Relational Constraint Solving in SMT

Baoluo Meng[1], Andrew Reynolds[1], Cesare Tinelli[1], and Clark Barrett[2]

[1] Department of Computer Science, The University of Iowa
[2] Department of Computer Science, Stanford University

Abstract. Relational logic is useful for reasoning about computational problems with relational structures, including high-level system design, architectural configurations of network systems, ontologies, and verification of programs with linked data structures. We present a modular extension of an earlier calculus for the theory of finite sets to a theory of finite relations with such operations as transpose, product, join, and transitive closure. We implement this extension as a theory solver of the SMT solver CVC4. Combining this new solver with the finite model finding features of CVC4 enables several compelling use cases. For instance, native support for relations enables a natural mapping from Alloy, a declarative modeling language based on first-order relational logic, to SMT constraints. It also enables a natural encoding of several description logics with concrete domains, allowing the use of an SMT solver to analyze, for instance, Web Ontology Language (OWL) models. We provide an initial evaluation of our solver on a number of Alloy and OWL models which shows promising results.

Keywords: Relational Logic, SMT, Alloy, OWL

1 Introduction

Many computational problems require reasoning about relational structures. Examples include high-level system design, architectural configuration of network systems, reasoning about ontologies, and verification of programs with linked data structures. Relational logic is an appealing choice for reasoning about such problems. In this paper, we consider a many-sorted relational logic where relations of arity n are defined as sets of n-tuples with parametrized sorts for tuple elements. We define a version of this logic as a first-order theory of finite relations where relation terms are built from relation constants and variables, set operators, and relational operators such as join, transpose, product, and transitive closure.

In previous work [3], Bansal et al. developed a decision procedure for a theory of finite sets with cardinality constraints. The theory of finite relations presented here is an extension of that theory to relational constraints. We present a calculus for the satisfiability of quantifier-free formulas in this theory. Our calculus is in general refutation-sound and model-sound. It is also terminating and refutation-complete for a restricted class of quantifier-free formulas that has useful applications.

The calculus is explicitly designed to be implementable as a theory solver in SMT solvers based on the DPLL(T) architecture [14]. We have implemented a modular component for it in our SMT solver CVC4 [4], allowing CVC4 to solve constraints on relations over elements from any of the theories it supports. This relational extension of

CVC4's native input language enables natural mappings to SMT formulas from several modeling languages based on relations. This includes Alloy, a formal language based on first-order relational logic, as well as ontology languages such as OWL. A significant potential advantage of these mappings is that they bring to these languages the power of SMT solvers to reason natively about a variety of interpreted types, something that is challenging for existing reasoners for these languages.

1.1 Related Work

Alloy is a well-known declarative modeling language based on a relational logic with transitive closure, set cardinality, and integer arithmetic operators [13]. Alloy specifications, or *models*, can be analyzed for consistency or other entailment properties with the Alloy Analyzer, a static analyzer based on encoding models as propositional logic formulas, and passing them to an off-the-shelf propositional satisfiability (SAT) solver. This approach limits the analysis to models with explicit and concrete cardinality bounds on the relations involved; hence it is appropriate only for proving the consistency of a model or for disproving that a given property, encoded as a formula, holds for a model. Despite these limitations, Alloy and its analyzer have been quite useful for lightweight modeling and analysis of software systems. An earlier attempt to solve Alloy constraints without artificial cardinality bounds on relations was made by Ghazi and Taghdiri [8] using SMT solvers. They developed a translation from a subset of Alloy's language to the SMT-LIB language [5] and used the SMT solver Yices [7] to solve the resulting constraints. That approach can prove some properties of certain Alloy models, but it still requires explicitly *finitizing* relations when dealing with transitive closure, limiting the kind of properties that can be proven in models that contain applications of transitive closure. Later, the same authors introduced more general methods [9, 10], implemented in the AlloyPE tool, to axiomatize relational operators as SMT formulas without finitization while covering the entire core language of Alloy. However, since quantified relational logic is in general undecidable and quantifiers are heavily used in the translation, the resulting SMT formulas translated from Alloy are often difficult or even impossible for SMT solvers to solve, especially when transitive closure is involved.

Description Logics (DLs) [1, 2] are decidable fragments of relational logic explicitly developed for efficient knowledge representation and reasoning. They consider on purpose only unary and binary relations. The main building blocks of DLs are individuals, concepts, roles as well as operations over these, where concepts represent sets of individuals, roles represent binary relations between individuals, and operations include membership, subset, relational composition (join) and equality. Restricted use of quantifiers is allowed in DLs to encode more expressive constraints on roles and concepts. OWL [20], a standardized semantic web ontology language, represents an important application of DLs to ontological modeling. It consists of entities similar to those in DLs except for superficial differences in concrete syntax and for the inclusion of additional features that make reasoning about OWL models undecidable in general. Many efficient reasoners have been built for reasoning about OWL ontologies written in restricted fragments of OWL. These include Konclude [16], FaCT++ [18], and Chainsaw [19].

1.2 Formal Preliminaries

We define our theory of relations and our calculus in the context of many-sorted first-order logic with equality. We assume the reader is familiar with the following notions from that logic: signature, term, literal, formula, free variable, interpretation, and satisfiability of a formula in an interpretation (see, e.g., [6] for more details). Let Σ be a many-sorted signature. We will use $\approx$ as the (infix) logical symbol for equality—which has type $\sigma \times \sigma$ for all sorts σ in Σ and is always interpreted as the identity relation. We assume all signatures Σ contain the Boolean sort Bool, always interpreted as the binary set $\{\mathit{true}, \mathit{false}\}$, and a Boolean constant symbol true for true. Without loss of generality, we assume $\approx$ is the only predicate symbol in Σ, as all other predicates may be modeled as functions with return sort Bool. We will commonly write, e.g. $P(x)$, as shorthand for $P(x) \approx \mathsf{true}$ where $P(x)$ has sort Bool. We write $s \not\approx t$ as an abbreviation of $\neg\, s \approx t$. If e is a term or a formula, we denote by $\mathrm{Vars}(e)$ the set of e's free variables, extending the notation to tuples and sets of terms or formulas as expected. We write $\varphi[\boldsymbol{x}]$ to indicate that all the free variables of a formula φ are from tuple $\boldsymbol{x}$.

If φ is a Σ-formula and $\mathcal{I}$ a Σ-interpretation, we write $\mathcal{I} \models \varphi$ if $\mathcal{I}$ satisfies φ. If t is a term, we denote by $t^{\mathcal{I}}$ the value of t in $\mathcal{I}$. A *theory* is a pair $T = (\Sigma, \mathbf{I})$, where Σ is a signature and $\mathbf{I}$ is a class of Σ-interpretations that is closed under variable reassignment (i.e., every Σ-interpretation that differs from one in $\mathbf{I}$ only in how it interprets the variables is also in $\mathbf{I}$). $\mathbf{I}$ is also referred to as the *models* of T. A Σ-formula φ is *satisfiable* (resp., *unsatisfiable*) *in* T if it is satisfied by some (resp., no) interpretation in $\mathbf{I}$. A set Γ of Σ-formulas *entails in* T a Σ-formula φ, written $\Gamma \models_T \varphi$, if every interpretation in $\mathbf{I}$ that satisfies all formulas in Γ satisfies φ as well. We write $\models_T \varphi$ as an abbreviation for $\emptyset \models_T \varphi$. We write $\Gamma \models \varphi$ to denote that Γ entails φ in the class of all Σ-interpretations. Two Σ-formulas are *equisatisfiable in* T if for every model $\mathcal{A}$ of T that satisfies one, there is a model of T that satisfies the other and differs from $\mathcal{A}$ at most over the free variables not shared by the two formulas. When convenient, we will treat a finite set of formulas as the conjunction of its elements and vice versa.

2 A Relational Extension to a Theory of Finite Sets

A many-sorted theory of finite sets with cardinality T_S is described in detail in our previous work [3]. The theory T_S includes a set sort constructor $\mathsf{Set}(\alpha)$ parametrized by the sort of the set elements. The theory T_S can be combined with any other theory T Nelson-Oppen-style, by instantiating the parameter α with any sort in T. The signature of theory T_S includes function and predicate symbols for set union ($\sqcup$), intersection ($\sqcap$), difference ($\setminus$), the empty set ($[\,]$), singleton set construction ($[\_]$), set inclusion ($\sqsubseteq$), and membership ($\sqsubset$), all interpreted as expected. A sound, complete and terminating tableaux-style calculus for the theory T_S is implemented in the CVC4 SMT solver [4].

