First-Order Transitive Closure Axiomatization via Iterative Invariant Injections

Aboubakr Achraf El Ghazi, Mana Taghdiri, and Mihai Herda

Karlsruhe Institute of Technology, Germany
{elghazi,mana.taghdiri,herda}@kit.edu

Abstract. This paper presents an approach for proving the validity of first-order relational formulas that involve transitive closure. Given a formula F that includes the transitive closure of a relation R, our approach can deduce a complete (pure) first-order axiomatization of the paths of R that occur in F. Such axiomatization enables full automated verification of F using an automatic theorem prover like Z3. This is done via an iterative detection and injection of R-invariants —invariant formulas with respect to R-transitions in the context of F. This paper presents a proof for the correctness of the approach, and reports on its application to non-trivial Alloy benchmarks.

Keywords: First-order relational logic, Transitive closure, Axiomatization, Specification, Verification, Alloy, SMT solving.

1 Introduction

Many computational problems, especially those that encode manipulations of linked data structures, can be efficiently specified in a relational first-order logic. Alloy [16] is a popular such language that has been successfully used for specifying and checking several different systems for different purposes at both the design and implementation level (see e.g. [5, 17, 24, 26]). The transitive closure (TC) operator is a crucial, powerful tool for encoding structure-rich systems.

While the Alloy Analyzer can efficiently check Alloy specifications within a bounded scope, their full verification is a long-standing challenge. This is especially due to the known difficulty of reasoning about transitive closure —adding transitive closure even to very tame logics makes them undecidable [15]. Due to this undecidability, most attempts to verify the correctness of Alloy specifications are based on *interactive* theorem proving (e.g. [1, 12, 25]).

In our previous work [8], we proposed to verify Alloy specifications by first trying a fully automatic proof engine before switching to interactive theorem proving. The automatic engine, described in [9], relies on an efficient, semi-satisfiable[1] translation of Alloy into the satisfiability modulo theories (SMT) language, and is able to automatically provide (full) proofs of some non-trivial Alloy specifications. Given the increasing capability of SMT solvers in handling

[1] If the result is unsatisfiable, the input is unsatisfiable too.

quantifiers [4, 13, 14], the success of our automatic engine is not surprising at a first glance. However, given the fact that some of the proofs required transitive closure which causes undecidability, the success was thought-provoking. Since transitive closure is not expressible in pure first-order logic, in [9], we used an integer-based axiomatization. Although such axiomatization guaranties TC-satisfiability —any model of the axiomatisation is a TC model— it is not generally sufficient for refuting proof obligations; one still needs the integer induction (IND) principle. The latter claim roots in our experiments with Kelloy [25], an interactive proof engine for Alloy specifications. We observed that most specifications that require the transitive closure theory, need the IND principle and have to be proven manually.

In this paper, we investigate the following questions: (1) when can the integer based axiomatization of TC refute proof obligations without requiring the IND principle? (2) for the logic fragment of question 1, can one use an integer-free axiomatization? and (3) to refute a proof obligation outside the fragment of question 1, what kind of integer-free axiomatization can be used?

Let F be a refutable first-order relational formula in which the semantics of all symbols except for transitive closure are precisely encoded, and the transitive closure of a relation R is encoded by an uninterpreted binary relation tc_R. To answer questions (1) and (2), we use a pure first-order, weak axiomatization (WTC) which constrains tc_R to a transitive relation containing R but not the smallest one. We prove that WTC is complete for any *negative* transitive closure occurrence in the clause normal form (CNF) of F. Therefore, if the solver (falsely) reports F as satisfiable modulo WTC, it is only because of *positive* transitive closure occurrences. To extend the WTC fragment and to answer question (3), we introduce a technique which automatically detects relevant invariants about the paths of tc_R and adds them as additional assumptions to F. If any of such invariants cause contradiction, F has been refuted and the process stops. Otherwise, more invariants will be detected and added to F.

We applied our technique to a total of 20 Alloy benchmarks known to be valid, all of which require the transitive closure theory for their proof. Out of the 20 benchmarks 18 were successfully proven correct fully automatically using our invariant detection technique.

2 Background

Let $\Sigma = (\mathit{Typ}, \sqsubseteq, \mathit{Fun}, \alpha)$ be a typed *signature* consisting of a set of type symbols Typ that always includes boolean (Bool) and integer (Int) types, a partial order $\sqsubseteq$ over Typ, a set of function symbols Fun[2], and a typing function $\alpha : \mathit{Fun} \to \mathit{Typ}^+$ that gives the type of each function symbol, i.e. $\alpha(f) = (T_1, \ldots, T_n)$ iff $f : T_1 \times \cdots \times T_{n-1} \to T_n$. If $T_n = \mathit{Bool}$, function f can also be called a predicate and the notation $f \subseteq T_1 \times \cdots \times T_{n-1}$ can be used alternatively. We assume that

[2] Certain interpreted functions (e.g. equality and logical connectives) are always included and denoted in infix notation.

for each type $T \in \mathit{Typ}$, there exists an infinite set of variables of type T denoted by Var^T. Variable sets are mutually disjoint and their union is denoted by Var.

*Term*s of the logic are built recursively from variables in Var, function symbols in Fun and quantifiers. The type of a term is determined recursively based on the types of its variables and functions. Function symbols with arity 0 are constants, and denoted by the set Con. Terms without variables are ground terms, and denoted by Gr. The notation $t[x_1, \ldots, x_n]$ denotes that the variables $x_1, \ldots, x_n$ (for short $x_{1:n}$) occur in t. For two terms t and s and a variable x, the notation $t[s/x]$ denotes the result of substituting s for x in t. We use $t[S]$ for applying a set of substitutions S when the substitution order does not matter.

Formulas are boolean terms. A formula is *atomic* if it is a function application, i.e. $f(t_{1:n})$, and the top-level function f is not a logical connective. A *literal* is an atomic formula or its negation. A *clause* is a disjunction of literals. A formula F is in *clause normal form* (CNF) if it is a conjunction of clauses $C_1 \wedge \cdots \wedge C_n$, where each C_i is quantifier-free and its variables are implicitly universally quantified. We view CNF formulas as sets of clauses and clauses as sets of literals.

