Universidade do Minho
Escola de Engenharia

Renato Jorge Araújo Neves

Proof support for hybridised logics

September, 2013

Universidade do Minho
Escola de Engenharia

Renato Jorge Araújo Neves

Proof support for hybridised logics

Dissertação de
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Manuel António Martins
Luís Soares Barbosa

Setembro, 2013

DECLARAÇÃO

Nome

_____

Endereço electrónico: _____ Telefone: _____ / _____

Número do Bilhete de Identidade: _____

Título dissertação □ / tese □

_____

_____

_____

Orientador(es):

_____

_____ Ano de conclusão: _____

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

_____

Universidade do Minho, ____ / ____ / _____

Assinatura: _____

This dissertation was partially supported by a research grant awarded within the NASONI project (FCT), under the contract FCOMP-01-0124-FEDER-028923.

Abstract

Formal methods are mathematical techniques used to certify safe systems. Such methods abound and have been successfully used in classical Engineering domains, yet informatics is the exception. There, they are still immature and costly; furthermore, software engineers frequently view them with "fear". Thus, the use of formal methods is typically restricted to cases where they are essential. In other words, they are mostly used in the class of systems where safety is imperative, as the lack of it can lead to significant losses (material or human). We denote such systems critical. The present is leading us to a future where critical systems are ubiquitous.

Recent research in the MONDRIAN project emphasises the need for expressive logics to formally specify *reconfigurable* systems, *i.e.*, systems capable of evolving in order to adapt to the different contexts induced by the dynamics of their surroundings. In the same project, theoretical foundations for the formal specification of *reconfigurable* systems, were developed in a sound, generic, and systematic way, resorting for this to *hybrid* logics – their intrinsic properties make them natural candidates for such job. From those foundations a methodology for specifying *reconfigurable* systems was built and proposed: Instead of choosing a logic for the specification, build an *hybrid ad-hoc* one, by taking into account the particular characteristics of each *reconfigurable* system to be specified.

The purpose of this dissertation is to bring the proposed methodology into practice, by creating suitable tools for it, and by illustrating its application to relevant case studies.

Keywords : Formal methods, modelling, *reconfigurable* systems, *hybrid* logics, *institutions*.

Resumo

Métodos formais são técnicas matemáticas usadas para certificar sistemas fiáveis. Tais métodos são comuns e usados com sucesso nas engenharias clássicas. No entanto, informática é a excepção. No que respeita este campo, os métodos formais são prematuros e relativamente dispendiosos; para além disso, os engenheiros de software vêem estas técnicas com alguma apreensão. Assim, o emprego de métodos formais está tipicamente restrito a casos onde são absolutamente essenciais. Por outras palavras, são maioritariamente usados na classe de sistemas, cujas falhas têm o potencial de tragédia, seja ela material ou humana; tais sistemas têm a denominação de críticos. O presente leva-nos para um futuro em que os sistemas críticos são ubíquos.

Investigação recente no project MONDRIAN enfatiza a necessidade de lógicas expressivas, para especificar formalmente sistemas *reconfiguráveis*, *i.e.*, sistemas que evoluem de modo a se adaptarem aos diferentes contextos, induzidos pela dinâmica do meio que os rodeia. No mesmo projecto, bases teóricas para a especificação formal de sistemas *reconfiguráveis* foram establecidas de forma sólida, genérica e sistemática, recorrendo-se para isso às lógicas *híbridas* – as suas propriedades intrínsecas, fazem delas candidatos naturais para a especificação de sistemas reconfiguráveis. Dessas teorias foi inferida e proposta uma metodologia para especificar sistemas *reconfiguráveis*: Em vez de escolher uma lógica para a especificação, construir uma outra, híbrida *ad-hoc*, tendo em conta as características particulares de cada sistema reconfigurável a especificar.

O propósito desta dissertação é de trazer a metodologia proposta à práctica, criando-se para isso, ferramentas que a suportem, e ilustrando a sua aplicação a casos de estudo relevantes.

Keywords : Métodos formais, modelação, sistemas *reconfiguráveis*, lógicas *híbridas*, *instituições*.

Acknowledgements

A thesis or a dissertation is developed by a student. However, that same student is embraced by sundry institutions (advisers, family, friends...), that make such work possible. Among other things, they provide work conditions, academical/life wisdom, distractions and support.

