An institution for Alloy and its translation to second-order logic

Renato Neves[1], Alexandre Madeira[2], Manuel Martins[3], and Luís Barbosa[1]

[1] INESC TEC (HASLab) & Univ. Minho
{nevrenato@gmail.com, lsb@di.uminho.pt}
[2] INESC TEC (HASLab) & Univ. Minho and Dep. Mathematics, Univ. Aveiro
madeira@ua.pt
[3] Center for Research and Development in Mathematics and Applications - Dep. Mathematics, Univ. Aveiro
martins@ua.pt

Abstract. Lightweight formal methods, of which Alloy is a prime example, combine the rigour of mathematics without compromising simplicity of use and suitable tool support. In some cases, however, the verification of safety or mission critical software entails the need for more sophisticated technologies, typically based on theorem provers. This explains a number of attempts to connect Alloy to specific theorem provers documented in the literature. This paper, however, takes a different perspective: instead of focusing on one more combination of Alloy with still another prover, it lays out the foundations to fully integrate this system in the Hets platform which supports a huge network of logics, logic translators and provers. This makes possible for Alloy specifications to "borrow" the power of several, non dedicated proof systems. The paper extends the authors' previous work on this subject by developing in full detail the semantical foundations for this integration, including a formalisation of Alloy as an institution, and introducing a new, more general translation of the latter to second-order logic.

Keywords: Model finding, theorem proving, second–order logic.

1 Introduction

In [17] the authors discussed the integration of Alloy [9] in Hets platform of logics, logic translators and provers, by scketching its formalisation as an institution [7, 6] and defining its encoding into Casl [14], an extension of multi-sorted first order logic with partiality and free types. The motivation was clear: to offer a systematic way to connect Alloy to a huge network of logics and logical systems in order to complement the model finder strategies of the former with suitable theorem provers already linked into the latter.

Actually, Alloy, based on a single sorted relational logic whose models can be automatically tested with respect to bounded domains, is a most successful tool for the working software engineer. Its simple but powerful language combined with an analyser which can promptly give counter-examples depicted graphically, makes it increasingly popular both in academia and industry: Successful stories report on the discovery of faults in software designs, supposedly faultless, by taking advantage of ALLOY. 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 our research claims the use of ALLOY should be framed into broader toolchains involving more general, even if often less friendly, theorem provers. In such a toolchain properties can be first tested within the ALLOY analyser; if no counter-examples are found, a theorem prover would be asked to generate a proof, at least in what concerns critical design fragments.

There is a number of results on connecting ALLOY to specific theorem provers. The perspective taken in [17], however, and complemented in this paper goes a step further by "plugging" ALLOY into the HETS network, providing a number of effective and sound connections to several logical systems and tools at once. Currently, HETS integrates several world-class reasoners, namely ISABELLE [18], LEO-II [4], SPASS [21], VAMPIRE [19], DARWIN [2], among many others. Plugging ALLOY to HETS, makes thus possible the translation of its models to a number of languages in the network, and naturally, borrowing for free the corresponding proof support. Experiments can then be carried out in different tools, typically tuned to specific application areas.

Actually, HETS [15] may be regarded 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* [6] and logic translations as *comorphisms*. This is the price to be paid: the integration of ALLOY in this network entails the need for formalising ALLOY as an institution and providing a translation to a relevant logic in the HETS network.

The present paper addresses this challenge by developing in full detail the semantical foundations for this integration, including a formalisation of ALLOY as an institution, and introducing a new, more general translation of it to second-order logic (SOL) [12]. This new translation allows a more natural embedding of ALLOY, when compared to the previous, essentially first-order translation to CASL introduced in [17]. Moreover, this translation establishes a connection between ALLOY and higher-order provers, such as LEO-II and ISABELLE, using the logic THF [3] which integrates second-order features. The original translation to CASL, presented in [17], is re-framed in this context, because, in practice, it opens doors to a broad number of theorem provers, namely for first-order systems. Both translations are depicted in Fig. 1 as dashed arrows.

Related work. The idea of connecting ALLOY to a theorem prover is not new — see, for example, references [20, 10, 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, [20], one of the most recent proposals

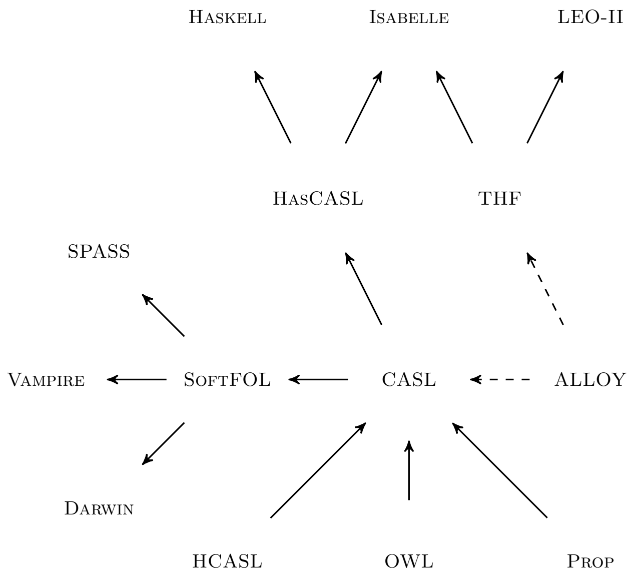


Fig. 1. HETS sub–network extended with ALLOY

in this trend, translates models into a first-order dialect supported by the KEY theorem prover.

Paper structure. The formalisation of ALLOY as an institution and the definition of suitable comorphisms is presented in sections 3, 4 and 5. Before that, in section 2, a brief overview of the theory of institutions is provided as a background for the paper. Section 6 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 7 concludes.

2 Background: Institutions

2.1 Institutions and comorphisms

An *institution* [7] 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 computing science as possible in a general and uniform way, independently of particular logical systems. This has now been achieved to an extent even greater than originally thought, as the theory of *institutions* became the most fundamental mathematical theory underlying the algebraic specification discipline.

Definition 1. *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,*
- $Sen^{\mathcal{I}} : Sign^{\mathcal{I}} \to \mathbb{S}et$ *is a functor relating signatures to the corresponding sentences, where* $\mathbb{S}et$ *is the category of Sets,*
- $Mod^{\mathcal{I}} : (Sign^{\mathcal{I}})^{op} \to \mathbb{C}at$ *is a functor giving for each signature* Σ*, the category of its models, where* $\mathbb{C}at$ *is the category of categories,*
- $\models^{\mathcal{I}}_{\Sigma} \subseteq |Mod^{\mathcal{I}}(\Sigma)| \times Sen^{\mathcal{I}}(\Sigma)$ *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* $\rho \in Sen^{\mathcal{I}}(\Sigma)$*,*

$$M' \models^{\mathcal{I}}_{\Sigma'} Sen^{\mathcal{I}}(\varphi)(\rho) \ \textit{ iff } Mod^{\mathcal{I}}(\varphi)(M') \models^{\mathcal{I}}_{\Sigma} \rho$$

The reduct of M' through a signature morphism φ is defined by $Mod^{\mathcal{I}}(\varphi)(M')$, and denoted by $M' \upharpoonright \varphi$. Dually, M' is called a model φ–expansion of $M' \upharpoonright \varphi$.

Example 1. To illustrate the concept of an institution, consider, in this example, the construction of an institution for first-order logic (FOL).

Signatures. $Sign^{\mathrm{FOL}}$ is a category whose objects are triples (S, F, P), where S is the set of sort symbols, F a family of function symbols indexed by their arity, $F = \{F_{w \to s} | w \in S^*, s \in S\}$ and P a family of relational symbols also indexed by their arity, $P = \{P_w | w \in S^*\}$. A signature morphism in this category is also a triple $(\varphi_{st}, \varphi_{op}, \varphi_{rl})$ such that for $\varphi : (S, F, P) \to (S', F', P')$, if $\sigma \in F_{w \to s}$, then $\varphi_{op}(\sigma) \in F'_{\varphi_{st}(w) \to \varphi_{st}(s)}$, and if $\pi \in P_w$ then $\varphi_{rl}(\pi) \in P'_{\varphi_{st}(w)}$.