We extend our first-order logic to a relational one by introducing two distinct types Rel and Tuple, and a binary predicate $\in \subseteq \mathit{Tuple} \times \mathit{Rel}$ that represent relations, tuples, and the membership predicate respectively. We use $ar : \mathit{Rel} \cup \mathit{Tuple} \to \mathbb{N}$ to denote the arity of relations and tuples. Unary relations are called sets and unary tuples are called atoms. We use $R \subseteq T_1 \times \cdots \times T_n$ to denote that R is an n-ary relation, and that the i^{th} element of every tuple in R is of type T_i.

Constant relations form basic relational expressions[3]. Complex relational expressions are built using the usual relational operators. Particularly, the join $R \mathbin{.} S$ of two relational expressions R and S is a relational expression Q of arity $n := ar(R) + ar(S) - 2$, such that $(a_{1:n}) \in Q$ iff there exists an atom a where $(a_{1:ar(R)-1}, a) \in R$ and $(a, a_{ar(R):n}) \in S$; the transitive closure R^+ of a binary relation R is the smallest transitive relation containing R; the restriction $R|_S$ of a relational expression R and a set of tuples S of arity $ar(R)$ is a relational expression Q such that $Q = R \cap S$. For a binary relation R, an *R-path* is a sequence of atoms $a_{1:n}$ where $n \geq 2$ and $(a_i, a_{i+1}) \in R$ for $1 \leq i < n$. An R-path of length $n = 2$ is an *R-step*. An R-path may not list the intermediate atoms explicitly. That is, $(a, b) \in R^+$ is also an R-path. For an R-path p we use p_u to denote its tuple and p_s and p_e to denote its start and end boundaries respectively.

Let $|M|^T$ denote the universe of all semantical values of a type $T \in \mathit{Typ}$. A model $\mathcal{M}$ for a signature Σ is a pair $(|M|, M)$ where $|M|$ is a class of universes defined as $\{|M|^T \mid T \in \mathit{Typ}\}$ such that if $T_i \sqsubseteq T_j$, then $|M|^{T_i} \subseteq |M|^{T_j}$, and M is an interpretation that maps every function $f : T_1 \times .. \times T_{n-1} \to T_n$ to an interpretation $M(f) : |M|^{T_1} \times \ldots \times |M|^{T_{n-1}} \to |M|^{T_n}$ and every variable $x : T$ to a value in $|M|^T$. For a relation R, we write $M(R)$ as a shorthand for $\{u \mid M(\in (u, R))\}$. By default, $|M|^{\mathit{Bool}} = \{\mathit{true}, \mathit{false}\}$ and $|M|^{\mathit{Int}} = \mathbb{Z}$. The interpretation $M(t)$ of a term t is defined recursively using the rule $M(f(t_{1:n})) = M(f)(M(t_1), \ldots, M(t_n))$ for a function symbol f. Satisfaction, denoted by $\mathcal{M} \models \varphi$, for a model $\mathcal{M}$ and a formula φ is defined as usual (see elsewhere [11, p. 80]).

[3] Conventionally, relational terms are called relational expressions.

F :
(1) $h_1 \,.\, mark = \emptyset$
(2) $h_0 \,.\, ref \subseteq h_1 \,.\, ref$
(3) $\forall n. \neg((root, n) \in tc_{H.ref}(h_1)) \vee n \in h_2 \,.\, mark$
(4) $h_1 \,.\, ref \subseteq h_2 \,.\, ref$
(5) $\forall n. \neg(n \notin h_2 \,.\, mark) \vee n \,.\, (h_3 \,.\, ref) = \emptyset$
(6) $\forall n. \neg(n \in h_2 \,.\, mark) \vee n \,.\, (h_3 \,.\, ref) = n \,.\, (h_2 \,.\, ref)$
(7) $(root, live) \in tc_{H.ref}(h_0)$
(8) $live \,.\, (h_0 \,.\, ref) \not\subseteq live \,.\, (h_3 \,.\, ref)$
WTC : (9) $\forall h. h \,.\, ref \subseteq tc_{H.ref}(h)$
(10) $\forall h. Transitive(tc_{H.ref}(h))$

(a)

Essential R-path p:
$(root, live) \in tc_{H.ref}(h_0)$
Path invariant for p:
$n \in h_2 \,.\, mark$

(b)

F' :
$F \wedge WTC \wedge$
$\forall x.\ (root, x) \in tc_{H.ref}(h_0) \rightarrow x \in h_2 \,.\, mark$

(c)

Fig. 1: Example. (a) Original formula and a weak transitive closure theory, (b) a difficult R-path in F and its invariant, (c) augmented formula.

A theory $\mathcal{T}$ is a set of deductively closed formulas. A class of models ω induces a theory $Th(\omega)$, namely the theory of all formulas φ where $\mathcal{M} \models \varphi$ and $\mathcal{M} \in \omega$ —the resulting set is deductively closed by definition. We use $\mathcal{T}^g_f$ to denote the theory which agrees with $\mathcal{T}$ except for the interpretations $\mathcal{M}$ of f, where it is interpreted the same way as $M(g)$. A $\mathcal{T}$-model $\mathcal{M}$ is a model that satisfies all formulas of $\mathcal{T}$. Especially, a formula φ is satisfiable modulo a theory $\mathcal{T}$ if there exists a $\mathcal{T}$-model $\mathcal{M}$ where $\mathcal{M} \models \varphi$, for short $\mathcal{M} \models_{\mathcal{T}} \varphi$.

Let Ax be the finite set of axioms for all the interpreted symbols of *Fun* except transitive closure. Only sub-theories of the theory built by the deductive closure $Cl(Ax)$ of Ax are considered here. For the transitive closure R^+, we introduce a fresh uninterpreted binary relation tc_R.

3 Example

Figure 1(a) gives a relational first-order formula F in CNF form —lines correspond to clauses. Symbols h_0 to h_3 are constants of type H that represents the system state; *root* and *live* are two constants of type Obj that represents objects; $mark \subseteq H \times Obj$ represents the marked objects in each state; $ref \subseteq H \times Obj \times Obj$ represents references between objects in each state; and $tc_{H.ref} : H \rightarrow Obj \times Obj$ is a function that maps each state h to a binary relation $tc_{H.ref}(h) \subseteq Obj \times Obj$ which aims at representing the transitive closure of the relation $h \,.\, ref$. The last two lines (WTC) give a weak semantics for $tc_{H.ref}(h)$. They constrain it to be transitive and to include the base relation, but not necessarily the smallest such relation. F gives the negated proof obligation of a safety property of an extremely simplified version of mark-and-sweep algorithm. The state transition (h_0–h_1) resets all the marks (Lines 1-2), (h_1–h_2) marks objects reachable from *root* (Lines 3-4), and (h_2–h_3) sweeps references of non-marked objects (Lines 5-6). The safety property is negated, thus it checks if in the final state, there is a *live* object that was originally reachable from *root* in the beginning state (Line 7), but some of its references have been swept (Line 8).

