Giving ALLOY a family

Renato Neves, Luís Soares Barbosa
HASLab - INESC TEC
Univ. Minho
nevrenato@gmail.com
lsb@di.uminho.pt

Alexandre Madeira
HASLab - INESC TEC
Dep. Mathematics, Univ. Aveiro
Critical Software S.A.
madeira@ua.pt

Manuel A. Martins
CIDMA
Dep. Mathematics
Univ. Aveiro
martins@ua.pt

Abstract

Lightweight formal methods ought to provide to the end user the rigorousness of mathematics, without compromising simplicity and intuitiveness. ALLOY *is a powerful tool, particularly successful on this mission. Limitations on the verification side, however, are known to prevent its wider use in the development of safety or mission critical applications. A number of researchers proposed ways to connect Alloy to other tools in order to meet such challenges. This paper's proposal, however, is not establishing a link from* ALLOY *to another single tool, but rather to "plunge" it into the* HETS *network of logics, logic translators and provers. This makes possible for Alloy specifications to "borrow" the power of several, non dedicated proof systems. Semantical foundations for this integration are discussed in detail.*

1. Introduction

Lightweight formal methods combine mathematical rigour with simple notations and ease-of-use support platforms. ALLOY [6], based on a single sorted relational logic whose models can be automatically tested with respect to bounded domains, is one of the most successful examples. Its simple but powerful language combined with an analyser which can promptly give counter-examples depicted graphically, makes ALLOY increasingly popular both in academia and industry. Successful stories report on the discovery of faults in software designs previously thought to be faultless. The tool, however, may also bring a false sense of security, as absence of counter-examples does not imply model's correctness. Therefore, in the project of critical systems the use of ALLOY should be framed into wider toolchains involving more general, even if often less friendly theorem provers.

Actually, ALLOY impairments on the verification side may be overcome by "connecting" it to reasoners able to guarantee correctness. In such a toolchain properties can be first tested within the ALLOY analyser; if no counter-examples are found, a theorem prover is then asked to generate a proof, at least in what concerns some critical design fragments. The rationale is that typically, finding counter-examples is easier than generating a proof – how often has one tried to prove a property, only to find out a simple example invalidating it?

A number of attempts have been made in this direction (*cf.* [14], [7] and [1]). The usual approach is to translate ALLOY models into the input language of a given theorem prover and (re-)formulate the proof targets accordingly. For instance, [14], one of the most recent proposals in this trend, translates models into a first-order dialect supported by the KEY theorem prover.

The perspective taken in this paper goes a step further "plugging" ALLOY into the HETS network, as depicted in Fig. 1.

HETS [12] has been described as a "motherboard" for logics where different "expansion cards" can be plugged. The latter are individual logics (with associated analysers and proof tools) as well as logic translations to "transport" properties and proofs between them. To make them *compatible*, logics are formalised as *institutions* [3] and logic translations as *comorphisms*[1].

Plugging ALLOY to HETS brings for free the power of several provers and model checkers connected into the network, including, for instance, VAMPIRE, SPASS, EPROVER, DARWIN, ISABELLE, among many others. Experiments can then be carried out in different tools, typically tuned to specific application areas. Moreover, ALLOY models can also be translated into a number of languages available in HETS, including CASL, HASCASL, or even HASKELL itself.

There is, however, a price to be paid. To interconnect ALLOY with the HETS network, one needs first

- to formalise ALLOY underlying logic system as an *in-*

[1] The background section below provides a brief introduction to institutions as canonical representatives of logical systems.

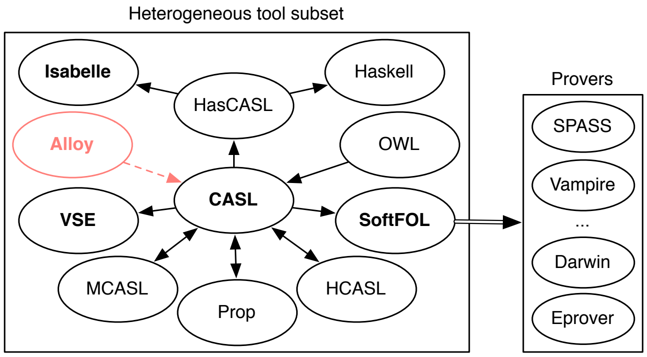


Figure 1. "Plugging" ALLOY into the HETS network

stitution,

- and to provide an effective translation to CASL, the *lingua franca* in the HETS platform, formalised as a *comorphism*.

Meeting these two challenges, and therefore laying sound foundations for the envisaged integration, is the main technical contribution of this paper.

Paper structure. The formalisation of ALLOY as an institution and the definition of a suitable comorphism to CASL is presented in sections 3 and 4. Before that, in section 2, a brief overview of the theory of institutions, CASL and ALLOY is provided as a background for the paper. Section 5 reports on a (fragment of a) case study in the medical domain on the combined use of ALLOY and HETS, to illustrate the potential and limits of the approach proposed here. Finally, section 6 concludes.

