rightarrow R(x,y)\big) \wedge R(x,y') \wedge \neg R(x',y')$ - this conjunct is as Θ above.

(2) $\bigwedge_i \big(T_i(x,y) \Leftrightarrow \neg \bigvee_{j \neq i} T_j(x,y)\big)$ - the pair (x,y) is tiled by exactly one tile.

(3) $\bigwedge_i \big[T_i(x,y) \rightarrow \vee_{\{j \ : \ Adj_h(t_i,t_j)\}} T_j(x',y)\big]$ - tiling respects the horizontal adjacency relation.

(4) $\bigwedge_i \big[T_i(x,y) \rightarrow \vee_{\{j \ : \ Adj_v(t_j,t_i)\}} T_j(x,y')\big]$ - tiling respects the vertical adjacency relation.

We claim that if Φ_3 is satisfiable then $\mathcal{T}$ has a solution.

Let $H \ : \ h \rightarrow h$ and $V \ : \ h \times v \rightarrow v$ be new functional symbols. The Skolem normal form of Φ_3 is the formula

$$\Phi_3' ::= \forall x : h \forall y : v \varphi_3 \{H(x)/x', V(x,y)/y'\}.$$

Φ_3 is satisfiable if Φ_3' is satisfiable. Assume that M is a model of Φ_3'. Define x_i and y_i as in the proof that Θ' has no finite model. The same arguments show that all x_i are different and all y_i are different. Define the tiling of $Nat \times Nat$ as follows: put a tile of type t_k on (i,j) if $T_k(x_i,y_j)$ holds. The second conjunct ensure that every pair (x,y) is tiled by exactly one tile. The third and fourth conjuncts ensure that the tiling respects horizontal and vertical adjacency relations.

Finally, we claim that if $\mathcal{T}$ has a solution then Φ_3 is satisfiable. Indeed, in this case the domains for v and h can be interpreted as Nat; R can be interpreted as "$\leq$" and $T_k(i,j)$ holds if (i,j) is tiled by t_k. $\square$

Corollary 21 (Finite Model Property Fails)**.** Each of the following fragments has a formula that is satisfiable only in infinite structures: $[\exists^* \forall, all, (1)]^{ext}_{=}$, $[FO^2]^{ext}$, and $[\exists^* \forall \exists^*, all]^{ext}_{=}$.

Proof. Suppose, toward contradiction, that one of the fragments has the finite model property. Therefore, the satisfiability problem for this fragment is recursive enumerable. The validity problem is recursively enumerable for the whole many-sorted logic. Hence, the satisfiability problem for this fragment is decidable, and this contradicts Theorem 20. $\square$

It is well known that $[\forall\ \forall\ \exists]_=$ and $[\forall\ \exists\forall]_=$ are undecidable classes for one-sorted first-order logic (see [9]). The following theorem says that for many-sorted first-order logic the only undecidable three quantifier prefix classes are these two one-sorted classes.

The next theorem has some theoretical interest. Unfortunately we have not found any practical use for it. We also do not give a complete classification for many-sorted logic and even the proofs of the following theorems are just direct proofs on the different cases and not a general method.

Theorem 22. *The satisfiability problem is decidable for sentences of the form $Q_1Q_2Q_3\psi$, where ψ is a quantifier-free many-sorted formula with equality without function symbols, and $Q_1Q_2Q_3$ is a quantifier prefix not of the form $[\forall x_1 : A\forall x_2 : A\exists x_3 : A]$ or $[\forall x_1 : A\exists x_2 : A\forall x_3 : A]$ for some sort A.*

Proof. It suffices to prove the decidability of the following quantifier prefixes. The result for the other prefixes follows from the one-sorted logic results.