In this section, we describe an extension T_R of this theory which, however, does not include a set cardinality operator or cardinality constraints.[3] The new theory T_R extends T_S with a parametric tuple sort $\mathsf{Tup}_n(\alpha_1, \ldots, \alpha_n)$ for every $n > 0$ and various relational operators defined over sets of tuples, that is, over values whose sort is an instance

[3] A further extension of the theory to cardinality constraints is planned for future work.

Set symbols:

$$[\,] : \mathsf{Set}(\alpha) \qquad \sqcup, \sqcap, \setminus : \mathsf{Set}(\alpha) \times \mathsf{Set}(\alpha) \to \mathsf{Set}(\alpha) \qquad \sqsubset\!\!\!\!- : \alpha \times \mathsf{Set}(\alpha) \to \mathsf{Bool}$$

$$[\,\_\,] : \alpha \to \mathsf{Set}(\alpha) \qquad \sqsubseteq : \mathsf{Set}(\alpha) \times \mathsf{Set}(\alpha) \to \mathsf{Bool}$$

Relation symbols:

$$\langle \_, \ldots, \_ \rangle : \alpha_1 \times \cdots \times \alpha_n \to \mathsf{Tup}_n(\alpha_1, \ldots, \alpha_n)$$

$$* : \mathsf{Rel}_m(\boldsymbol{\alpha}) \times \mathsf{Rel}_n(\boldsymbol{\beta}) \to \mathsf{Rel}_{m+n}(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

$$\bowtie : \mathsf{Rel}_{p+1}(\boldsymbol{\alpha}, \gamma) \times \mathsf{Rel}_{q+1}(\gamma, \boldsymbol{\beta}) \to \mathsf{Rel}_{p+q}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \text{ with } p + q > 0$$

$$\_^{-1} : \mathsf{Rel}_m(\alpha_1, \ldots, \alpha_m) \to \mathsf{Rel}_m(\alpha_m, \ldots, \alpha_1) \qquad \_^{+} : \mathsf{Rel}_2(\alpha, \alpha) \to \mathsf{Rel}_2(\alpha, \alpha)$$

where $m, n > 0$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$, and $\mathsf{Rel}_n(\boldsymbol{\gamma}) = \mathsf{Set}(\mathsf{Tup}_n(\boldsymbol{\gamma}))$.

Fig. 1: Signature Σ_R of our relational theory T_R.

of $\mathsf{Set}(\mathsf{Tup}_n(\alpha_1, \ldots, \alpha_n))$. We call any sort σ of the form $\mathsf{Set}(\mathsf{Tup}_n(\sigma_1, \ldots, \sigma_n))$ a *relational sort (of arity n)* and abbreviate it as $\mathsf{Rel}_n(\sigma_1, \ldots, \sigma_n)$.

The full signature Σ_R of T_R is defined in Figure 1. Note that the function symbols $*$, $\bowtie$, and $^{-1}$ are not only parametric but also overloaded, as they apply to relational sorts $\mathsf{Rel}_k(\boldsymbol{\sigma})$ of different arities k. The models of T_R are the expansions of the models of T_S that interpret $\langle \_, \ldots, \_ \rangle$ as the n-tuple constructor, $*$ as relational product, $\bowtie$ as relational join, $\_^{-1}$ as the transpose operator, and $\_^{+}$ as the transitive closure operator. A *relation term* is a Σ_R-term of some relational sort. A *tuple term* is a Σ_R-term of some tuple sort. A *T_R-constraint* is a (dis)equality of the form $(\neg)s \approx t$, where s and t are Σ_R-terms. We write $s \not\sqsubset\!\!\!\!- t$ and $[\,t_1, \ldots, t_n\,]$ with $n > 1$ as an abbreviation of $\neg(s \sqsubset\!\!\!\!- t \approx \mathsf{true})$ and $[\,t_1\,] \sqcup \cdots \sqcup [\,t_n\,]$, respectively.

3 A Calculus for the Relational Extension

In this section, we describe a tableaux-style calculus for determining the satisfiability of finite sets of T_R-constraints. The calculus consists of a set of derivation rules similar to those in the calculus from [3] that deal with set constraints as well as new rules to process T_R-constraints. For simplicity, we will implicitly assume that the sort of any set or relation term is flat (i.e., set or relation elements are not themselves sets or relations) and allow only variables as terms of element sorts. Nested sets and relations and more complex element terms can be processed in a standard way by using a Nelson-Oppen-style approach which we will not discuss here.

The derivation rules modify a *state* data structure, where a state is either the distinguished state unsat or a set $\mathcal{S}$ of T_R-constraints. The rules are provided in Figure 2 and 3 in *guarded assignment form*. In such form, the premises of a rule refer to the current state $\mathcal{S}$ and the conclusion describes how $\mathcal{S}$ is changed by the rule's application. Rules with two or more conclusions, separated by the symbol $\|$, are non-deterministic branching rules. In the rules, we write $\mathcal{S},\ c$ as an abbreviation of $\mathcal{S} \cup \{c\}$, and denote by $\mathcal{T}(\mathcal{S})$ the set of all terms and subterms occurring in $\mathcal{S}$. We define the following closure

$$\textsc{Inter Up}\quad \frac{x \sqsubseteq s \in \mathcal{S}^* \qquad x \sqsubseteq t \in \mathcal{S}^* \qquad s \sqcap t \in \mathcal{T}(\mathcal{S})}{\mathcal{S} := \mathcal{S},\, x \sqsubseteq s \sqcap t} \qquad \textsc{Inter Down}\quad \frac{x \sqsubseteq s \sqcap t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S},\, x \sqsubseteq s,\, x \sqsubseteq t}$$

$$\textsc{Union Up}\quad \frac{x \sqsubseteq u \in \mathcal{S}^* \qquad u \in \{s,t\} \qquad s \sqcup t \in \mathcal{T}(\mathcal{S})}{\mathcal{S} := \mathcal{S},\, x \sqsubseteq s \sqcup t} \qquad \textsc{Union Down}\quad \frac{x \sqsubseteq s \sqcup t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S},\, x \sqsubseteq s \;\parallel\; \mathcal{S} := \mathcal{S},\, x \sqsubseteq t}$$

$$\textsc{Diff Up}\quad \frac{x \sqsubseteq s \in \mathcal{S}^* \qquad s \setminus t \in \mathcal{T}(\mathcal{S})}{\mathcal{S} := \mathcal{S},\, x \sqsubseteq t \;\parallel\; \mathcal{S} := \mathcal{S},\, x \sqsubseteq s \setminus t} \qquad \textsc{Diff Down}\quad \frac{x \sqsubseteq s \setminus t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S},\, x \sqsubseteq s,\, x \not\sqsubseteq t}$$

$$\textsc{Single Up}\quad \frac{[x] \in \mathcal{T}(\mathcal{S})}{\mathcal{S} := \mathcal{S},\, x \sqsubseteq [x]} \qquad \textsc{Single Down}\quad \frac{x \sqsubseteq [y] \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S},\, x \approx y} \qquad \textsc{Empty Unsat}\quad \frac{x \sqsubseteq [\,] \in \mathcal{S}^*}{\mathsf{unsat}}$$

$$\textsc{Set Diseq}\quad \frac{s \not\approx t \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S},\, z \sqsubseteq s,\, z \not\sqsubseteq t \quad\parallel\quad \mathcal{S} := \mathcal{S},\, z \not\sqsubseteq s,\, z \sqsubseteq t} \qquad \textsc{Eq Unsat}\quad \frac{(t \not\approx t) \in \mathcal{S}^*}{\mathsf{unsat}}$$

Fig. 2: Basic rules for set intersection, union, difference, singleton, disequality and contradiction. In SET DISEQ, z is a fresh variable.

operator for $\mathcal{S}$ where $\models_{\text{tup}}$ denotes entailment in the Σ_R-theory of tuples:[4]

$$\begin{aligned}\mathcal{S}^* = {} & \{s \approx t \mid s,t \in \mathcal{T}(\mathcal{S}),\, \mathcal{S} \models_{\text{tup}} s \approx t\} \cup \\ & \{s \not\approx t \mid s,t \in \mathcal{T}(\mathcal{S}),\, \mathcal{S} \models_{\text{tup}} s \approx s' \wedge t \approx t' \text{ for some } s' \not\approx t' \in \mathcal{S}\} \cup \\ & \{s \sqsubseteq t \mid s,t \in \mathcal{T}(\mathcal{S}),\, \mathcal{S} \models_{\text{tup}} s \approx s' \wedge t \approx t' \text{ for some } s' \sqsubseteq t' \in \mathcal{S}\}\end{aligned}$$

The set $\mathcal{S}^*$ is computable by extending congruence closure procedures with a rule for deducing consequences of tuple equalities of the form $\langle s_1, \ldots, s_n\rangle \approx \langle t_1, \ldots, t_n\rangle$.