Sentences. For each signature object $(S, F, P) \in |Sign^{\mathrm{FOL}}|$, $Sen^{\mathrm{FOL}}(S, F, P)$ is the smallest set of first order sentences:

$t \approx t'$, for $t, t' \in terms$
$\pi(t_1, \ldots, t_n)$, for $t_1, \ldots, t_n \in terms$ and $\pi \in P_w$
$\neg\rho$, for $\rho \in Sen^{\mathrm{FOL}}(S, F, P)$
$\rho \Rightarrow \rho'$, $\rho, \rho' \in Sen^{\mathrm{FOL}}(S, F, P)$
$\forall x : s \,.\, \rho$, $s \in S$, $\rho \in Sen^{\mathrm{FOL}}(S, F \uplus \{x\}_{\to s}, P)$

where a term of sorts is a syntactic structure $\sigma(t_1, \ldots, t_n)$, such that $\sigma \in F_{s_1,\ldots,s_n \to s}$ and $t_1, \ldots, t_n$ are terms of sort $s_1, \ldots, s_n$, respectively. A signature morphism φ defines a term translation function $terms(\varphi)$, given by

$$terms(\varphi)(\sigma(t_1, \ldots, t_n)) = \varphi_{op}(\sigma)(terms(\varphi)(t_1), \ldots, terms(\varphi)(t_n)).$$

Given a signature morphism φ in $Sign^{\mathrm{FOL}}$, sentences are mapped in the following way:

$$\begin{aligned}
Sen^{\mathrm{FOL}}(\varphi)(t \approx t') &= terms(\varphi)(t) \approx terms(\varphi)(t') \\
Sen^{\mathrm{FOL}}(\varphi)(\pi(t_1, \ldots, t_n)) &= \varphi_{rl}(\pi)(terms(\varphi)(t_1), \ldots, terms(\varphi)(t_n)) \\
Sen^{\mathrm{FOL}}(\varphi)(\neg\rho) &= \neg Sen^{\mathrm{FOL}}(\varphi)(\rho) \\
Sen^{\mathrm{FOL}}(\varphi)(\rho \Rightarrow \rho') &= Sen^{\mathrm{FOL}}(\varphi)(\rho) \Rightarrow Sen^{\mathrm{FOL}}(\varphi)(\rho') \\
Sen^{\mathrm{FOL}}(\varphi)(\forall x : s \,.\, \rho) &= \forall x : \varphi_{st}(s) \,.\, Sen^{\mathrm{FOL}}(\varphi')(\rho), \\
&\quad \text{where } \varphi' \text{ canonically extends } \varphi \text{ with } \varphi'_{op}(x) = x
\end{aligned}$$

Models. For each signature $(S, F, P) \in |Sign^{\mathrm{FOL}}|$, $Mod^{\mathrm{FOL}}(S, F, P)$ is a category whose objects are models with the following components : a carrier set $|M_s|$, for each $s \in S$; a function $M_\sigma : |M_w| \to |M_s|$, for each $\sigma \in F_{w \to s}$; a relation $M_\pi \subseteq |M_w|$, for each $\pi \in P_w$.

For any signature morphism $\varphi : (S, F, P) \to (S', F', P')$, and any (S', F', P')–model M', $Mod^{\mathrm{FOL}}(\varphi)(M')$, or $M' \restriction \varphi$, is defined as:

- for any $s \in S$, $|(M' \restriction \varphi)_s| = |M'_{\varphi_{st}(s)}|$
- for any $\sigma \in F_{w \to s}$, $(M' \restriction \varphi)_\sigma = M'_{\varphi_{op}(\sigma)}$
- for any $\pi \in P_w$, $(M' \restriction \varphi)_\pi = M'_{\varphi_{rl}(\pi)}$

Satisfaction. For any Σ–model $M \in |Mod^{\mathrm{FOL}}(\Sigma)|$, with $\Sigma \in |Sign^{\mathrm{FOL}}|$, the satisfaction relation is inductively defined in the following way:

$$\begin{aligned}
M &\models^{\mathrm{FOL}}_{\Sigma} t \approx t' && \text{iff } M_t = M_{t'} \\
M &\models^{\mathrm{FOL}}_{\Sigma} \pi(t_1, \ldots, t_n) && \text{iff } (t_1, \ldots, t_n) \in M_\pi \\
M &\models^{\mathrm{FOL}}_{\Sigma} \neg\rho && \text{iff } M \not\models^{\mathrm{FOL}}_{\Sigma} \rho \\
M &\models^{\mathrm{FOL}}_{\Sigma} \rho \Rightarrow \rho' && \text{iff } M \models^{\mathrm{FOL}}_{\Sigma} \rho' \text{ whenever } M \models^{\mathrm{FOL}}_{\Sigma} \rho \\
M &\models^{\mathrm{FOL}}_{\Sigma} \forall x : s \,.\, \rho && \text{iff for any model } x\text{–expansion } M' \text{ of } M,\ M' \models^{\mathrm{FOL}}_{\Sigma'} \rho
\end{aligned}$$

A comorphism is a mapping that "embeds" a structurally "simpler" institution into a more "complex" one.

Definition 2. *Formally, given two institutions $\mathcal{I}, \mathcal{I}'$, a comorphism from $\mathcal{I}$ to $\mathcal{I}'$ is a triple (Φ, α, β) consisting of*

- *a functor Φ: $Sign^{\mathcal{I}} \to Sign^{\mathcal{I}'}$,*
- *a natural transformation α: $Sen^{\mathcal{I}} \Rightarrow Sen^{\mathcal{I}'}.\Phi$,*
- *a natural transformation β: $Mod^{\mathcal{I}'}.\Phi^{op} \Rightarrow Mod^{\mathcal{I}}$,*

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

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

Definition 3. *Let (Φ, α, β) be a comorphism. We say that (Φ, α, β) is conservative whenever, for each Σ–model M in $\mathcal{I}$, there exists a $\Phi(\Sigma)$–model M' in $\mathcal{I}'$ such that $M = \beta_\Sigma(M')$.*

The notion of conservative comorphism is typically used to "borrow" proof support from an institution in a sound way. In this paper we resort to such notion to complement ALLOY's proof environment by "borrowing" the proof support from other logical systems within HETS.

Definition 4. *Given an* institution $\mathcal{I}$*, one defines the institution of 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* $M \models^{\mathcal{I}}_{\Sigma} \Gamma$*.*

The latter definition (from [6]) 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, as discussed later in the paper, due to a sort of "hidden" rules in semantics of ALLOY, each comorphism defined in sections 4 and 5 does not go to the institution of the target logic, but to the corresponding institution of presentations.

The notion of an amalgamation square is often essential for proving the satisfaction conditions of institutions and comorphisms. This justifies recalling it here.

Definition 5. *A commuting square of functors,*

$$\begin{tikzcd}
A' \arrow[r, "G_2"] \arrow[d, "G_1"'] & A_2 \arrow[d, "F_2"] \\
A_1 \arrow[r, "F_1"'] & A
\end{tikzcd}$$

is a weak amalgamation square *if and only if, for each* $M_1 \in |A_1|$*,* $M_2 \in |A_2|$*, such that* $F_1(M_1) = F_2(M_2)$*, there is an object* $M' \in |A'|$ *such that* $G_1(M') = M_1$ *and* $G_2(M') = M_2$*. When* M' *is unique the amalgamation square is called strong.*

The model amalgamation of an institution consists of the model amalgamation of the diagrams in $Mod^{\mathcal{I}}$ (external square) induced by the pushout of signatures in $Sign^{\mathcal{I}}$ (internal square),

$$\begin{tikzcd}
Mod^{\mathcal{I}}(\Sigma') \arrow[rrr, "Mod^{\mathcal{I}}(\theta_2)"] \arrow[ddd, "Mod^{\mathcal{I}}(\theta_1)"'] & & & Mod^{\mathcal{I}}(\Sigma_2) \arrow[ddd, "Mod^{\mathcal{I}}(\varphi')"] \\
& \Sigma' & \Sigma_2 \arrow[l, "\theta_2"'] & \\
& \Sigma_1 \arrow[u, "\theta_1"] & \Sigma \arrow[l, "\varphi"] \arrow[u, "\varphi'"'] & \\
Mod^{\mathcal{I}}(\Sigma_1) \arrow[rrr, "Mod^{\mathcal{I}}(\varphi)"'] & & & Mod^{\mathcal{I}}(\Sigma)
\end{tikzcd}$$

i.e., for each $M_1 \in |Mod^{\mathcal{I}}(\Sigma_1)|$, $M_2 \in |Mod^{\mathcal{I}}(\Sigma_2)|$, such that $Mod^{\mathcal{I}}(\theta_1)(M_1) = Mod^{\mathcal{I}}(\theta_2)(M_2)$, there is an object $M' \in |Mod^{\mathcal{I}}(\Sigma')|$ called the amalgamation of M_1 and M_2, such that $Mod^{\mathcal{I}}(\varphi)(M') = M_1$ and $Mod^{\mathcal{I}}(\varphi')(M') = M_2$. The square is strong if M' is unique.

Consider comorphism (Φ, α, β). The model amalgamation of β–transformations and functor reducts of $Mod^{\mathcal{I}}$ consists of the weak model amalgamation of the following commutative square,

$$\begin{tikzcd}
Mod'^{\mathcal{I}}(\Phi(\Sigma')) \arrow[r, "\beta_{\Sigma'}"] \arrow[d, "Mod^{\mathcal{I}}(\theta_1)"'] & Mod^{\mathcal{I}}(\Sigma') \arrow[d, "Mod^{\mathcal{I}}(\varphi')"] \\
Mod^{\mathcal{I}'}(\Phi(\Sigma)) \arrow[r, "\beta_{\Sigma}"'] & Mod^{\mathcal{I}}(\Sigma)
\end{tikzcd}$$

i.e., for each $M_\Phi \in |Mod^{\mathcal{I}'}(\Phi(\Sigma))|$, $M' \in |Mod^{\mathcal{I}}(\Sigma')|$, such that $\beta_\Sigma(M_\Phi) = Mod^{\mathcal{I}}(\varphi)(M')$, there is a model $M'_\Phi \in |Mod^{\mathcal{I}}(\Phi(\Sigma'))|$, such that $Mod^{\mathcal{I}'}(\varphi')(M'_\Phi) = M_\Phi$ and $\beta_{\Sigma'}(M'_\Phi) = M'$. When M' is unique, the amalgamation square is called strong.

