Decidable Fragments of Many-Sorted Logic

Aharon Abadi, Alexander Rabinovich, and Mooly Sagiv

School of Computer Science, Tel-Aviv University, Israel
{aharon,rabinoa,msagiv}@post.tau.ac.il

Abstract. We investigate the possibility of developing a decidable logic which allows expressing a large variety of real world specifications. The idea is to define a decidable subset of many-sorted (typed) first- order logic. The motivation is that types simplify the complexity of mixed quantifiers when they quantify over different types. We noticed that many real world verification problems can be formalized by quantifying over different types in such a way that the relations between types remain simple.

Our main result is a decidable fragment of many-sorted first-order logic that captures many real world specifications.

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 many-sorted first-order logic and (ii) developing mechanisms for proving validity of formulas in that logic over finite models (which is actually harder than validity over arbitrary 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 [10,11]. The Alloy Analyzer is similar to a bounded model checker [8], which means that every reported error is real, but Alloy can miss errors (i.e., produce false positives). Indeed, the Alloy tool performs an under-approximation of the set of reachable states, and is designed for falsifying rather than verifying properties.

Main Results. This paper investigates the applicability of first-order tools to reason about Alloy specifications. It is motivated by our initial experience with employing off-the-shelf resolution-based first-order provers to prove properties of formulas in many-sorted first-order logic.

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 resolution-based theorem prover. The rest of this subsection elaborates on these results.

Motivation: Employing Ordered Resolution-Based Theorem Provers
Ordered Resolution-Based First-Order Systems such as `SPASS` [16] and `Vampire` [14] have been shown to be quite successful in proving first-order theorems. Ordering dramatically improves the performance of a prover and in some cases can even guarantee decidability.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 17–31, 2007.

As a motivating experience reported in [3], we converted Alloy specifications into formulas in first-order logic with transitive closure. We then conservatively modeled transitive closure via sound first-order axioms similar to the ones in [12]. The result is that every theorem proved by SPASS about the Alloy specification is valid over finite models, but the first-order theorem prover may fail due to: (i) timeout in the inference rules, (ii) infinite models which violate the specification, and (iii) models that violate the specifications and the transitive closure requirement.

Encouragingly, SPASS was able to prove 8 out of the 12 Alloy examples tried without any changes or user intervention. Our initial study indicates that in many of the examples SPASS failed due to the use of transitive closure and the fact that SPASS considers infinite models that violate the specifications. It is interesting to note that SPASS was significantly faster than Alloy when the scope exceeded 7 elements in each type.

Adding Types. Motivated by our success with SPASS we investigated the possibility of developing a decidable logic which allows to express many of the Alloy examples. The idea is to define a decidable subset of first-order logic. Since Alloy specifications include different types, natural specifications use many-sorted first-order logic.

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. In [7] the complete classification of fragments with decidable validity problem and fragments with finite model property according to quantifier prefixes and vocabulary is provided. 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, finite model property fails for the formulas with the quantifier prefix $\forall\forall\exists$ and equality. Sorts 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 functions 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 quantifier prefix $\forall x : A \forall y : A \exists z : B$ have the finite-model property. Usually, like in the above example, the inclusion of sorts simplifies the verification task.

Our Contribution. The main technical contribution of this paper is identification of a fragment of many-sorted logic which is (1) decidable (2) useful — can formalize many of the Alloy examples that do not contain transitive closure and (3) has a finite counter-model property which guarantees that a formula has a counter-model iff it has a finite counter-model (equivalently 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 finite model property.

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 by formulas in these fragments. In Section 3 we prove that our fragments are decidable for validity over

finite models. In Section 4, we investigate ways of generalizing decidable fragments from first-order logic to many-sorted logic.

The reader is referred to [3] for proofs, more examples of formalizing interesting properties using decidable logic, extensions for transitive closure, and a report on our experience with SPASS.

2 Three Fragments of Many-Sorted First-Order Logic

Safety properties of programs/systems can be usually formalized by universal sentences. The task of the verification that a program P satisfies a property θ can be reduced to the validity problem for sentences of the form $\psi \Rightarrow \theta$, where sentence ψ formulate the behavior of P.

In this section we introduce three fragments St_0, St_1 and St_2 of many-sorted logic for description of the behavior of programs and systems. The validity (and validity over the finite models) problems for formulas of the form $\psi \Rightarrow \theta$, where $\psi \in St_i$ and θ is universal are decidable. This allows us to prove that a given program/system satisfy a property expressed as a universal formula.