This dissertation is not an exception to that. As such I would like to devote this page to those who have helped me in one way or another.

Firstly, I would like to thank my supervisors, Prof. Luis Barbosa and Prof. Manuel Martins. They have projected many times, patience, wisdom, friendship, and many other qualities, that I took as lessons and will pursue throughout my life. Of course, I cannot forget Alexandre who was almost a third advisor, with lots of insightful comments to offer. I doubt that this dissertation would ever reach its current quality without his support.

My colleagues were also critical in this work by giving their support in "unconventional" ways; for that I am grateful.

Finally, I would like to thank my family by resorting to my native language.

Pai e mãe, durante toda a minha vida tive vários professores, muitos deles excelentes. No entanto, foi com vocês que mais aprendi. Vocês foram essenciais nesta crusada e sei que posso contar convosco para as próximas.

Como me poderia esquecer das xonas? Estou muito grato pela paciência que sempre tiveram comigo. Ambas pensam demasiado de mim.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Software is an integral part of our lives. We have it in our cars, planes, medical devices, cellphones, and even in our toasters. This ubiquity, and the increased dependency of our daily lives on computer-based devices is potentially critical because software is generally built in an informal, non coherent and unscientific way.

To give a sense of safety and quality to software products, testing still is the main, and often the only, method used. Yet as Dijkstra said, it only shows the presence, not the absence of bugs. Moreover, when an error is uncovered, often is the case were it would be detected earlier with alternative practices, thus preventing wasted time and money.

Due to the limitations of this trial-and-error approach one ought to consider additional techniques in software development. In particular modelling, whose importance has been recognised centuries ago in several Engineering domains, albeit in software development being often neglected.

Currently, UML is the *de facto* standard for modelling. Indeed, UML is a simple language that can help in structuring and schematising projects, and even in finding inconsistencies early in the project. However it is also an ambiguous language that like testing, cannot prove the correctness of

a design or the corresponding code.

Therefore, one may legitimately reach the conclusion that at least when millions of euros and/or human lives are at stake, most of the software is not trustworthy. To tackle this (and being us in an Engineering area) there is an agenda for developing sound mathematical based theories, that can provide efficient ways to guarantee that software behaves as expected. Significant progress has been made on this subject in the last 30 years, however it still is considered far behind against other Engineerings. Such fact can been supported by the following statement presented on [MBMNed] and given by a software engineer : *I'm not afraid of working with symbols and formulas, like most software engineers and consultants I encounter are*.

Moreover, most formal methods for software engineering are still expensive to use, and require highly qualified personnel. Hence, they are scarcely used out of the critical systems' frontier. The lack of tool support is another obstacle to the general use of formal methods. Reinforcing this last statement, a study referred in [MBMNed] asking for suggestions for curricular improvement in the formal methods area, pointed out that the most frequent one was indeed, increased tool support.

In broad terms, this dissertation aims at contributing to fill the gap between theory and practice, *i.e.*, between mathematically sound and expressive methods, and suitable support tools.

1.2 Context

More concretely, our focus is on a specific class of specification methods – those based on the *hybrid* extension of modal logic, which is particularly useful to reason about *reactive* systems, *i.e.*, systems whose interaction with their environment, proceeds along computation and whose evolution is, to a large extent, determined by such interactions.

Hybrid logics add expressive power to ordinary modal logic, through the introduction of propositional symbols of a new sort, called nominals,

each being true at exactly one possible state or world (of the underlying Kripke frame). Identifying each world with a particular execution mode in a reactive system, provides an interesting framework to specify *reconfigurable* systems. Those are systems whose "form" (*i.e.* resources involved, network topology, etc) changes along the computational process in response to varying context conditions. Their behaviour is indexed to a set of different run-time *modes* between which the system commutes dynamically. *Reconfigurable* systems are now everywhere, from service-oriented applications to cloud management or robot controllers.

Such systems can be modelled as classical state-machines or labelled transition systems. Modal logic, in which the value of an assertion is relative to a particular mode, state or point in the system's evolution, is the standard specification language for the *transitional* behaviour of this sort of models. As mentioned above, the *hybrid* extension further provides a way to refer to particular, suitably named states. Several variants of such logics exist, with corresponding axiomatizations, model classes and proof theories. In some cases, as discussed below, they come equipped with specific theorem provers.