3 Alloy as an institution

Alloy[9] 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 the *motto* of Alloy 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 *Abstract*, i.e., included in the union of its descendants, *Some*, i.e., required to have at least one element, or *One*, i.e., exactly with one element. The Alloy analyser checks an assertion against a specification by seeking for counter-examples within bounded domains.

In this section we define an institution for Alloy, $\mathcal{A} = (Sign^{\mathcal{A}}, Sen^{\mathcal{A}}, Mod^{\mathcal{A}}, \models^{\mathcal{A}})$. We proceed as follows:

Signatures. Objects in $Sign^{\mathcal{A}}$ are tuples, (S, m, R, X), composed by:

- A family of sets containing kinds and indexed by a type, $S = \{S_t\}_{t\in\{All,Abs,Som,One\}}$. S_{All} is the set of all kinds, S_{Abs} the set of the abstract ones, S_{Som} the non-empty ones, and, finally, S_{One} collects the kinds containing exactly one element. Notice that, for all S_t, $S_t \subseteq S_{All}$.
- $m : S_{All} \to S_{All}$ is a function that gives the parent of each kind, *i.e.*, $m(s) = s'$ means that s' is the parent of s. Top level kinds are considered the parents 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 unary relational symbols X, representing the variable symbols declared on quantified expressions. Despite being the same than the elements in S_{One}, once encoded they must be treated differently.

Morphisms $\varphi : (S, m, R, X) \to (S', m', R', X')$ in this category are triples $\varphi = (\varphi_{kd}, \varphi_{rl}, \varphi_{vr})$ such that:

- $\varphi_{kd} : S \to S'$ is a function that, for any $S_t \in S$, if $\pi \in S_t$ then $\varphi_{kd}(\pi) \in S'_t$, 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_{rl} is a family of functions such that, $\varphi_{rl} = \{\varphi_{kd} : R_w \to R'_{\varphi_{kd}(w)}\}_{w\in(S_{All})^+}$;
- $\varphi_{vr} : X \to X'$ is a function.

Sentences. Consider Exp a functor of the same type of $Sen^{\mathcal{A}}$. 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 one containing

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

where the length $|e|$ of an expression e is computed as follows:

$$\begin{array}{ll}
|\pi| & = |w|, \text{ for } \pi \in (R_\Sigma)_w \\
|\pi| & = 1, \text{ for } \pi \in (S_\Sigma)_{All} \cup X_\Sigma \\
|\hat{}e| & = |e| \\
|\sim e| & = |e| \\
|e \odot e'| & = |e|, \text{ for } \odot \in \{+, -, \&\} \\
|e \;.\; e'| & = (|e| + |e'|) - 2 \\
|e \mathrel{{-}{>}} e'| & = |e| + |e'|
\end{array}$$

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

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

Note that other standard boolean connectives may be built from the above. For instance, the conjunction, denoted in ALLOY with symbol `and`, is usually defined with the implication and negation constructors.

Given a signature morphism $\varphi : \Sigma \to \Sigma'$ in $\mathit{Sign}^{\mathcal{A}}$, expressions and sentences are mapped as follows:

$$\begin{array}{ll}
\mathit{Exp}(\varphi)(\pi) & = \varphi_{kd}(\pi), \text{ for } \pi \in (S_\Sigma)_{\mathit{All}} \\
\mathit{Exp}(\varphi)(\pi) & = \varphi_{rl}(\pi), \text{ for } \pi \in (R_\Sigma)_{\mathit{All}} \\
\mathit{Exp}(\varphi)(\pi) & = \varphi_{vr}(\pi), \text{ for } \pi \in X_\Sigma \\
\mathit{Exp}(\varphi)(\hat{}\,e) & = \hat{}\,\mathit{Exp}(e) \\
\mathit{Exp}(\varphi)(e \texttt{ -> } e') & = \mathit{Exp}(\varphi)(e)\texttt{->}\ \mathit{Exp}(\varphi)(e') \\
\mathit{Exp}(\varphi)(e \odot e') & = \mathit{Exp}(\varphi)(e) \odot \mathit{Exp}(\varphi)(e') \\
\mathit{Exp}(\varphi)(e \,.\, e') & = \mathit{Exp}(\varphi)(e) \,.\, \mathit{Exp}(\varphi)(e')
\end{array}$$

$$\begin{array}{ll}
\mathit{Sen}^{\mathcal{A}}(\varphi)(e\ \texttt{in}\ e') & = \mathit{Sen}^{\mathcal{A}}(\varphi)(e)\ \texttt{in}\ \mathit{Sen}^{\mathcal{A}}(\varphi)(e') \\
\mathit{Sen}^{\mathcal{A}}(\varphi)(\texttt{not}\ \rho) & = \texttt{not}\ \mathit{Sen}^{\mathcal{A}}(\varphi)(\rho) \\
\mathit{Sen}^{\mathcal{A}}(\varphi)(\rho\ \texttt{implies}\ \rho') & = \mathit{Sen}^{\mathcal{A}}(\varphi)(\rho)\ \texttt{implies}\ \mathit{Sen}^{\mathcal{A}}(\varphi)(\rho') \\
\mathit{Sen}^{\mathcal{A}}(\varphi)((\texttt{all}\ \pi : e)\ \rho) & = (\texttt{all}\ x\ : \mathit{Exp}(\varphi)(e))\ \mathit{Sen}^{\mathcal{A}}(\varphi')(\rho), \text{ where } \varphi'
\end{array}$$

canonically expands φ with $\varphi'_{vr}(x) = x$.

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

1. a carrier set $|M|$;
2. an unary relation $M_\pi \subseteq |M|$, for each $\pi \in S_{\mathit{All}}$;
3. a relation $M_\pi \subseteq M_w$, for each $\pi \in R_w$;
4. a singleton relation, $M_\pi \subseteq |M|$, for each $\pi \in X$,

satisfying the following properties for any $\pi, \pi' \in S_{\mathit{All}}$,

1. $M_\pi \subseteq M_{m(\pi)}$
2. if $\pi \in S_{\mathit{Som}}$, then $M_\pi \not\subseteq \emptyset$
3. if $\pi \in S_{\mathit{One}}$, then $\# M_\pi = 1$
4. if $\pi \in S_{\mathit{Abs}}$, then $M_\pi \subseteq \bigcup_{\tau \in m^\circ(\pi)} M_\tau$
5. if π, π' are not related by the transitive closure of m, then $M_\pi \cap M_{\pi'} = \emptyset$

Evaluation of expressions in such models, is done in the following way:

$$\begin{array}{ll}
M_{\sim\, e} & = (M_e)^\circ \\
M_{e \,+\, e'} & = M_e + M_{e'} \\
M_{e \,-\, e'} & = M_e - M_{e'} \\
M_{e \,\&\, e'} & = M_e \cap M_{e'} \\
M_{e \,.\, e'} & = M_e \,.\, M_{e'} \\
M_{e \texttt{ -> } e'} & = M_e \times M_{e'} \\
M_{\hat{}\,e} & = (M_e)^+
\end{array}$$

A signature morphism, $\varphi : \Sigma \to \Sigma'$, 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, $M' \restriction \varphi \in |Mod^{\mathcal{A}}(\Sigma)|$. The latter is defined as follows:

$|(M' \restriction \varphi)| = |M'|$
$(M' \restriction \varphi)_\pi = M'_{\varphi_{kd}(\pi)}$, for any $\pi \in (S_\Sigma)_{All}$
$(M' \restriction \varphi)_\pi = M'_{\varphi_{rl}(\pi)}$, for any $\pi \in (R_\Sigma)_w$
$(M' \restriction \varphi)_\pi = M'_{\varphi_{vr}(\pi)}$, for any $\pi \in X_\Sigma$

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

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

for all model x–expansions M' of M, with Σ' canonically extending Σ with the variable x.

The next step is to prove that $\mathcal{A} = (Sign^{\mathcal{A}}, Sen^{\mathcal{A}}, Mod^{\mathcal{A}}, \models^{\mathcal{A}})$ forms an institution. The proof will proceed through a number of auxiliary results.

Lemma 1. *On the conditions above the commuting diagram below is a strong amalgamation square.*

$$\begin{tikzcd}
(S, m, R, X) \arrow[r, "\varphi"] \arrow[d, hook, "x"'] & (S', m', R', X') \arrow[d, hook, "x^{\varphi}"] \\
(S, m, R, X + \{x\}) \arrow[r, "\varphi'"'] & (S', m', R', X' + \{x\})
\end{tikzcd}$$

Proof. Proof in appendix A.1. □

Lemma 2. *For any signature morphism $\varphi : \Sigma \to \Sigma'$ in $Sign^{\mathcal{A}}$, any Σ–expression e, and any Σ'–model M',*

$$(M' \restriction \varphi)_e = M'_{Exp(\varphi)(e)}$$

Proof. Proof in appendix A.2. □

Finally, we can prove the satisfaction condition for $\mathcal{A}$ and conclude its characterisation as an institution.