2. Background

2.1. Institutions and comorphisms

An *institution* [4] is a formalisation of the concept of a logical system, introduced by Joseph Goguen and Rod Burstall in the late 70's, as a response to the increasing number of logics emerging for software specification. Its original aim was to develop as much as computing science as possible in a general, uniform way independently of particular logical systems. This has now been achieved to an extent even greater than originally thought, as *institution* theory became the most fundamental mathematical theory underlying algebraic specification theory.

Formally, an *institution* is a tuple

$$(Sign^{\mathcal{I}}, Sen^{\mathcal{I}}, Mod^{\mathcal{I}}, (\models^{\mathcal{I}}_{\Sigma})_{\Sigma \in |Sign^{\mathcal{I}}|})$$

where

- $Sign^{\mathcal{I}}$ is a category of signatures and signature morphisms
- $Sen^{\mathcal{I}} : Sign^{\mathcal{I}} \to Set$, is a functor relating signatures to the corresponding sentences
- $Mod^{\mathcal{I}} : (Sign^{\mathcal{I}})^{op} \to C$, is a functor, giving for each signature Σ, the category of its models
- $\models^{\mathcal{I}}_{\Sigma} \subseteq |Mod^{\mathcal{I}}(\Sigma)| \times Sen^{\mathcal{I}}(\Sigma)$, is the satisfaction relation between models and sentences such that, for each morphism $\varphi : \Sigma \to \Sigma'$ in $Sign^{\mathcal{I}}$, and for any $M' \in |Mod^{\mathcal{I}}(\Sigma')|$ and $\rho \in Sen^{\mathcal{I}}(\Sigma)$,
$$M' \models^{\mathcal{I}}_{\Sigma'} Sen^{\mathcal{I}}(\varphi)(\rho) \text{ iff } Mod^{\mathcal{I}}(\varphi)(M') \models^{\mathcal{I}}_{\Sigma} \rho$$

A comorphism is a map through which theorems in the source institutions can be translated to the target one. Formally, given two institutions $\mathcal{I}, \mathcal{I}'$, a comorphism, $\mathcal{I} \to \mathcal{I}'$ is a triple (Φ, α, β) consisting of :

- a functor, $\Phi : Sign^{\mathcal{I}} \to Sign^{\mathcal{I}'}$
- a natural transformation, $\alpha : Sen^{\mathcal{I}} \Rightarrow Sen^{\mathcal{I}'}.\Phi$
- a natural transformation, $\beta : Mod^{\mathcal{I}'}.\Phi^{op} \Rightarrow Mod^{\mathcal{I}}$

such that, for any $\Sigma \in |Sign^{\mathcal{I}}|, M' \in |Mod^{\mathcal{I}'}(\Phi(\Sigma))|$ and $\rho \in Sen^{\mathcal{I}}(\Sigma)$,

$$\beta_{\Sigma}(M') \models^{\mathcal{I}}_{\Sigma} \rho \text{ iff } M' \models^{\mathcal{I}'}_{\Phi(\Sigma)} \alpha_{\Sigma}(\rho)$$

A comorphism is *conservative* whenever, for any $\Sigma \in |Sign^{\mathcal{I}}|$, β_{Σ} is surjective. This extra condition makes translations between logical systems well-behaved and is essential to correctly "borrow" proof support from one system to another.

Given an *institution* $\mathcal{I}$ one defines the institution of its *presentations* over $\mathcal{I}$ by extending signatures $\Sigma \in |Sign^{\mathcal{I}}|$, to pairs (Σ, Γ), where $\Gamma \subseteq Sen^{\mathcal{I}}(\Sigma)$, signature morphisms to presentation morphisms and restricting models $M \in |Mod^{\mathcal{I}}(\Sigma)|$ to the ones in which Γ is satisfied, *i.e.*, such that $M \models^{\mathcal{I}}_{\Sigma} \Gamma$ (see [3]).

This definition is very useful to deal with comorphisms where the source institution is too complex to be transformed into the target one in a straightforward way. Actually, this is the case here due to its "hidden" rules in the ALLOY semantics that one need to take into account.

2.2. CASL

CASL, the *Common Algebraic Specification Language* [11], was developed within the COFI initiative with the purpose of creating a suitable language for specifying requirements and to design conventional software packages. CASL specifications extend *multi-sorted first order logic* with partial functions, subsorting and free types, *i.e.*, types whose elements are restricted to be generated by the corresponding constructors and whose distinct constructor terms must denote different elements. Currently, CASL is regarded as the *de facto* standard language for algebraic specification. It is integrated into HETS along with many of its expansions, acting, as suggested in Fig. 1, as the glue language inside the HETS network of logics.

2.3. Alloy

ALLOY [6] is based on a single sorted relational language extended with a transitive closure operator.