From the point of view of the working software engineering, however, there is a need to combine *hybrid* logic with whatever formalism one may think more suitable to specify the *functional* behaviour of the system, inside each state or operation mode. For example, in very simple cases, propositional logic may be enough; if data types are present one may need, however, equational logic or even the full power of first-order. In other cases more exotic formalisms may be in order: for example fuzzy or other sort of probabilistic logic are being increasingly used to reason about, run–time composition of software services and bio-inspired algorithms.

The need to combine *hybrid* logics, to cater for *transitional* behaviour, with a variety of other logics to capture *functional* requirements in concrete system states, lead to the development of a generic, broad scope method to introduce *hybrid* features in whatever logic one wants to take for the underlying functional specifications. This is referred as the *hybridi-*

sation method and was introduced in [MMDB11]. The process amounts to a systematic way to extend any logic, defined as an *institution* [GB92, Dia08], with the properties characteristic of *hybrid* logic. The result is usually called an *hybridised* logic.

This method was developed in the context of the MONDRIAN [1] project, and its application to the specification of *reconfigurable* systems, has been discussed in a number of papers by the project team starting in [MFMB11].

Actually, the MONDRIAN project gave a special focus to *reconfigurable* systems. A concern justified by the fact that a significant part of existing software is in fact *reconfigurable*. For instance, the well known auto pilot mechanism in a plane, supports different configurations: While in the ON mode the engine follows directions from a map and a GPS device; if in the OFF mode, directions are to be given by the pilot.

The MONDRIAN project and the *hybridisation* process of arbitrary logics, mentioned above, provides the context for the present dissertation.

1.3 Contributions

The overall aim of this dissertation is to provide effective tool support for stating and verifying formal specifications written in *hybridised* logics, constructed accordingly to the systematic *hybridisation* method documented in [MMDB11]. Adopting the same generic approach underlying the method, our aim is *not* to build dedicated proof tools for each possible *hybridised* logic, which moreover would be a giant task, but to develop support for defining *hybrid* versions of whatever logics one may be interested in, on top of the HETS [MML07] platform.

What makes the *hybridisation* method developed in [MMDB11] effective, is the representation of logics as *institutions* [GB92]. As an abstract representation of a logical system, encompassing syntax, semantics and satisfaction, an *institution* provides modular structuring and parameterization mechanisms which are defined "once and for all", abstracting from

[1] `http://www.di.uminho.pt/mondrian`

the concrete particularities of each specification logic. Intuitively, and slightly forcing the meaning of words, one may regard *institutions* as a kind of (formal) boilerplates...for logics. But *institutions*, originally proposed by Joseph Goguen and Burstall in the 70's, also provide a systematic way to relate logics and transport results from one to another, through the so-called *comorphism*. This means that a theorem prover for the latter, can be used to reason about specifications written in the former. The *hybridisation* method, whose implementation this dissertation discusses, takes advantage of this to "freely" provide suitable tool support for specifications, through suitable translations to *first-order* logic. HETS concretises the theory of *institutions* and respective *comorphisms*, in the form of a sound framework. We take advantage of it, to implement the *hybridisation* method.

In this context, the research programme of this dissertation lead to the following contributions:

1. The integration and exploration of the *hybridisation* method (proposed in [MMDB11]) on top of HETS the platform. In particular,

   (a) The implementation of an *hybridised* version of CASL [MHST03] into HETS, which provided a first step for implementing the more generic version of the *hybridisation* process;

   (b) A performance analysis and comparison, involving the tool support achieved for the *hybridised* version of CASL, and dedicated provers for *hybrid* logics;

   (c) The implementation of the *hybridisation* process into HETS, in a generic setting.

   The results achieved in this line of work were published in [NMMB13b], and the implementation results patched into the offical version of HETS [2], by the respective community.

[2] `http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/hets/index_e.htm`

2. The development of several illustrative examples and an extensive case study with *hybrid* specifications, which entails the need to resort to a number of different base logics.

3. The extension of the *hybridisation* process, to the relational framework underlying Alloy [Jac06], one of the most successful *lightweight formal* tools. In particular,

   (a) the formalisation of Alloy as an *institution*;

   (b) followed by the definition of a suitable *comorphism* from Alloy to *first order* logic;

   (c) then on the practical side, the exploration of a case study involving DCR graphs [HM10], that brings the theory developed into practice.

   The results achieved in this line of work were published in [NMMB13a, NMMBar].

