An Equational Calculus for Alloy

Marcelo F. Frias^{*,1}, Carlos G. López Pombo^2, and Nazareno M. Aguirre^3

^{1}Department of Computer Science, School of Exact and Natural Sciences, University of Buenos Aires, Argentina, and CONICET
{mfrias,clpombo}@dc.uba.ar

^{2}Department of Computer Science, School of Exact and Natural Sciences, University of Buenos Aires, Argentina

^{3}Department of Computer Science, FCEFQyN, Universidad Nacional de Río Cuarto, Argentina
aguirre@dc.exa.unrc.edu.ar

Abstract. In this paper we show that, by translating Alloy formulas to formulas in the language of fork algebras, we obtain a complete, equational, and purely relational calculus for Alloy.

1 Introduction

Alloy [10] is a formal modeling language that is gaining acceptance within the formal methods community. Its main feature is its tool support. The Alloy Analyzer [8] performs a kind of bounded model checking that allows one to find counterexamples of supplied assertions within a model description written in Alloy. Another important feature of Alloy is its amenable language, which incorporates constructs ubiquitous in object modeling notations. Also, its easy to understand semantics, based on relations, makes the language appealing.

Contributions of This Paper: At the same time a vast research has been carried on about the analysis of Alloy specifications using SAT solving [9, 10], not much research has been done so far on the analysis of Alloy specifications using theorem proving. In order to prove Alloy assertions from Alloy specifications, in this paper we propose the following procedure:

1. Translate both the assertions and the specification to equations in the language of fork algebras [4].
2. Verify equationally whether the translated assertions can be derived from the translation of the specification.

The language of fork algebras is quite similar to Alloy in that it predicates exclusively about relations (binary in this case). The advantage of moving to fork algebras comes from the fact there is a complete (equational) calculus for reasoning about this class of algebras. Therefore, proving an Alloy formula reduces to proving its translation in this complete equational calculus.

* Research partially funded by Antorchas foundation and project UBACYT X094.

Relevance to Formal Engineering Methods: The Alloy modeling language and the Alloy Analyzer are being applied in domains such as air traffic control [3] or verification of medical software [12]. While the kind of analysis provided by Alloy is extremely useful, it does not substitute analysis techniques such as theorem proving. There has been a growing interest in combining the SAT-solving approach of Alloy with theorem proving (we will elaborate on this in the next paragraph), and our hope is that new tools developed in this direction take into account the limitations pointed out here, as well as the possible solutions we offer.

Comparison with Previous Work: Since a broad comparison with other formal methods that combine model checking and theorem proving might make us loose focus, we will compare with the specific work that has already been done on theorem proving from Alloy specifications. Prioni [2] is a tool that combines the model checking capabilities offered by Alloy with the theorem proving capabilities provided by the (semi-automatic) theorem prover Athena [1]. Athena's characterisation of Alloy allows one to reason about specifications, using (semi automatic) theorem proving. However, the proposed characterisation does not capture well some features of Alloy and its relational logic, such as, for instance, the uniform treatment for scalars, singletons and relations. Quoting the authors,

> "Recall that all values in Alloy are relations. In particular, Alloy blurs the type distinction between scalars and singletons. In our Athena formalization, however, this distinction is explicitly present and can be onerous for the Alloy user." (cf. [2, p. 6])

Prioni has also a number of further shortcomings, such as the, in our opinion, awkward representations of *Alloy*'s composition operation '.' and of ordered pairs [2, p. 4]. This tool, however, manages to integrate the use of theorem proving with the SAT solving based analysis of the Alloy Analyzer, cleverly assisting theorem proving with SAT solving and vice versa. The mechanisms used to combine SAT solving and theorem proving are independent of the theorem prover used and the axiomatic characterisation of Alloy. Thus, they could also be employed to combine the Alloy Analyzer with other approaches to reasoning about Alloy specifications, such as, for instance, the one presented in this paper.

In a previous paper [6] we presented a semantics for Alloy based on fork algebras. In that paper, the way in which we assigned semantics to quantification followed Alloy's definition, which is not equational. In this paper we present a complete and equational calculus for reasoning about Alloy assertions, in the context of Alloy specifications. Moreover, the language of fork algebras is closer to Alloy than the language used in Prioni because fork algebras talk about relations at the same level of abstraction that Alloy does.

Structure of the Paper: In Section 2 we present the syntax and semantics of Alloy, as well as a sample of Alloy specification that will be used in Section 3.4. In Section 3 we present the main results of the paper. It contains a description of fork algebras, including ways to deal with relations that are not binary.

It presents the translation of Alloy formulas to fork algebra equations, and a theorem that shows that the translation indeed captures the meaning of Alloy formulas. The section finishes with an example. Finally, in Section 4, we state our conclusions about the presented work.