Roughly speaking, an ALLOY specification is divided into declarations, of both relations and signatures, and sentences. Signatures will be called *kinds* from now on to distinguish them from signatures in an institution. Actually, kinds are nothing more than unary relations whose purpose is to restrict other relations. This is in line with ALLOY's *motto* which regards *everything as a relation*. Additionally, kinds may be given parents by an annotation with the keyword `extends`, establishing the obvious inclusion relation. When two kinds are in different subtrees (i.e. one is not a descendant of the other) they are supposed to be mutually disjoint. Finally, kinds may be of type

1. *Abstract*, i.e., included in the union of its descendants
2. *Some*, i.e., required to have at least one element
3. *One*, i.e., exactly with one element

The ALLOY analyser checks an assertion against a specification by seeking for counter-examples within bounded domains.

One of non standard features in ALLOY is the support for transitive closure over arbitrary expressions. This cannot be directly encoded into CASL, since it is not an higher order logic construction. Consequently, in the sequel only the transitive closure of atomic relations will be considered[2]. This is done, however, without loss of generality: for an arbitrary expression we just declare an extra binary relation and state that the latter is equal to the former.

3. Alloy as an institution

The purpose of this section is to define an institution $\mathcal{A} = (Sign^{\mathcal{A}}, Sen^{\mathcal{A}}, Mod^{\mathcal{A}}, \models^{\mathcal{A}})$ corresponding to the logical system underlying ALLOY. We proceed as follows:

Signatures. Objects (S, m, R, X) are tuples composed by:

- A family of sets containing kinds and indexed by a type, $S = \{S_t\}_{t \in \{All, Abs, Som, One\}}$. S_{All} represents all kinds, S_{Abs} the abstract ones, S_{Som} the non-empty ones, and S_{One} the kinds containing exactly one element.
- $m : S_{All} \to S_{All}$ is a function that returns the ascendent of a given kind, *i.e.*, $m(s) = s'$ means that s' is the parent of s. Top level kinds are considered the ascendents of themselves, and therefore, m takes the form of a forest structure.
- A family of relational symbols $R = \{R_w\}_{w \in (S_{All})^+}$.
- A set of singleton relational symbols X, representing the variables declared on quantified expressions. Despite being the same as the elements in S_{One}, once encoded they must be treated differently.

Morphisms $(S, m, R, X) \xrightarrow{\varphi} (S', m', R', X')$ are triples $\varphi = (\varphi_s, \varphi_r, \varphi_v)$ where:

- $\varphi_s : S \to S'$ is a function such that, for any $S_t \in S$, if $s \in S_t$ then $\varphi_s(s) \in S'_t$, and the following diagram commutes:

$$\begin{tikzcd} S \arrow[r, "\varphi_s"] \arrow[d, "m"'] & S' \arrow[d, "m'"] \\ S \arrow[r, "\varphi_s"'] & S' \end{tikzcd}$$

- φ_r is a family of functions such that, $\varphi_r = \{\varphi_w : R_w \to R'_{\varphi_s(w)}\}_{w \in (S_{All})^+}$
- $\varphi_v : X \to X'$ is a function.

[2] To the corresponding encoding an extra relation is added for each binary one as the transitive closure of the latter.

Sentences. Given a signature $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma) \in |Sign^{\mathcal{A}}|$, the set of expressions $Exp(\Sigma)$ is the smallest set containing

$$\begin{array}{ll}
p, & p \in (S_\Sigma)_{All} \cup (R_\Sigma)_w \cup X_\Sigma \\
\hat{\ }r, & r \in (R_\Sigma)_w \text{ and } |w| = 2 \\
\sim e, & e \in Exp(\Sigma) \\
e \mathrel{-\!\!>} e', & e, e' \in Exp(\Sigma) \\
e \odot e', & \text{such that } e, e' \in Exp(\Sigma), \\
 & |e| = |e'|, \text{ and } \odot \in \{+, -, \&\} \\
e \,.\, e', & \text{such that } e, e' \in Exp(\Sigma), \\
 & \text{and } |e| + |e'| > 2
\end{array}$$

with $|e|$ standing for the length of expression e.

Finally, the set of sentences, $Sen^{\mathcal{A}}(\Sigma)$, is the smallest one containing:

$$\begin{array}{ll}
e \,\texttt{in}\, e' & e, e' \in Exp(\Sigma), \text{ and } |e| = |e'| \\
\texttt{not}\ \rho & \rho \in Sen^{\mathcal{A}}(\Sigma) \\
\rho\ \texttt{implies}\ \rho' & \rho, \rho' \in Sen^{\mathcal{A}}(\Sigma) \\
(\texttt{all}\ x : e)\ \rho & e \in Exp(\Sigma), \rho \in Sen^{\mathcal{A}}(\Sigma'), |e| = 1
\end{array}$$

where $\Sigma' = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma + \{x\})$.