St_0 is a natural fragment of the universal formulas which has 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 finite extension* property which we introduce in Section 3. This property implies that the validity problem over finite models for the sentences of the form $\psi \Rightarrow \theta$ where $\psi \in St_0$ and θ is universal, is decidable. In Section 2.3 we formalize birthday book example in St_0.

Motivated by examples from Alloy we introduce in Section 2.1, a more expressive (though less natural) set of formulas St_1. The St_1 formulas also have satisfiability with finite extension property, and therefore might be suitable for automatic verification of safety properties. The behavior of many specifications from [1] can be formalized by 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 finite extension property. All except one specifications from [1] which do not use transitive closure can be formalized by formulas of the form $\psi \Rightarrow \theta$, where $\psi \in St_2$ and θ are universal.

2.1 St_0 Class

In this subsection we will 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 $f : A_1 \times \ldots \times A_m \to B$ $level(B) < level(A_i)$ 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 finitely many terms over Σ with the variables in V.

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 Herbrand model over St_0 vocabulary. We will 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 new atomic formula $x \in Im[f]$, where f is a function symbol. The formula $x \in Im[f]$ is a shorthand for $\exists y_1 : A_1 \ldots \exists y_n : A_n \ (x = f(y_1, \ldots, y_n))$.

This is formalized below.

St_1 Vocabulary. Contains predicates, function symbols, equality symbol and atomic formulas $x \in Im[f]$ where f is a function symbol.

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

The semantics is as in 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 will prove that St_1 has satisfiability with finite extension property which generalize finite model property.

2.3 Examples

Most of our examples come from Alloy [10,11][1]. The vast majority of Alloy examples include transitive closure, and thus cannot be formalized in our logic. We examined eight Alloy specifications without transitive closure and seven of them fit into our logic. This is illustrated by the birthday book example. The second example is a Railway Safety specification. This example cannot be formalized by formulas in St_1. However, it fits in St_2 which is an extension of St_1, and will be described in Section 2.4.

Birthday Book. Table 1 is used to model a simple Birthday book program[2]. A birthday book has two fields: $known$, a set of names (of persons whose birthdays are known), and $date$, set of triples (birthday book, person, the birthday date of that person). The operation $getDate$ gets the birthday date for a given birthday book and person. The operation $AddBirthday$ adds an association between a name and a date. The assertion $Assert$ checks that if you add an entry and then look it up, you get back what you just entered.

The specification assertion has the form $\psi \Rightarrow \theta$ where $\psi \in St_0$ 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)* $= 0$.

Railway Safety Example. A policy for controlling the motion of trains in a railway system is analyzed. Gates are placed on track segments to prevent trains from colliding. We need a criterion to determine when gates should be closed. In [4] a different

[1] For details, see [1].

[2] The example originates in [15] and the translation to Alloy is given as an example in the Alloy distribution found at http://alloy.mit.edu

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

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

Railroad crossing problem is formalized, where the time is treated as continuous time, while we use a discrete time. The Alloy formalization in [1,2] differs slightly from our formalization, however both of them represent the same specification. In our formalization the type *Movers* and the relation *moving* were added to represent sets of moving trains. In addition some of the relations and the functions have suffix *_current* or *_next* to represent interpretation at the current and the next period. 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.

Formulas Description

- $safe\_current$ 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 in which gate conditions it is legal for a set of trains to move.
- $trainMove$ is a physical constraint: a driver may not choose to cross from one segment into another segment which is not connected to it. 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 the previous $current$ time. The second ensures that the trains that do not move 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

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

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 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 $\forall s_1, s_2 : Segment\ \ Overlaps(s_1, s_2) \Leftrightarrow Overlaps(s_2, s_1)$ $\forall s : Segment\ \ Overlaps(s, s)$

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 will result in a state that is also safe. In other words, safety is preserved.

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

The specification contains functions $getSegment\_current : Train \rightarrow Segment$ and $getSegment\_next : Train \rightarrow Segment$ where $getSegment\_current$ participates in formula $x \epsilon Im[getSegment\_current]$ in contrast to our requirements from St_1 formulas.