2 Syntax and Semantics of Alloy

In this section we present the syntax and semantics of Alloy's relational logic as extracted from [10]. The relational logic is the kernel of Alloy. The language is then defined by means of extensions of the kernel that can be reduced to relational logic formulas. Constructs such as 'Functions' allow one to define abbreviations for formulas. 'Facts' are axioms constraining the model, and 'Assertions' are properties intended to hold that are subject to analysis within the model. For a careful description of these concepts, the reader is directed to [10]. A good source of information on Alloy is the paper [9], but the semantics is given in terms of binary relations unlike the latest version of Alloy [10], where relations have arbitrary finite ranks. In Fig. 1, we present the grammar and semantics of *Alloy*'s relational logic. Composition of binary relations is well understood; but for relations of higher rank, the following definition for the composition of relations has to be considered:

$$R;S = \{\langle a_1,\dots,a_{i-1},b_2,\dots,b_j\rangle : \\ \exists b\,(\langle a_1,\dots,a_{i-1},b\rangle \in R \;\wedge\; \langle b,b_2,\dots,b_j\rangle \in S)\}\ .$$

Example 1. Let us consider the following fragment of Alloy specification for memories, extracted from [10].

sig *Addr* { }   sig *Data* { }

These are basic signatures. We do not assume any special properties regarding the structures of data and addresses.

A possible way of defining memories is by saying that a memory consists of set of addresses, and a (total) mapping from these addresses to data values:

```
sig Memory {
    addrs: set Addr
    map: addrs ->! Data
}
```

The symbol "!" in the above definition indicates that "map" is functional and total (for each element a of addrs, there exists exactly one element d in *Data* such that map(a) = d).

Alloy allows for the definition of signatures as subsets of the set denoted by other "parent" signature. This is done via what is called *signature extension.* For the example, one could define other (perhaps more complex) kinds of memories as extensions of the *Memory* signature:

problem ::= decl*form
decl ::= *var* : *typexpr*
typexpr ::=
type
| *type* $\to$ *type*
| *type* $\Rightarrow$ *typexpr*

form ::=
expr *in* expr (subset)
|!form (neg)
| form && form (conj)
| form || form (disj)
| *all v* : *type*/form (univ)
| *some v* : *type*/form (exist)

expr ::=
expr + expr (union)
| expr & expr (intersection)
| expr − expr (difference)
|$\sim$ expr (transpose)
| expr.expr (navigation)
| +expr (closure)
| $\{v : t/$form$\}$ (set former)
| Var

Var ::=
var (variable)
| $Var[var]$ (application)

$M : \text{form} \to env \to Boolean$
$X : \text{expr} \to env \to value$
$env = (var + type) \to value$
$value = (atom \times \cdots \times atom) +$
$\quad (atom \to value)$

$M[a\ in\ b]e = X[a]e \subseteq X[b]e$
$M[!F]e = \neg M[F]e$
$M[F\&\&G]e = M[F]e \wedge M[G]e$
$M[F\ ||\ G]e = M[F]e \vee M[G]e$
$M[all\ v : t/F] =$
$\quad \bigwedge\{M[F](e \oplus v \mapsto \{\,x\,\})/x \in e(t)\}$
$M[some\ v : t/F] =$
$\quad \bigvee\{M[F](e \oplus v \mapsto \{\,x\,\})/x \in e(t)\}$

$X[a + b]e = X[a]e \cup X[b]e$
$X[a\&b]e = X[a]e \cap X[b]e$
$X[a - b]e = X[a]e \setminus X[b]e$
$X[\sim a]e = (X[a]e)^{\smile}$
$X[a.b]e = X[a]e;X[b]e$
$X[+a]e =$ the smallest r such that
$\quad r;r \subseteq r$ and $X[a]e \subseteq r$
$X[\{v : t/F\}]e =$
$\quad \{x \in e(t)/M[F](e \oplus v \mapsto \{\,x\,\})\}$
$X[v]e = e(v)$
$X[a[v]]e = \{\langle y_1, \ldots, y_n\rangle/$
$\quad \exists x.\ \langle x, y_1, \ldots, y_n\rangle \in e(a) \wedge \langle x\rangle \in e(v)\}$

Fig. 1. Grammar and semantics of Alloy

sig *MainMemory* extends *Memory* {}

sig *Cache* extends *Memory* {
    dirty: set addrs
}

With these definitions, *MainMemory* and *Cache* are special kinds of memories. In caches, a subset of addrs is recognised as *dirty*.

A system might now be defined to be composed of a main memory and a cache:

sig *System* {
    cache: *Cache*
    main: *MainMemory*
}

3 A Complete Equational Calculus for Alloy, Based on Fork Algebras

We begin this section by introducing in Section 3.1 the class of fork algebras. Fork algebras are closely related to Alloy's semantics because they provide a formalism for dealing with (binary) relations. Actually, fork algebras are even closer to NP [7], a specification language also developed by Jackson and that later evolved to Alloy. Since fork algebras allow us to deal with binary relations and Alloy handles relations of arbitrary arity, in Sections 3.2 and 3.3 we show how to emulate these kinds of relation in fork algebras. Finally, in Section 3.4 we present the mapping and present a theorem connecting Alloy terms with their translation. Finally, we present an example illustrating the way the translation works.