Models. For each $(S, m, R, X) \in |Sign^{\mathcal{A}}|$, a model $M \in |Mod^{\mathcal{A}}((S, m, R, X))|$ has

- A carrier set $|M|$
- An unary relation $M_s \subseteq |M|$, for each $s \in S_{All}$
- A relation $M_r \subseteq M_{s_1} \times \cdots \times M_{s_n}$, for each $r \in R_{s_1 \cdots s_n}$
- A singleton relation, $M_x \subseteq |M|$, for each $x \in X$

and satisfies the following axioms, for all $s, s' \in S_{All}$,

1. $M_s \subseteq M_{m(s)}$
2. if $s \in S_{Som}$, then $M_s \neq \varnothing$
3. if $s \in S_{One}$, then $\# M_s = 1$
4. if $s \in S_{Abs}$, then $M_s \subseteq \bigcup_{q \in m^\circ(s)} M_q$
5. if s, s' are not related by the transitive closure of m, then $M_s \cap M_{s'} \subseteq \varnothing$

Evaluation of expressions is as follows:

$$\begin{array}{lcl}
M_{\sim\, e} & = & (M_e)^\circ \\
M_{e\, +\, e'} & = & M_e + M_{e'} \\
M_{e\, -\, e'} & = & M_e - M_{e'} \\
M_{e\, \&\, e'} & = & M_e \cap M_{e'} \\
M_{e\, .\, e'} & = & M_e \,.\, M_{e'} \\
M_{e\, -> e'} & = & M_e \times M_{e'} \\
M_{\hat{\ }r} & = & \bigcup_{n \in N} M_{r^n}, \text{ such that } M_{r^0} = M_r \\
 & & \text{and } M_{r^{n+1}} = (M_r \,.\, M_{r^n})
\end{array}$$

Each signature morphism, $\Sigma \xrightarrow{\varphi} \Sigma' \in |Sign^{\mathcal{A}}|$, is mapped to $Mod^{\mathcal{A}}(\varphi) : Mod^{\mathcal{A}}(\Sigma') \to Mod^{\mathcal{A}}(\Sigma)$, giving, for each $M' \in |Mod^{\mathcal{A}}(\Sigma')|$, its φ-reduct, $M' \restriction_\varphi \ \in |Mod^{\mathcal{A}}(\Sigma)|$ defined by:

$$\begin{array}{lcl}
|(M' \restriction_\varphi)| & = & |M'| \\
(M' \restriction_\varphi)_s & = & M'_{\varphi_s(s)}, \text{ for any } s \in (S_\Sigma)_{All} \\
(M' \restriction_\varphi)_r & = & M'_{\varphi_r(r)}, \text{ for any } r \in (R_\Sigma)_w \\
(M' \restriction_\varphi)_x & = & M'_{\varphi_v(x)}, \text{ for any } x \in X_\Sigma
\end{array}$$

Satisfaction. Given a Σ-model M, for $\Sigma \in |Sign^{\mathcal{A}}|$, the satisfaction relation is defined for each Σ-sentence as follows:

$$\begin{array}{lll}
M \models^{\mathcal{A}}_{\Sigma} e \,\texttt{in}\, e' & \text{iff} & M_e \subseteq M_{e'} \\
M \models^{\mathcal{A}}_{\Sigma} \texttt{not}\ \rho & \text{iff} & M \not\models^{\mathcal{A}}_{\Sigma} \rho \\
M \models^{\mathcal{A}}_{\Sigma} \rho\ \texttt{implies}\ \rho' & \text{iff} & M \models^{\mathcal{A}}_{\Sigma} \rho' \\
 & & \text{whenever } M \models^{\mathcal{A}}_{\Sigma} \rho \\
M \models^{\mathcal{A}}_{\Sigma} (\texttt{all}\ x : e)\rho & \text{iff} & M' \models^{\mathcal{A}}_{\Sigma'} (x \,\texttt{in}\, e)\ \texttt{implies}\ \rho
\end{array}$$

for all model expansions M' of M, by the corresponding inclusion morphism.

Lemma 1. *$\mathcal{A} = (Sign^{\mathcal{A}}, Sen^{\mathcal{A}}, Mod^{\mathcal{A}}, \models^{\mathcal{A}})$, as defined above, is an institution.*

Proof. See the accompanying technical report [9].

□

4. From Alloy to CASL

This section characterises a conservative *comorphism* from ALLOY to the institution of presentations over CASL. The latter needs to be an institution of presentations to deal appropriately with ALLOY implicit rules over kinds and the transitive closure. Both features will be encoded into Γ, thereby restricting the class of available models. An object in the category $Sign^{\mathcal{CASL}}$ of CASL signatures is a tuple (S, TF, PF, P) where S is the set of sorts, TF a family of function symbols indexed by their arity, PF a family of partial function symbols indexed by their arity, and finally P is a family of relational symbols also indexed by their arity. Then, we define

Signature functor. For any signature $(S, m, R, X) \in |Sign^{\mathcal{A}}|$, Φ gives a tuple $((S', TF, PF, P), \Gamma)$ where

$$\begin{array}{lcl}
S' & = & \{U, Nat\} \\
TF & = & (\{0\}_{Nat}, \{suc\}_{Nat \to Nat}, \{x | x \in X\}_{\to U}) \\
PF & = & \varnothing \\
P & = & (\{s | s \in S_{All}\}_U,\ \{r | r \in R_{s_1,\ldots,s_n}\}_{U_1,\ldots,U_n}, \\
 & & \{t_r | r \in R_{s_1,s_2}\}_{Nat,U,U})
\end{array}$$

and Γ is the smallest set containing the following axioms:

1. $\{(\forall u : U)\, s(u) \Rightarrow s'(u) | s \in S,\, s' = m(s)\}$
2. $\{(\exists u : U)\, s(u) | s \in (S_{One} \cup S_{Som})\}$
3. $\{(\forall u, u' : U)\, (s(u) \wedge s(u')) \Rightarrow u = u' | s \in S_{One}\}$
4. $\{(\forall u : U)\, s(u) \Rightarrow (\bigvee_{s' \in m^{\circ}(s)} s'(u)) | s \in S_{Abs}\}$
5. $\{(\neg(\exists u : U)\, s(u) \wedge s'(u)) | s, s' \in S_{All} \wedge \neg m^{+}(s, s')\}$ where m^{+} is the transitive closure of m
6. $\{(\forall u_1, \cdots, u_n : U)\, r(u_1, \cdots, u_n)$ $\Rightarrow \;\; \bigwedge_{i=1}^{n} s_i(u_i) | r \in R_{s_1,\ldots,s_n}\}$
7. $\{$ `free type` $Nat ::= (0 \mid suc(Nat))$ $\}$
8. $\{(\forall u, v : U)\, t_r(0, u, v) \Leftrightarrow r(u, v) \wedge$ $(\forall n : Nat)\, t_r(suc(n), u, v) \Leftrightarrow (\exists x : U)\, t_r(0, u, x) \wedge$ $t_r(n, x, v) | r \in R_{s_1,s_2}\}$

Sentence transformation. Given any signature $\Sigma \in |Sign^{\mathcal{A}}|$, where $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma)$, function $\alpha_\Sigma : Sen^{\mathcal{A}}(\Sigma) \to Sen^{\mathcal{CASL}}(\Phi(\Sigma))$ is defined by

$$\begin{array}{lcl}
\alpha_\Sigma(e \;\texttt{in}\; e') & = & (\forall V : U)\, \eta_V(e) \Rightarrow \eta_V(e'), \\
 & & \text{such that } V = (v_1, \ldots, v_n), \\
 & & \text{and } n = |e| \\
\alpha_\Sigma(\texttt{not}\; \rho) & = & \neg \alpha_\Sigma(\rho) \\
\alpha_\Sigma(\rho \;\texttt{implies}\; \rho') & = & \alpha_\Sigma(\rho) \;\texttt{implies}\; \alpha_\Sigma(\rho') \\
\alpha_\Sigma((\texttt{all}\; x : e)\; \rho) & = & (\forall x : U) \\
 & & \quad \alpha_{\Sigma'}\; ((x \;\texttt{in}\; e) \;\texttt{implies}\; \rho)
\end{array}$$

Where η is defined as follows :

$$\begin{array}{lcl}
\eta_V(p) & = & p(V), p \in ((S_\Sigma)_{All} \cup (R_\Sigma)_w) \\
\eta_V(x) & = & x = V, x \in X_\Sigma \\
\eta_V(\hat{\ } r) & = & (\exists n : Nat)\, t_r(n, V) \\
\eta_V(\sim e) & = & \eta_{V'}(e), \text{ such that } V' = (v_n, \ldots, v_1) \\
 & & \text{for } V = (v_1, \ldots, v_n) \\
\eta_V(e + e') & = & \eta_V(e) \vee \eta_V(e') \\
\eta_V(e \;\texttt{-}\; e') & = & \eta_V(e) \wedge \neg \eta_V(e') \\
\eta_V(e \,\&\, e') & = & \eta_V(e) \wedge \eta_V(e') \\
\eta_V(e \;\texttt{->}\; e') & = & \eta_{V'}(e) \wedge \eta_{V''}(e'), \text{ such that} \\
 & & V' = (v_1, \ldots, v_n) \text{ is a prefix of } V \\
 & & \text{where } n = |e|, \text{ and} \\
 & & V'' = (v_{n+1}, \ldots, v_m) \text{ is a suffix of } V \\
 & & \text{where } (m - n) = |e'| \\
\eta_V(e \,.\, e') & = & (\exists y : U)\eta_{(V',y)}(e) \wedge \eta_{(y,V'')}(e'), \\
 & & \text{such that } V' = (v_1, \ldots, v_n) \text{ is a prefix} \\
 & & \text{of } V \text{ where } n + 1 = |e|, \text{ and} \\
 & & V'' = (v_{n+1}, \ldots, v_m) \text{ is a suffix of } V, \\
 & & \text{where } (m - n + 1) = |e'|
\end{array}$$

Model transformation. Given a signature $\Sigma \in |Sign^{\mathcal{A}}|$, where $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma)$, function $\beta_\Sigma : Mod^{\mathcal{CASL}}(\Phi(\Sigma)) \to Mod^{\mathcal{A}}(\Sigma)$ is defined as

$$\begin{array}{lcl}
|\beta_\Sigma(M)| & = & |M_U|, \text{ where } |M_U| \text{ is the carrier of } U \text{ in } M \\
\beta_\Sigma(M)_p & = & M_p, \text{ for } p \in ((S_\Sigma)_{All} \cup (R_\Sigma)_w \cup X_\Sigma)
\end{array}$$

Lemma 2. *The construction (Φ, α, β) detailed in this section defines a conservative comorphism from the institution $\mathcal{A}$, corresponding to* ALLOY *underlying logical system, to a presentation $\mathcal{CASL}$ of* CASL.