(1) $[\forall x_1 : B\forall x_2 : A\exists x_3 : A]$
(2) $[\forall x_1 : A\forall x_2 : B\exists x_3 : A]$
(3) $[\forall x_1 : A\forall x_2 : A\exists x_3 : B]$
(4) $[\forall x_1 : A\forall x_2 : B\exists x_3 : C]$
(5) $[\forall x_1 : B\exists x_2 : A\forall x_3 : A]$
(6) $[\forall x_1 : A\exists x_2 : B\forall x_3 : A]$
(7) $[\forall x_1 : A\exists x_2 : B\forall x_3 : C]$
(8) $[\forall x_1 : A\exists x_2 : A\forall x_3 : B]$

The Skolem form of (3)-(7) is in St_0; therefore, the satisfiability problems for (3)-(7) are decidable. The decidability for (8) follows from the fact that if a formula from (8) has a model, then the formula has a model with one element of type *B*. So the satisfiability of $[\forall x_1 : A\exists x_2 : A\forall x_3 : B]$ is equivalent to the satisfiability of $[\forall x_1 : A\exists x_2 : A]$, which is decidable [6]. The decidability for (1) and (2) is similar to (8). □

The following theorem gives a classification of many-sorted first-order logic with four quantifiers except for the case $[\forall x_1 : A\forall y_2 : B\exists x_2 : A\exists y_2 : B]$. It is still an open question whether the satisfiability problem for this fragment is decidable or undecidable.

Theorem 23. *The satisfiability problem for sentences of the form $Q_1Q_2Q_3Q_4\psi$, where ψ is a quantifier-free many-sorted formula with equality without function symbols and Q_i is a quantifier prefix $[\forall x : A]$ or $[\exists x : A]$ for some sort A, is as follows:*

(1) When all quantifiers are over the same sort, this is a problem of one-sorted logic.
(2) When there are three quantifiers with the same type, only the one-sorted cases are undecidable, i.e., there is a pattern like $[\forall\ \forall\ \exists]_=$ or $[\forall\ \exists\forall]_=$ over the same type.
(3) When there are two quantifiers with the same type and two quantifiers with different types, the satisfiability problem is decidable.
(4) When there are two quantifiers with one type and two quantifiers with another type, then
  (a) $[\exists x_2 : A\exists y_2 : B\forall x_1 : A\forall y_1 : B]$ is decidable.
  (b) $[\exists x_2 : A\forall x_1 : A\exists y_2 : B\forall y_1 : B]$ is decidable.
  (c) $[\exists y_2 : B\forall x_1 : A\exists x_2 : A\forall y_1 : B]$ is decidable.
  (d) $[\exists x_2 : A\forall x_1 : A\forall y_1 : B\exists y_2 : B]$ is decidable.
  (e) $[\exists y_2 : B\forall x_1 : A\forall y_1 : B\exists x_2 : A]$ is decidable.
  (f) $[\forall x_1 : A\exists x_2 : A\exists y_2 : B\forall y_1 : B]$ is decidable.

(g) $[\forall x_1 : A \exists y_2 : B \forall y_1 : B \exists x_2 : A]$ *is decidable.*
(h) $[\forall x_1 : A \exists x_2 : A \forall y_1 : B \exists y_2 : B]$ *is undecidable.*

Proof. 1: Is trivial.
2:
Suppose that three quantifiers are over type *A* and one quantifier is over type *B*. There are two cases: (a) the quantifier of type *B* is universal (b) the quantifier of type *B* is existential. If (a), then ψ has a model iff ψ has a model with one element of type *B*. Therefore, the universal quantifier of type *B* can be eliminated. From one-sorted results it is known that a formula with three quantifiers is undecidable iff there is a pattern like $[\forall\, \forall\, \exists]_=$ or $[\forall\, \exists \forall]_=$. If (b), then let $\bar{\psi}$ be the Skolem form of ψ. If $\bar{\psi}$ contains no function from *A* to *A*, then $\bar{\psi}$ is in St_0 and therefore is decidable. If $\bar{\psi}$ does not contain a pattern like $[\forall\, \forall\, \exists]_=$ or $[\forall\, \exists \forall]_=$ over sort *A* but contains a function from *A* to *A*, then there is only one universal quantifier over type *A* and all other quantifiers are existential. Hence, $\bar{\psi}$ is decidable from the one-sorted results.
3:
Suppose that two quantifiers are over type *A* and two quantifiers are over types *B* and *C*. If one of the quantifiers over *B* or *C* is universal, then it can be eliminated, because ψ has a model iff ψ has a model with one element of one of these types. Therefore, without loss of generality, we assume that the two quantifiers over *B* and *C* are existential. If the two quantifiers over *A* are universal, then the Skolem form of ψ is in St_0 and is therefore decidable. If one of the quantifiers over *A* is existential, then ψ is decidable from one-sorted results.
4:
The Skolem forms of 4a and 4b are in St_0 and are therefore decidable.
For case 4c we can eliminate $\forall y_1 : B$ because ψ has a model iff ψ has a model with one element of type *B*. Hence, the decidability follows from the one-sorted results.
Cases 4d, 4e, 4f, and 4g are similar to case 4c. In Theorem 20, we already proved that case 4h is undecidable . □