3.1 Fork Algebras

Fork algebras [4] are described through few equational axioms. The intended models of these axioms are structures called *proper fork algebras*, in which the domain is a set of binary relations (on some base set, let us say B), closed under the following operations for sets:

- *union* of two binary relations, denoted by $\cup$,
- *intersection* of two binary relations, denoted by $\cap$,
- *complement* of a binary relation, denoted, for a binary relation r, by $\overline{r}$,
- the *empty* binary relation, which does not relate any pair of objects, and is denoted by $\emptyset$,
- the *universal* binary relation, usually $B \times B$, that will be denoted by 1.

Besides the previous operations for sets, the domain has to be closed under the following operations for binary relations:

- *transposition* of a binary relation. This operation swaps elements in the pairs of a binary relation. Given a binary relation r, its transposition is denoted by $\breve{r}$,
- *composition* of two binary relations, which, for binary relations r and s is denoted by $r\,;s$,
- *reflexive–transitive closure*, which, for a binary relation r, is denoted by r^*,
- the *identity* relation (on B), denoted by Id.

Finally, a binary operation called *fork* is included, which requires the base set B to be closed under an injective function $\star$. This means that there are elements x in B that are the result of applying the function $\star$ to elements y and z. Since $\star$ is injective, x can be seen as an encoding of the pair $\langle y, z\rangle$. The application of fork to binary relations R and S is denoted by $R\nabla S$, and its definition is given by:

$$R\nabla S = \{\, \langle a, b \star c\rangle : \langle a, b\rangle \in R \text{ and } \langle a, c\rangle \in S \,\} \ .$$

The operation *cross* (denoted by $\otimes$) performs a kind of parallel product. Its set-theoretical definition is given by:

$$R\otimes S = \{\, \langle a \star c, b \star d\rangle : \langle a, b\rangle \in R \ \wedge \ \langle c, d\rangle \in S \,\} \ .$$

Once the class of proper fork algebras has been presented, the class of fork algebras is axiomatized with the following formulas:

1. Your favorite set of equations axiomatizing Boolean algebras. These axioms define the meaning of union, intersection, complement, the empty set and the universal relation.
2. Formulas defining composition of binary relations, transposition, reflexive–transitive closure and the identity relation:

$$\begin{aligned}
&x;(y;z) = (x;y);z,\\
&x;Id = Id;x = x,\\
&(x;y) \cap z = \emptyset \text{ iff } (z;\breve{y}) \cap x = \emptyset \text{ iff } (\breve{x};z) \cap y = \emptyset,\\
&x^* = Id \cup (x;x^*),\\
&x^*;y;1 \leq (y;1) \cup \left(x^*;(\overline{y;1} \ \cap \ (x;y;1))\right).
\end{aligned}$$

3. Formulas defining the operator ∇:

$$\begin{aligned}
&x \nabla y = (x;(Id \nabla 1)) \cap (y;(1 \nabla Id)),\\
&(x \nabla y);(w \nabla z)^{\smile} = (x;\breve{w}) \cap (y;\breve{z}),\\
&(Id \nabla 1)^{\smile} \nabla (1 \nabla Id)^{\smile} \leq Id.
\end{aligned}$$

The axioms given above define a class of models. Proper fork algebras satisfy the axioms [5], and therefore belong to this class. It could be the case that there are models for the axioms that are not proper fork algebras. Fortunately, as was proved in [5], [4, Thm. 4.2], if a model is not a proper fork algebra then it is isomorphic to one.

In Section 3.4 we will need to handle fork terms involving variables denoting relations. Following the definition of the semantics of Alloy, we define a mapping Y that, given an environment in which these variables get values, homomorphically allows to calculate the values of terms. The definition is given in Fig. 2. The set U is the domain of a fork algebra, and therefore a set of binary relations.

$$\begin{aligned}
&Y : \text{expr} \rightarrow env \rightarrow U\\
&env = (var + type) \rightarrow U.\\
\\
&Y[\emptyset]e = \text{smallest element in } U\\
&Y[1]e = \text{largest element in } U\\
&Y[\overline{a}]e = \overline{Y[a]e}\\
&Y[a \cup b]e = Y[a]e \cup Y[b]e\\
&Y[a \cap b]e = Y[a]e \cap Y[b]e\\
&Y[\breve{a}]e = (Y[a]e)^{\smile}\\
&Y[Id]e = Id\\
&Y[a;b]e = Y[a]e;Y[b]e\\
&Y[a \nabla b]e = Y[a]e \nabla Y[b]e\\
&Y[a^*]e = (Y[a]e)^*\\
&Y[v]e = e(v)
\end{aligned}$$