Theorem 1. *The satisfaction condition holds for $\mathcal{A}$.*

Proof. Let $\varphi : \Sigma \to \Sigma'$ be a $Sign^{\mathcal{A}}$ morphism, ρ a Σ–sentence, and M' a Σ'–model:

Consider first the case $\rho := e \;\texttt{in}\; e'$:

$$M' \restriction \varphi \models^{\mathcal{A}}_{\Sigma} e \; \texttt{in} \; e'$$

$\Leftrightarrow$ $\{\models$ defn. $\}$

$$(M' \restriction \varphi)_e \subseteq (M' \restriction \varphi)_{e'}$$

$\Leftrightarrow$ $\{$Lemma 2 $\}$

$$M'_{Exp(\varphi)(e)} \subseteq M'_{Exp(\varphi)(e')}$$

$\Leftrightarrow$ $\{\models$ defn. $\}$

$$M' \models^{\mathcal{A}}_{\Sigma'} Exp(\varphi)(e) \; \texttt{in} \; Exp(\varphi)(e')$$

$\Leftrightarrow$ $\{Sen^{\mathcal{A}}$ defn. $\}$

$$M' \models^{\mathcal{A}}_{\Sigma'} Sen^{\mathcal{A}}(\varphi)(e \; \texttt{in} \; e')$$

Proofs for the negation and implication are analogous.

When $\rho := (\texttt{all} \; x : e) \; \rho$:

$$M' \restriction \varphi \models^{\mathcal{A}}_{\Sigma} (\texttt{all} \; x : e) \; \rho$$

$\Leftrightarrow$ $\{\models$ defn. $\}$

For all model x–expansions $(M' \restriction \varphi)'$ of $M' \restriction \varphi$,
$(M' \restriction \varphi)' \models^{\mathcal{A}}_{\Sigma^x} (x \; \texttt{in} \; e) \; \texttt{implies} \; \rho$

$\Leftrightarrow$ $\{$Lemma 1; I.H. $\}$

For all model x–expansions M'' of M',
$M'' \models^{\mathcal{A}}_{\Sigma'^x} Sen^{\mathcal{A}}(\varphi')((x \; \texttt{in} \; e) \; \texttt{implies} \; \rho)$

$\Leftrightarrow$ $\{\models$ defn. $\}$

$$M' \models^{\mathcal{A}}_{\Sigma'} (\texttt{all} \; x : Sen^{\mathcal{A}}(\varphi)(e)) \; Sen^{\mathcal{A}}(\varphi')(\rho)$$

$\Leftrightarrow$ $\{Sen^{\mathcal{A}}$ defn. $\}$

$$M' \models^{\mathcal{A}}_{\Sigma'} Sen^{\mathcal{A}}(\varphi)((\texttt{all} \; x : e) \; \rho)$$

□

We have proved that $\mathcal{A} = (Sign^{\mathcal{A}}, Sen^{\mathcal{A}}, Mod^{\mathcal{A}}, \models^{\mathcal{A}})$ is an institution.

4 From Alloy to SOL

Second-order logic (SOL), extends first–order logic with quantification over functions and predicates, a feature required to canonically encode the notion of transitive closure primitive in Alloy. The corresponding institution, SOL =

$(\mathit{Sign}^{\mathrm{SOL}}, \mathit{Sen}^{\mathrm{SOL}}, \mathit{Mod}^{\mathrm{SOL}}, \models^{\mathrm{SOL}})$, extends the one in example 1, by allowing quantification over functions and predicates in $\mathit{Sen}^{\mathrm{SOL}}$.

This section describes a conservative *comorphism* from Alloy to the institution of presentations over SOL, where the notion of presentations is used as a key tactic for dealing appropriately with Alloy's implicit rules over kinds.

We proceed as follows: Let us define $(\Phi, \alpha, \beta) : \text{Alloy} \rightarrow \mathrm{SOL}^{pres}$.

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

- $S = \{U\}$
- $F_w = \begin{cases} \{\pi | \pi \in X\} & \text{if } w = U \\ \emptyset & \text{otherwise} \end{cases}$
- $P_w = \begin{cases} \{\pi | \pi \in S_{All}\} \cup \{\pi | \pi \in R_s, |s| = 1\} & \text{if } w = U \\ \{\pi | \pi \in R_s, |s| > 1\} & \text{if } w = (U, \ldots, U), |w| > 1 \\ \emptyset & \text{otherwise} \end{cases}$

and Γ is the least set containing the following sets of axioms:

1. $\{(\forall u : U)\, \pi(u) \Rightarrow \pi'(u) | \pi \in S_{All},\, \pi' = m(\pi)\}$
2. $\{(\exists u : U)\, \pi(u) | \pi \in S_{One} \cup S_{Som}\}$
3. $\{(\forall u, u' : U)\, (\pi(u) \wedge \pi(u')) \Rightarrow u = u' | \pi \in S_{One}\}$
4. $\{(\forall u : U)\, \pi(u) \Rightarrow (\bigvee_{\pi' \in m^{\circ}(\pi)} \pi'(u)) | \pi \in S_{Abs}\}$
5. $\{\neg(\exists u : U)\, \pi(u) \wedge \pi'(u) | \pi, \pi' \in S_{All}$, π,π' not related by the transitive closure of $m\}$
6. $\{(\forall u_1, \ldots, u_n : U)\, \pi(u_1, \ldots, u_n) \Rightarrow \bigwedge_{i=1}^{n} s_i(u_i) | \pi \in R_{s_1, \ldots, s_n}\}$

Sentence transformation. Given any signature $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma) \in |\mathit{Sign}^{\mathcal{A}}|$, $\alpha_\Sigma : \mathit{Sen}^{\mathcal{A}}(\Sigma) \rightarrow \mathit{Sen}^{\mathrm{SOL}^{pres}}(\Phi(\Sigma))$ is defined as:

$$\begin{array}{ll} \alpha_\Sigma(\texttt{not}\ \rho) & = \neg\alpha_\Sigma(\rho) \\ \alpha_\Sigma(\rho\ \texttt{implies}\ \rho') & = \alpha_\Sigma(\rho) \Rightarrow \alpha_\Sigma(\rho') \\ \alpha_\Sigma((\texttt{all}\ x : e)\ \rho) & = (\forall x : U)\ \alpha_{\Sigma^x}\ ((x\ \texttt{in}\ e)\ \texttt{implies}\ \rho), \text{ where} \\ & \quad \Sigma^x = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma + \{x\}) \\ \alpha_\Sigma(e\ \texttt{in}\ e') & = (\forall V : U_1 \ldots U_n)\, \eta_V(e) \Rightarrow \eta_V(e'), \text{ such that} \\ & \quad V = (v_1, \ldots, v_n), \text{ and } |V| = |e|, \end{array}$$

with η_V[4] being defined as follows:

[4] To represent a tuple of n elements, $v_1, \ldots, v_n$, we use notations $(v_1, \ldots, v_n)$ and $v_1, \ldots, v_n$ interchangeably, the latter being usually chosen if potential ambiguity is ruled out by the context.

$$\begin{aligned}
\eta_V(\pi) &= \pi(V), \pi \in (S_\Sigma)_{All} \cup (R_\Sigma)_w \\
\eta_v(\pi) &= \pi = v, \pi \in X_\Sigma \\
\eta_{(v1,v2)}(\hat{\ }e) &= (\exists R : U_1, U_2) R(v_1, v_2) \wedge \alpha_{\Sigma^R}(e \texttt{ in } R \texttt{ and } e.R \texttt{ in } R) \wedge \\
&\quad (\forall S : U_1, U_2) \alpha_{\Sigma^{R,S}}((e \texttt{ in } S \texttt{ and } e.S \texttt{ in } S) \texttt{ implies } R \texttt{ in } S), \\
&\quad \text{where } \Sigma^R = (S_\Sigma, m_\Sigma, R_\Sigma + R, X_\Sigma) \text{ and} \\
&\quad \Sigma^{R,S} = (S_\Sigma, m_\Sigma, R_\Sigma + R + S, X_\Sigma) \\
\eta_V(\sim e) &= \eta_{V'}(e), \text{ such that } V' = (v_n, \dots, v_1) \\
\eta_V(e + e') &= \eta_V(e) \vee \eta_V(e') \\
\eta_V(e \texttt{ - } e') &= \eta_V(e) \wedge \neg \eta_V(e') \\
\eta_V(e \,\&\, e') &= \eta_V(e) \wedge \eta_V(e') \\
\eta_V(e \texttt{ -> } e') &= \eta_{V'}(e) \wedge \eta_{V''}(e'), \text{ where } V' \text{ is the prefix of } V \text{ such that} \\
&\quad |V'| = |e| \text{ and } V'' \text{ the suffix of } V \text{ such that } |V''| = |e'| \\
\eta_V(e \texttt{ . } e') &= (\exists y : U) \eta_{(V',y)}(e) \wedge \eta_{(y,V'')}(e'), \text{ where } V' \text{ is the prefix of } V \\
&\quad \text{such that } |V'| + 1 = |e|, V'' \text{ the suffix of } V \text{ such that} \\
&\quad |V''| + 1 = |e'|
\end{aligned}$$