4. Further development of the theory underlying the *hybridisation* method, by addressing the following points:

   (a) Identification of the *hybridisation* method as an endofunctor, paving the way for a more generic proof support for hybridised logics, than the one introduced in [MMDB11];

   (b) Development of a technique that at some extent, provides to *hybrid* specifications the means to reason about evolving interfaces, alongside with the already existing machinery for talking about evolving configurations.

   Results achieved, were published in [MNMB13].

Overall, this dissertation is supported by four publications in international events with peer–review: [NMMB13b], [NMMB13a], [NMMBar] and [MNMB13].

1.4 Roadmap

This chapter sums up the motivation, context and contributions for this *MSc.* dissertation.

The remaining chapters are structured as follows:

- Chapter 2 summarises the theory supporting this dissertation. It also describes the HETS platform and dedicated provers for *hybrid* logics.

- The core of this dissertation, *i.e.*, the development of tool support for *hybridised* logics, is presented in Chapter 3. It starts by introducing, the *hybridisation* of a singular logic with the respective proof support. Then, HETS implementation of the generic *hybridisation* process is introduced.

  Technical details are complemented with (small) case studies resorting to the *hybridisation*.

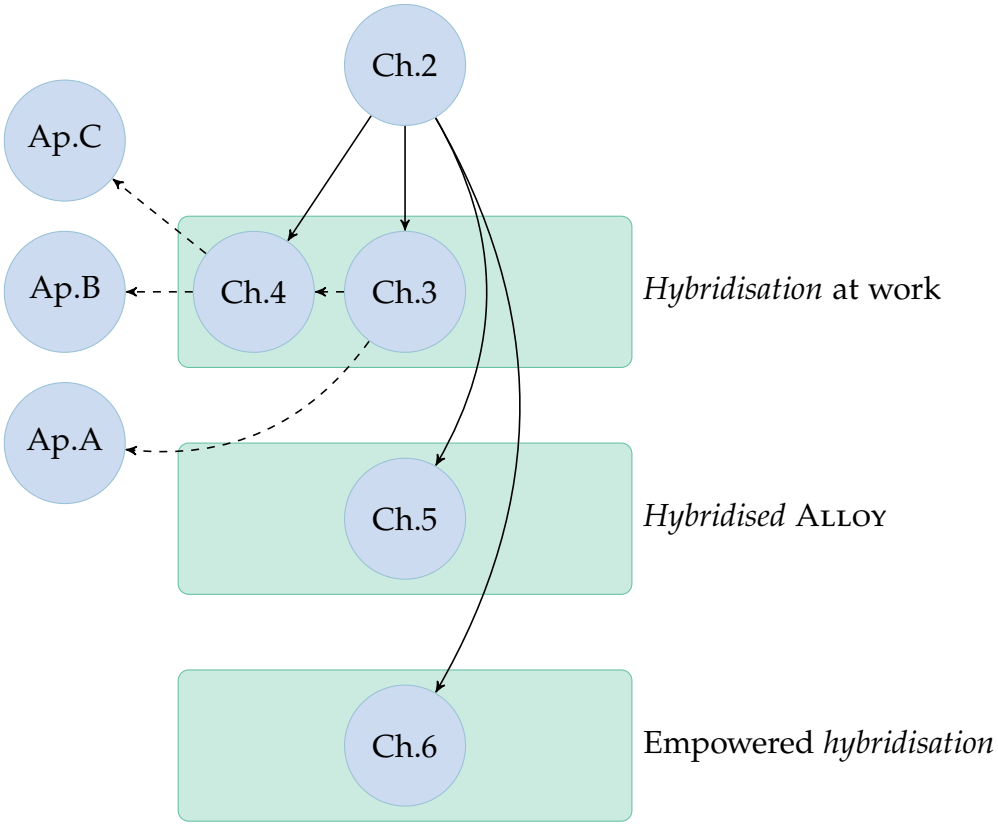
- Chapter 4 illustrates the methodology behind the *hybridisation* process with an extensive case study on a critical medical device. In this context, several *hybridised* logics are employed, ranging from the more conventional to the more exotic ones.

- Chapter 5 presents an extension of the method suitable to *hybridise* Alloy and provides tool support to its *hybridised* version. The theory is put into practice with a case study on medical workflows.

- Chapter 6 discusses further extensions to the *hybridisation* method.

