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. Semantic 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 comorphisms1.

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-

1The background section below provides a brief introduction to institutions as canonical representatives of logical systems.
Figure 1. “Plugging” ALLOY into the HETS network

... (Sign^I, Sen^I, Mod^I, (|=^I_{Σ}|_{Sign^I}))

where

- Sign^I is a category of signatures and signature morphisms
- Sen^I : Sign^I → Set, is a functor relating signatures to the corresponding sentences
- Mod^I : (Sign^I)op → C, is a functor, giving for each signature Σ, the category of its models
- |=^I_{Σ} ⊆ |Mod^I(Σ)| × Sen^I(Σ), is the satisfaction relation between models and sentences such that, for each morphism ϕ : Σ → Σ' in Sign^I, and for any M' ∈ |Mod^I(Σ')| and ρ ∈ Sen^I(Σ),
  
  \( M' \models_{Σ} Sen^I(ϕ)(ρ) \) \text{ iff } Mod^I(ϕ)(M') \models_{Σ'} ρ

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

- a functor, Φ : Sign^I → Sign^I'
- a natural transformation, α : Sen^I → Sen^I'.Φ
- a natural transformation, β : Mod^I'.Φ^op → Mod^I

such that, for any Σ ∈ |Sign^I|, M' ∈ |Mod^I'(Φ(Σ))| and ρ ∈ Sen^I(Σ),

\( β_{Σ}(M') \models_{Σ} ρ \) \text{ iff } M' \models_{Φ(Σ)} α_{Σ}(ρ)

A comorphism is conservative whenever, for any Σ ∈ |Sign^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 \( \Sigma \in |\text{Sign}\Sigma| \), to pairs \( (\Sigma, \Gamma) \), where \( \Gamma \subseteq \text{Sen}^2(\Sigma) \), signature morphisms to presentation morphisms and restricting models \( M \in |\text{Mod}^2(\Sigma)| \) to the ones in which \( \Gamma \) is satisfied, i.e., such that \( M \models^2 \Gamma \) (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, 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 \( \mathcal{A} = (\text{Sign}^{\mathcal{A}}, \text{Sen}^{\mathcal{A}}, \text{Mod}^{\mathcal{A}}, \models^{\mathcal{A}}) \) 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, \( S = \{S_t\}_{t \in \{\text{Abs}, \text{Non}, \text{One}\}} \). \( S_{\text{Abs}} \) represents all kinds, \( S_{\text{Non}} \) the abstract ones, \( S_{\text{Non}} \) the non-empty ones, and \( S_{\text{One}} \) the kinds containing exactly one element.
- \( m : S_{\text{Abs}} \to S_{\text{Abs}} \) is a function that returns the ancestor of a given kind, i.e., \( m(s) = s' \) means that \( s' \) is the parent of \( s \). Top level kinds are considered the ascendants of themselves, and therefore, \( m \) takes the form of a forest structure.
- A family of relational symbols \( R = \{R_w\}_{w \in S_{\text{Abs}}} \).
- A set of singleton relational symbols \( X \), representing the variables declared on quantified expressions. Despite being the same as the elements in \( S_{\text{One}} \), 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 \( S_t \subset S \), if \( s \in S_t \) then \( \varphi_s(s) \in S'_t \), and the following diagram commutes:

\[
\begin{array}{ccc}
S & \xrightarrow{\varphi_s} & S' \\
m \downarrow & & \downarrow m' \\
S & \xrightarrow{\varphi} & S'
\end{array}
\]

- \( \varphi_r \) is a family of functions such that, \( \varphi_r = \{\varphi_w : R_w \to R'_w\}_{w \in S_{\text{Abs}}} \)
- \( \varphi_v : X \to X' \) is a function.

\(^2\)The corresponding encoding an extra relation is added for each binary one as the transitive closure of the latter.
Sentences. Given a signature $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma) \in |Sign^A|$, the set of expressions $Exp(\Sigma)$ is the smallest set containing
\[
\begin{align*}
p, & \quad p \in (S_\Sigma)_{All} \cup (R_\Sigma)_w \cup X_\Sigma \\
r, & \quad r \in (R_\Sigma)_w, \text{ and } |w| = 2 \\
\sim e, & \quad e \in Exp(\Sigma) \\
e \to e', & \quad e, e' \in Exp(\Sigma), \\
e \cap e', & \quad e, e' \in Exp(\Sigma), \\
e \cdot e', & \quad e, e' \in Exp(\Sigma), \text{ and } |e| + |e'| > 2
\end{align*}
\]
with $|e|$ standing for the length of expression $e$.