Model transformation. Consider a signature $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma) \in |Sign^{\mathcal{A}}|$, and a $\Phi(\Sigma)$–model M.
Then $\beta_\Sigma : |Mod^{\mathrm{SOL}^{pres}}(\Phi(\Sigma))| \to |Mod^{\mathcal{A}}(\Sigma)|$ is defined as:

$|\beta_\Sigma(M)| = |M_U|$, where $|M_U|$ is the carrier of U in M,
$\beta_\Sigma(M)_\pi = M_\pi$, for any $\pi \in (S_\Sigma)_{All} \cup (R_\Sigma)_w$,
$\beta_\Sigma(M)_\pi = \{M_\pi\}$, for any $\pi \in X_\Sigma$

Lemma 3. *On the conditions above, the commuting diagram below is a strong amalgamation square,*

$$\begin{tikzcd}
Mod^{\mathcal{A}}(S, m, R, X) \arrow[d, hook, "x^\beta"'] & & Mod^{\mathrm{SOL}^{pres}}(S', F', P') \arrow[ll, "\beta_{(S,m,R,X)}"'] \arrow[d, hook, "x"] \\
Mod^{\mathcal{A}}(S, m, R, X + \{x\}) & & Mod^{\mathrm{SOL}^{pres}}(S', F' + \{x\}, P') \arrow[ll, "\beta_{(S,m,R,X+\{x\})}"]
\end{tikzcd}$$

Proof. Proof in appendix A.3 □

Note that Lemma 3 says that, for any (S', F', P')–model M the x^β–expansion of its transformation by $\beta_{(S,m,R,X)}$, is equal to the transformation by $\beta_{(S,m,R,X+\{x\})}$ of its x–expansion, whenever the value taken by both x's in the corresponding expansions is the same.

Lemma 4. *Let $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma)$ be a signature in $Sign^{\mathcal{A}}$, M' a $\Phi(\Sigma)$-model M', and e a Σ-expression. Then, for any tuple $(v_1, \dots, v_n) \in X_\Sigma$ with $n = |e|$, we have*

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_{(v_1,\dots,v_n)}(e) \textit{ iff } \beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} (v_1 \texttt{->} \dots \texttt{->} v_n) \texttt{ in } e$$

Proof. Proof in appendix A.4 □

Theorem 2. *On the conditions above the satisfaction condition holds in (Φ, α, β). I.e., given a signature $\Sigma \in |Sign^{\mathcal{A}}|$, a $\Phi(\Sigma)$-model M', and a Σ-sentence ρ*

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \alpha_{\Sigma}(\rho) \textit{ iff } \beta_{\Sigma}(M') \models^{\mathcal{A}}_{\Sigma} \rho$$

Proof. Proof in appendix A.5 □

Theorem 3. *The triple (Φ, α, β) defined above is a conservative comorphism (in $\mathcal{SOL}^{pres}$).*

Proof. Let M be an (S, m, R, X)-model. Now, lets consider the $\Phi(S, m, R, X)$-model M' defined by:

1. $|M'_U| = |M|$
2. $M'_\pi = M_\pi$, for any $\pi \in S_{All} \cup R_w$
3. $M'_\pi = a$ where $\{a\} = M_\pi$, for any $\pi \in X$

Clearly $\beta_{(S,m,R,X)}(M') = M$ and M' satisfies Γ. Hence β is surjective and, therefore, (Φ, α, β) is conservative.

□

5 From Alloy to Casl

Casl, the *Common Algebraic Specification Language* [14], 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; we use free types and the notion of presentations to encode Alloy's transitive closure in Casl. Signatures in Casl are as in SOL, but extended with a family of partial functions symbols PF indexed by their arity, and a partial order $\leq$ over the symbols in S. As usual, $PF \cap F = \emptyset$.

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 figure 1, as a glue language inside the Hets network of logics.

A comorphism $(\Phi', \alpha', \beta') :$ Alloy→Casl^{pres} may be defined in a very similar way to the comorphism defined in the previous section. Let us, then, analyse the things that change in each component.

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

$$\begin{aligned} S' &= \{U, Nat\} \\ PF &= \emptyset \end{aligned}$$

$$F_w = \begin{cases} \{\pi | \pi \in X\} & \text{if } w = U \\ \{0\} & \text{if } w = Nat \\ \{suc\} & \text{if } w = Nat, Nat \\ \emptyset & \text{for the other cases} \end{cases}$$

$$P_w = \begin{cases} \{\pi | \pi \in S_{All}\} \cup \{r | r \in R_s, |s| = 1\} & \text{if } w = U \\ \{r | r \in R_s, |s| > 1\} & \text{if } w = (U, \ldots, U), |w| > 1 \\ \{\pi_r | \pi \in R_s, |s| = 2\} & \text{if } w = Nat, U, U \\ \emptyset & \text{for the other cases} \end{cases}$$

and Γ contains two additional rules:

1. { `free type` $Nat ::= (0 \mid suc(Nat))$ }
2. $\{(\forall u, v : U)\ \pi_r(0, u, v) \Leftrightarrow r(u, v) \wedge (\forall n : Nat)\ \pi_r(suc(n), u, v) \Leftrightarrow (\exists x : U)$ $\pi_r(0, u, x) \wedge \pi_r(n, x, v) | \pi_r \in R_s, |s| = 2\}$

Sentence transformation. Given any signature $\Sigma \in |Sign^{\mathcal{A}}|$, where $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma)$, $\alpha'_\Sigma : Sen^{\mathcal{A}}(\Sigma) \to Sen^{\mathrm{CASL}^{pres}}(\Phi(\Sigma))$ is defined in the same way as α (introduced in section 4), with the following replacing the case of transitive closure over expressions:

$$\eta'_V(\hat{}r) = (\exists n : Nat)\, \pi_r(n, V)$$

Note that only the transitive closure of atomic relations is considered. 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.

Model transformation. Nothing changes in the model transformation component, i.e., for each $\Sigma \in |Sign^{\mathcal{A}}|$, $\beta_\Sigma = \beta'_\Sigma$.

Theorem 4. *On the conditions above the satisfaction condition holds in* (Φ', α', β').

Proof. We can prove the satisfaction condition by using the following treatment in the case of transitive closure:

When $e := \hat{}r, r \in R_w$, with $|w| = 2$:

$$M' \models^{\textsc{Casl}^{pres}}_{\Phi(\Sigma)} \eta_V(\hat{}r)$$

$\Leftrightarrow$ $\{\alpha \text{ definition }\}$

$$M' \models^{\mathrm{CASL}^{pres}}_{\Phi(\Sigma)} (\exists n : Nat)\, \pi_r(n, V)$$

$\Leftrightarrow$ $\{\models \text{ defn. and } \pi_r \text{ in } \Gamma \text{ defn. }\}$

$$M'_V \in M'_{(r^+)}$$

$\Leftrightarrow$ $\{\text{elements of } V \text{ are constants, } \beta \text{ definition }\}$

$$\beta_\Sigma(M')_V \subseteq \beta_\Sigma(M')_{(r^+)}$$

$\Leftrightarrow$ $\big\{$Expression evaluation and $\models$ definition $\big\}$

$$\beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} V \texttt{ in } \hat{\ } r$$

Then for all other cases the proof is analogous to the one performed in the last section. □

Theorem 5. *The above comorphism is conservative.*

Proof. We have just to define a model in the same way as in Theorem 3. In addition, the following must also be included:

(a) $M'_{Nat} = \mathbb{N}$;
(b) For any π in $R_{s,s}$, M' has a relation, π_r, defining the transitive closure of r.

Clearly, M' satisfies the additional rules 1,2 in Γ. □

We have proved that (Φ', α', β') is a comorphism, and furthermore that is conservative. This means that (Φ', α', β') is a sound method for validating ALLOY's specifications through the proof environment of CASL.

6 ALLOY and HETS at work

6.1 An introduction to DCR graphs

DCR graphs, short for *Distributed Condition Response Graphs*, were introduced in [8] 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 $\texttt{condition}, \texttt{response} \subseteq E \times E$ which restrict control flow, regarded as a sequence of event executions. In detail,

- $(e, e') \in \texttt{condition}$ iff e' can only be executed after e;
- $(e, e') \in \texttt{response}$ 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 [16] 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.