- Finally, chapter 7 concludes this dissertation.

The diagram below gives the dependency relation between technical chapters (solid lines), and suggests reading sequences (dashed lines):


Ch.2
Ap.C
Ap.B
Ch.4
Ch.3
Hybridisation at work
Ap.A
Ch.5
Hybridised Alloy
Ch.6
Empowered *hybridisation*

Chapter 2

Background

2.1 Hybrid logics

2.1.1 Modal logics

Reconfigurable systems by definition, have different modes of operation. They evolve by reacting to different events, changing their attributes along time. This leads to a notion of relative truth, *i.e.*, properties hold not statically, but rather relative to stages of evolution – one property true at one mode of operation, may not be at another.

Actually not only reconfigurable systems, but many others have relativeness intrisinc in them, which is only expectable since it is a natural law in our reality. For example, the proposition "is raining" is only true at certain points in time.

Modal logics [BdRV01] are suitable languages to deal with this relative truth pattern. They distinguish themselves by being able to express transitions between states and making restrictions to the worlds associated with those transitions. Therefore, modal logics are considered to be natural candidates for the specification of reconfigurable systems, if the underlying states and transitions are regarded as the modes of operation and triggers for them, respectively.

States and transitions form what is the called Kripke frame. Modal logic formulas make assertions about the underlying transition structure,

but do not fix the sort of model one may want to associate with a specific state. This for instance, may be a propositional, first order, or whatever structure is found appropriate.

Arguably the simplest modal logic is the modal propositional logic (henceforth $\mathcal{MPL}$). As its name suggests, states correspond to propositional models, with the propositions changing their truth value along them. From another perspective, one may see $\mathcal{MPL}$ as an expansion of propositional logic, with a modality operator, resulting in the following grammar:

$$\rho := p \mid \neg\rho \mid \rho \Rightarrow \rho' \mid [r]\rho, \text{ where}$$

p is a proposition, *i.e.*, a symbol interpreted in models as *true* or *false*. r is a *modality*, *i.e.*, the identification of a particular transition intepreted in models as a binary relation over worlds.

As usual, $\rho \vee \rho'$ and $\rho \wedge \rho'$ are defined by $\neg\rho \Rightarrow \rho'$ and $\neg(\rho \Rightarrow \neg\rho')$, respectively.

The new operator, $[r]$, is what turns propositional logic into a modal logic. It provides the machinery to restrict properties in all states that can be accessed by a given modality (*i.e.* transition), considering a state as a point of evaluation – the state at which an expression is to be evaluated. To give an example: $[r]p$ is one of the sentences that is possible to build with the grammar presented above. Roughly speaking, when evaluated at world w, it states: *p holds for any world w', that is accessible from state w, through a transition corresponding to modality r.* Dually, one can also say that there is a transition by r to a world where p holds : $\neg[r]\neg p$. $\langle r\rangle p$, is a compact way of expressing the same, as for any sentence ρ, $\neg[r]\neg\rho \equiv \langle r\rangle\rho$. As already mentioned, one of the limitations of modal logics is their limited inability to address a specific state. Hybrid logics naturally overcome this limitation. This explains why we focus on them, in this dissertation.

2.1.2 Hybrid logics

Arthur Prior [Bla06] paved the path for hybrid logics in the late 60's. Notably, this class of logics boosts the expressivity of modal logics without affecting neither decidability nor complexity [BBW06].

The standard hybrid logics as presented in [AtC06], extend modal logics with special propositions, which can only be true at exactly one state. They are called nominals, and regarded as state names or identifiers. Additionally, we have a new operator that moves the point of evaluation to a world referred by a nominal. Thus, hybrid logics provide the machinery to refer directly to states.

Analogous to $\mathcal{MPL}$, $\mathcal{HPL}$ is the hybrid logic whose worlds are propositional models. Its sentences are given by the following grammar:

$$\rho = p \mid \neg\rho \mid \rho \Rightarrow \rho' \mid [r]\rho \mid n \mid @_n\rho, \text{ where}$$

p is a proposition, r a modality, and n a nominal

The novelty, is the operator @; it changes the point of evaluation to the state referred by the respective nominal. *I.e.*, the formula coming after this operator, is evaluated at the state pointed by the nominal associated with @. For example, $@_n p$ means that : *At state n, p holds*.