Fig. 2. Semantics of fork terms involving variables

3.2 Representing Objects and Sets

We will represent sets by binary relations contained in the identity relation. Thus, for an arbitrary type t and an environment *env*, $env(t) \subseteq Id$ must hold. That is, for a given type t, its meaning in an environment *env* is a binary relation contained in the identity binary relation. Similarly, for an arbitrary variable v of type t, $env(v)$ must be a relation of the form $\{ \langle x, x \rangle \}$, with $\langle x, x \rangle \in env(t)$. This is obtained by imposing the following conditions on $env(v)$[1]:

$$\begin{aligned} & env(v) \subseteq env(t), \\ & env(v);1;env(v) = env(v), \\ & env(v) \neq \emptyset \ . \end{aligned}$$

Actually, given binary relations x and y satisfying the properties:

$$y \subseteq Id, \quad x \subseteq y, \quad x;1;x = x, \quad x \neq \emptyset, \tag{1}$$

it is easy to show that x must be of the form $\{ \langle a, a \rangle \}$ for some object a. Thus, given an object a, by a we will also denote the binary relation $\{ \langle a, a \rangle \}$. Since y represents a set, by $x : y$ we assert the fact that x is an object of type y, which implies that x and y satisfy the formulas in (1).

3.3 Representing and Navigating Relations of Higher Rank in Fork Algebras

In a proper fork algebra the relations π and ρ defined by

$$\pi = (1\text{'}\nabla 1)^{\smile}, \quad \rho = (1 \nabla 1\text{'})^{\smile}$$

behave as projections with respect to the encoding of pairs induced by the injective function $\star$. Their semantics in a proper fork algebra $\mathfrak{A}$ whose binary relations range over a set B, is given by

$$\pi = \{ \langle a \star b, a \rangle : a, b \in B \},$$

$$\rho = \{ \langle a \star b, b \rangle : a, b \in B \} \ .$$

From the definitions of fork, π and ρ, operation $\otimes$ is definable as follows:

$$R \otimes S = (\pi;R) \nabla (\rho;S) \ .$$

Given a n-ary relation $R \subseteq A_1 \times \cdots \times A_n$, we will represent it by the binary relation

$$\{ \langle a_1, a_2 \star \cdots \star a_n \rangle : \langle a_1, \ldots, a_n \rangle \in R \} \ .$$

[1] The proof requires relation 1 to be of the form $B \times B$ for some nonempty set B.

This will be an invariant in the representation of n-ary relations by binary ones.

For instance, given an Alloy ternary relation

$$\text{map} \subseteq \mathit{Memory} \times \text{addrs} \times \text{Data},$$

in our framework it is encoded as a binary relation map whose elements are pairs of the form $\langle m, a \star d\rangle$ for $m : \mathit{Memory}$, a : Addr and d : Data. We will in general denote the encoding of a relation C as a binary relation, by C. Given an object (in the relational sense — cf. 3.2) m : Memory, the navigation of the relation map through m should result in a binary relation contained in Addr × Data. Given a relational object $a : t$ and a binary relation R encoding a relation of rank higher than 2, we define the navigation operation $\bullet$ by

$$a \bullet R = \breve{\pi} ; \mathit{Ran}\,(a ; R)\,; \rho\ . \tag{2}$$

Operation Ran in (2) returns the range of a relation as a partial identity. It is defined by

$$\mathit{Ran}\,(x) = (x ; 1) \cdot 1'\ .$$

Its semantics in terms of binary relations is given by

$$\mathit{Ran}\,(R) = \{\, \langle a, a\rangle : \exists b \text{ s.t. } \langle b, a\rangle \in R\,\}\ .$$

If we denote by $x \overset{R}{\text{—}} y$ the fact that x and y are related via the relation R, then Fig. 3 gives a graphical explanation of operation $\bullet$.

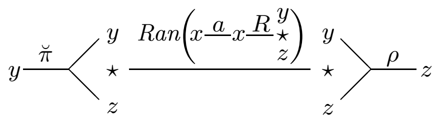
y
$\breve{\pi}$
$y \star z$
$\mathit{Ran}\left(x \overset{a}{\text{—}} x \overset{R}{\text{—}} \begin{smallmatrix} y \\ \star \\ z \end{smallmatrix}\right)$
$y \star z$
ρ
z

Fig. 3. Semantics of $\bullet$

For a binary relation R representing a relation of rank 2, navigation is easier. Given a relational object $a : t$, we define

$$a \bullet R = \mathit{Ran}\,(a ; R)\ .$$

Going back to our example about memories, it is easy to check that for a relational object $m' : \mathit{Memory}$ such that $m' = \{\, \langle m, m\rangle \,\}$,

$$m' \bullet \mathsf{map} = \{\langle a, d\rangle : a \in \mathit{Addr}, d \in \mathit{Data}\ \ \mathit{and}\ \ \langle m, a \star d\rangle \in \mathsf{map}\}\ .$$