6.2 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 tool inside the network has to be called, provided that ALLOY is already plugged there. Applying the comorphism to CASL defined in the previous section we get the following encoding of the property:

```
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.

6.3 A medical workflow

Consider now the following example of a DCR graph representing a medical workflow as introduced in [16]. 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 = ˆ(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
```

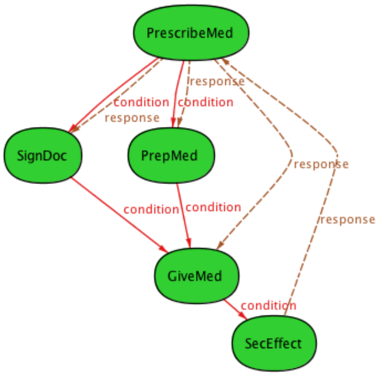


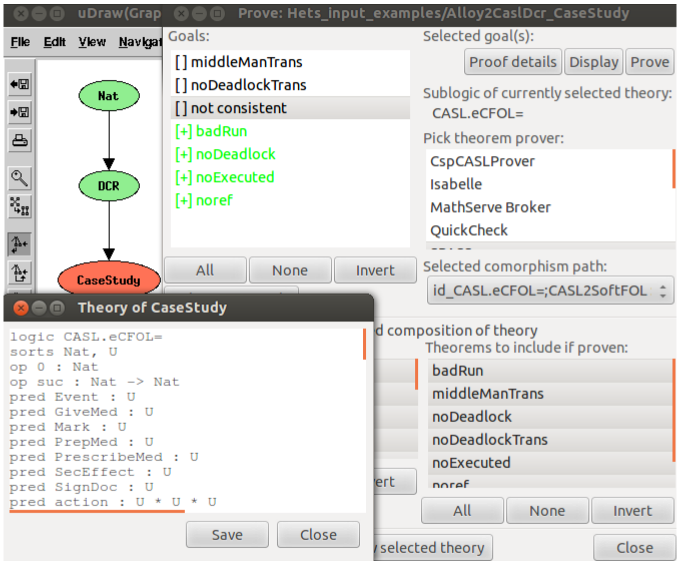
Fig. 2. A medical workflow diagram

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 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[5] using both ALLOY, with scope 15, and automatic theorem provers available in HETS, namely SPASS and EProver, through the second 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.