A derivation rule *applies* to a state $\mathcal{S}$ if all the conditions in the rule's premises hold for $\mathcal{S}$ *and* the rule application is not redundant. An application of a rule to a state $\mathcal{S}$ with a conclusion $\mathcal{S} \cup \{\varphi_1[\boldsymbol{x}_1, \boldsymbol{z}], \ldots, \varphi_n[\boldsymbol{x}_n, \boldsymbol{z}]\}$, where $\boldsymbol{z}$ are the fresh variables introduced by the rule's application (if any), is *redundant* if $\mathcal{S}$ already contains $\varphi_1[\boldsymbol{x}_1, \boldsymbol{t}], \ldots, \varphi_n[\boldsymbol{x}_n, \boldsymbol{t}]$ for some terms $\boldsymbol{t}$.

For simplicity and without loss of generality, we consider only initial states $\mathcal{S}_0$ that contain no variables of tuple sorts $\mathsf{Tup}_n(\sigma_1, \ldots, \sigma_n)$, since such variables can be replaced by a tuple $\langle x_1, \ldots, x_n\rangle$ where each x_i is a variable of sort σ_i. We also assume that $\mathcal{S}_0$ contains no atoms of the form $s \sqsubseteq t$, since they can be replaced by $s \approx s \sqcap t$, or disequalities $\langle s_1, \ldots, s_n\rangle \not\approx \langle t_1, \ldots, t_n\rangle$ between tuple terms, since those can be treated by guessing a disequality $s_i \not\approx t_i$ between two of their respective components. All derivation rules preserve these restrictions.

Figure 2 presents the basic rules for the core set constraints in our theory. For each set operator, Figure 2 contains a *downward* rule and an *upward* rule. Given a mem-

[4] Note that this theory has all the function symbols of Σ_R, not just the tuple constructors $\langle \_, \ldots, \_\rangle$. The extra symbols are treated as *uninterpreted*.

TRANSP UP

$$\frac{\langle x_1,\ldots,x_n\rangle \sqsubseteq R \in \mathcal{S}^* \quad R^{-1} \in \mathcal{T}(\mathcal{S})}{\mathcal{S} := \mathcal{S},\ \langle x_n,\ldots,x_1\rangle \sqsubseteq R^{-1}}$$

TRANSP DOWN

$$\frac{\langle x_1,\ldots,x_n\rangle \sqsubseteq R^{-1} \in \mathcal{S}^*}{\mathcal{S} := \mathcal{S},\ \langle x_n,\ldots,x_1\rangle \sqsubseteq R}$$

PROD UP

$$\frac{\langle x_1,\ldots,x_m\rangle \sqsubseteq R_1 \in \mathcal{S}^* \quad \langle y_1,\ldots,y_n\rangle \sqsubseteq R_2 \in \mathcal{S}^* \quad R_1 * R_2 \in \mathcal{T}(\mathcal{S})}{\mathcal{S} := \mathcal{S},\ \langle x_1,\ldots,x_m,y_1,\ldots,y_n\rangle \sqsubseteq R_1 * R_2}$$

PROD DOWN

$$\frac{\langle x_1,\ldots,x_m,y_1,\ldots,y_n\rangle \sqsubseteq R_1 * R_2 \in \mathcal{S}^* \quad \mathrm{ar}(R_1) = m}{\mathcal{S} := \mathcal{S},\ \langle x_1,\ldots,x_m\rangle \sqsubseteq R_1,\ \langle y_1,\ldots,y_n\rangle \sqsubseteq R_2}$$

JOIN UP

$$\frac{\langle x_1,\ldots,x_m,z\rangle \sqsubseteq R_1, \langle z,y_1,\ldots,y_n\rangle \sqsubseteq R_2 \in \mathcal{S}^* \quad m+n>0 \quad R_1 \bowtie R_2 \in \mathcal{T}(\mathcal{S})}{\mathcal{S} := \mathcal{S},\ \langle x_1,\ldots,x_m,y_1,\ldots,y_n\rangle \sqsubseteq R_1 \bowtie R_2}$$

JOIN DOWN

$$\frac{\langle x_1,\ldots,x_m,y_1,\ldots,y_n\rangle \sqsubseteq R_1 \bowtie R_2 \in \mathcal{S}^* \quad \mathrm{ar}(R_1) = m+1}{\mathcal{S} := \mathcal{S},\ \langle x_1,\ldots,x_m,z\rangle \sqsubseteq R_1,\ \langle z,y_1,\ldots,y_n\rangle \sqsubseteq R_2}$$

TCLOS UP I

$$\frac{\langle x_1,x_2\rangle \sqsubseteq R \in \mathcal{S}^* \quad R^+ \in \mathcal{T}(\mathcal{S})}{\mathcal{S} := \mathcal{S},\ \langle x_1,x_2\rangle \sqsubseteq R^+}$$

TCLOS UP II

$$\frac{\langle x_1,x_2\rangle \sqsubseteq R^+,\ \langle x_2,x_3\rangle \sqsubseteq R^+ \in\ \mathcal{S}^*}{\mathcal{S} := \mathcal{S},\ \langle x_1,x_3\rangle \sqsubseteq R^+}$$

TCLOS DOWN

$$\frac{\langle x_1,x_2\rangle \sqsubseteq R^+ \in \mathcal{S}^*}{\begin{array}{c}\mathcal{S} := \mathcal{S},\ \langle x_1,x_2\rangle \sqsubseteq R \quad \| \quad \mathcal{S} := \mathcal{S},\ \langle x_1,z\rangle \sqsubseteq R,\ \langle z,x_2\rangle \sqsubseteq R \\ \| \quad \mathcal{S} := \mathcal{S},\ \langle x_1,z_1\rangle \sqsubseteq R,\ \langle z_1,z_2\rangle \sqsubseteq R^+,\ \langle z_2,x_2\rangle \sqsubseteq R,\ z_1 \not\approx z_2\end{array}}$$

Fig. 3: Basic relational derivation rules. Letters z, z_1, z_2 denote fresh variables.

bership constraint $x \sqsubseteq s$, the downward rules infer either additional membership constraints over the immediate subterms of s, or an equality in the case where s is $\{y\}$. For example, rule INTER DOWN, infers the constraints $x \sqsubseteq s$ and $x \sqsubseteq t$ if $\mathcal{S}^*$ contains the constraint $x \sqsubseteq s \sqcap t$. The upward rules handle the case where some set s occurs in $\mathcal{S}$, and infer membership constraints of the form $x \sqsubseteq s$ based on other constraints from $\mathcal{S}$. Rule SET DISEQ introduces a witness for a disequality between two sets s and t by using a fresh variable z to assert that there is an element that is in s but not t, or in t but not in s. There are two rules for deriving unsat from trivially unsatisfiable constraints in $\mathcal{S}$: membership constraints of the form $x \sqsubseteq \emptyset$ (EMPTY UNSAT) and disequalities of the form $t \not\approx t$ (EQ UNSAT).