3.4 Translating Alloy Formulas to Fork Algebra Equations

It is well known [13, p. 26] that Boolean combinations of relation algebraic equations can be translated into a single equation of the form $R = 1$. Since

Alloy terms are typed, the translation must be modified slightly. We denote by 1 the untyped universal relation. By 1_k we will denote the universal k-ary relation, and by Id_k we denote the k-ary identity relation. The transformation, for n-ary Alloy terms a and b, is:

$$a \text{ in } b \rightsquigarrow 1_n - (a - b) = 1_n$$

For a formula of the form $!(a = 1_n)$, we reason as follows:

$$!(a = 1_n) \iff !(1_n - a = 0)\ .$$

Now, from a nonempty n-ary relation, we must generate a universal n-ary relation. Notice that if $1_n - a$ is nonempty, then $1_1.(1_n - a)$ is nonempty, and has arity $n - 1$. Thus, the term

$$\underbrace{1_1.(\cdots .(1_1}_{n-1}.(1_n - a))\cdots)$$

yields a nonempty 1-ary relation. If we post compose it with 1_2, we obtain the universal 1-ary relation. If the resulting relation is then composed with the $(n+1)$-ary universal relation, we obtain the desired n-ary universal relation. We then have

$$!(a = 1_n) \rightsquigarrow (\underbrace{1_1.(\cdots .(1_1}_{n-1}.(1_n - a))\cdots).1_2).1_{n+1} = 1_n\ .$$

If we are given a formula of the form

$$a = 1_n \ \&\& \ b = 1_m,$$

with $n = m$, then the translation is trivial:

$$a = 1_n \ \&\& \ b = 1_m \rightsquigarrow a\&b = 1_n\ .$$

If $m > n$, we will convert a into a m-ary relation a' such that $a' = 1_m$ if and only if $a = 1_n$. Let a' be defined as

$$a.Id_3.1_{m-n+1}\ .$$

Then,

$$a = 1_n \ \&\& \ b = 1_m \rightsquigarrow a'\&b = 1_m\ .$$

Therefore, we will assume that whenever a quantifier occurs in a formula, it appears being applied to an equation of the form $R = 1_n$, for some n. Since variables in RL stand for single objects, we will first convert term R into a term of the form

$$(Id_{S_1} \otimes \cdots \otimes Id_{S_k}) \otimes R\ .$$

Intuitively, the term $Id_{S_1} \otimes \cdots \otimes Id_{S_k}$ allows us to pass on the value of the free variables occurring in R so that the values can be retrieved at any time. If we define relations $X_i (1 \leq i \leq k)$ by

$$X_i = \begin{cases} \rho^{;(i-1)};\pi & \text{if } 1 \leq i < k, \\ \rho^{;(i-1)} & \text{if } i = k \ , \end{cases}$$

an input $a_1 \star \cdots \star a_k$ is related through term X_i to a_i. Notice then that the term $Dom\,(\pi;X_i \ \cap \ \rho)$ filters those inputs $(a_1 \star \cdots \star a_k) \star b$ in which $a_i \neq b$ (i.e., the value b is bound to be a_i). The translation is defined as follows:

$$\begin{aligned} T(C) &= (Id_{S_1} \otimes \cdots \otimes Id_{S_k}) \otimes \mathsf{C}, \\ T(x_i) &= Dom\,(\pi;X_i \ \cap \ \rho)\,, \\ T(r+s) &= T(r) \cup T(s), \\ T(r\&s) &= T(r) \cap T(s), \\ T(r-s) &= T(r) \cap \overline{T(s)} \cap ((Id_{S_1} \otimes \cdots \otimes Id_{S_k}) \otimes 1)\,, \\ T(\sim r) &= T(r)^{\smile}, \\ T(+r) &= T(r);T(r)^*. \end{aligned}$$

In order to define the translation for navigation $r.s$ and application $s[v]$, we need to distinguish whether s is a binary relation, or if it has greater arity. The definition is as follows:

$$T(r.s) = \begin{cases} T(r) \bullet T(s) & \text{if } s \text{ is binary}, \\ T(r) \bullet (T(s);\, ((Id \otimes \pi) \nabla (Id \otimes \rho))) & \text{otherwise.} \end{cases}$$

$$T(s[v]) = \begin{cases} T(v) \bullet T(s) & \text{if } s \text{ is binary}, \\ T(v) \bullet (T(s);\, ((Id \otimes \pi) \nabla (Id \otimes \rho))) & \text{otherwise.} \end{cases}$$