Along with @, comes the possibility of using nominals on their own. This allows assertions regarding equality between worlds. For instance, $@_n m$, means that n and m point to the same world. Another example is $@_n n$, which is actually a valid sentence in the context of this particular logic.

Typically $\mathcal{HPL}$ model semantics is given by Kripke frames along with an assignment function, as shown below:

Definition 2.1.1. *An $\mathcal{HPL}$ model is a triple (W, R, g) where :*

- *W is the set of worlds,*
- *R is a set of binary relations over $W \times W$. Each relation in R is uniquely identified by a modality symbol,*

- *g the assignment function that for each world, gives a truth value to a proposition, $g : W \to P \to \{0,1\}$. As a side note, the latter may also be seen as a family of functions, $(g_w : P \to \{0,1\})_{w \in W}$.*

Finally the satisfaction is defined as follows:

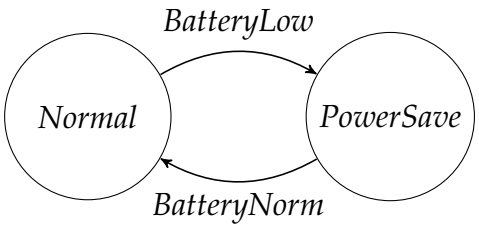
Definition 2.1.2. *Given an $\mathcal{HPL}$ model M = (W,R,g) and a naming function f, assigning to each nominal a world. The relation $M, f, w \models \rho$, for w a* world, *and ρ an $\mathcal{HPL}$ formula, is defined as:*

$$
\begin{array}{lll}
M, f, w \models p & \textit{iff} & g(w, p) = 1 \\
M, f, w \models \neg\rho & \textit{iff} & M, f, w \not\models \rho \\
M, f, w \models \rho \Rightarrow \rho' & \textit{iff} & M, f, w \models \rho' \textit{ whenever } M, f, w \models \rho \\
M, f, w \models [r]\rho & \textit{iff} & \textit{for all } (w, w') \in M_r,\ M, f, w' \models \rho \\
M, f, w \models n & \textit{iff} & f(n) = w \\
M, f, w \models @_n\rho & \textit{iff} & M, f, f(n) \models \rho
\end{array}
$$

The following example illustrates the use of hybrid logics for specification.

Example 2.1.1. *Consider a smoke detector, that can work at normal or low battery mode. When functioning at normal mode, the sensor is queried at an high frequency. But, when the battery is low, the detector switches to a lower querying frequency so that battery may be preserved.*

The description given above, is depicted by the following Kripke frame:



Clearly in this system, there are exactly two possible modes of operation (1), which furthermore must be mutually exclusive (2). Although with $\mathcal{MPL}$ is not possible to specify requirements (1) and (2), with $\mathcal{HPL}$ we can do it resorting to nominals, as the following shows:

$$(Normal \vee PowerSave)\ (1)\ \neg(Normal \wedge PowerSave)\ (2)$$

From the example's description, is also clear that when a transition by *BatteryLow* happens, the achieved state must always be at a low frequency mode (3). This can be written in $\mathcal{MPL}$ as follows:

$$[BatteryLow]\neg HighFreq\ (3)$$

However, we also know that such transition happens between modes *Normal* and *PowerSave* (4). This cannot be expressed with $\mathcal{MPL}$, due to the lack of nominal support and control of the evaluation point. On the other hand, with $\mathcal{HPL}$, requirements (3,4) are straightforward:

$$@_{Normal}[BatteryLow](PowerSave \wedge \neg HighFreq)(3,4)$$

$\mathcal{HPL}$ provides machinery to specify other interesting properties. For instance, that the available transitions form a cycle: *From Normal mode, we can jump to a state through BatteryLow, and then to Normal again, through BatteryNorm* (5).

$$@_{Normal}\langle BatteryLow\rangle\langle BatteryNorm\rangle Normal\ (5)$$

2.1.3 Expressiveness of hybrid logics

Although, hybrid logics already give a boost to modal logics' expressive power, there has been work trying push such expressiveness even further, even if, some times at the cost of decidability and increased complexity. One of the most significant results is a new operator that significantly increases hybrid logics expressiveness, but turns them undecidable in the process. This operator, often called *binder* ($\downarrow$) and introduced by Goranko [Gor96], names states dynamically. *I.e.*, binds a new name to a state, but just in a sentence's context. One classical example of a property that cannot be expressed using standard hybrid logics is *reflexivity*, as in the following Kripke frame:

With the binder such property is stated in the following way:

$$\downarrow x \,.\, \langle r \rangle x$$

Note that the world did not have a pre–defined name, and therefore we could never refer it with standard hybrid logics. Using the binder, we gave it a name in an *ad hoc* way, thus overcoming the reference impossibility.