Proof. See the accompanying technical report [9].

□

5. Alloy and Hets at work

5.1. An introduction to DCR graphs

DCR graphs, short for *Distributed Condition Response Graphs*, were introduced in [5] to specify workflow models in an implicit way through a number of conditions. A functional style and precise semantics make DCR graphs excellent candidates for modelling critical workflows.

Formally, a DCR graph consists of a set E of events and two relations `condition`, `response` $\subseteq E \times E$ which restrict control flow, regarded as a sequence of event executions. In detail,

- $(e, e') \in$ `condition` iff e' can only be executed after e;
- $(e, e') \in$ `response` iff whenever e is executed the control flow may only come to terminal configuration after the execution of e'.

A mark, or execution state, in a DCR G, is a tuple $(Ex, Res) \in \mathbb{P}(E) \times \mathbb{P}(E)$, where Ex is the set of the events that already occurred and Res the set of events scheduled for execution. A valid execution step in G is a triple (M, M', e) where $M, M' \in \mathbb{P}(E) \times \mathbb{P}(E)$ and $e \in E$ such that, for $M = (Ex, Res), M' = (Ex', Res')$,

1. $\{e' | \texttt{condition}(e', e)\} \subseteq Ex$
2. $Ex' = Ex \cup \{e\}$
3. $Res' = (Res \backslash \{e\}) \cup \{e' | response(e, e')\}$

Mukkamala [13] suggests a translation of DCR graphs to PROMELA so that the specification of workflows can be checked with the SPIN model checker. The encoding, however, is not easy. For example, the language has only arrays as a basic data structure, thus events and relations have to be encoded as arrays, relations becoming two-dimensional bit arrays. Moreover, SPIN based verification is limited by possible state explosion.

An encoding into ALLOY, on the other hand, seems an attractive alternative. Not only it comes out rather straightforwardly, due to the original relational definition of DCR graphs, but also the ALLOY analyser is eager to avoid potential state space explosion by restricting itself to bounded domains. This restricts, of course, the scope of what can be verified in a specification. However, as illustrated below, ALLOY plugged into the HETS family offers a really interesting alternative to the verification of DCR based workflows.

5.2. DCR graphs in Alloy

DCR graphs are encoded in ALLOY as follows,

```
abstract sig Event {
    condition : set Event,
    response : set Event
}

sig Mark {
    executed : set Event,
    toBeExecuted : set Event,
    action : set Mark −> set Event
}

fact {
    all m,m' : Mark, e : Event |
        (m −> m' −> e) in action <=>
            (condition.e in m.executed and
            m'.executed = m.executed + e and
            m'.toBeExecuted = (m.toBeExecuted - e) + e.response )
}
```

This includes the declaration of two kinds (`sig`), one of events and another to define markings. Relations are declared in an object oriented style as fields of kinds (objects). For example, what the declaration of `action` entails is, as expected, a subset of the product Mark × Mark × Event. Finally note how the invariant for valid execution steps is directly captured in the `fact` above. Other DCR properties can be directly checked in ALLOY. For example,

```
all m,m' : Mark, e : Event |
    (m −> m' −> e) in action and e in m'.toBeExecuted
        implies e in e.response
```

formalises the claim that ‘after executing an event e, if in the next mark e is still to be executed, then `response` contains a reflexive pair at e”.

Of course, this property cannot be proved in ALLOY for an arbitrary domain. To do it another member of the ’family has to be called, provided ALLOY is already plugged into the wider HETS network. Applying the comorphism defined in the previous section we get the following encoding of the property in CASL:

```
forall m : U . Mark(m) =>
forall m' : U . Mark(m') =>
forall e : U . Event(e) =>
 (forall v1,v2,v3 : U . v1 = m /\ v2 = m' /\ v3 = e =>
    action(v1,v2,v3)) /\
 (forall v : U . v = e => exists y : U . y = m' /\
    toBeExecuted(y,v)) =>
    (forall v : U . v = e => exists y : U . y = e /\ response(y,v))
```

which, after a few reduction steps simplifies to

```
forall m,m',e : U .
    Mark(m) /\ Mark(m') /\ Event(e) =>
        (action(m,m',e) /\ toBeExecuted(m',e) => response(e,e))
```

which is can then be verified by the SPASS theorem prover.

5.3. A medical workflow

Consider now the following example of a DCR graph representing a medical workflow as introduced in [13]. It concerns the administration of a medicine to a patient. The workflow diagram obtained from the ALLOY analyser is depicted in Fig. 2.