In case there are no quantified variables, there is no need to carry the values on, and the translation becomes:

$$\begin{aligned} T(C) &= \mathsf{C}, \\ T(r+s) &= T(r) \cup T(s), \\ T(r\&s) &= T(r) \cap T(s), \\ T(r-s) &= T(r) \cap \overline{T(s)}, \\ T(\sim r) &= T(r)^{\smile}, \\ T(+r) &= T(r);T(r)^*, \\ T(r.s) &= T(r) \bullet T(s), \\ T(s[r]) &= T(r) \bullet T(s)\ . \end{aligned}$$

It is now easy to prove a theorem establishing the relationship between Alloy terms and the corresponding translation. Notice that:

– Given a type T, $e(T)$ is a nonempty set.
– Given a variable v, $e(v)$ is a n-ary relation for some $n \in \mathbb{N}$.

We define the environment e' by:

– Given a type T, $e'(T) = \{ \langle a, a \rangle : a \in e(T) \}$.
– Given a variable v such that $e(v)$ is a n-ary relation,

$$e'(v) = \begin{cases} \{ \langle a, a \rangle : a \in e(v) \} & \text{if } n = 1, \\ \{\langle a_1, a_2 \star \cdots \star a_n \rangle : \langle a_1, a_2, \ldots, a_n \rangle \in e(v)\} & \text{otherwise.} \end{cases}$$

Theorem 1 establishes the relationship between the semantics of Alloy terms and their translation, and allows us to assess that the translation is sound in the sense that it captures the semantics of Alloy terms. For the theorem we assume that whenever the transpose operation or the transitive closure occur in a term, they affect a binary relation. Notice that this is the assumption in [10]. We also assume that whenever the navigation operation is applied, the argument on the left-hand side is a unary relation (set). This is because our representation of relations of arity greater than two makes defining the generalized composition more complicated than desirable. At the same time, use of navigation in object-oriented settings usually falls in the situation modeled by us. In order to simplify notation, we will denote by $b^\star$ the element $b_1 \star \cdots \star b_m$.

Theorem 1. *For every Alloy term t such that:*

1. *$X[t]e$ defines a n-ary relation,*
2. *there are m free variables $x_1, \ldots, x_m$ in t,*

$$Y[T(t)]e' = \begin{cases} \{\langle b^\star \star a, b^\star \star a \rangle : \\ \qquad a \in X[t]e(\overline{b} \mapsto \overline{x})\} & \textit{if } n = 1 \\ \{\langle b^\star \star a_1, b^\star \star (a_2 \star \cdots \star a_n) \rangle : \\ \qquad \langle a_1, \ldots, a_n \rangle \in X[t]e(\overline{b} \mapsto \overline{x})\} & \textit{if } n > 1 \end{cases}$$

Proof. (Sketch of the proof) With the aim of using a shorter notation, the value (according to the standard semantics) of an Alloy term t in an environment e will be denoted by $e(t)$ rather than by $X[t]e$. Similarly, the value of a fork algebra term t in an environment e' will be denoted by $e'(t)$ rather than by $Y[t]e'$. The proof follows by induction on the structure of term t. As a sample we prove it for the variables, the remaining cases end up being simple applications of the semantics of the fork algebra operators.

If v is a quantified variable (namely, x_i), then $e(t)$ is a unary relation.

$$\begin{aligned} & e'\left(T(x_i)\right) \\ & \quad = e'\left(\mathit{Dom}\left(\pi ; X_i \ \cap \ \rho\right)\right) && \text{(by def. } T\text{)} \\ & \quad = \{ \langle b^\star \star a, b^\star \star a \rangle : a = b_i \} && \text{(by semantics)} \\ & \quad = \{ \langle b^\star \star a, b^\star \star a \rangle : a \in \{ b_i \} \} && \text{(by set theory)} \\ & \quad = \left\{ \langle b^\star \star a, b^\star \star a \rangle : a \in \left(e(\overline{b} \mapsto \overline{x})\right)(x_i) \right\} \ . && \text{(by def. } e(\overline{b} \mapsto \overline{x})\text{)} \end{aligned}$$

If v is a variable distinct of $x_1, \ldots, x_m$, there are two possibilities.

1. $e(v)$ denotes a unary relation.
2. $e(v)$ denotes a n-ary relation with $n > 1$.