Finally, the set of sentences, $Sen^A(\Sigma)$, is the smallest one containing:
\[
\begin{align*}
e \in e' & \quad e, e' \in Exp(\Sigma), \text{ and } |e| = |e'| \\
\text{not} \ \rho & \quad \rho \in Sen^A(\Sigma) \\
\rho \text{ implies } \rho' & \quad \rho, \rho' \in Sen^A(\Sigma) \\
(\text{all } x : e) \ \rho & \quad e \in Exp(\Sigma), \rho \in Sen^A(\Sigma), |e| = 1
\end{align*}
\]
where $\Sigma' = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma + \{x\})$.

Models. For each $(S, m, R, X) \in |Sign^A|$, a model $M \in |Mod^A((S, m, R, X))|$ has
\begin{itemize}
\item A carrier set $|M|
\item An unary relation $M_s \subseteq |M|$, for each $s \in S_{All}$
\item A relation $M_r \subseteq M_{s_1} \times \cdots \times M_{s_n}$, for each $r \in R_{s_1 \ldots s_n}$
\item A singleton relation, $M_x \subseteq |M|$, for each $x \in X$
\end{itemize}
and satisfies the following axioms, for all $s, s' \in S_{All}$,
\begin{enumerate}
\item $M_s \subseteq M_{m(s)}$
\item if $s \in S_{Som}$, then $M_s \neq \emptyset$
\item if $s \in S_{One}$, then $\#M_s = 1$
\item if $s \in S_{Abs}$, then $M_s \subseteq \bigcup_{q \in m^>(s)} M_q$
\item if $s, s'$ are not related by the transitive closure of $m$, then $M_s \cap M_{s'} \subseteq \emptyset$
\end{enumerate}

Evaluation of expressions is as follows:
\[
\begin{align*}
M_{- e} & = (M_e)^0 \\
M_{e + e'} & = M_e + M_{e'} \\
M_{e \cdot e'} & = M_e \cdot M_{e'} \\
M_{e \cap e'} & = M_e \cap M_{e'} \\
M_{e \cdot e'} & = M_e \cdot e' \\
M_{e \cdot e'} & = M_e \cdot e' \\
M_{r - e'} & = M_{r - e'} \\
M_{r - s} & = \bigcup_{n \in N} M_{r^n}, \text{ such that } M_{r^0} = M_r \\
\text{and } M_{r^{n+1}} = (M_r \cdot M_{r^n})
\end{align*}
\]

Each signature morphism, $\Sigma \xrightarrow{\varphi} \Sigma' \in |Sign^A|$, is mapped to $Mod^A(\varphi) : Mod^A(\Sigma) \to Mod^A(\Sigma)$, giving, for each $M' \in |Mod^A(\Sigma')|$, its $\varphi$-reduct, $M' \mid \varphi \in |Mod^A(\Sigma)|$ defined by:
\[
\begin{align*}
|\{M' \mid \varphi\}| & = |M'| \\
(M' \mid \varphi)_s & = M'_{\varphi(s)}, \text{ for any } s \in (S_\Sigma)_{All} \\
M'_{\varphi(r)} & = M'_{\varphi(r)}, \text{ for any } r \in (R_\Sigma)_w \\
M'_{\varphi(x)} & = M'_{\varphi(x)}, \text{ for any } x \in X_\Sigma
\end{align*}
\]

Satisfaction. Given a $\Sigma$-model $M$, for $\Sigma \in |Sign^A|$, the satisfaction relation is defined for each $\Sigma$-sentence as follows:
\[
\begin{align*}
M \models^A e \in e' & \quad \text{iff } M_e \subseteq M_{e'} \\
M \models^A \text{ not } \rho & \quad \text{iff } M \models^A \rho \\
M \models^A \rho \text{ implies } \rho' & \quad \text{iff } M \models^A \rho \\
& \quad \text{whenever } M \models^A \rho \\
M \models^A (\text{all } x : e) \rho & \quad \text{iff } M' \models^A (x \in e) \implies \rho
\end{align*}
\]
for all model expansions $M'$ of $M$, by the corresponding inclusion morphism.