We supplement the set-specific rules with an additional set of rules for T_R-constraints, given in Figure 3. From the membership of a tuple in the transpose of a relation R, rule TRANSP DOWN concludes that the reverse of the tuple is in R. Conversely, rule TRANSP UP ensures that the reverse of a tuple is in the transpose of a relation R if the tuple is in R and R^{-1} occurs in $\mathcal{S}$. From the constraint that a tuple t belongs the join of two relations R_1 and R_2 with arities m and n respectively, using a fresh Skolem variable z, rule JOIN DOWN infers that R_1 contains a tuple t_1 whose last element is the first element of a tuple t_2 in R_2 such that t is the join of t_1 and t_2. The JOIN UP rule computes the join of pairs of tuples explicitly asserted to belong to a relation R_1 and a relation R_2, respectively, provided that $R_1 \bowtie R_2$ is a term in $\mathcal{S}$. The PROD DOWN and PROD UP rules are defined similarly for the product of relations. The rules TCLOS UP I and TCLOS UP II compute members of the transitive closure of R based on the (currently asserted) members of R. When a tuple $\langle x_1, x_2\rangle$ can be inferred to belong to the transitive closure of a binary relation R, TCLOS DOWN can produce three alternative conclusions. In reachability terms, the first conclusion considers the case that x_1 is directly reachable from x_1 in the graph induced by R, the second that x_2 is reachable from x_1 in two steps, and the third that it is reachable in more steps. Note that the third case may lead to additional applications of TCLOS DOWN, possibly indefinitely, if the other constraints in $\mathcal{S}$ (implicitly) entail that $\langle x_1, x_2\rangle$ does not in fact belong to R.

Example 1. Let $\mathcal{S} = \{\langle x, y\rangle \sqsubseteq R^{-1},\ R \approx S,\ \langle y, x\rangle \not\sqsubseteq S\}$. By rule TRANSP DOWN, we can derive a constraint $\langle y, x\rangle \sqsubseteq R$, leading to a new $\mathcal{S}$: $\{\langle x, y\rangle \sqsubseteq R^{-1},\ R \approx S,\ \langle y, x\rangle \not\sqsubseteq S,\ \langle y, x\rangle \sqsubseteq R\}$. Then, $\langle y, x\rangle \sqsubseteq R$ is both equal and disequal to true in $\mathcal{S}^*$. Thus, we can derive unsat by EQ UNSAT, and conclude that $\mathcal{S}$ is T_R-unsatisfiable. □

Example 2. Let $\mathcal{S}$ be $\{\langle x\rangle \sqsubseteq R,\ \langle y\rangle \sqsubseteq R,\ R * R \approx S \sqcap T,\ \langle y, x\rangle \not\sqsubseteq T\}$. By rule PROD UP, we derive constraints $\langle x, y\rangle \sqsubseteq R * R$, $\langle y, x\rangle \sqsubseteq R * R$, $\langle x, x\rangle \sqsubseteq R * R$ and $\langle y, y\rangle \sqsubseteq R * R$. By set reasoning rule INTER DOWN, we derive another four constraints $\langle x, y\rangle \sqsubseteq S$, $\langle y, x\rangle \sqsubseteq S$, $\langle x, y\rangle \sqsubseteq T$, and $\langle y, x\rangle \sqsubseteq T$, leading to a contradiction with $\langle y, x\rangle \not\sqsubseteq T$. Thus, we can derive unsat by rule EQ UNSAT. □

Example 3. Let $\mathcal{S}$ be $\{\langle x, y\rangle \sqsubseteq R, \langle y, z\rangle \sqsubseteq R, \langle x, z\rangle \not\sqsubseteq R^+\}$. By rule TCLOS UP I, we derive two new constraints $\langle x, y\rangle \sqsubseteq R^+$ and $\langle y, z\rangle \sqsubseteq R^+$. Then, we can derive another constraint $\langle x, z\rangle \sqsubseteq R^+$, by rule TCLOS UP II, in contradiction with $\langle x, z\rangle \not\sqsubseteq R^+$. Thus, we can derive unsat by rule EQ UNSAT. □

Example 4. Let $\mathcal{S}$ be $\{\langle x, y\rangle \sqsubseteq R^+, \langle x, y\rangle \not\sqsubseteq R\}$. By rule TCLOS DOWN, we construct a derivation tree with three child branches, which add to $\mathcal{S}$ the sets $\{\langle x, y\rangle \sqsubseteq R\}$, $\{\langle x, z\rangle \sqsubseteq R, \langle z, y\rangle \sqsubseteq R\}$, and $\{\langle x, z_1\rangle \sqsubseteq R, \langle z_1, z_2\rangle \sqsubseteq R^+, \langle z_2, y\rangle \sqsubseteq R, z_1 \not\approx z_2\}$ respectively, where z_1 and z_2 are fresh variables. By rule EQ UNSAT, we can derive unsat in the first branch. Since no rules apply to the second branch, we can conclude, as we will see, that $\mathcal{S}$ is T_R-satisfiable. □

4 Calculus Correctness

In this section, we formalize the correctness properties satisfied by our calculus. These include refutation and model soundness in general and termination over a fragment of our language of constraints.[5] The rules of the calculus define a notion of *derivation trees*. These are possibly infinite trees whose nodes are states where the children of each non-leaf node are the result of applying one of the derivation rules of the calculus to that node. A finite branch of a derivation tree is *closed* if it ends with unsat; it is

[5] All proofs of the propositions below can be found in a longer version of this paper available at `http://cvc4.cs.stanford.edu/papers/CADE2017-relations/`.

(element)	$e := x$
(unary relation)	$u := x \mid [\,] \mid u_1 \sqcup u_2 \mid u_1 \sqcap u_2 \mid [\,\langle e \rangle\,] \mid b \bowtie u$
(binary relation)	$b := x \mid [\,] \mid b_1 \sqcup b_2 \mid b_1 \sqcap b_2 \mid [\,\langle e_1, e_2 \rangle\,] \mid b^{-1}$
(constraint)	$\varphi := e_1 \approx e_2 \mid \langle e \rangle \sqsubseteq u \mid \langle e_1, e_2 \rangle \sqsubseteq b \mid \neg\varphi_1$

Fig. 4: A restricted fragment of T_R-constraints. Letter x denotes variables.

saturated if no rules apply to its leaf. A derivation tree is *closed* if all of its branches are closed.

Proposition 1 (Refutation Soundness). *If there is a closed derivation tree with root node $\mathcal{S}$, then $\mathcal{S}$ is T_R-unsatisfiable.*

Proposition 2 (Model Soundness). *Let $\mathcal{S}$ be the leaf of a saturated branch in a derivation tree. There is a model $\mathcal{I}$ of T_R that satisfies $\mathcal{S}$ and is such that (i) for all $S \in \mathrm{Vars}(\mathcal{S})$ of set sort, $S^{\mathcal{I}} = \left\{ x^{\mathcal{I}} \;\middle|\; x \sqsubseteq S \in \mathcal{S}^* \right\}$, and (ii) for all other $x, y \in \mathrm{Vars}(\mathcal{S})$, $x^{\mathcal{I}} = y^{\mathcal{I}}$ if and only if $x \approx y \in \mathcal{S}^*$.*

Our calculus is terminating for a sublanguage of constraints involving only unary and binary relations and excluding transitive closure, product, or equality between relations. While this sublanguage, defined in Figure 4, is quite restricted, it is useful in reductions of description logics to relational logic, which we discuss in Section 5.2.