If $e(v)$ denotes a unary relation,

$$\begin{aligned}
e'(T(v)) &= e'((Id_{S_1} \otimes \cdots \otimes Id_{S_m}) \otimes v) && \text{(by def. } T\text{)}\\
&= (Id_{S_1} \otimes \cdots \otimes Id_{S_m}) \otimes e'(v) && \text{(by semantics)}\\
&= (Id_{S_1} \otimes \cdots \otimes Id_{S_m}) \otimes \{ \langle a, a\rangle : a \in e(v) \} && \text{(by def. } e'\text{)}\\
&= \{ \langle b^\star \star a, b^\star \star a\rangle : a \in e(v) \} && \text{(by semantics)}\\
&= \left\{ \langle b^\star \star a, b^\star \star a\rangle : a \in (e(\overline{b} \mapsto \overline{x})))(v) \right\} \ . && \text{(by def. } e(\overline{b} \mapsto \overline{x})\text{)}
\end{aligned}$$

If $e(v)$ denotes a n-ary relation $(n > 1)$,

$$\begin{aligned}
e'(T(v)) &= e'((Id_{S_1} \otimes \cdots \otimes Id_{S_m}) \otimes v) && \text{(by def. } T\text{)}\\
&= (Id_{S_1} \otimes \cdots \otimes Id_{S_m}) \otimes e'(v) && \text{(by semantics)}\\
&= (Id_{S_1} \otimes \cdots \otimes Id_{S_m}) \otimes \{ \langle a_1, a_2 \star \cdots \star a_n\rangle : \langle a_1, \ldots, a_n\rangle \in e(v) \} \\
& && \text{(by def. } e'\text{)}\\
&= \{ \langle b^\star \star a_1, b^\star \star (a_2 \star \cdots \star a_n)\rangle : \langle a_1, \ldots, a_n\rangle \in e(v) \} && \text{(by semantics)}\\
&= \left\{ \langle b^\star \star a_1, b^\star \star (a_2 \star \cdots \star a_n)\rangle : \langle a_1, \ldots, a_n\rangle \in (e(\overline{b} \mapsto \overline{x})))(v) \right\} \ . \\
& && \text{(by def. } e(\overline{b} \mapsto \overline{x})\text{)}
\end{aligned}$$

■

From the set-theoretical definition of fork and the remaining relational operators, it follows that[2]

$$(x_1 \star \cdots \star x_n) \star x \ T(t) \ (x_1 \star \cdots \star x_n) \star y \iff (x_1 \star \cdots \star x_n) \star x \star y \in \mathsf{ran}\,(Id \ \nabla \ T(t);\rho) \ . \quad (3)$$

Now, it only remains to adequately quantify, using relational expressions, variables $x_1, \ldots, x_n, x, y$. We define the relational term $\exists_{x_i}$ as follows:

$$\exists_{x_i} = X_1 \nabla \cdots \nabla X_{i-1} \nabla 1_S \nabla X_{i+1} \nabla \cdots \nabla X_k \ .$$

For instance, if $k = 3$, we have $\exists_{x_2} = X_1 \nabla 1_S \nabla X_3$. This term defines the binary relation

$$\{ \langle a_1 \star a_3, a_1 \star a_2 \star a_3\rangle : a_2 \in S \} \ .$$

Notice that the term generates all possible values for variable x_2. The term

$$\exists_{x_2} ; Ran\,(Id \ \nabla \ T(t(x_1, x_2, x_3));\rho) \, ;1$$

describes the binary relation

[2] Given a binary relation R, $\mathsf{ran}\,(R)$ denotes the set $\{ b : \exists a \text{ such that } \langle a, b\rangle \in R \}$.

$$\{ \langle (a_1 \star a_3) \star a \star b, c\rangle : (\exists a_2 : S) \left(\langle a_1 \star a_2 \star a_3) \star a, b\rangle \in \ T(t(x_1, x_2, x_3))\right) \ \} \ .$$

This term allows us to quantify over the right domain. Profiting from the interdefinability of $\exists$ and $\forall$, the term

$$\overline{\exists_{x_2} ; \overline{T(t)}}$$

allows us to quantify variable x_2 universally. We will denote such a term as $\forall_{x_2} T(t)$.

Since variables x and y abstract the input and output values for term $T(t)$ (cf. (3)) and the original equation is of the form $T(t) = 1$, variables x and y must be quantified universally.

Example 2. Let us consider the following Alloy assertion, to be understood in the context of the specification for memories provided in Section 2.

$$\text{some s} : \mathit{System} \mid \text{s.cache.map in s.main.map} \ . \tag{4}$$

Once converted to an equation of the form $R = 1$, assertion (4) becomes

$$\text{some s} : \mathit{System} \mid 1_2 - (\text{s.cache.map} - \text{s.main.map}) = 1_2 \ . \tag{5}$$

If we apply translation T to the term on the left-hand side of the equality in (5), it becomes

$$\begin{array}{l} \overline{\mathit{Dom}\,(\pi ; X_{\text{s}} \cap \rho) \bullet \begin{matrix} \mathit{Id}_S \\ \otimes \\ \text{cache} \end{matrix} \bullet \begin{pmatrix} \mathit{Id}_S & & \mathit{Id}\otimes\pi \\ \otimes & ; & \nabla \\ \text{map} & & \mathit{Id}\otimes\rho \end{pmatrix} \cap \begin{pmatrix} \mathit{Id}_S \\ \otimes \\ 1 \end{pmatrix}} \\ \qquad \cup \ \mathit{Dom}\,(\pi ; X_{\text{s}} \cap \rho) \bullet \begin{matrix} \mathit{Id}_S \\ \otimes \\ \text{main} \end{matrix} \bullet \begin{pmatrix} \mathit{Id}_S & & \mathit{Id}\otimes\pi \\ \otimes & ; & \nabla \\ \text{map} & & \mathit{Id}\otimes\rho \end{pmatrix} \ . \end{array} \tag{6}$$