5. Related Works

Lee Momtahan [14] proves the finite model property and decidability for a language that is a subset of Alloy. This language deals only with quantifies-free formulas. Even the birthday book example cannot be fully formalized in this language.

Fontaine and Gribomont [8] introduced a quantifier elimination procedure based on an enhanced Herbrand Theorem. Their results imply the decidability of a fragment of many-sorted logic which is similar to our St_0. Moreover, they proved that this fragment is decidable even in the case when for a type at level zero an interpretation is provided by a structure with decidable quantifier-free theory. They succeeded in formalizing in this fragment generalized railroad crossing and parameterized Burns algorithm.

Shuvendu Lahiri and Shaz Qadeer [13] introduce a new logic interpreted over a finite partially ordered set *D* of sorts. The aim of [13] and our work is to find decidable logics useful for verification. Both the logic of [13] and the logics considered in our work use a stratified vocabulary. Shuvendu Lahiri and Shaz Qadeer [13] logic uses the pre-image of functions, while we use the image of functions. We can easily translate the pre-image into our logic.

The formula $\forall x \in f^{-1}(t).\psi(x)$ with the pre-image f^{-1} is equivalent to the formula $\forall x : T.f(x) = t \Rightarrow \psi(x)$ without any pre-image. Hence, the pre-image can be eliminated without complicating the quantifier structure of the formula. However the elimination of the image is not

so easy. The formula $\forall x : T.x \in Im[f]$ can be translated to an equivalent formula $\forall x : T \exists y : T'.f(y) = x$. However, the translation is not in the fragment considered at [13] for two reasons: (1) formulas in [13] do not contains alternation of quantifiers (2) they use only bonded quantifiers of the form $\forall x \in S$, where S is a set term in their language. Moreover, it is impossible to express unbounded quantifier $\forall x : T.\alpha$ in their logic. Some additional differences follow:

(1) In our logic, all types are uninterpreted. In [13], logic allows using the type of *Nat* with the standard interpretation for $<$ and $+$.
(2) In [13] there are no relation symbols, except $<$. Because of the restriction of using only the bonded quantifiers, it seems that they cannot simulate relations. We have not even succeeded in formulating our simplest example, Birthday, in their logic.
(3) The transitive closure is central to the logic of [13]. Our use of the transitive closure is an adaptation of other results to a typed fragment.
(4) The complexity of the satisfiability problem for the logic considered in [13] is in *NP*. We have not analyzed the complexity of our fragments. For $St_1 \Rightarrow Un$ formulas, we can show that if a formula has a counter-model then it has a counter-model of the exponential size. Therefore, the non-validity problem is in *NEXPTIME*. However, for $St_2 \Rightarrow Un$, it is impossible to provide a complexity bound, because it uses a semantical requirement.

6. Conclusion

In this paper we initiated a systematic study of fragments of many-sorted logic, which are decidable, have the finite model property and have potential for practical use. To the best of our knowledge, the idea of looking at this problem in a systematic way has not been explored previously, despite the well-known complete classification in the one-sorted case, presented in the book by Boerger, Graedel, and Gurevich [6].

We presented a number of decidable fragments of many-sorted first-order logic. The first one, St_0, is based on a stratified vocabulary. The stratification property guarantees that only a finite number of terms can be built with a given finite set of variables. As a result, the Herbrand universe is finite and the small model property holds. Moreover, a stronger property named "the satisfiability with a finite extension property" holds.