As mentioned in the introduction, ALLOY may give a false sense of security as the scope set for a simulation session may not be wide enough to produce a counter example. To illustrate this situation consider the following property in which we assume transRun = ^(action.Event). In English it reads: “starting with an empty mark $(\emptyset, \emptyset)$, if by continuously executing events a mark is reached where SecEffect was executed and no further events are to be executed, then this mark has no executed events”. In ALLOY,

```
all m,m' : Mark |
    (no m.(executed+toBeExecuted) and
    m' in m.transRun and
    SecEffect in m'.executed and
    no m'.toBeExecuted)
        implies no m'.executed
```

An analysis of the workflow diagram shows the property is false. Actually, if the left side of the implication is true, it may happen that the right hand side is false: the former says there are executed events while the latter contradicts it. The ALLOY analyser, however, is unable to find a counter-example within a scope below 15 (recall the default scope is 3). The problem of this, is that with a scope smaller than 15 (10 marks + 5 events) the ALLOY analyser can never reach a mark where the left side of the implication is true, and therefore no counter examples are found.

On the other hand, after encoding into CASL and calling another prover in the HETS network, such as VAMPIRE, the result pops out in a few seconds. A HETS session for

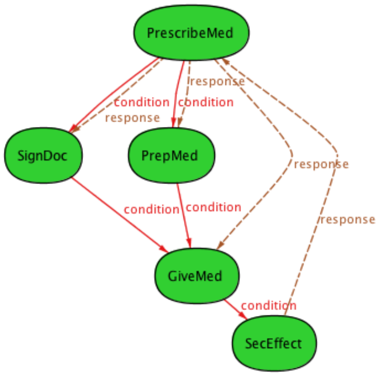


Figure 2. A medical workflow diagram

this example is reproduced in Fig. 3. In general the ALLOY analyser has difficulties when dealing with similar properties and diagrams with just two more events. In some cases the search, if successful, may exceed 30 minutes.

We have checked several other properties[3] using both ALLOY, with scope 15, and an automatic theorem prover available in HETS, namely SPASS and EPROVER, through the encoding proposed in this paper. The experimental results seem to confirm the advantages of the hybrid approach proposed here, with automatic theorem provers taking the job whenever ALLOY is unable to proceed or requires an excessive processing time. In some cases, namely when dealing with encodings of ALLOY models that make heavy use of transitive closure, another member of the HETS network — an interactive theorem prover — has to be called.

6. Discussion and conclusions

As suggested by its title, this paper is an attempt to *give* ALLOY *a family*. I.e., a first step towards a methodology for modelling and validating software designs in which ALLOY is integrated into a network of logical tools rather than connected, once and for all, to a single one.

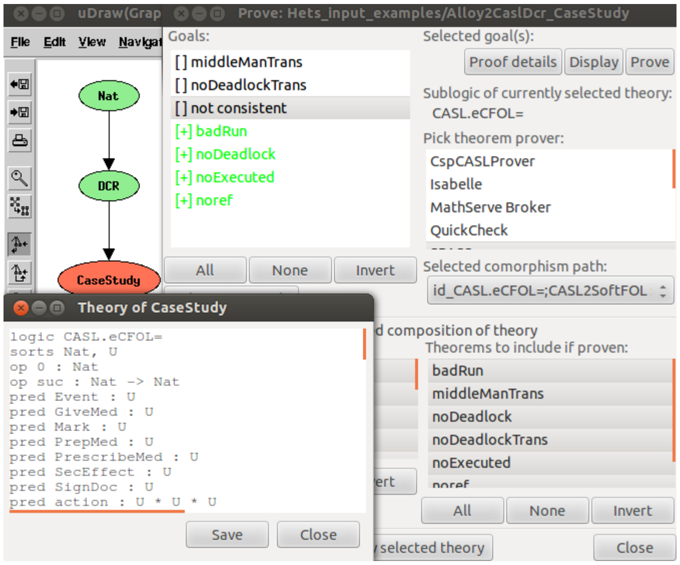
Going generic has, as one could expect, a price to be paid. In our case, this was the development of a proper formalisation of the ALLOY logical system as an institution, together with a conservative comorphism from it into an institution of presentations over CASL as an entry point in the HETS network. These two results are the main technical contributions of this paper. They are stated in lemmas 1 and 2, whose proofs were omitted due to strict page limits but can be found in [9], available from `github.com/nevrenato/IRI_FMI_Annex`.

Adopting an institutional framework brings to scene a notational burden the working software engineer may find hard to bear. It should be noted, however, this is done once and for all: our results, once proved, provide a simple method to translate ALLOY models into CASL specifications. In applications there is no need to recall how the underlying construction was formulated.