Proposition 3 (Termination). *If $\mathcal{S}$ is a finite set of constraints generated by the grammar in Figure 4, then all derivation trees with root node $\mathcal{S}$ are finite.*

Proof. Assume that $\mathcal{S}$ is a finite set containing only constraints φ from the grammar in Figure 4. First, we construct the following mapping from relation terms to tuple terms. Let D_{u} (resp. D_{b}) be a mapping from unary (resp. binary) relation terms to sets of unary (resp. binary) tuple terms defined as the least solution to the following set of constraints, where the e's, the u's and the b's are implicitly universally quantified metavariables ranging over terms respectively of element, unary relation and binary relation sort:

$$
\begin{array}{rl}
\langle e \rangle \in D_{\mathrm{u}}(u) & \text{if } \langle e \rangle \sqsubseteq u \in \mathcal{S} \\
\langle e \rangle \in D_{\mathrm{u}}(u_1) & \text{if } \langle e \rangle \in D_{\mathrm{u}}(u_2) \text{ and } u_1 \in \mathcal{T}(u_2) \\
\langle z_{e,b,u} \rangle \in D_{\mathrm{u}}(u) & \text{if } \langle e \rangle \in D_{\mathrm{u}}(b \bowtie u) \\
\langle e_1, e_2 \rangle \in D_{\mathrm{b}}(b) & \text{if } \langle e_1, e_2 \rangle \sqsubseteq b \in \mathcal{S} \\
\langle e_1, e_2 \rangle \in D_{\mathrm{b}}(b_1) & \text{if } \langle e_i, e_j \rangle \in D_{\mathrm{b}}(b_2) \text{ for } \{i, j\} = \{1, 2\} \text{ and } b_1 \in \mathcal{T}(b_2) \\
\langle e, z_{e,b,u} \rangle \in D_{\mathrm{b}}(b) & \text{if } \langle e \rangle \in D_{\mathrm{u}}(b \bowtie u)
\end{array}
$$

where $z_{e,b,u}$ denotes a unique fresh variable for each value of e, b, and u. We require only one such variable for each triple (e, b, u) since our redundancy criteria for rule applications ensures that JOIN DOWN cannot be applied more than once for the same premise $\langle e \rangle \in D_{\mathrm{u}}(b \bowtie u)$. Intuitively, D_{u} maps each relation term u in $\mathcal{S}$ to an overapproximation of the set of unary tuples $\langle e \rangle$ for which our calculus can infer the constraint $\langle e \rangle \sqsubseteq u$ using downward rules only, and similarly for the binary case D_{b}. The domain of D_{u} and that of D_{b} contain only relation terms occurring in $\mathcal{S}$, and thus are finite.

All sets in the ranges of D_u and D_b are also finite. To show this, we argue that only a finite number of fresh variables $z_{e,b,u}$ are introduced by this construction. We define a measure depth on element terms such that $\mathsf{depth}(e) = 0$ for all $e \in \mathcal{T}(\mathcal{S})$, and $\mathsf{depth}(z_{e,b,u}) = 1 + \mathsf{depth}(e)$. For all variables $z_{e,b,u}$ in the range of D_u and D_b, we have that $b \bowtie u \in \mathcal{T}(\mathcal{S})$, and if e is a variable of the form $z_{e',b',u'}$, then $b \bowtie u$ is either a subterm of b' or u'. Thus, the depth of all element terms in the range of D_u and D_b is finite. Since there are finitely many element terms in $\mathcal{T}(\mathcal{S})$, and finitely many terms of the form $b \bowtie u$ in $\mathcal{T}(\mathcal{S})$, there are finitely many variables of the form $z_{e,b,u}$ and thus finitely many element terms occur in tuple terms in the range of D_u and D_b. Therefore, there are finitely many tuple terms in sets in the ranges of D_u and D_b.

Now, let U_u (resp. U_b) be a mapping from unary (resp. binary) relation terms to sets of unary (resp. binary) tuple terms, constructed to be the least solution to the following set of constraints (where again the e's, the u's and the b's are implicitly universally quantified metavariables as above):

$$\begin{array}{rl}
\langle e\rangle \in U_\mathrm{u}(u) & \text{if } \langle e\rangle \in D_\mathrm{u}(u)\\
\langle e\rangle \in U_\mathrm{u}(u_1) & \text{if } \langle e\rangle \in U_\mathrm{u}(u_2), u_2 \in \mathcal{T}(u_1) \text{ and } u_1 \in \mathcal{T}(\mathcal{S})\\
\langle e\rangle \in U_\mathrm{u}(u) & \text{if } u = [\,e\,] \text{ and } u \in \mathcal{T}(\mathcal{S})\\
\langle e_1\rangle \in U_\mathrm{u}(u_1) & \text{if } u_1 = b_1 \bowtie u_2 \text{ and } \langle e_1, e_2\rangle \in U_\mathrm{b}(b_1)\\
\langle e_1, e_2\rangle \in U_\mathrm{b}(b) & \text{if } \langle e_1, e_2\rangle \in D_\mathrm{b}(b)\\
\langle e_1, e_2\rangle \in U_\mathrm{b}(b_1) & \text{if } \langle e_i, e_j\rangle \in U_\mathrm{b}(b_2) \text{ for } \{i,j\} = \{1,2\}, b_2 \in \mathcal{T}(b_1), b_1 \in \mathcal{T}(\mathcal{S})\\
\langle e_1, e_2\rangle \in U_\mathrm{b}(b) & \text{if } b = [\,\langle e_1, e_2\rangle\,] \text{ and } b \in \mathcal{T}(\mathcal{S})
\end{array}$$

Similar to the previous construction, U_u maps each relation term u in $\mathcal{S}$ to an over-approximation of the set of unary tuples $\langle e\rangle$ for which our calculus can infer the constraint $\langle e\rangle \sqsubseteq u$ using both downward and upward rules, and similarly for the binary case U_b. By construction, since the domains of D_u and D_b are subsets of $\mathcal{T}(\mathcal{S})$, the domains of U_u and U_b are also subsets of $\mathcal{T}(\mathcal{S})$, and thus are finite, hence their respective ranges R_u and R_b are finite too. Each set in R_u or R_b is finite as well, since the tuples in these ranges are built from element terms e that occur in the range of D_u and D_b or in singleton sets of the form $[\,\langle e\rangle\,]$, $[\,\langle e, e'\rangle\,]$ or $[\,\langle e', e\rangle\,]$ in $\mathcal{T}(\mathcal{S})$.

Now let $\widehat{\mathcal{S}}$ be the following set of constraints:

$$\begin{aligned}
\widehat{\mathcal{S}} = {} & \{(\neg)\langle e\rangle \sqsubseteq u \mid \langle e\rangle \in U_\mathrm{u}(u),\, u \in \mathcal{T}(\mathcal{S})\} \cup\\
& \{(\neg)\langle e_1, e_2\rangle \sqsubseteq b \mid \langle e_1, e_2\rangle \in U_\mathrm{b}(b),\, b \in \mathcal{T}(\mathcal{S})\} \cup\\
& \{(\neg)e_1 \approx e_2 \mid e_1, e_2 \in \mathcal{T}(R_\mathrm{u} \cup R_\mathrm{b})\} \cup\\
& \{\langle e_1\rangle \approx \langle e_2\rangle \mid e_1, e_2 \in \mathcal{T}(R_\mathrm{u} \cup R_\mathrm{b})\} \cup\\
& \{\langle e_1, e_2\rangle \approx \langle e_3, e_4\rangle \mid e_1, e_2, e_3, e_4 \in \mathcal{T}(R_\mathrm{u} \cup R_\mathrm{b})\}
\end{aligned}$$

From the arguments above we can conclude that $\widehat{\mathcal{S}}$ is finite. By construction, $\mathcal{S} \subseteq \widehat{\mathcal{S}}$. One can show by structural induction on derivation trees that all descendants of $\mathcal{S}$ in a derivation tree are also subsets of $\widehat{\mathcal{S}}$. Since the size of a state strictly grows with each rule application not deriving unsat, we can conclude that no derivation tree can be grown indefinitely. Hence all derivation trees with root $\mathcal{S}$ are finite. □

By Proposition 1, 2, and 3, we have that *any* rule application strategy for the calculus is a decision procedure for finite sets of constraints in the language generated by the grammar from Figure 4.

5 Applications of T_R

Our main motivation for adding native support for relations in an SMT solver is that it enables more natural mappings from other logical formalisms ultimately based on relations. This opens the possibility of leveraging the power and flexibility of SMT to reason about problems expressed in those formalisms. We discuss here two potential applications: reasoning about Alloy specifications and reasoning about OWL ontologies. It should be clear to the knowledgeable reader though that the set of potential applications is much larger, encompassing description logics in general as well as various modal logics—via an encoding of their accessibility relation.

5.1 Alloy specifications

Alloy is a formal specification language based on relational logic which is widely used for modeling structurally-rich problems [12]. Alloy specifications, called *models* in the Alloy literature, are built from relations and relational algebra operations in addition to the usual logical connectives and quantifiers. One can also specify expected properties of a specification as formulas, called *assertions* in Alloy, that should be entailed by the specification.