In our previous work [25], we solved such formulas by adding general axioms about transitive closure. Here, for example, F can be refuted using the subset

preservation axiom, namely $R \subseteq S \rightarrow R^+ \subseteq S^+$ for binary relations R and S. The only state transition in F that allows for sweeping object references is (h_2–h_3) —Line 5. Since (5) is guarded by the condition that the objects are not marked at h_2, to refute the formula, it is sufficient to show that all live objects are marked at h_2. Applying the above axiom to Lines 2, 4 and using Line 7, we have $(root, live) \in tc_{H.ref}(h_2)$ and can easily close the proof. In [25] we collected more than 100 such transitive closure axioms, proved and added them as further deduction rules. Although the approach was useful for interactive and semi-interactive solving, the results of [3] suggest that this approach does not scale for automatic provers such as SMT solvers. [3] proposes to add these lemmas only on-demand based on some heuristics. In this paper, we go one step further and detect and add only the actually needed properties on-the-fly (as opposed to always include some general properties).

Our new approach refutes F by first solving $F \wedge WTC$ using an SMT solver. In this example, the safety property holds, and thus F must be unsatisfiable. The solver, however, (falsely) reports F as satisfiable. This is because WTC only fixes the semantics of *negative* R-paths in F —those that only appear in negated literals in CNF; *positive* R-paths remain as sources of incompleteness (thus called *difficult* R-paths). We are interested in those difficult R-paths whose refutation is mandatory for refuting F (*essential* R-paths). Fig. 1(a) contains only one difficult R-path: $(root, live) \in tc_R$ (Line 7) (denoted by p), and it is essential since it is a unit clause in F[4]. We refute p by searching for some property $\varphi[x]$, called p-invariants, that (1) holds for all objects reachable from the beginning of p, namely $root$, by one R-step —p-step test— and (2) if it holds for an object x, it holds for all objects reachable from x by one R-step —R-invariant test. Given a p-invariant $\varphi[x]$, the induction principle allows us to add the assumption $\forall x_2.(root, x_2) \in tc_R \rightarrow \varphi[x_2/x]$ as an additional clause to F without affecting its validity. If one of the p-invariants is known to not hold for $live$ (the end object of p), then p is refuted and we are done. Fig. 1(b) shows the p-invariant which is sufficient to refute our essential R-path. It is the subclause of clause (3) which passes both p-step and R-invariant tests. Details on the search procedure for p-invariants are presented in sec. 6. After adding the p-invariant assumption to F (Fig. 1(c)), the SMT solver reports it as unsatisfiable, and thus the example has been verified fully automatically.

4 Weak TC Axiomatization and its Fragment

In this section, we discuss a general[5], weak, first-order, integer-free axiomatization for transitive closure (denoted by WTC) and describe a fragment for which this is complete. The WTC axioms are given in 1. They constrain the symbol tc_R to be a transitive relation that contains R —denoted by $tr(R)$. Therefore, their deductive closure $Cl(WTC)$ describes $\mathcal{T}^{tr(R)}_{tc_R}$. Although the WTC is very weak, there exists a *non-trivial* fragment for which this axiomatization is complete.

[4] In general the test for essential R-paths is not trivial.

[5] Independent of the considered formula

$$\forall x_1, x_2.\ (x_1, x_2) \in R \rightarrow (x_1, x_2) \in tc_R$$
$$\forall x_1, x_2, x_3.\ (x_1, x_2) \in tc_R \wedge (x_2, x_3) \in tc_R \rightarrow (x_1, x_3) \in tc_R \quad (1)$$

Theorem 1 (WTC complete fragment). *Let F be a first-order relational formula, R and tc_R two binary relations, and u a tuple such that the R-path $u \in tc_R$ occurs only as negative literal in $CNF(F)$. Then, F is unsatisfiable modulo $\mathcal{T}^{R^+}_{tc_R|u}$ iff it is unsatisfiable modulo $\mathcal{T}^{tr(R)}_{tc_R|u}$.*

Proof. Let u denote a tuple (a, b) and the R-path $(a, b) \in tc_R$ be denoted by p. Assuming that p occurs only as negative literal in $CNF(F)$, we need to prove that (1) if F is unsatisfiable modulo $\mathcal{T}^{tr(R)}_{tc_R|u}$, then it is unsatisfiable modulo $\mathcal{T}^{R^+}_{tc_R|u}$ too, and (2) if F has a $\mathcal{T}^{tr(R)}_{tc_R|u}$-model, it has a $\mathcal{T}^{R^+}_{tc_R|u}$-model too. Case (1) is trivial since $R^+ \subseteq tr(R)$. For case (2) we assume that $\mathcal{M}$ is a $\mathcal{T}^{tr(R)}_{tc_R|u}$-model of F. For all clauses in $CNF(F)$ in which a literal other than $\neg p$ is satisfied, $\mathcal{M}$ is especially a $\mathcal{T}^{R^+}_{tc_R|u}$-model because $\mathcal{T}^{tr(R)}_{tc_R|u}$ and $\mathcal{T}^{R^+}_{tc_R|u}$ coincide in symbols other than $tc_R|_u$. For all other clauses C, we can assume that $C := \neg p \vee C_{rest}$ and $\mathcal{M} \models \neg(a, b) \in tc_R$. Since $\mathcal{T}^{tr(R)}_{tc_R} = Cl(WTC)$, $\mathcal{M}$ is especially a model for the second WTC axiom instantiated with a and b; $\mathcal{M} \models \forall x_2.\ (a, x_2) \notin tc_R \vee (x_2, b) \notin tc_R$. By induction, using the first axiom, there is no R-path from a to b in $\mathcal{M}$. Therefore $\mathcal{M}$ is a $\mathcal{T}^{R^+}_{tc_R|u}$-model for $\neg p$ and thus for C. □

In other words, theorem 1 states that if all R-paths in $CNF(F)$ are negative literals, then WTC is a correct and complete R^+-axiomatization of tc_R in F. It describes, therefore, a WTC complete fragment. The fragment conditions are syntactic and allow categorizing R-paths into easy —with only negative literals in the $CNF(F)$— and difficult —otherwise. Hereafter, we denote the set of all difficult R-paths by DP.