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

Proof. See the accompanying technical report [9].

\[
\square
\]

4. From Alloy to CASL

This section characterises a conservative combination 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 $\Gamma$, thereby restricting the class of available models. An object in the category $Sign_{CASL}$ 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 $P$ a family of relational symbols also indexed by their arity. Then, we define

Signature functor. For any signature $(S, m, R, X) \in |Sign^A|$, $\Phi$ gives a tuple $((S', TF, PF, P), \Gamma)$ where
\[
\begin{align*}
S' & = \{U, Nat\} \\
TF & = \{\{0\}_Nat, \{\text{su}c\}_Nat\to Nat, \{x | x \in X \to U\} \\
PF & = \emptyset \\
P & = \{(s | s \in S_{All} | U, \{r | r \in R_{s_1, \ldots, s_n} \}_U, \ldots, u_n, \{t_r | r \in R_{s_1, s_2} \}_Nat, U\}
\end{align*}
\]
and $\Gamma$ is the smallest set containing the following axioms:
Sentence transformation. Given any signature 
\[ \Sigma \in |Sign^A| , \text{ where } \Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma) \], function \( \alpha_\Sigma : Sen^A(\Sigma) \rightarrow Sen^CASL(\Phi(\Sigma)) \) is defined by

\[
\alpha_\Sigma(e \ in \ e') = (\forall V : U) \eta_V(e) \Rightarrow \eta_V(e'), \quad \text{such that } V = (v_1, \ldots, v_n), \text{ and } n = |e| \\
\alpha_\Sigma(\text{not } \rho) = \neg \alpha_\Sigma(\rho) \\
\alpha_\Sigma(\rho \ implies \ \rho') = \alpha_\Sigma(\rho) \ implies \ \alpha_\Sigma(\rho') \\
\alpha_\Sigma((\text{all } x : e) \ \rho) = (\forall x : U) \alpha_\Sigma((x \ in \ e) \ implies \ \rho)
\]

Where \( \eta \) is defined as follows:

\[
\eta_V(p) = p(V), p \in (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma) \\
\eta_V(x) = x \in V, x \in X_\Sigma \\
\eta_V(\bot) = (\exists n : Nat) t_r(n, V) \\
\eta_V(- e) = \eta_{V'}(e), \text{ such that } V' = (v_n, \ldots, v_1) \\
\eta_V(e + e') = \eta_V(e) \land \eta_V(e') \\
\eta_V(e - e') = \eta_V(e) \land \neg \eta_V(e') \\
\eta_V(e \ & e') = \eta_V(e) \land \eta_V(e'), \text{ such that } V' = (v_1, \ldots, v_n) \\
\eta_V(e \Rightarrow e') = \eta_{V'}(e) \land \eta_{V''}(e'), \text{ such that } V'' = (v_n, \ldots, v_m) \text{ is a prefix of } V \\
\eta_V(e \cdot e') = (\exists y : U) \eta_{V', y}(e) \land \eta_{V}(y, V'')(e'), \text{ such that } V' = (v_1, \ldots, v_n) \text{ is a prefix of } V \text{ where } n = |e'| \\
\eta_V(e \cdot e') = (\exists y : U) \eta_{V', y}(e) \land \eta_{V}(y, V'')(e'), \text{ such that } V'' = (v_{n+1}, \ldots, v_m) \text{ is a suffix of } V \text{ where } (m - n + 1) = |e'|
\]

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

\[
|\beta_\Sigma(M)| = |M_U|, \text{ where } |M_U| \text{ is the carrier of } U \text{ in } M \\
\beta_\Sigma(M)_p = M_p, \text{ for } p \in (\langle S_\Sigma \rangle, (R_\Sigma)_w \cup X_\Sigma)
\]

**Lemma 2.** The construction \((\Phi, \alpha, \beta)\) detailed in this section defines a conservative comorphism from the institution \(A\), corresponding to ALLOY underlying logical system, to a presentation CASL of CASL.

**Proof.** See the accompanying technical report [9].

\[ \square \]

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,

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

A mark, or execution state, in a DCR \( G \), is a tuple \( (Ex, Res) \in P(E) \times P(E) \), 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 \( M, M' \in P(E) \times P(E) \) and \( e \in E \) such that, for \( M = (Ex, Res), M' = (Ex', Res') \),

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

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 c : U . Mark(m) =>
forall c : U . Mark(m') =>

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 counterexample 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
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\footnote{Full models at github.com/nevrenato/IRI_FMI_Annex.} 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 \textit{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-
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