Since s is the only variable, $X_{\text{s}} = \rho^0 = \mathit{Id}$, and therefore (6) becomes

$$\begin{array}{l} \overline{\mathit{Dom}\,(\pi \cap \rho) \bullet \begin{matrix} \mathit{Id}_S \\ \otimes \\ \text{cache} \end{matrix} \bullet \begin{pmatrix} \mathit{Id}_S & & \mathit{Id}\otimes\pi \\ \otimes & ; & \nabla \\ \text{map} & & \mathit{Id}\otimes\rho \end{pmatrix} \cap \begin{pmatrix} \mathit{Id}_S \\ \otimes \\ 1 \end{pmatrix}} \\ \qquad \cup \ \mathit{Dom}\,(\pi \cap \rho) \bullet \begin{matrix} \mathit{Id}_S \\ \otimes \\ \text{main} \end{matrix} \bullet \begin{pmatrix} \mathit{Id}_S & & \mathit{Id}\otimes\pi \\ \otimes & ; & \nabla \\ \text{map} & & \mathit{Id}\otimes\rho \end{pmatrix} \ . \end{array} \tag{7}$$

Certainly (7) is harder to read than the equation in (4). This can probably be improved by adding adequate syntactic sugar to the language. Let us denote by E the term in (7). Following our recipe, we arrive to the following equation

$$\exists_{\text{s}} ; \forall_x \forall_y \left(\mathit{Ran}\,(\mathit{Id} \ \nabla \ E ; \rho) \ ; 1\right) = 1 \ . \tag{8}$$

4 Conclusions

Even although theorem proving was not considered a critical issue during the development of Alloy, it has been lately recognized as a significant problem that deserves to be addressed. The tool Prioni allows one to prove properties from Alloy specification but at the expense of requiring the user to learn another language conceptually different of Alloy in order to understand proof steps. Although the language of fork algebras is not the same as Alloy, from a conceptual viewpoint the language of fork algebras shares common principles with Alloy, as for instance the fact that objects are relations. This is one of the cornerstones of Alloy, and is shared by fork algebras but not by the language used in Athena.

References

1. Arkoudas K., *Type-ω DPLs*, MIT AI Memo 2001-27, 2001.
2. Arkoudas K., Khurshid S., Marinov D. and Rinard M., *Integrating Model Checking and Theorem Proving for Relational Reasoning*, to appear in Proceedings of RelMiCS'03.
3. Gregory Dennis, *TSAFE: Building a Trusted Computing Base for Air Traffic Control Software.* MIT Masters Thesis, January 2003.
4. Frias M., *Fork Algebras in Algebra, Logic and Computer Science*, World Scientific Publishing Co., Series Advances on Logic, 2002.
5. Frias, M. F., Haeberer, A. M. and Veloso, P. A. S., *A Finite Axiomatization for Fork Algebras*, Logic Journal of the IGPL, Vol. 5, No. 3, 311–319, 1997.
6. Frias M., López Pombo C., Baum G., Aguirre N. and Maibaum T., *Taking Alloy to the Movies*, in Proceedings of FME'03, Pisa, Italy, 2003, LNCS 2805, pp. 678–697.
7. Jackson D., *Nitpick: A checkable specificacion language*, in proceedings of Formal Methods in Software Practice, January 1996.
8. Jackson D., *Micromodels of Software: Lightweight Modelling and Analysis with Alloy*, 2002.
9. Jackson D., *Alloy: A Lightweight Object Modelling Notation*, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 11, Issue 2 (April 2002), pp. 256-290.
10. Jackson, D., Shlyakhter, I., and Sridharan, M., *A Micromodularity Mechanism.* Proc. ACM SIGSOFT Conf. Foundations of Software Engineering/European Software Engineering Conference (FSE/ESEC '01), Vienna, September 2001.
11. Maddux R., *Pair-Dense Relation Algebras*, Transactions of the American Mathematical Society, Vol. 328, N. 1, 1991.
12. Andrew Rae, Prasad Ramanan, Daniel Jackson, and Jay Flanz. *Critical feature analysis of a radiotherapy machine.* International Conference of Computer Safety, Reliability and Security (SAFECOMP 2003), Edinburgh, September 2003.
13. Tarski, A. and Givant, S.,*A Formalization of Set Theory without Variables*, A.M.S. Coll. Pub., vol. 41, 1987.