5 *R*-Invariants for Axiomatizing Difficult *R*-Paths

This section introduces *R-invariants* as a means for providing a transitive closure axiomatisation that is *context-complete*, i.e. complete with respect to the context in which the transitive closure is used. This axiomatisation handles difficult R-paths, those for which the weak axiomatisation is not complete, and thus provides a proof possibility for formulas beyond the WTC-fragment described in Sec. 4.

Definition 1 (Essential difficult R-paths). *Let R be a binary relation and F be a refutable first-order relational formula modulo $\mathcal{T}^{R^+}_{tc_R}$. Then, a difficult R-path $p \in DP$ is essential —for refuting F— if there exists a model $\mathcal{M}$ where $\forall p' \in DP \setminus p.\ M(tc_R|_{p'_u}) = M(R^+|_{p'_u})$, $M(tc_R|_{p_u}) = M(tr(R)|_{p_u})$ and $\mathcal{M} \models F$. The set of all essential (difficult) R-paths is denoted by EDP.*

Definition 1 describes difficult R-paths that require further axiomatization in order to refute F. The definition condition, however, requires a complete axiomatization of difficult R-paths, which is in fact our ultimate goal. Therefore, we will later give a practical *heuristic* to check for essential R-paths.

Definition 2 (R-invariant). *Let F be a first-order formula and R a binary relation. Then, a formula $\varphi[x]$ is a forward (resp. backward) R-invariant with respect to x, F and a theory $\mathcal{T}$ if*

$$F \models_{\mathcal{T}} \forall x_1, x_2.\ \varphi[x_1/x] \wedge (x_1, x_2)^d \in R \rightarrow \varphi[x_2/x]$$

for $d = 1$ (resp. $d = -1$) , where $(x_1, x_2)^{-1} = (x_2, x_1)$.

Definition 3 (p-invariant). *Let F be a first-order formula, R a binary relation and p an R-path of the form $(a, b) \in tc_R$. Then, a forward (resp. backward) R-invariant formula $\varphi[x]$ is forward (resp. backward) p-invariant with respect to x, F and a theory $\mathcal{T}$ if a (resp. b) is ground and*

$$F \models_{\mathcal{T}} \forall x_2.\ (a, x_2)^{-d} \in R \rightarrow \varphi[x_2/x]$$

for $d = 1$ (resp. $d = -1$).

When using def. 3 and 2, we may skip mentioning x, F and $\mathcal{T}$ when clear from the context. Unless explicitly stated, the forward definitions are meant.

Definition 4 (TC induction schema). *The first-order relational version of the induction axiom, denoted by IND^r, is a schema of axioms which states that for any closed first-order formula $\varphi[z_1, z_2]$, containing variables z_1 and z_2, the following hold:*

$$[\forall x_{1:2}.\ (x_1, x_2) \in R \rightarrow \varphi[x_1/z_1, x_2/z_2] \wedge \tag{2}$$

$$\forall x_{1:3}.\ \varphi[x_1/z_1, x_2/z_2] \wedge (x_1, x_2) \in tc_R \wedge (x_2, x_3) \in R \rightarrow \varphi[x_1/z_1, x_3/z_2]] \tag{3}$$

$$\rightarrow \quad [\forall x_{1:2}.\ (x_1, x_2) \in R^+ \rightarrow \varphi[x_1/z_1, x_2/z_2]] \tag{4}$$

For any refutable formula F modulo $\mathcal{T}^{R^+}_{tc_R|p_u}$ that contains an essential R-path p of the form $(a, b) \in tc_R$, we would like to claim the existence of a p-invariant formula φ, such that $(\forall x.(a, x) \in tc_R \rightarrow \varphi) \wedge F$ is refutable modulo $\mathcal{T}^{tr(R)}_{tc_R|p_u}$. We found it difficult to prove this claim using a $\mathcal{T}^{R^+}_{tc_R}$ theory, especially since any refutation proof of F has to be considered in a second-order proof system. Instead, we consider the $\mathcal{T}^{ind}_{tc_R}$ theory, which consists of the extension of $\mathcal{T}^{tr(R)}_{tc_R}$ with our induction schema for transitive closure (def. 4). This is indeed a restriction, since $\mathcal{T}^{ind}_{tc_R}$ only covers a recursively-enumerable set of properties — similar argument as in [19]. This is comparable to the gap between the first- and second-order Peano axiomatization of arithmetic (cf. [2, page 1133]). In practice, however, it imposes no restriction to the proof power and this is the common practice in literature (cf. [1, 25]).

Theorem 2 (Main theorem). *Let R be a binary relation, F a first-order relational formula and p a difficult R-path of the form $(a, b) \in tc_R$ in a clause C of F. If F is refutable modulo $\mathcal{T}^{ind}_{tc_R|a,b}$ but satisfiable modulo $\mathcal{T}^{tr(R)}_{tc_R|a,b}$, then there exists a p-invariant $\varphi[x]$ w.r.t. x, $F \setminus C$ and $\mathcal{T}^{tr(R)}_{tc_R|a,b}$, such that*

$$F \setminus C \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} \neg\varphi[b/x] \text{ and} \tag{5}$$

$$(\forall x_2.\ (a, x_2) \in tc_R \rightarrow \varphi[x_2/x]) \wedge F \text{ is refutable modulo } \mathcal{T}^{tr(R)}_{tc_R|a,b}. \tag{6}$$

Proof. Without lost of generality, we can assume that $\mathcal{T}^{tr(R)}_{tc_R|a,b}$ differs from $\mathcal{T}^{ind}_{tc_R|a,b}$ only in the interpretation of $tc_R|_{(a,b)}$, and p only occurs in C. Therefore, $F \setminus C$ must be satisfiable modulo $\mathcal{T}^{ind}_{tc_R|a,b}$. This means that since F is refutable modulo $\mathcal{T}^{ind}_{tc_R|a,b}$ but satisfiable modulo $\mathcal{T}^{tr(R)}_{tc_R|a,b}$, for each $\mathcal{T}^{ind}_{tc_R|a,b}$-model $\mathcal{M}$ of $F \setminus C$, $\mathcal{M} \models (a,b) \notin tc_R$, which in turn means $F \setminus C \models_{\mathcal{T}^{ind}_{tc_R|a,b}} (a,b) \notin tc_R$.