[5] 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
```

Fig. 3. A Hets session.

7 Discussion and conclusions

The paper laid the first steps toward establishing a rigorous methodology for modelling and validating software designs by connecting Alloy to a network of logics and logical tools, rather than, 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 underlying logical system as an institution, together with conservative comorphisms into institutions of presentations over SOL and Casl as entry points in the Hets network. The work reported here extends [17] in working out all the proof details and, mainly, providing a new, sound translation to SOL.

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 not only into both SOL and Casl specifications. On the other hand, following this approach 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 [13] we have developed a systematic way to build a hybrid logic layer on top of an arbitrary institution.

Hybrid logic [5] 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 [11] 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 that 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 [16]. More experimental work, however, is necessary 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, within projects FCOMP-01-0124-FEDER-028923, project FCOMP-01-0124-FEDER-022690 and NORTE-01-0124-FEDER-000060.

References

1. Konstantine Arkoudas, Sarfraz Khurshid, Darko Marinov, and Martin 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. Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.
3. Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. Thf0 — the core of the tptp language for higher-order logic. In *Proceedings of the 4th international joint conference on Automated Reasoning*, IJCAR '08, pages 491–506, Berlin, Heidelberg, 2008. Springer-Verlag.
4. Christoph Benzmüller, Frank Theiss, Larry Paulson, and Arnaud Fietzke. LEO-II - a cooperative automatic theorem prover for higher-order logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.
5. T Braüner. *Proof-Theory of Propositional Hybrid Logic.* Hybrid Logic and its Proof-Theory, 2011.
6. Răzvan Diaconescu. *Institution-independent Model Theory.* Birkhäuser Basel, 2008.
7. Joseph A. Goguen and Rod M. Burstall. Institutions: abstract model theory for specification and programming. *J. ACM*, 39:95–146, January 1992.

8. Thomas T. Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *Proc. 3rd PLACES Workshop*, volume 69 of *EPTCS*, pages 59–73, 2010.
9. Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis.* The MIT Press, 2006.
10. Nuno Macedo and Alcino Cunha. Automatic unbounded verification of Alloy specifications with Prover9. *CoRR*, abs/1209.5773, 2012.
11. Alexandre Madeira, José M. Faria, Manuel A. Martins, and Luís Soares Barbosa. Hybrid specification of reactive systems: An institutional approach. In G. Barthe, A. Pardo, and G. Schneider, editors, *Software Engineering and Formal Methods (SEFM 2011, Montevideo, Uruguay, November 14-18, 2011)*, volume 7041 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 2011.
12. María Manzano. *Extensions of first order logic.* Cambridge tracts in theoretical computer science. Cambridge University Press, Cambridge, New York, 1996.
13. Manuel A. Martins, Alexandre Madeira, Răzvan Diaconescu, and Luís Soares Barbosa. Hybridization of institutions. In A. Corradini, B. Klin, and C. Cîrstea, editors, *Algebra and Coalgebra in Computer Science (CALCO 2011, Winchester, UK, August 30 - September 2, 2011)*, volume 6859 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2011.
14. Till Mossakowski, Anne Haxthausen, Donald Sannella, and Andrzej Tarlecki. CASL: The common algebraic specification language: Semantics and proof theory. *Computing and Informatics*, 22:285–321, 2003.
15. Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, Hets. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007 - Braga, Portugal, March 24 - April 1, 2007)*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer, 2007.
16. R. R. Mukkamala. *A Formal Model For Declarative Workflows : Dynamic Condition Response Graphs.* PhD thesis, IT University of Copenhagen, 2012.
17. Renato Neves, Alexandre Madeira, Manuel A. Martins, and Luís S. Barbosa. Giving alloy a family. In Chengcui Zhang, James Joshi, Elisa Bertino, and Bhavani Thuraisingham, editors, *Proceedings of 14th IEEE International conference on information reuse and intergration*, pages 512–519. IEEE Press, 2013.
18. Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: a proof assistant for higher-order logic.* Springer-Verlag, Berlin, Heidelberg, 2002.
19. Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *AI Commun.*, 15(2,3):91–110, August 2002.
20. Mattias Ulbrich, Ulrich Geilmann, Aboubakr Achraf El Ghazi, and Mana Taghdiri. A proof assistant for alloy specifications. In Cormac Flanagan and Barbara 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.
21. Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Renate A. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction, CADE 2009*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 140–145. Springer, 2009.

A Proofs

A.1 Lemma 1

The following commuting diagram of ALLOY signature morphisms is a strong amalgamation square.

$$\begin{tikzcd}
(S,m,R,X) \arrow[r, "\varphi"] \arrow[d, hook, "x"'] & (S',m',R',X') \arrow[d, hook, "x^{\varphi}"] \\
(S,m,R,X+\{x\}) \arrow[r, "\varphi'"'] & (S',m',R',X'+\{x\})
\end{tikzcd}$$

Proof. Let M_1 be an $(S,m,R,X+\{x\})$–model, and M_2 an (S',m',R',X')–model, such that $Mod^{\mathcal{A}}(x)(M_1) = Mod^{\mathcal{A}}(\varphi)(M_2)$. Thus, for any $\pi \in S_t, M_{1\pi} = M_{2\varphi_{kd}(\pi)}$, for any $\pi \in R_w, M_{1\pi} = M_{2\varphi_{rl}(\pi)}$, for any $\pi \in X$, $M_{1\pi} = M_{2\varphi_{vr}(\pi)}$, and $|M_1| = |M_2|$.

Then, let us define an $(S',m',R,',X'+\{x\})$-model M' by stating that: For all $\pi \in S'_t, M_{2\pi} = M'_\pi$; for all $\pi \in R'_w, M'_\pi = M_{2\pi}$; for all $\pi \in X', M_{2\pi} = M'_\pi$; $|M_2| = |M'|$, and $M_{1x} = M'_x$. Clearly, $M_1 = Mod^{\mathcal{A}}(\varphi')(M')$ and $M_2 = Mod^{\mathcal{A}}(x^{\varphi})(M')$. Also it is not difficult to show that M' is unique. Therefore the diagram above is a strong amalgamation square.

□

A.2 Lemma 2

For any signature morphism $\varphi : \Sigma \to \Sigma'$ in $Sign^{\mathcal{A}}$, any Σ–expression e, and any Σ'–model M',

$$(M' \restriction \varphi)_e = M'_{Exp(\varphi)(e)}$$

Proof. Consider first the case $e := \pi$, for $\pi \in (R_\Sigma)_w$:

$$(M' \restriction \varphi)_\pi$$

$=$ $\{$Reduct defn. $\}$

$$M'_{\varphi_{rl}(\pi)}$$

$=$ $\{$*Exp* defn. $\}$

$$M'_{Exp(\varphi)(\pi)}$$

Proofs for when $\pi \in (S_\Sigma)_{All}$ or $\pi \in X_\Sigma$ are analogous.

When $e := e + e'$:

$$(M' \upharpoonright \varphi)_{e+e'}$$

$=$ $\{$Expression evaluation $\}$

$$(M' \upharpoonright \varphi)_{e} + (M' \upharpoonright \varphi)_{e'}$$

$=$ $\{$Induction hypothesis $\}$

$$M'_{Exp(\varphi)(e)} + M'_{Exp(\varphi)(e')}$$

$=$ $\{$Expression evaluation $\}$

$$M'_{Exp(\varphi)(e)+Exp(\varphi)(e')}$$

$=$ $\{$*Exp* defn. $\}$

$$M'_{Exp(\varphi)(e+e')}$$

Proofs for the remaining operators are analogous. □

A.3 Lemma 3

The following commuting square of model morphisms and model transformations is a strong amalgamation square,

$$\begin{tikzcd}
Mod^{\mathcal{A}}(S,m,R,X) \arrow[d, hook, "x^{\beta}"'] & Mod^{\mathrm{SOL}^{pres}}(S',F',P') \arrow[l, "\beta_{(S,m,R,X)}"'] \arrow[d, hook, "x"] \\
Mod^{\mathcal{A}}(S,m,R,X+\{x\}) & Mod^{\mathrm{SOL}^{pres}}(S',F'+\{x\},P') \arrow[l, "\beta_{(S,m,R,X+\{x\})}"]
\end{tikzcd}$$

Proof. Let M_1 be an $(S, m, R, X + \{x\})$–model and M_2 a (S', F', P')–model such that $Mod^{\mathcal{A}}(x^{\beta})(M_1) = \beta_{(S,m,R,X)}(M_2)$. *I.e.*, for any $\pi \in S_{All} \cup R_w \cup X$, $M_{1\pi} = M_{2\pi}$ and $|M_1| = |M_2|$.

Then let us define an $(S', F' + \{x\}, P')$–model M' such that for any $s \in S', |M'_s| = |M_{2s}|$; for any $\sigma \in F'_w, M'_\sigma = M_{2\sigma}$; for any $\pi \in P'_w, M'_\pi = M_{2\pi}$; $M'_x = \{M_{1x}\}$. Clearly, we have $M_1 = \beta_{(S,m,R,X+\{x\})}(M')$ and $M_2 = Mod^{\mathrm{SOL}^{pres}}(x)(M')$. Moreover, it is not difficult to show that M' is unique. Therefore the diagram above is a strong amalgamation square. □

A.4 Lemma 4

Let $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma)$ be a signature in $Sign^{\mathcal{A}}$, M' a $\Phi(\Sigma)$-model M', and e a Σ-expression. Then, for any tuple $(v_1, \ldots, v_n) \in X_\Sigma$ with $n = |e|$, we have

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_{(v_1,\ldots,v_n)}(e) \text{ iff } \beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} (v_1 \texttt{->} \ldots \texttt{->} v_n) \texttt{ in } e$$

Proof. When $e := \pi, \pi \in (R_\Sigma)_w \cup (S_\Sigma)_{All}$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_{(v_1,\ldots,v_n)}(\pi)$$

$\Leftrightarrow$ $\big\{\eta$ defn. $\big\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \pi(v_1,\ldots,v_n)$$

$\Leftrightarrow$ $\big\{\models$ defn. $\big\}$

$$(M'_{v_1},\ldots,M'_{v_n}) \in M'_\pi$$

$\Leftrightarrow$ $\big\{v_i$ elements are constants $\big\}$

$$M'_{v_1} \times \cdots \times M'_{v_n} \subseteq M'_\pi$$

$\Leftrightarrow$ $\big\{\beta$ defn. $\big\}$

$$\beta_\Sigma(M')_{v_1} \times \cdots \times \beta_\Sigma(M')_{v_n} \subseteq \beta_\Sigma(M')_\pi$$

$\Leftrightarrow$ $\big\{$Expression evaluation; $\models$ defn. $\big\}$

$$\beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} (v_1\texttt{->}\ldots\texttt{->}v_n)\ \texttt{in}\ \pi$$

When $e := \pi, \pi \in X_\Sigma$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_v(\pi)$$

$\Leftrightarrow$ $\big\{\eta$ defn. $\big\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} v = \pi$$

$\Leftrightarrow$ $\big\{\models$ defn. $\big\}$

$$M'_v = M'_\pi$$

$\Leftrightarrow$ $\big\{v$ and π are constants $\big\}$

$$\{M'_v\} \subseteq \{M'_x\}$$

$\Leftrightarrow$ $\big\{\beta$ defn. $\big\}$

$$\beta_\Sigma(M')_v \subseteq \beta_\Sigma(M')_\pi$$

$\Leftrightarrow$ $\big\{\models$ defn. $\big\}$

$$\beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} v\ \texttt{in}\ \pi$$

When $e := e\,.\,e'$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_V(e\,.\,e')$$

$\Leftrightarrow$ $\big\{\eta$ defn. $\big\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} (\exists y : U)\, \eta_{(V',y)}(e) \wedge \eta_{(y,V'')}(e')$$

$\Leftrightarrow$ $\big\{$⊨ defn. $\big\}$

$$\text{There is a } y\text{-expansion } M'' \text{ of } M' \text{ such that}$$