The analysis of Alloy specifications can be performed automatically by a tool called the Alloy Analyzer which uses as its reasoning engine Kodkod, a SAT-based finite model finder [17]. This requires the user to impose an (artificial, concrete) finite upper bound on the size of the domains of each relation, limiting the analyzer's ability to determine that a specification is consistent or has a given property. In the first case, the user has to manually increase the bounds until the Alloy Analyzer is able to find a satisfying interpretation for the specification; in the second case, until it can find a counter-example for the property. As a consequence, the analyzer cannot be used to prove that (i) a specification is inconsistent or (ii) it does have a certain property.

In contrast, thanks to its new theory solver for relational constraints based on the calculus described earlier, CVC4 is now able in many cases to do (i) and (ii) automatically, with no artificial upper bounds on domain sizes. Also, because of its own finite model finding capabilities [15], it can find minimal satisfying interpretations for consistent specifications or minimal counter-examples for properties without the need of user-provided artificial upper bounds on domain sizes. Finally, since its relational theory solver is fully integrated with its theory solvers for other theories (such as linear arithmetic, strings, arrays, and so on), CVC4 can natively support mixed constraints using relations over its various built-in types, something that is possible in the Alloy Analyzer only in rather limited form.[6] To evaluate CVC4's capabilities in solving Alloy problems, we have defined a translation from Alloy specifications to semantically equivalent SMT formulas that leverages our theory of relations. The translation focuses on Alloy's kernel language since non-kernel features can be rewritten to the kernel language by the Alloy Analyzer itself. We sketch our translation below.[7]

[6] The Alloy Analyzer currently has built-in support for bounded integers. Any other data types need to be axiomatized in the specification.

[7] The translation is sound only assuming all Alloy signatures to be finite. A full account of the translation and a proof of its soundness are beyond the scope of this paper.

In Alloy, a *signature* is a set of uninterpreted atomic elements, called *atoms*. Signatures are defined with a syntax that is reminiscent of classes in object-oriented languages. A *relation (of arity n)* is a set of n-tuples of atoms and is declared as a *field* of some signature S, which acts as the domain of the elements in the first component. Multiplicity constraints on signatures and fields can be added with keywords such as **`some`**, **`no`**, **`lone`**, and **`one`**, which specify that a signature is non-empty, empty, has cardinality at most one, and is a singleton, respectively. Other keywords specify that a relation is one-to-one, one-to-many, and so on. One or more signatures can be declared to be subsets of another signature with **`extends`** or **`in`**. With **`extends`**, all specified signatures are additionally assumed to be mutually disjoint.

In the translation, we introduce an uninterpreted sort Atom for Alloy atoms. An Alloy signature **`sig S`** is translated as a constant[8] S of sort $\mathsf{Rel}_1(\mathsf{Atom})$, that is, a set of unary tuples. A field $\mathtt{f:S}_1 \mathtt{\to} \cdots \mathtt{\to} \mathtt{S}_n$ of a signature **`S`** is translated as a constant f of sort $\mathsf{Rel}_{n+1}(\mathsf{Atom}, \ldots, \mathsf{Atom})$ together with the additional constraint $\mathsf{f} \sqsubseteq \mathsf{S} * \mathsf{S}_1 * \cdots * \mathsf{S}_n$ to ensure that the components of f's tuples are from the intended signatures. The signature hierarchy is encoded using subset constraints. For example, the Alloy constraint $\mathtt{sig\ S}_1\mathtt{,\ \ldots,\ S}_n\ \mathtt{extends\ S}$ is translated as the set of constraints $\{\mathsf{S}_1 \sqsubseteq \mathsf{S}, \ldots, \mathsf{S}_n \sqsubseteq \mathsf{S}\} \cup \{\mathsf{S}_i \sqcap \mathsf{S}_j \approx [\,] \mid 1 \leq i < j \leq n\}$. If **`S`** above is also declared to be *abstract* (a notion similar to abstract classes in object-oriented languages), the additional constraint $\mathsf{S}_1 \sqcup \cdots \sqcup \mathsf{S}_n \approx \mathsf{S}$ is added to enforce that. Similarly, signature declarations of the form $\mathtt{sig\ S}_1\mathtt{,\ \cdots,\ S}_n\ \mathtt{in\ S}$, are translated just as $\{\mathsf{S}_1 \sqsubseteq \mathsf{S}, \ldots, \mathsf{S}_n \sqsubseteq \mathsf{S}\}$. Multiplicity constraints are translated as quantified formulas.

Since our theory supports all constructs and operators in Alloy's kernel language, Alloy expressions and formulas, which can include quantifiers ranging over atoms, can be more or less directly translated to their counterparts in CVC4's language. It is worth mentioning that our translation supports Alloy's set comprehension construct, by introducing a fresh relational constant for the set and adding definitional axioms for it. In addition, we partially support Alloy cardinality constraints of the form $\mathtt{\#(}r\mathtt{)}\ op\ n$ where r is a relation term, $op \in \{\mathtt{<}, \mathtt{>}, \mathtt{=}\}$, $n \in \mathbb{N}$, and **`#`** is the cardinality operator, by encoding them as subset constraints. For example, the Alloy constraint **`#(S) < 3`** on a signature S is translated to $\mathsf{S} \sqsubseteq [\,\langle k_1\rangle, \langle k_2\rangle\,]$ where S is the corresponding unary relation and k_1 and k_2 are two fresh constants of sort Atom.

5.2 OWL DL Ontologies

OWL is an ontology language whose current version, OWL 2, was adopted as a standard Semantic Web language by the W3 consortium. It includes a sublanguage, called OWL DL, that corresponds to the expressive, yet decidable, description logic $\mathcal{SHOIN}(\mathbf{D})$ [2]. We have defined an initial, partial translation from $\mathcal{SHOIN}(\mathbf{D})$ to SMT formulas that again leverages our theory of relations.

A mapping from salient $\mathcal{SHOIN}(\mathbf{D})$ constructs to their SMT counterparts is illustrated in Figure 5. The figure shows only relations whose elements do not belong to the so-called *concrete* domain(s) $\mathbf{D}$ of $\mathcal{SHOIN}(\mathbf{D})$.[9] Similarly to the Alloy translation,

[8] Free constants have the same effect as free variables for satisfiability purposes.

[9] Some of those domains in OWL correspond to built-in sorts in CVC4. A full translation from OWL concrete domains to CVC4 built-in sorts is beyond the scope of this work.

DL	**CVC4**
individual name: a	$\mathsf{a} : \mathsf{Atom}$
atomic concept C, role R	$\mathsf{C} : \mathsf{Rel}_1(\mathsf{Atom}),\ \mathsf{R} : \mathsf{Rel}_2(\mathsf{Atom}, \mathsf{Atom})$
intersection $\mathsf{C} \sqcap \mathsf{D}$, union $\mathsf{C} \sqcup \mathsf{D}$	$\mathsf{C} \sqcap \mathsf{D},\ \mathsf{C} \sqcup \mathsf{D}$
inverse role R^-, complement $\neg\mathsf{C}$	$\mathsf{R}^{-1},\ \mathsf{Univ} \setminus \mathsf{C}$
top concept $\top$, bottom concept $\bot$	$\mathsf{Univ},\ [\,]$
existential restriction $\exists\mathsf{R.C}$	$\mathsf{R} \bowtie \mathsf{C}$
universal restriction $\forall\mathsf{R.C}$	$[\, x \mid x \sqsubseteq \mathsf{Univ} \wedge [\, x \,] \bowtie \mathsf{R} \sqsubseteq \mathsf{C} \,]$
at-least restriction $\geq_n\mathsf{R.C}$	$[\, x \mid x \sqsubseteq \mathsf{Univ} \wedge \exists a_1, \ldots, a_n : \mathsf{Atom}\, ([\, x \,] \bowtie \mathsf{R}) \sqcap \mathsf{C} \sqsupseteq [\, \langle a_1 \rangle, \ldots, \langle a_n \rangle \,] \wedge \mathsf{dist}(a_1, \ldots, a_n) \,]$
at-most restriction $\leq_n\mathsf{R.C}$	$[\, x \mid x \sqsubseteq \mathsf{Univ} \wedge \exists a_1, \ldots, a_n : \mathsf{Atom}\, ([\, x \,] \bowtie \mathsf{R}) \sqcap \mathsf{C}) \sqsubseteq [\, \langle a_1 \rangle, \ldots, \langle a_n \rangle \,] \wedge [\, \langle a_1 \rangle, \ldots, \langle a_n \rangle \,] \sqsubseteq \mathsf{C} \,]$
local reflexivity $\exists\mathsf{R.Self}$	$[\, \langle x, y \rangle \mid \langle x, y \rangle \sqsubseteq R \wedge x \approx y \,]$
nominal $\{\mathsf{a}\}$	$[\, \langle \mathsf{a} \rangle \,]$
concept, role assertion $\mathsf{C(a)},\ \mathsf{R(a, b)}$	$\mathsf{a} \sqsubseteq \mathsf{C},\ \langle \mathsf{a}, \mathsf{b} \rangle \sqsubseteq \mathsf{R}$
individual (dis)equality $\mathsf{a} \approx \mathsf{b},\ \mathsf{a} \not\approx \mathsf{b}$	$\mathsf{a} \approx \mathsf{b},\ \mathsf{a} \not\approx \mathsf{b}$
concept, role inclusion $\mathsf{C} \sqsubseteq \mathsf{D},\ \mathsf{R} \sqsubseteq \mathsf{S}$	$\mathsf{C} \sqsubseteq \mathsf{D},\ \mathsf{R} \sqsubseteq \mathsf{S}$
concept, role equiv. $\mathsf{C} \equiv \mathsf{D},\ \mathsf{R} \equiv \mathsf{S}$	$\mathsf{C} \approx \mathsf{D},\ \mathsf{R} \approx \mathsf{S}$
complex role inclusion $\mathsf{R}_1 \circ \mathsf{R}_2 \sqsubseteq S$	$\mathsf{R}_1 \bowtie \mathsf{R}_2 \sqsubseteq \mathsf{S}$
role disjointness $\mathsf{Disjoint(R, S)}$	$\mathsf{R} \sqcap \mathsf{S} \approx [\,]$