Let us further consider a proof object pr (e.g. in sequent style) for $F \setminus C \models_{\mathcal{T}^{ind}_{tc_R|a,b}} (a,b) \notin tc_R$, then the set of all formulas IP of all essential IND^r applications in pr is non-empty. Let $\Gamma := \{\phi_i[x_1,x_2] := \forall x_1,x_2.\ (x_1,x_2) \in tc_R \rightarrow \varphi_i(x_1,x_2) \mid \varphi_i \in IP\}$. Since IP contains all formulas of all essential IND^r applications in pr, we can conclude that $\Gamma, F \setminus C \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} (a,b) \notin tc_R$. Note, that a proof pr' of the last sequent does not contain any IND^r application, but have to make use of Γ in order to close, since $F \setminus C \not\models_{\mathcal{T}^{ind}_{tc_R|a,b}} (a,b) \notin tc_R$. Therefore, we can assume, w.l.o.g. the existence of a formula $\psi[x_1,x_2]$ where $F \setminus C \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} \neg\psi(a,b)$ and $\Gamma, \neg\psi(a,b) \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} (a,b) \notin tc_R$. Because of the form of the formulas in Γ, there must exist a $\phi_i \in \Gamma$ where $\Gamma, \neg\psi(a,b) \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} \neg\varphi_i(a,b)$. Having this, $(a,b) \notin tc_R$ can be directly concluded from ϕ_i —left to right. Note that this argument will already work if we only had the instantiation of x_1 in ϕ_i with a. Now we construct $\varphi[x_2] := \varphi_i[a/x_1]$ and prove that φ fulfills all the conditions of the theorem.

By instantiating x_1 with a in the first and second IND^r conditions (2) and (3) for $\varphi_i[x_1,x_2]$, we get directly that $\varphi[x_2]$ is a p-invariant w.r.t. x_2, F and $\mathcal{T}^{R^+}_{tc_R|a,b}$. For the theorem condition (5), let us assume that $F \setminus C \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} \varphi_i(a,b)$, then we get that $F \setminus C \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} \psi(a,b)$, which contradicts our earlier results. The last condition (6) holds since $F \setminus C \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} \neg\psi(a,b)$ and $F \setminus C, \varphi(b) \models_{\mathcal{T}^{tr(R)}_{tc_R|a,b}} \psi(a,b)$. □

Theorem 2 offers a basis for a framework capable of proving the validity of transitive closure formulas beyond the WTC fragment. Especially, for each essential R-path p, the theorem guaranties the existence of a p-invariant which is deducible from F modulo $\mathcal{T}^{ind}_{tc_R}$ and can together with F refute p. In the next section we show how the conditions of the theorem on φ can be turned into practical rules and heuristic algorithms to direct the search for p-invariants.

6 Algorithm for Detecting *p*-invariants

In order to provide an automatic procedure capable of proving transitive closure specifications, we present an algorithm which tries to bring the theoretical results of the previous sections into action. Before discussing the actual algorithm, some definitions and lemmas are needed.

We first discuss two concepts introduced and used in the last section: (1) *essential R-paths*, and (2) *R-path isolation*, i.e. the consideration of F modulo $(\mathcal{T}^{R^+}_{tc_R})^{tr(R)}_{tc_R|p_u}$ for an R-path p (cf. proof of theorem 2). The latter concept subsumes the former one and is of particular importance for the automation process. It allows for detecting essential R-paths and for handling the WTC incompleteness

for each R-path individually regardless of other paths. However, the second concept requires $\mathcal{T}^{R^+}_{tc_R}$ which is our actual goal. In order to overcome this, in def. 5, we introduce the idea of *bounded R-paths isolation*. Here, an R-path —of an arbitrary length— is replaced with a corresponding R-path of length less equal n, where n is the isolation confidence and R^i denotes joining R with itself i times.

Definition 5 (n confident R-path isolation). *Let R be a binary relation, F a first-order relational formula, p a difficult R-path in F and n a positive natural number. Then, the n confident isolation of p in F is*

$$F|^n_p := F[\{[u \in \bigcup_{i \leq n} R^i \ / \ u \in tc_R] \mid (u \in tc_R) \in DP \setminus \{p\}\}].$$

Data: F : *Term*
Result: *Term*
1 $F^{ini} \leftarrow CNF(\neg F)$; $F \leftarrow F^{ini}$; $n \leftarrow 1$
2 **repeat**
3 **for** $p := (p_s, p_e) \in tc_R \in \{p \in DP(F^{ini}) \mid sat(F^{ini}|^n_p)\}$ **do**
4 **for** $\langle p_g, d \rangle \in \{\langle p_s, 1\rangle, \langle p_e, -1\rangle\}$ **do**
5 **if** $p_g \in Gr$ **then**
6 $F \leftarrow pathInv(p, p, p_g, F, F^{ini}, R, d, n)$
7 **if** $unsat(F)$ **then**
8 **return** F
9 **else**
10 $x_{1:n} \leftarrow Var(p_g)$
11 **for** $p' := (p'_s, p'_e) \in \{p[a_{1:n}/x_{1:n}] \mid a_i \in sufGT^1(x_i)\}$ **do**
12 **if** $sat(F[p'/p]|^n_{p'})$ **then**
13 $p'_g \leftarrow d \ ? \ p'_s : p'_e$
14 $F \leftarrow pathInv(p, p', p'_g, F, F^{ini}, R, d, n)$
15 **if** $unsat(F)$ **then**
16 **return** F
17 **if** $(\forall p'.\ unsat(F[p'/p]|^n_{p'})) \wedge sat(F|^n_p)$ **then**
18 Further/General techniques are needed
19 **if** $\forall p : EDP.\ unsat(F|^n_p)$ **then**
20 $n \leftarrow n + 1$
21 **until** *F and n are unchanged*;
22 **return** F

Algorithm 1: Main Procedure

Algorithm 1 shows the main procedure of our approach. Given a refutable formula F modulo $\mathcal{T}^{R^+}_{tc_R}$, it will first detect all essential R-paths by checking the satisfiability of the n bounded isolation $F|^n_p$ of all difficult R-paths p (line 3). The isolation confidence n, is only increased if $F|^n_p$ is unsatisfiable for all essential R-paths in EDP but F is not (lines 19-20).