Subsequently, we extended the class St_0 to class St_1 and then to St_2, and proved that these classes also have the satisfiability with a finite extension property (and therefore, the finite model property). The added expressive power of St_2 is the ability to test whether an element is in the image of a function. Even though this particular extension may seem less natural from a syntactic viewpoint, it is very useful in many formalizations.

We provided sufficient semantical conditions for decidability. As a consequence, we obtained that for sentences of the form $\psi \Rightarrow \varphi$, where $\psi \in St_2$ and φ is universal, the validity problem is decidable. To illustrate the usefulness of the fragment, we formalized in it many examples from Alloy.

We extended our results to logic that allows restricted use of the transitive closure. We succeeded in formalizing some of Alloy specifications by formulas of this logic; however, the vast majority of Alloy examples that contain the transitive closure are not covered by this fragment. Future work is needed to evaluate its usefulness and to find its decidable extensions.

Finally, we looked at classes corresponding to decidable classes (or classes with the finite-model property) of first-order logic. We observed that just requiring the decidability of projections of the quantifier prefix for each type individually is not a sufficient condition for the decidability (respectively, the finite-model property). Future work is needed to carry out complete classification for many-sorted logic.

We plan to consider a less restricted use of the transitive closure which plays a very important role in numerous practical specifications. Another topic to be considered is the extension of our result to cases where some of the types and functions are interpreted. The third direction is evaluating the practical usefulness of our methods. Our decidability results do not provide concrete complexity bounds. We have not yet implemented decision procedures for our decidable classes.

Acknowledgements

We thank Tal Lev-Ami and Greta Yorsh for their insightful comments. We are also grateful to the anonymous referees for pointing out [8, 13] and for their numerous suggestions that improved this paper.

References

[1] A. Abadi, A. Rabinovich and M. Sagiv. Decidable Fragments of Many-Sorted Logic. In LPAR 2007, Springer LNCS 4790, 17-31, 2007.

[2] D. Beauquier and A. Slissenko. Decidable verification for reducible timed automata specified in a first order logic with time. *Theoretical Computer Science*, 275:347–388, 2002.

[3] D. Beauquier and A. Slissenko. A first order logic for specification of timed algorithms: Basic properties and a decidable class. *Annals of Pure and Applied Logic*, 113:13–52, 2002.

[4] E. Boerger, E. Graedel, and Y. Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1996.

[5] G. Boolos and R. Jeffrey. *Computability and Logic*. Cambridge University Press, 2nd edition, 1980.

[6] E. Borger, E. Gradel, and Y. Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[7] C. C. Chang and H. Jerome Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Netherlands, 1973.

[8] P. Fontaine and E. P. Gribomont. Decidability of invariant validation for parameterized systems. Tools and Algorithms for Construction and Analysis of Systems (TACAS). pages 97–112, 2003. Lecture Notes in Computer Science. Springer-Verlag.

[9] W.D. Goldfarb. The unsolvability of the godel class with identity. *The Journal of Symbolic Logic*, 49 , Number 4:1237–1252, 1984.

[10] N. Immerman, A. Rabinovich, T. Reps, M. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *Computer Science Logic (CSL*, pages 160–174, 2004.

[11] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng . Methodol*, 11(2):256–290, 2002.

[12] D. Jackson. Micromodels of software:lightweight modelling and analysis with alloy. Technical report, MIT Lab for Computer Science, 2002.

[13] Shuvendu K. Lahiri and Shaz Qadeer. Back to the future: revisiting precise program verification using smt solvers. In *POPL*, pages 171–182, 2008.

[14] Lee Momtahan. Towards a small model theorem for data independent systems in alloy. *Electronic Notes in Theoretical Computer Science*, 128(6):37–52, May 2005.

[15] M. Mortimer. On languages with two variables. *Zeitschr.f.math.Logik u.Grundlagen d.Math*, pages 135–140, 1975.

[16] J.M. Spivey. *The Z notation:a reference manual*. Prentice-Hall, 1992.
[17] B. A. Trakhtenbrot. The impossibility of an algorithm for the decidability problem on finite classes. *Doklady AN SSR*, 70(4):569–572, 1950.