Table 3. Formulas and assert used in the train example

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*

2.4 St_2 Class

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

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

For every function symbol $g : \bar{A}_1 \times \ldots \times \bar{A}_{\bar{k}} \rightarrow B$:

(*) $\forall a_1 : A_1, \ldots, \forall a_k : A_k\ \forall \bar{a}_1 : \bar{A}_1, \ldots, \forall \bar{a}_{\bar{k}} : \bar{A}_{\bar{k}}\ f(a_1, \ldots, a_k) = g(\bar{a}_2, \ldots, \bar{a}_{\bar{k}}) \Rightarrow$
$k = \bar{k} \wedge a_1 = \bar{a}_2 \wedge \ldots \wedge a_k = \bar{a}_{\bar{k}}$.

Notice 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 which 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 can notice that the specification contains functions $getSegment\_current : Train \rightarrow Segment$ and $getSegment\_next : Train \rightarrow Segment$. 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*$]$

participates in the formula. Let *M* be model such that $M \models$ *Assert_Train*. It 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 $t_1 \neq t_2$ 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 next time. But $M \models$ *safe_current* so $s \neq$ *getSegment_current*(t_2). From this and from the fact that no train comes to s at next time 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 will be denoted by UN. The main results of this section is 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 in [5,6] we provide sufficient semantical conditions for decidability of 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 $D'_{s_1}, \ldots, D'_{s_n}$.*
- *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 $(f_s^i)^{M'} : D'_{s_1} \times \ldots \times D'_{s_n} \rightarrow D'_s$ is total.

The following definition strengthens the notion of 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 finite extension definition was inspired by (but is quite different from) the definition of *C-satisfiable* with augmentation for complexity (k, n) in [5,6].

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 every 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 :

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

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 t_2 occupies. 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' sub model 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 still is a counter-model.

In the rest of this section we will 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 will 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 will complete 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 finite extension and φ is finitely refutable. Then $\neg\psi$ has the finite model property.*

Notice 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 which are finitely refutable and let $\mathcal{F}_{sat-fin-ext}$ be a set of sentences in many-sorted first-order logic which are satisfiable with 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. By lemma 7 if a sentence in this class is not valid then it has a finite counter-model. Hence, in order to check whether a sentence φ in this class is valid we can start (1) to enumerate proofs looking for a proof of φ and (2) to enumerate all finite models looking for a counter-model for φ. Either (1) or (2) will succeed. If (1) succeeds then φ is valid, if (2) succeeds φ it 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 which ensure decidability of validity problem for this class. Unfortunately, these semantical conditions are undecidable.

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

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

In the next two subsections we describe syntactical conditions which ensure

(1) finitely refutable property.
(2) satisfiability with finite extension.

3.3 Syntactical Conditions for Decidability

The proof of the following lemma uses the preservation theorem from first-order logic, which says that a sentence ψ is equivalent to 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 formula ψ is k-refutable for some k iff ψ is equivalent to a universal formula.*

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 decidable validity problem.

The next theorem is our main technical theorem.

Theorem 11. *Let ψ be a formula in St_2 then ψ is satisfiable with finite extension.*

Proof (Sketch)
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 level of types in Σ are in the set $\{0, \ldots, m\}$.

Let $M_0 = M'$ and 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 $M \models b \in Im[f]$, however, 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 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 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 $\bar{a'} \in M$ such that $M \models f(\bar{a'}) = b$.

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 $M \models f(\bar{a}) = b$.

Next, let M'' be defined 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 $\bar{M}$ follows that $\bar{M}$ is a submodel of M. Moreover, it is not difficult to show using the semantic requirement (*) , that the interpretations of $Im[f]$ in M and in $\bar{M}$ agree, i.e. for every $b \in Dom(\bar{M})$, $M \models b \in Im[f]$ iff $\bar{M} \models b \in Im[f]$. □

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

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 which have the finite-model property. We try to find a way to extend these classes to many-sorted logic.

We use the notation from [7]. According to [7] the following classes have the finite model property:

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

- $[\exists^*\forall\exists^*, all]_=$ (Ackermann 1928) the class of all sentences with quantifier prefix $\exists^*\forall\exists^*$ over arbitrary relational vocabulary with equality.
- $[\exists^*, all, all]_=$ (Gurevich 1976) the class of all sentences with quantifier prefix $\exists^*$ over arbitrary vocabulary with equality.
- $[\exists^*\forall, all, (1)]_=$ (Grädel 1996) the class of all sentences with quantifier prefix $\exists^*\forall$ over vocabulary that contain unary function and arbitrary predicate symbols with equality.
- FO^2 (Mortimer 1975) [13] the class of all sentences of relational vocabulary that contain two variables and equality.