For each essential R-path p we search for forward p-invariants with respect to its start boundary p_s and backward p-invariants with respect to its end boundary p_e. If the currently handled path boundary, p_g, is ground, which corresponds

Data: $p, p', p_g, F, F^{ini} : Term, R \subseteq T \times T, d, n : Int$
Result: $Term$
1 **for** $\varphi[x_{1:n}] \in (F^{ini} \setminus C_p)$ *with* $p_g \sqsubseteq type(x_i)$ **do**
2 **for** $x_i \in \{x_{1:n}\}$ **do**
3 $F \leftarrow concPathInv(\varphi, p, p', p_g, F, F^{ini}, x_i, R, d, n)$
4 **if** $unsat(F[p'/p]|^n_{p'})$ **then**
5 **return** F
6 **return** F

Algorithm 2: *pathInv*

Data: $\varphi, p, p', p_g, F, F^{ini} : Term, x : Var, R \subseteq T \times T, d, n : Int$
Result: $Term$
1 **for** $\varphi_i[x] \subseteq \varphi$ **do**
2 $F \leftarrow checkPathInv(\varphi_i, x, p_g, F, R, d)$
3 **if** $unsat(F[p'/p]|^n_{p'})$ **then**
4 **return** F
5 **for** $\varphi'_i[x] \in abst(\varphi_i, F^{ini}, x, R, n)$ **do**
6 $F \leftarrow checkPathInv(\varphi'_i, x, p_g, F, R, d)$
7 **if** $unsat(F[p'/p]|^n_{p'})$ **then**
8 **return** F
9 **return** F

Algorithm 3: *concPathInv*

Data: $\varphi, F : Term, x : Var, R \subseteq T \times T, n : Int$
Result: $Set < Term >$
1 $S \leftarrow \{\varphi\}$; $A \leftarrow \emptyset$
2 **for** $\varphi_i \in S$ **do**
3 **for** $abst \in \{$*applicable abstraction rules to* $\varphi_i\}$ **do**
4 $A \leftarrow A \cup abst(\varphi_i, x, R, n)$; $S \leftarrow S \cup abst(\varphi_i, x, R, n)$
5 $S \leftarrow S \setminus \{\varphi_i\}$
6 **return** A

Algorithm 4: *abst*

Data: $\varphi, t, p_g, F : Term, R \subseteq T \times T, d : Int$
Result: $Term$
1 **begin**
2 $PO_{ini} \leftarrow \forall x_2.\ (p_g, x_2)^d \in R \rightarrow \varphi[x_2/t]$
3 $PO_{ind} \leftarrow \varphi[x_2/t] \wedge (p_g, x_2)^d \in tc_R \wedge (x_2, x_3)^d \in R$
4 $PO_{ind} \leftarrow \forall x_2, x_3.\ PO_{ind} \rightarrow \varphi[x_3/t]$
5 **if** $unsat(F \wedge \neg PO_{ini}) \wedge unsat(F \wedge \neg PO_{ind})$ **then**
6 $F \leftarrow (\forall x_2.\ (p_g, x_2)^d \in tc_R \rightarrow \varphi[x_2/t]) \wedge F$
7 **return** F

Algorithm 5: *checkPathInv*

exactly to the considered case in theorem 2, the search is performed for the original R-path p by algo. 2. Otherwise, instances of p are used (line 11-12). The p instances are generated by instantiating the variables of p_g with their essential ground terms of complexity 1 —constants— using a slightly modified version of the framework in [10][6]. The R-path instantiation approach is motivated by the guess that probably only a *small* finite set of p instances are refutable.

In algo. 2, each clause φ of $CNF(F)$ —after excluding p's clauses— that contains a non empty set of variables $x_{1:n}$ of a type compatible to p_g is considered for the p-invariant search, namely with respect to each x_i in $\{x_{i:n}\}$ (line 1-2). Since all variables in φ are universally quantified, φ is obviously a p-invariant with respect to any variable x_i, however, we are interested in more concrete forms of φ. This is described in algo. 3, where, each sub clause φ_i that contains x_i is considered a candidate. The actual check for p-invariance is performed in algo. 5. Depending on weather p_g is a start or end boundary, the forward or backward definition of p-invariants is used respectively. If the p-invariant check fails for a candidate φ_i, syntactically-driven abstractions are generated and tried (algo. 4). Our abstraction rules are shown in fig. 2. The first rule abstracts a φ_i by instantiating their variables —x_i excluded— with their essential ground terms of complexity equal to the current calculation round r. The second rule relaxes positive literals —conclusions— in φ_i by their *syntactic* consequences in F. The third rule is only used if a p-invariant candidate passes the p-step test (cf. PO_{ini} in algo. 5) but fails in the R-invariant test. It then relaxes *unary* assumptions

[6] The essential ground terms are calculated in rounds with increasing term complexity, regardless of whether the set is finite or not.

on a single path boundary such that they hold for all reachable nodes from that boundary including itself —reachability direction is stated by d. Let's assume a clause C in F of the form $(a, x_2) \in R \wedge \phi(a) \rightarrow \varphi_{rest}[x_2]$ and an R-path p of the form $(a, x_2) \in tc_R$. Then, the p-invariant candidate φ equal to $\phi(a) \rightarrow \varphi_{rest}[x_2]$ will pass the p-step check using C only. If φ does not pass the R-invariant test, then our third abstraction rule can abstract it such that it passes both tests using C only.

$Abst_1$: Variable instantiations with essential ground terms of complexity r, using [10]
$Abst_2 : \varphi := (l \vee \varphi_{rest}), (\neg l \vee C_{rest}) \in CNF(F) \Longrightarrow \varphi \rightsquigarrow \varphi[C_{rest} \ / \ l]$
$Abst_3 : \varphi := (\neg\phi(t) \vee \varphi_{rest}), t := p_g \Longrightarrow \varphi \rightsquigarrow \varphi[(\forall x.\ x = t \vee (t, x)^d \in tc_R \rightarrow \phi(x)) \ / \ \phi(t)]$

Fig. 2: Abstraction rules

If, in the case of a non-ground p_g, all R-path instances p' can be refuted but not the original path p, we directly switch to a more general technique (algo. 1 line 17-18). Basically, the technique is a natural extension of the framework presented in section 5 to explicitly consider R-paths with non-ground boundaries. This technique was employed in only one of our benchmarks. Details of the technique are skipped in the interest of space.