Other possible extension is the addition of the so called *global operator* ($\overset{\circ}{\forall}$) [GG93] (and the dual *existential operator*) to hybrid logics. The catch is that the respective satisfiability problem is turned EXPTIME-complete. This operator, behaves in a similar way to universal quantification in first order logic. However with the difference that it can only quantify over states and cannot bind a variable's name to them. When using the global operator, as its name suggests, the point of evaluation becomes global. *I.e.*, the sentence associated to it, is not evaluated at just one, but at all available worlds.

Another notion worth mentioning is that of *rigid designators*. Designators are called rigid, when they must "behave" in the same way at all states of the underlying model. Therefore, they are typically used to express invariant properties. Returning to the smoke detector system, a simple requirement that can be formalised resorting to the notion of rigidness is: *When smoke is detected, a warning ensues independently of which state the system is.*

To conclude, nominal quantification ($\textcircled{\forall}$,$\textcircled{\exists}$) may also be considered. The latter works like the quantification in first order logic, with the difference that only nominals may be quantified. As expected, decidability is lost and complexity becomes exponential. However, it should be noted that nominal quantification is enough to "cover" all the other extensions mentioned above: Given a sentence ρ, we have

- $\downarrow x \,.\, \rho \equiv \textcircled{\exists} x \,.\, x \wedge \rho$;
- $\overset{\circ}{\forall} \rho \equiv \textcircled{\forall} x \,.\, @_x \rho$;

- To assert rigidness of a predicate p, one simply writes: $\textcircled{\triangledown} x, x' \,.\, @_x\, p \Rightarrow @_{x'}\, p$.

We conclude this brief survey of hybrid logics, mentioning that the hybridisation method, which is presented further in this document, incorporates nominal quantification. However, the latter can always be discarded whenever decidability and low complexity are a priority.

2.2 Hybridisation

2.2.1 Institutions

The navel of this dissertation, namely, the hybridisation process, relies on a number of formal concepts[1] whose definitions are recalled here. Being more specific, the present section recalls the notion of an institution – the framework in which the theory of hybridisation was developed.

The institution concept [GB92, Dia08] formalises the essence of a what a logical system is, by encompassing syntax, semantics and satisfaction. Introduced by Joseph Goguen and Rod Burstall in the late 70's, its original aim was to provide a sound foundation for the

"development of as much computing science as possible, in a general and uniform way,"

as a response to the increasing number of emerging logics in software specification.

The hybridisation process reaches its generic setting, because the theory of institutions provides means to completely abstract from each particularly a logic may have, therefore paving the way to develop theory "once and for all" logics. Formally,

Definition 2.2.1. *An institution is a tuple* $(Sign^{\mathcal{I}}, Sen^{\mathcal{I}}, Mod^{\mathcal{I}}, (\models^{\mathcal{I}}_{\Sigma})_{\Sigma \in |Sign^{\mathcal{I}}|})$, *where :*

[1] typically framed in the language of category theory (*cf* [Awo10]).

- $Sign^{\mathcal{I}}$ *is a category, whose objects are called signatures;*
- $Sen^{\mathcal{I}} : Sign^{\mathcal{I}} \to \mathbb{S}et$, *is a functor that for each signature* Σ, *gives a set whose elements are the* Σ*–sentences;*
- $Mod^{\mathcal{I}} : (Sign^{\mathcal{I}})^{op} \to \mathbb{C}$, *is a functor giving for each signature* Σ *a category whose objects are the* Σ*-models.* $\mathbb{C}$ *is the category of categories;*
- $\models^{\mathcal{I}}_{\Sigma} \subseteq |Mod^{\mathcal{I}}(\Sigma)| \times Sen^{\mathcal{I}}(\Sigma)$, *is the satisfaction relation such that for each morphism* $\varphi : \Sigma \to \Sigma'$, *the following holds :*