Below we describe a generic natural 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 $Q_1 \dots Q_m$ 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 decidability nor 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 12 (Projection of a Formula Onto 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 which 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 $P_{i_1,i_2,\dots,i_m}$ with an 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 $f_{i_1,i_2,\dots,i_m}$ with an 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 13 (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 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 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 fragment of St_0 and has the finite model property and thus is decidable. It is easy to prove that the naive extension of Gurevich class is decidable. The next two theorems state that the naive extensions of Ackermann, Grädel and Mortimer classes do not have the finite model property and, even more disappointing, are undecidable.

Theorem 14 (Finite Model Property Fails). *Each of the following fragments has a formula which is satisfiable only in infinite structures:* $[\exists^*\forall, all, (1)]^{ext}_{=}$, $[FO^2]^{ext}$ *and* $[\exists^*\forall\exists^*, all]^{ext}_{=}$.

Proof: see [3].

Theorem 15 (Undecidability). *The satisfiability problem is undecidable for each of the following fragments:* $[\exists^*\forall, all, (1)]^{ext}_{=}$, $[FO^2]^{ext}$ *and* $[\exists^*\forall\exists^*, all]^{ext}_{=}$.

Our proof of the above theorem (see [3]) provides formalization of two register machine and is similar to the proofs in [7].

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.

Although we did not found any practical use for this result we think that it has some theoretical interest.

Theorem 16. *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 functions 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: see [3] .

5 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 a potential for practical use. To 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 [7]).

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 of satisfiability with finite extension 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 finite extension property (and therefore, the finite model property). The added expressive power in 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 semantical sufficient conditions for decidability. As a consequence, we obtained that for the sentences of the form $\psi \Rightarrow \varphi$, where $\psi \in St_2$ and φ is universal, the validity problem is decidable. In order to illustrate the usefulness of the fragment, we formalized in it many examples from [1] - the Alloy finite model finder.

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 onto each type individually is not a sufficient condition for the decidability (respectively, the finite-model property) in general. Future work is needed in order to carry out a complete classification for the many-sorted logic.

In [3] we extended our results to a logic which allows a 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.

Acknowledgments

We are grateful to the anonymous referees for their suggestions and remarks. We also thank Tal Lev-Ami and Greta Yorsh for their comments.

References

1. The alloy analyzer home page: `http://alloy.mit.edu`
2. `http://alloy.mit.edu/case-studies.php`
3. Abadi, A.: Decidable fragments of many-sorted logic. Master's thesis, Tel-Aviv University (2007)
4. Beauquier, D., Slissenko, A.: Verification of timed algorithms: Gurevich abstract state machines versus first order timed logic. In: Proc. of ASM 2000 International Workshop (March 2000)

5. Beauquier, D., Slissenko, A.: Decidable verification for reducible timed automata specified in a first order logic with time. Theoretical Computer Science 275, 347–388 (2002)
6. Beauquier, D., Slissenko, A.: 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)
7. Borger, E., Gradel, E., Gurevich, Y.: The Classical Decision Problem. Springer, Heidelberg (1997)
8. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design 19(1), 7–34 (2001)
9. Goldfarb, W.D.: The unsolvability of the godel class with identity. The Journal of Symbolic Logic 49(4), 1237–1252 (1984)
10. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. 11(2), 256–290 (2002)
11. Jackson, D.: Micromodels of software:lightweight modelling and analysis with alloy. Technical report, MIT Lab for Computer Science (2002)
12. Lev-Ami, T., Immerman, N., Reps, T.W., Sagiv, M., Srivastava, S., Yorsh, G.: Simulating reachability using first-order logic with applications to verification of linked data structures. In: CADE, pp. 99–115 (2005)
13. Mortimer, M.: On languages with two variables. Zeitschr. f. math. Logik u. Grundlagen d. Math., 135–140 (1975)
14. Riazanov, A., Voronkov, A.: The design and implementation of vampire. AI Communications 15(2-3), 91–110 (2002)
15. Spivey, J.M.: The Z notation: a reference manual. Prentice-Hall, Englewood Cliffs (1992)
16. Weidenbach, C., Gaede, B., Rock, G.: Spass & flotter version 0.42. In: CADE-13. Proceedings of the 13th International Conference on Automated Deduction, pp. 141–145. Springer, Heidelberg (1996)