7 Evaluation

We have implemented a prototype version of the procedure described in section 6. In the current implementation, we fixed both the isolation confidence (algo. 1 line 19-20) and the ground term complexity (fig. 2 $Abst_1$) parameters to 1. To evaluate our technique, we checked 20 Alloy specifications that were expected to be correct. These benchmarks were taken from the Alloy Analyzer 4.2 distribution and involve transitive closure of varying complexities. In order to provide a fair evaluation of the technique, we have restricted the considered benchmarks to those that require the semantics of transitive closure for their correctness proof.

Since most Alloy benchmarks that involve transitive closure also involve trace specifications (based on the Alloy ordering library), we developed a reduction of Alloy trace specifications to transitive closure specifications. That is, we represent any ordered signature S which forms the base of a trace specification, as the set $\mathit{first} \cup \mathit{first} \,.\, \mathit{next}^+$ where *first* denotes the starting atom of the trace and $\mathit{next} \subseteq S \times S$ is a fresh acyclic relation denoting the ordering. If a trace invariant is known, we divide the original specification to (1) an invariant proof and (2) an invariant use specification. Such reduction is used for two of our Alloy benchmarks: *addrbooktrace* and *hotelroom*.

Table 1 shows the experimental results[7] performed using Z3 4.3.1 on an Intel Xeon, 2.7 GHz, 64GB memory. For each checked benchmark, we collect the number of R-paths, difficult R-paths, essential R-paths, checked p-invariant candidates, proved and injected p-invariants and the total analysis time (in seconds). Time-out is set to 12 hours for the entire analysis and to 1 minute for each call to the SMT solver. Out of 20 benchmarks assumed to be valid, 18 were proven correct by our tool. It should be noted that these benchmarks are absolutely not trivial. For example, our previous axiomatization using Z3 could not prove any of the benchmarks with essential R-paths at all (cf. [9]), and although Kelloy could prove all benchmarks, it required substantial human interactions, even for the *com* benchmarks, which do not contains essential R-paths at all (cf. [25]).

A surprising observation is that quite a large number, 13 out of 20, of Alloy specifications, that involve transitive closure, do not contain any essential R-paths, which lets them be *effectively* in the WTC fragment, although not syntactically. This fully answers our question of why in our earlier investigation [9], some transitive closure benchmarks could be proven but not others. It shows that only a very small part of our previous [9] transitive closure axiomatization, namely the WTC axioms, was actually responsible for the success.

All of the 13 benchmarks with no essential R-paths could be proven fully automatically in less than 2 seconds using WTC and without the need of any p-invariant injection. For these examples, according to theorem 1, if the SMT solver had reported a satisfying model, it would have been a valid one. Out of the remaining 7 benchmarks containing essential R-paths, our tool could prove 5. The number of injected p-invariants varies between 1, for *soundness1*, and 159, for *completeness*. The number of injected p-invariants is not guaranteed to reflect the number of needed p-invariants since it depends very much on the ordering of essential R-paths and CNF clauses. However, it does reflect that for all of our proven benchmarks except the last two. The benchmarks *hotelroom-locking* and *javatypes-soundness* could not be proven by our tool. For both benchmarks, the main difficulty lies in the complexity of our generated SMT formulas which makes them too difficult to solve by Z3. For *hotelroom-locking*, the proof obligations for the essential R-path checks could be handled, but none of the p-invariant checks, whereas for *javatypes-soundness* every single call of the solver times-out. This shows the dependency of the current version of our approach on analysable SMT representations.

8 Related Work

Several approaches have addressed the verification of Alloy specifications in general. Due to the undecidability of the Alloy language, most of these approaches are based on interactive solving. Prioni [1] and Kelloy [25] rely on reasoning in first-order logic and integer arithmetic, Dynamite [12] chose a reasoning in fork algebras —a higher-order logic. In all these general approaches the verification of

[7] Benchmarks, results and tool are available at http://i12www.ira.uka.de/~elghazi/tcAx_via_p-inv/

Benchmarks	Result	All/Dif/Ess Paths	Che. p-inv	Inj. p-inv	Time
addrbook-addIdempotent	proved	5 / 2 / 0	0	0	0,08
addrbook-delUndoesAdd	proved	5 / 2 / 0	0	0	0,10
addrbooktrace-addIdempotent	proved	23 / 17 / 0	0	0	0,25
addrbooktrace-delUndoesAdd	proved	20 / 14 / 0	0	0	0,21
addrbooktrace-lookupYields-use	proved	22 / 13 / 0	0	0	0,24
grandpa-noSelfFather	proved	6 / 3 / 0	0	0	0.09
grandpa-noSelfGrandpa	proved	6 / 3 / 0	0	0	0.09
com-theorem1	proved	5 / 2 / 0	0	0	0,18
com-theorem2	proved	5 / 2 / 0	0	0	1.73
com-theorem3	proved	5 / 2 / 0	0	0	0.24
com-theorem4a	proved	5 / 2 / 0	0	0	0.25
com-theorem4b	proved	5 / 2 / 0	0	0	0.13
filesystem-noDirAliases	proved	7 / 4 / 0	0	0	0.12
filesystem-someDir	proved	5 / 3 / 1	2	1	0.15
marksweepgc-soundness1	proved	15 / 9 / 1	38	1	9,29
marksweepgc-soundness2	proved	16 / 10 / 2	75	2	5,92
marksweepgc-completeness	proved	16 / 8 / 2	1021	159	66,58
addrbooktrace-lookupYields-proof	proved	18 / 11 / 2	271	41	79,67
hotelroom-locking	timeout	6 / 3 / 1	–	–	–
javatypes-soundess	timeout	116 / 19 / –	–	–	–

Table 1: Evaluation results

transitive closure specifications is in general *interactive*. In addition to definition rules, an induction schema is involved either directly or indirectly —for proving general lemmas.