On the other hand, following this path has a number of advantages. First of all this is a sound way to integrate systems based on a formal relationship between their underlying logical systems. This contrasts with *ad hoc* combinations, often attractive at first sight but not always consistent, which abound in less careful approaches to Software Engineering. A second advantage concerns the possibility of, once an institutional representation for ALLOY is obtained, combining it with other logical systems through a number of techniques available in the institutional framework. For example, in [10] we have developed a systematic way to build a hybrid logic layer on top of an arbitrary institution. Hybrid logic [2] adds to the modal description of transition structures the ability to refer to specific states, which makes it a suitable language to describe properties of individual states in any sort of structured transition system. A typical application of this method discussed in [8] is the design of reconfigurable systems, where each state corresponds to an execution configuration and transitions are labelled by triggers. The institutional rendering of ALLOY makes possible, the hybridisation of its models and their integration in the development cycle of reconfigurable software.

A second motivation was defining a tool chain for the validation of workflows represented by DCR graphs. Results obtained so far suggest that ALLOY, suitably integrated into a wider network of theorem provers, provides an intuitive alternative to the PROMELA formalisation presented in [13]. More experimental work, however, is neces-

[3] Full models at `github.com/nevrenato/IRI_FMI_Annex`.



Goals:

[] middleManTrans
[] noDeadlockTrans
[] not consistent
[+] badRun
[+] noDeadlock
[+] noExecuted
[+] noref

Selected goal(s):

Sublogic of currently selected theory:
CASL.eCFOL=

Pick theorem prover:
CspCASLProver
Isabelle
MathServe Broker
QuickCheck

Selected comorphism path:
id_CASL.eCFOL=;CASL2SoftFOL

d composition of theory
Theorems to include if proven:
badRun
middleManTrans
noDeadlock
noDeadlockTrans
noExecuted
noref

```
logic CASL.eCFOL=
sorts Nat, U
op 0 : Nat
op suc : Nat -> Nat
pred Event : U
pred GiveMed : U
pred Mark : U
pred PrepMed : U
pred PrescribeMed : U
pred SecEffect : U
pred SignDoc : U
pred action : U * U * U
```

Figure 3. A Hets session.

sary to substantiate this claim on general grounds.

Acknowledgements: This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through FCT, the Portuguese Foundation for Science and Technology, project FCOMP-01-0124-FEDER-028923, and by CIDMA - Universidade de Aveiro, project FCOMP-01-0124-FEDER-022690.

References

[1] K. Arkoudas, S. Khurshid, D. Marinov, and M. Rinard. Integrating model checking and theorem proving for relational reasoning. In *7th Inter. Seminar on Relational Methods in Computer Science (RelMiCS 2003)*, volume 3015 of *Lecture Notes in Computer Science*, pages 21–33, 2003.

[2] T. Brauner. *Hybrid Logic and its Proof-Theory*. Applied Logic Series. Springer, 2010.

[3] R. Diaconescu. *Isntitution-independent Model Theory*. Series in Universal Logic. Birkhauser, 2008.

[4] J. A. Goguen and R. M. Burstall. Institutions: abstract model theory for specification and programming. *J. ACM*, 39:95–146, January 1992.

[5] T. T. Hildebrandt and R. R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Proc. 3rd PLACES Workshop*, volume 69 of *EPTCS*, pages 59–73, 2010.

[6] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[7] N. Macedo and A. Cunha. Automatic unbounded verification of Alloy specifications with Prover9. *CoRR*, abs/1209.5773, 2012.

[8] A. Madeira, J. M. Faria, M. A. Martins, and L. S. Barbosa. Hybrid specification of reactive systems: An institutional approach. In G. Barthe, A. Pardo, and G. Schneider, editors, *Proc. 9th International Conference on Software Engineering and Formal Methods (SEFM 2011)*, volume 7041 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 2011.

[9] A. Madeira, R. Neves, M. A. Martins, and L. S. Barbosa. Giving Alloy a family - the proofs. TR-HASLab:01:2013, HASLab - INESC TEC and Universidade do Minho, 2013.

[10] M. A. Martins, A. Madeira, R. Diaconescu, and L. S. Barbosa. Hybridization of institutions. In A. Corradini, B. Klin, and C. Cirstea, editors, *4th Inter. Conf. on Algebra and Coalgebra in Computer Science*, volume 6859 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2011.

[11] T. Mossakowski, A. Haxthausen, D. Sannella, and A. Tarlecki. CASL: The common algebraic specification language: Semantics and proof theory. *Computing and Informatics*, 22:285–321, 2003.

[12] T. Mossakowski, C. Maeder, and K. Lüttich. The heterogeneous tool set (Hets). In *Proc. 4th Intern. Verification Workshop (VERIFY)*, volume 259 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[13] R. R. Mukkamala. *A Formal Model For Declarative Workflows : Dynamic Condition Response Graphs*. PhD thesis, IT University of Copenhagen, 2012.

[14] M. Ulbrich, U. Geilmann, A. A. E. Ghazi, and M. Taghdiri. A proof assistant for alloy specifications. In C. Flanagan and B. König, editors, *Proc. 18th Inter. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 7214 of *Lecture Notes in Computer Science*, pages 422–436. Springer, 2012.