$$M'' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^y} \eta_{(V',y)}(e) \text{ and } M'' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^y} \eta_{(y,V'')}(e')$$

$\Leftrightarrow$ $\big\{$I.H., lemma 3, ⊨ defn. $\big\}$

$$\text{There is a } y\text{–expansion } M'' \text{ of } M' \text{ such that}$$
$$\beta_{\Sigma^y}(M'') \models^{\mathcal{A}}_{\Sigma^y} (V'{-}{>}y)\ \mathtt{in}\ e\ \mathtt{and}\ (y{-}{>}V'')\ \mathtt{in}\ e'$$

$\Leftrightarrow$ $\big\{$lemma 3, ⊨ defn. $\big\}$

$$\beta_{\Sigma}(M') \models^{\mathcal{A}}_{\Sigma} V\ \mathtt{in}\ e \,.\, e'$$

When $e := \hat{}e$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_{(v_1,v_2)}(\hat{}e)$$

$\Leftrightarrow$ $\big\{$ η defn. $\big\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \exists R.\big(R(v_1,v_2) \wedge \alpha_{\Sigma^R}(e\ \mathtt{in}\ R\ \mathtt{and}\ e.R\ \mathtt{in}\ R) \wedge$$
$$\forall S.\ \alpha_{\Sigma^{R,S}}\big((e\ \mathtt{in}\ S\ \mathtt{and}\ e.S\ \mathtt{in}\ S)\ \mathtt{implies}\ R\ \mathtt{in}\ S\big)\big)$$

$\Leftrightarrow$ $\big\{$ ⊨ defn. $\big\}$

$$\text{There is an } R\text{-expansion } M^R \text{ of } M' \text{ such that}$$
$$M^R \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^R} R(v_1,v_2) \wedge \alpha_{\Sigma^R}(e\ \mathtt{in}\ R\ \mathtt{and}\ e.R\ \mathtt{in}\ R)$$
$$\text{and for any } S\text{-expansion } M^{R,S} \text{ of } M^R,$$
$$M^{R,S} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,S}} \alpha_{\Sigma^{R,S}}\big((e\ \mathtt{in}\ S\ \mathtt{and}\ e.S\ \mathtt{in}\ S)\ \mathtt{implies}\ R\ \mathtt{in}\ S\big)$$

$\Leftrightarrow$ $\big\{$⊨ defn, α defn. $\big\}$

$$\text{There is an } R\text{-expansion } M^R \text{ of } M' \text{ such that}$$
$$\Big(M^R \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^R} R(v_1,v_2) \text{ and } M^R \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^R} (\forall u_1,u_2).$$
$$\big(\eta_{(u_1,u_2)}(e) \Rightarrow \eta_{(u_1,u_2)}(R)\big) \wedge \big(\eta_{(u_1,u_2)}(e.R) \Rightarrow R(u_1,u_2)\big)\Big)$$
$$\text{and for any } S\text{-expansion } M^{R,S} \text{ of } M^R,$$
$$M^{R,S} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,S}} \big((\forall u_1,u_2).(\eta_{(u_1,u_2)}(e) \Rightarrow \eta_{(u_1,u_2)}(S)) \wedge$$
$$(\eta_{(u_1,u_2)}(e.S) \Rightarrow S(u_1,u_2))\big) \Rightarrow \big((\forall u_1,u_2).R(u_1,u_2) \Rightarrow S(u_1,u_2)\big)$$

$\Leftrightarrow$ $\big\{$ α defn., ⊨ defn. $\big\}$

$$\text{There is an } R\text{-expansion } M^R \text{ of } M' \text{ such that}$$
$$M^R \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^R} R(v_1,v_2) \text{ and}$$

$\Big($for any (u_1, u_2)-expansion $(M^R)^{(u_1,u_2)}$ of M^R,
$\big((M^R)^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,u_1,u_2}} \eta_{(u_1,u_2)}(e)$ implies
$(M^R)^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,u_1,u_2}} \eta_{(u_1,u_2)}(R)\big)$ and
$\big((M^R)^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,u_1,u_2}} \eta_{(u_1,u_2)}(e.R)$ implies
$(M^R)^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,u_1,u_2}} R(u_1, u_2)\big)\Big)$
and for any S-expansion $M^{R,S}$ of M^R,
$\Big($ for any (u_1, u_2)-expansion $(M^{R,S})^{(u_1,u_2)}$ of $M^{R,S}$
$\big((M^{R,S})^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,S,u_1,u_2}} \eta_{(u_1,u_2)}(e)$ implies
$(M^{R,S})^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,S,u_1,u_2}} \eta_{(u_1,u_2)}(S)\big)$ and
$\big((M^{R,S})^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,S,u_1,u_2}} \eta_{(u_1,u_2)}(e.S)$ implies
$(M^{R,S})^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,S,u_1,u_2}} S(u_1, u_2)\big)\Big)$ implies
that for any (u_1, u_2)-expansion $(M^{R,S})^{(u_1,u_2)}$ of $M^{R,S}$
$\Big((M^{R,S})^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,S,u_1,u_2}} R(u_1, u_2)$ implies
$(M^{R,S})^{(u_1,u_2)} \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^{R,S,u_1,u_2}} S(u_1, u_2)\Big)$

$\Leftrightarrow$ $\{$I.H., Lemma 3 $\}$

There is an R-expansion M^R of M' such that
$\beta_{\Sigma^R}(M^R) \models^{\mathcal{A}}_{\Sigma^R} (v_1\texttt{->}v_2)\ \texttt{in}\ R$ and
$\Big($ for any (u_1, u_2)-expansion $(M^R)^{(u_1,u_2)}$ of M^R,
$\big(\beta_{\Sigma^{R,u_1,u_2}}((M^R)^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ e$ implies
$\beta_{\Sigma^{R,u_1,u_2}}((M^R)^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ R\big)$ and
$\big(\beta_{\Sigma^{R,u_1,u_2}}((M^R)^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ e.R$ implies
$\beta_{\Sigma^{R,u_1,u_2}}((M^R)^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ R\ \big)\Big)$
and for any S-expansion $M^{R,S}$ of M^R,
$\Big($for any (u_1, u_2)-expansion $(M^{R,S})^{(u_1,u_2)}$ of $M^{R,S}$
$\big(\beta_{\Sigma^{R,S,u_1,u_2}}((M^{R,S})^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,S,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ e$ implies
$\beta_{\Sigma^{R,S,u_1,u_2}}((M^{R,S})^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,S,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ S\big)$ and
$\big(\beta_{\Sigma^{R,S,u_1,u_2}}((M^{R,S})^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,S,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ e.S$ implies
$\beta_{\Sigma^{R,S,u_1,u_2}}((M^{R,S})^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,S,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ S\big)\Big)$ implies
that for any (u_1, u_2)-expansion $(M^{R,S})^{(u_1,u_2)}$ of $M^{R,S}$
$\Big(\beta_{\Sigma^{R,S,u_1,u_2}}((M^{R,S})^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,S,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ R$ implies
$\beta_{\Sigma^{R,S,u_1,u_2}}((M^{R,S})^{(u_1,u_2)}) \models^{\mathcal{A}}_{\Sigma^{R,S,u_1,u_2}} (u_1\texttt{->}u_2)\ \texttt{in}\ S\Big)$

$\Leftrightarrow$ $\{$inclusion defn., $\models$ defn. $\}$

There is an R–expansion M^R of M' such that

$$\begin{gathered}\beta_{\Sigma^R}(M^R) \models^{\mathcal{A}}_{\Sigma^R} (v_1\texttt{->}v_2)\ \texttt{in}\ R \text{ and} \\ \beta_{\Sigma^R}(M^R) \models^{\mathcal{A}}_{\Sigma^R} e\ \texttt{in}\ R\ \texttt{and}\ e.R\ \texttt{in}\ R \\ \text{and for all } S\text{–expansions } M^{R,S} \text{ of } M^R \\ \beta_{\Sigma^{R,S}} \models^{\mathcal{A}}_{\Sigma^{R,S}} (e\ \texttt{in}\ S\ \texttt{and}\ e.S\ \texttt{in}\ S)\ \texttt{implies}\ R\ \texttt{in}\ S\end{gathered}$$

$\Leftrightarrow$ $\{$transitive closure defn. $\}$

$$\beta_{\Sigma}(M') \models^{\mathcal{A}}_{\Sigma} (v_1\texttt{->}v_2)\ \texttt{in}\ \hat{}e$$

When $e := \sim e$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_{(v_1,\ldots,v_n)}(\sim e)$$

$\Leftrightarrow$ $\{\alpha$ defn. $\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_{(v_n,\ldots,v_1)}(e)$$

$\Leftrightarrow$ $\{$I.H $\}$

$$\beta_{\Sigma}(M') \models^{\mathcal{A}}_{\Sigma} (v_n\texttt{->}\ldots\texttt{->}v_1)\ \texttt{in}\ e$$

$\Leftrightarrow$ $\{$Galois connection $\}$

$$\beta(M') \models^{\mathcal{A}}_{\Sigma} (v_1\texttt{->}\ldots\texttt{->}v_n)\ \texttt{in}\ (\sim e)$$

When $e := e + e'$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_V(e + e')$$

$\Leftrightarrow$ $\{\alpha$ defn., $\models$ defn. $\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_V(e) \text{ or } M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \eta_V(e')$$

$\Leftrightarrow$ $\{$I.H $\}$

$$\beta_{\Sigma}(M') \models^{\mathcal{A}}_{\Sigma} V\ \texttt{in}\ e \text{ or } \beta_{\Sigma}(M') \models^{\mathcal{A}}_{\Sigma} V\ \texttt{in}\ e'$$

$\Leftrightarrow$ $\{\models$ defn., sum defn. $\}$

$$\beta_{\Sigma}(M') \models^{\mathcal{A}}_{\Sigma} V\ \texttt{in}\ e + e'$$

Proofs for the remaining cases are analogous.

□

A.5 Theorem 2

The satisfaction condition holds in $(\Phi, \alpha, \beta) :$ Alloy $\to \mathrm{SOL}^{pres}$. I.e., given a signature $\Sigma \in |Sign^{\mathcal{A}}|$, a $\Phi(\Sigma)$-model M', and a Σ-sentence ρ

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \alpha_\Sigma(\rho) \text{ iff } \beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} \rho$$

Proof. When $\rho := e\ \mathtt{in}\ e'$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \alpha_\Sigma(e\ \mathtt{in}\ e')$$

$\Leftrightarrow$ $\{\alpha \text{ defn.}\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} (\forall V : U_1, \ldots, U_n)\, \eta_V(e) \Rightarrow \eta_{V'}(e')$$

$\Leftrightarrow$ $\{\text{Satisfaction defn.}\}$

$$\begin{gathered}\text{For any } V\text{-expansion } M'' \text{ of } M',\\ M'' \models^{\mathcal{A}}_{\Phi(\Sigma)'} \eta_V(e') \text{ whenever } M'' \models^{\mathcal{A}}_{\Phi(\Sigma)'} \eta_V(e)\end{gathered}$$

$\Leftrightarrow$ $\{\text{Lemma 4 and} \models \text{defn.}\}$

$$\begin{gathered}\text{For any } V\text{-expansion } M'' \text{ of } M',\\ \beta_{\Sigma'}(M'') \models^{\mathcal{A}}_{\Sigma'} V\ \mathtt{in}\ e \Rightarrow V\ \mathtt{in}\ e'\end{gathered}$$

$\Leftrightarrow$ $\{\text{Inclusion defn., Lemma 3}\}$

$$\beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} e\ \mathtt{in}\ e'$$

When $\rho := \mathtt{not}\ \rho$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \alpha_\Sigma(\mathtt{not}\ \rho)$$

$\Leftrightarrow$ $\{\alpha \text{ defn.}\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \mathtt{not}\ \alpha_\Sigma(\rho)$$

$\Leftrightarrow$ $\{\models \text{defn.}\}$

$$M' \not\models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \alpha_\Sigma(\rho)$$

$\Leftrightarrow$ $\{\text{I.H.}\}$

$$\beta_\Sigma(M') \not\models^{\mathcal{A}}_{\Sigma} \rho$$

$\Leftrightarrow$ $\{\models \text{defn.}\}$

$$\beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} \mathtt{not}\ \rho$$

For implication the proof is analogous

When $\rho := (\mathtt{all}\ x : e)\ \rho$:

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} \alpha_\Sigma((\mathtt{all}\ x : e)\ \rho)$$

$\Leftrightarrow$ $\{\alpha \text{ defn.}\}$

$$M' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)} (\forall x : U)\alpha_{\Sigma}((x \;\texttt{in}\; e) \;\texttt{implies}\; \rho)$$

$\Leftrightarrow$ $\{\models \text{ defn.}\}$

$$\text{For any } x\text{-expansion } M'' \text{ of } M', \\ M'' \models^{\mathrm{SOL}^{pres}}_{\Phi(\Sigma)^x} \alpha_{\Sigma^x}((x \;\texttt{in}\; e) \;\texttt{implies}\; \rho)$$

$\Leftrightarrow$ $\{\text{I.H.}\}$

$$\text{For any } x\text{-expansion } M'' \text{ of } M', \\ \beta_{\Sigma^x}(M'') \models^{\mathcal{A}}_{\Sigma^x} (x \;\texttt{in}\; e) \;\texttt{implies}\; \rho$$

$\Leftrightarrow$ $\{\text{Lemma 3}\}$

$$\text{For any } x\text{-expansion } \beta_{\Sigma^x}(M'') \text{ of } \beta_{\Sigma}(M'), \\ \beta_{\Sigma^x}(M'') \models^{\mathcal{A}}_{\Sigma^x} (x \;\texttt{in}\; e) \;\texttt{implies}\; \rho$$

$\Leftrightarrow$ $\{\models \text{ defn.}\}$

$$\beta_{\Sigma}(M') \models^{\mathcal{A}}_{\Sigma} (\texttt{all}\; x : e)\; \rho$$

□