Closer to our approach, are the works of Nelson [23] and Ami [22]. Nelson proposes a set of first-order axioms for axiomatizing the reachability between two objects following a *functional* relation f. To handle the presence of cycles he uses a ternary predicate $a \xrightarrow[c]{f} b$ stating that b is reachable from a via arbitrary f applications, but never going through c. Later works, as in [7, 20, 21], revisited and extended Nelson's ideas. The main problem with such fixed first-order axiomatizations of transitive closure is that it is unlikely that they are complete. Ami proves in [22] that Nelson's axioms are not complete even in the functional setting. More directly, we can provide a very simple refutable formula modulo transitive closure which is satisfiable in Nelson' axioms, i.e. $a \xrightarrow[b]{f} b \wedge \forall x.\ f(x) \neq b$. In our approach, however, the f-path from a to b can be easily refuted since the empty clause —$false$— is a backward invariant for this path. Ami's work, also motivated by Nelson's work, proposes, instead, three axiom *schemas*, which follow from a transitive closure induction schema. This is very similar to our approach in that the axiom set is not fixed, but generated on-demand. However, their approach differs significantly from ours in that: (1) only a pure syntactical notion of *difficult* R-paths is used (2) only *unary* predicates and their boolean combinations are considered as instantiation formulas for the axiom schemas, (3) the search for instantiation formulas is not R-path directed, (4) no criteria

for detecting already refuted R-paths is involved, and finally (5) no abstractions are used, even not variable instantiations.

Other tools like ACL2 [18], and IsaPlanner [6] are well established in the automation of general induction schemas, for years. We think that our procedure and implementation can definitively profit from their ideas, especial their lemma discovering routine, called *lemma calculation*, and lemma abstraction ideas.

9 Conclusion

We have presented an approach capable of proving Alloy specifications that involve transitive closure fully automatically. For all transitive closure occurrences the WTC axiomatization is introduced. In case the Alloy specification includes neither *difficult* R-paths —syntactical check— nor essential R-path —semantical check— we have proved that WTC is a complete axiomatization of transitive closure and thus the solver result —either *sat* or *unsat*— can be trusted. Otherwise, each essential R-path can be handled on its own thanks to our bounded *R-path isolation* concept. The incompleteness of WTC is adjusted for an essential R-path p by a directed detection and injection of *p-invariants*.

Although in theory our p-invariant detection procedure is guaranteed to terminate, this has little significance in practical terms, as we could observe for some benchmarks. From both, the conceptual as well as the engineering point of view, there is plenty room for improvement. This includes (1) the reduction of redundancy w.r.t. p-invariant candidates, and instantiation of paths and formulas, (2) the introduction of heuristics for the prioritization of paths, clauses, instantiations and abstractions, and (3) the further, also conceptual, investigation of essential R-paths with non-ground boundaries. At least for (1) and (2) we think that we can profit from well established tools in the area of induction automation like ACL2 [18], and IsaPlanner [6], even though their focus is different.

References

1. K. Arkoudas, S. Khurshid, D. Marinov, and M. Rinard. Integrating model checking and theorem proving for relational reasoning. In *(RMICS)*, 2003.
2. Jon Barwise, editor. *Handbook of mathematical logic.* Number 90 in Studies in logic and the foundations of mathematics. North-Holland Publ., Amsterdam, 1977.
3. Jonathan Best. Proving Alloy models by introducing an explicit relational theory in SMT. Studienarbeit, Karlsruhe Institute of Technology, Dec. 2012.
4. M. P. Bonacina, C. L., and L. de Moura. On deciding satisfiability by DPLL and unsound theorem proving. In *CADE*, pages 35–50, 2009.
5. G. Dennis, F. Chang, and D. Jackson. Modular verification of code with SAT. In *ISSTA*, pages 109–120, 2006.
6. L. Dixon and J. Fleuriot. IsaPlanner: A prototype proof planner in Isabelle. In *CADE*, pages 279–283. January 2003.
7. Jan Van Eijck. *Defining (reflexive) transitive closure on finite models.* 2008.

8. Aboubakr Achraf El Ghazi, Ulrich Geilmann, Mattias Ulbrich, and Mana Taghdiri. A dual-engine for early analysis of critical systems. In *DSCI*, Berlin, 2011.
9. Aboubakr Achraf El Ghazi and Mana Taghdiri. Relational reasoning via SMT solving. In *FM*, pages 133–148, June 2011.
10. Aboubakr Achraf El Ghazi, Mattias Ulbrich, Mana Taghdiri, and Mihai Herda. Reducing the complexity of quantified formulas via variable elimination. In *SMT*, pages 87–99, July 2013.
11. Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, 1972.
12. Marcelo Frias, Carlos Lopez Pombo, and Mariano Moscato. Alloy Analyzer+PVS in the analysis and verification of Alloy specifications. In *(TACAS)*, 2007.
13. Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *AMAI*, 55(1):101–122, 2009.
14. Y. Ge and L. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *CAV*, pages 306–320, 2009.
15. 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*, pages 160–174. January 2004.
16. Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Apr. 2006.
17. E. Kang and D. Jackson. Formal modeling and analysis of a flash filesystem in Alloy. In *ABZ*, 2008.
18. Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.
19. Uwe Keller. Some remarks on the definability of transitive closure in first-order logic and datalog. 2004.
20. Shuvendu K Lahiri and Shaz Qadeer. Verifying properties of well-founded linked lists. In *ACM SIGPLAN Notices*, POPL, pages 115–126, New York, 2006. ACM.
21. K. Rustan M. Leino. Recursive object types in a logic of object-oriented programs. In *TOPLAS*, pages 170–184. January 1998.
22. T. Lev-Ami, N. Immerman, T. Reps, M. Sagiv, S. Srivastava, and G. Yorsh. Simulating reachability using first-order logic with applications to verification of linked data structures. In *CADE*, pages 99–115. January 2005.
23. Greg Nelson. Verifying reachability invariants of linked structures. In *POPL*, pages 38–47, New York, 1983. ACM.
24. M. Taghdiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *FORTE*, pages 240–256, 2003.
25. Mattias Ulbrich, Ulrich Geilmann, Aboubakr Achraf El Ghazi, and Mana Taghdiri. A proof assistant for Alloy specifications. In *TACAS*, pages 422–436, March 2012.
26. Mandana Vaziri-Farahani. *Finding bugs in software with a constraint solver*. Thesis, Massachusetts Institute of Technology, 2004.