Fig. 5: A mapping from DL language to Σ_R-constraints.

we use the single sort Atom for all elements of non-concrete domains. The set comprehension notation is used here for brevity: for a set comprehension term of the form $[\, x \mid \varphi \,]$ where x has n-tuple sort, we introduce a fresh set constant S accompanied by the defining axiom $\forall x : \mathsf{Tup}_n(\mathsf{Atom})\ (x \sqsubseteq S \Leftrightarrow \varphi)$. The constant Univ in the figure denotes the universal unary relation over Atom. Since this constant is currently not built-in as a symbol of T_R, it is accompanied by the defining axiom $\forall a : \mathsf{Atom}\ \langle a \rangle \sqsubseteq \mathsf{Univ}$. The expression $\mathsf{dist}(a_1, \ldots, a_n)$ states that $a_1, \ldots, a_n$ are pairwise different. We observe how the translation is immediate for most constructs, with the notable exception of universal and number restrictions, which require the use of complex quantified formulas.

6 Evaluation

To evaluate our theory solver for T_R in CVC4 we implemented translators from Alloy and from OWL based on the translations sketched in the previous section. This section presents an initial evaluation on a selection of Alloy and OWL benchmarks.[10]

[10] Detailed results and all benchmarks are available at `http://cvc4.cs.stanford.edu/papers/CADE2017-relations/`.

	Alloy Analyzer		CVC4		CVC4+AX		AlloyPE	
Problem	**Res.**	**Time/Scope**	**Res.**	**Time**	**Res.**	**Time**	**Res.**	**Time**
academia_0	sat	0.60/3	sat	1.55	-	to	unk	84.76
academia_1	sat	0.53/2	sat	1.93	-	to	-	to
academia_2	sat	0.45/2	sat	0.49	-	to	unk	0.15
social_1	sat	0.52/3	sat	1.20	-	to	n/a	-
social_5	sat	1.56/2	sat	0.49	-	to	n/a	-
social_6	sat	0.49/2	sat	0.52	-	to	n/a	-
cf_0	sat	0.47/3	sat	0.51	-	to	n/a	-
cf_1	sat	0.49/3	sat	0.78	-	to	n/a	-
javatypes	sat	0.50/3	sat	0.42	-	to	uns	2.35
set	sat	0.45/2	sat	0.46	-	to	unk	0.92
loc_int	sat	0.57/1	sat	2.82	-	to	n/a	-
genealogy	sat	0.64/6	sat	89.20	-	to	n/a	-
number_1	sat	0.81/2	sat	8.65	-	to	n/a	-
railway	sat	0.67/4	sat	156.45	-	to	n/a	-
academia_3	b-uns	162.17/63	uns	0.49	uns	1.05	uns	0.28
academia_4	b-uns	246.92/162	uns	0.43	uns	0.54	uns	0.13
family_1	b-uns	146.62/68	uns	0.41	uns	0.44	uns	0.15
family_2	b-uns	279.77/30	uns	1.02	uns	48.78	uns	0.23
social_2	b-uns	256.98/56	uns	0.66	-	to	n/a	-
social_3	b-uns	191.45/57	uns	0.49	uns	35.91	n/a	-
social_4	b-uns	171.26/64	uns	0.46	uns	18.13	n/a	-
birthday	b-uns	156.08/53	uns	0.45	uns	0.61	uns	0.13
library	b-uns	259.54/119	uns	0.42	uns	0.40	uns	1.11
lights	b-uns	228.89/122	uns	32.69	-	to	n/a	-
INSLabel	b-uns	198.53/8	uns	1.46	-	to	n/a	-
farmers_1	sat	1.04/8	-	to	-	to	n/a	-
views	sat	9.91/9	-	to	-	to	n/a	-

Fig. 6: Evaluation on Alloy benchmarks.

6.1 Experimental Evaluation on Alloy Models

We considered two sets of Alloy benchmarks; the first consists of 40 examples from the Alloy distribution and from a formal methods course taught by one of the authors; the second were used in [10] to evaluate AlloyPE. All benchmarks consist of an Alloy model together with a single property. We evaluated two configurations of CVC4. The first, denoted **CVC4**, enables full native support for relational operators via the calculus from Section 3. The second, denoted **CVC4+AX**, instead encodes all relational operators as uninterpreted functions and supplements the translation of benchmarks with additional axioms that specify their semantics. To compare CVC4 with other tools, we also evaluated the Alloy Analyzer, version 4.2, downloaded from the Alloy website, and El Ghazi et al.'s AlloyPE tool (kindly provided to us directly by its authors) using the SMT solver Z3 version 4.5.1 as a backend. All experiments were performed with a 300 second timeout on a machine with a 2.9 GHz Intel Core i7 CPU with 8 GB of memory.

Figures 6 and 7 show the results from running the Alloy Analyzer, CVC4 and AlloyPE on the two sets of Alloy benchmarks. We omit results for 13 of the benchmarks from the first set that no system solved. The second and third columns show the results of running the Alloy Analyzer. To evaluate the Alloy Analyzer on these benchmarks, we considered bounded scopes in an incremental fashion. Using a script, we set an initial upper bound, or *scope*, of 1 for the cardinality of all signatures in the problem, and kept increasing it by 1 if the Alloy Analyzer found the problem unsatisfiable (b-uns) in the current scope—meaning that it was not able to *disprove* the property in the problem. We terminated on time out (to), or when the analyzer was able to disprove the prop-

	Alloy Analyzer		CVC4		CVC4+AX		AlloyPE	
Problem	**Res.**	**Time**	**Res.**	**Time**	**Res.**	**Time**	**Res.**	**Time**
mem-wr	b-uns	195.98/35	uns	0.43	uns	0.48	uns	0.44
mem-wi	b-uns	260.66/29	uns	0.45	uns	0.50	uns	0.42
ab-ai	b-uns	185.06/28	uns	0.46	uns	0.79	uns	0.49
ab-dua	b-uns	193.33/27	uns	0.49	uns	0.48	uns	0.70
abt-dua	b-uns	137.87/14	uns	0.60	uns	0.81	uns	0.70
abt-ly-u	b-uns	261.23/9	uns	0.81	uns	28.26	uns	1.4
abt-ly-p	b-uns	277.86/8	uns	0.81	uns	1.77	uns	175.19
gp-nsf	b-uns	152.55/69	uns	0.41	uns	0.59	uns	0.43
gp-nsg	b-uns	166.75/66	uns	0.42	-	to	uns	0.44
com-1	b-uns	297.18/13	uns	2.95	-	to	uns	0.59
com-2	b-uns	295.73/13	-	to	-	to	uns	0.55
com-3	b-uns	295.33/14	uns	4.29	-	to	uns	0.64
com-4a	b-uns	301.57/13	uns	9.39	-	to	uns	0.99
com-4b	b-uns	299.77/13	uns	0.90	-	to	uns	0.61
fs-sd	b-uns	157.90/70	uns	0.42	-	to	uns	0.89
fs-nda	b-uns	271.38/44	uns	0.55	-	to	uns	0.83
gc-s1	b-uns	270.07/14	uns	4.92	uns	8.14	uns	14.27
gc-s2	b-uns	288.44/8	-	to	-	to	uns	10.66
gc-c	b-uns	287.73/8	-	to	-	to	uns	42.31
hr-l	b-uns	275.80/7	-	to	-	to	-	to

Fig. 7: Evaluation on AlloyPE benchmarks

erty (sat) or ran out of memory for a scope. We report the scope size for benchmarks where the tool returned an answer. The fourth and fifth columns show the results from CVC4 when invoked in finite model finding mode on the translated SMT problem. The last two columns are the results from AlloyPE, where n/a indicates that AlloyPE failed due either to the presence of unsupported Alloy constructs in the input problem or to internal errors during solving or translation.

As shown in the table, our approach is overall slower than the Alloy Analyzer for satisfiable benchmarks. For the unsatisfiable ones, CVC4 returns an answer within a reasonable time limit for most of the benchmarks and has advantages over the state of the art. It is important to note, however, that an unsat answer from CVC4 indicates that the property is valid as opposed to the b-uns answer from the Alloy Analyzer, which only means the property is valid within the given scope.

Compared to AlloyPE, we successfully solved all of their benchmarks but four, as indicated in Figure 7. For these benchmarks, AlloyPE benefits from performing a static analysis of the problem that involves sophisticated heuristics to discover invariants. For our own set of benchmarks, in Figure 6, AlloyPE failed on all sat benchmarks and was unable to solve many unsat ones due to failures during the translation or the solving phase. We observe that AlloyPE gives an unsound answer for the benchmark javatypes: it returns unsat, whereas both the Alloy Analyzer and CVC4 return sat.

Our results also indicate that native support for relational reasoning is important for reasoning efficiently for these benchmark sets. In fact, **CVC4+AX** is unable to report sat for any satisfiable benchmark due to its use of axioms for the relation operators, which quantify over set variables, where CVC4's finite model finding techniques are not applicable. More interestingly, **CVC4+AX** solves significantly fewer unsat benchmarks when compared to **CVC4**, indicating that using the calculus in Section 3 is superior to encoding the semantics of relational operators via an explicit axiomatization.

6.2 Experimental Evaluation on OWL Models

We built a preliminary translator from OWL to SMT, and we did a *consistency evaluation*, which checks whether or not an ontology is contradictory, for a set of OWL benchmarks in pure description logic from the 4th OWL Reasoner Evaluation competition.[11]. Of the original 7,704 benchmarks, we considered only those whose size was under 1 MB, and further excluded benchmarks that involved some of the more sophisticated features of the OWL language that are currently not supported by our translator. We ran the experiments with a 30 second time out, on a machine with a 3.2GHz Intel(R) Xeon CPU E5-2667 v3 and 20 GB of memory.

Among the selected 3,936 benchmarks, CVC4 found 3,639 consistent, found 7 inconsistent, and timed out on the remaining 290. By comparison, the state-of-the-art DL reasoner Konclude [16] gave an answer for all 3,936 benchmarks. However, Konclude and CVC4 disagreed on 9 benchmarks, all of which CVC4 found consistent. We determined that Konclude reports inconsistent for those 9 benchmarks possibly because the benchmarks are not syntactically compliant. For example, some include Boolean literals without a type declaration. In terms of performance, CVC4 takes on average 1.7 seconds per benchmark on the 3,646 it solves. This is significantly slower than Konclude, which takes on average 0.02 seconds per benchmark on the same set. We attribute this to the fact that CVC4 does not yet support the universal set and set complement natively, and has no specific quantifier instantiation heuristics for the quantified formulas generated by the translation of universal and number restrictions. Nevertheless, we find these results quite encouraging as they show that further investigation into efficient reasoning for OWL models in SMT solvers is an interesting direction of research.

7 Conclusion and Future Work

We presented a calculus for an extension to the theory of finite sets that includes support for relations and relational operators. We implemented the calculus as a modular extension to the set subsolver in our SMT solver CVC4. A preliminary evaluation has shown that our implementation is competitive with the state of the art when used to prove properties and verifying the consistency of Alloy specifications.

We are investigating more expressive fragments for which our calculus terminates, including those corresponding to fragments of description logic [11]. In future work, we would like to devise an approach for a theory that includes both relational constraints and cardinality constraints that is efficient in practice, together with specialized techniques geared toward reasoning about formulas resulting from the translation of description logic problems. In particular, we plan to extend our logic with the set complement operator and a constant for the universal set, and extend our calculus and its implementation to provide direct support for them.

Acknowledgements This work was partially supported by NSF grant no. 1228765 and by a gift from GE Global Research. We are grateful to Jasmin Blanchette and the anonymous reviewers for their very detailed comments and questions which helped improve the presentation of the paper.

[11] See `https://www.w3.org/community/owled/ore-2015-workshop/competition`.

References

1. F. Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.
2. F. Baader, I. Horrocks, and U. Sattler. Description logics. In V. L. Frank van Harmelen and B. Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 135 – 179. Elsevier, 2008.
3. K. Bansal, A. Reynolds, C. W. Barrett, and C. Tinelli. A new decision procedure for finite sets and cardinality constraints in SMT. In *Proceedings of IJCAR'16*, volume 9706 of *LNCS*, pages 82–98. Springer, 2016.
4. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proceedings of CAV'11*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.
5. C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB standard—Version 2.6. In A. Gupta and D. Kroening, editors, *SMT 2010*, 2010.
6. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185, chapter 26, pages 825–885. IOS Press, February 2009.
7. B. Dutertre and L. D. Moura. The YICES SMT solver. Technical report, SRI International, 2006.
8. A. A. E. Ghazi and M. Taghdiri. Analyzing alloy constraints using an SMT solver: a case study. In *5th International Workshop on Automated Formal Methods (AFM)*, 2010.
9. A. A. E. Ghazi and M. Taghdiri. Relational reasoning via SMT solving. In *Proceedings of FM'11*, volume 6664 of *LNCS*, pages 133–148. Springer, 2011.
10. A. A. E. Ghazi, M. Taghdiri, and M. Herda. First-order transitive closure axiomatization via iterative invariant injections. In *Proceedings of NFM'15*, volume 9058 of *LNCS*. Springer, 2015.
11. I. Horrocks and U. Sattler. Decidability of shiq with complex role inclusion axioms. *Artificial Intelligence*, 160(1-2):79–104, 2004.
12. D. Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.
13. D. Jackson. *Software Abstractions - Logic, Language, and Analysis*. MIT Press, 2006.
14. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, Nov. 2006.
15. A. Reynolds, C. Tinelli, A. Goel, and S. Krstic. Finite model finding in SMT. In *Proceedings of CAV'13*, volume 8044 of *LNCS*, pages 640–655. Springer, 2013.
16. A. Steigmiller, T. Liebig, and B. Glimm. Konclude: System description. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27(1), 2014.
17. E. Torlak and D. Jackson. Kodkod: a relational model finder. In *Proceedings of TACAS'07*, volume 4424 of *LNCS*, pages 632–647. Springer, 2007.
18. D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: system description. In *Proceedings of IJCAR'06*, volume 4130 of *LNCS*. Springer, 2006.
19. D. Tsarkov and I. Palmisano. Chainsaw: a metareasoner for large ontologies. In I. Horrocks, M. Yatskevich, and E. Jiménez-Ruiz, editors, *ORE*, 2012.
20. W3C. OWL 2 web ontology language, https://www.w3.org/2007/OWL/wiki/Syntax.