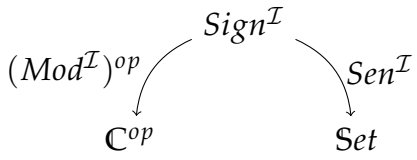
$$Mod^{\mathcal{I}}(\varphi)(M') \models^{\mathcal{I}}_{\Sigma} \rho \text{ iff } M' \models^{\mathcal{I}}_{\Sigma'} Sen^{\mathcal{I}}(\varphi)(\rho) \text{ for any}$$

$M' \in |Mod^{\mathcal{I}}(\Sigma')|$, $\rho \in Sen^{\mathcal{I}}(\Sigma)$

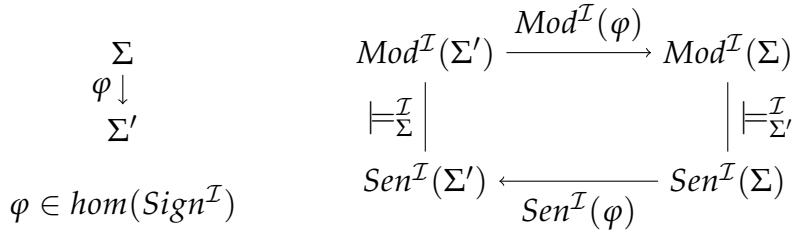
The reduct of M' through a signature morphism φ, is defined by $Mod^{\mathcal{I}}(\varphi)(M')$, and denoted as $M' \restriction_{\varphi}$. Dually, M' is called the model expansion by φ of $M' \restriction_{\varphi}$.

To achieve simple notations, for any signature object Σ, we may use notation C_{Σ} to express *"the component C of signature* Σ*"*.

Following the category theory tradition of describing notions in terms of diagrams, the structure of an institution $\mathcal{I}$ may be intuitively seen as follows,



along with the satisfaction condition, expressed by the diagram shown below:



The notion of an amalgamation square is often essential for "capturing" logics, and so is employed in the present document. We recall the correspondent definition.

Definition 2.2.2. *A commuting square of functors,*

$$\begin{array}{ccc} A' & \xrightarrow{G_2} & A_2 \\ {\scriptstyle G_1}\downarrow & & \downarrow{\scriptstyle F_2} \\ A_1 & \xrightarrow[F_1]{} & A \end{array}$$

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, we have a strong amalgamation square.

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),

$$\begin{array}{ccc} Mod^{\mathcal{I}}(\Sigma') & \xrightarrow{Mod^{\mathcal{I}}(\theta_2)} & Mod^{\mathcal{I}}(\Sigma_2) \\ \downarrow{\scriptstyle Mod^{\mathcal{I}}(\theta_1)} & \begin{array}{ccc} \Sigma' & \xleftarrow{\theta_2} & \Sigma_2 \\ {\scriptstyle \theta_1}\uparrow & & \uparrow{\scriptstyle \varphi'} \\ \Sigma_1 & \xleftarrow[\varphi]{} & \Sigma \end{array} & \downarrow{\scriptstyle Mod^{\mathcal{I}}(\varphi')} \\ Mod^{\mathcal{I}}(\Sigma_1) & \xrightarrow[Mod^{\mathcal{I}}(\varphi)]{} & Mod^{\mathcal{I}}(\Sigma) \end{array}$$

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.

The simplest example of an institution is the singular logic.

Example 2.2.1. The singular logic

Signatures. *$Sign^1$ is the singleton category, having only one (atomic) object and the respective identity arrow.*

Sentences. *Sen^1 is a functor that for each object in $Sign^1$ returns the empty set.*

Models. *Mod^1 is a functor that for each object in $Sign^1$ returns the 1 category.*

Satisfaction. *$\models^1$ is the family of relations indexed by the objects in $Sign^1$. For each $\Sigma \in |Sign^1|$, $\models^1_\Sigma = |Mod^1(\Sigma)| \times Sen^1(\Sigma)$.*

Satisfaction condition. *The proof for this institution comes for free, as for any $\Sigma \in |Sign^1|$, $Sen^1(\Sigma) = \emptyset$.*

□

The institution capturing propositional logic, is also an interesting example:

Example 2.2.2. Propositional logic

Signatures. *$Sign^{\mathcal{PL}}$ is a category whose objects are sets of propositional symbols. A morphism, φ, in this category is a function $\varphi : P \to P'$ that for each $p \in P$, gives a $\varphi(p) \in P'$.*

Sentences. *$Sen^{\mathcal{PL}}$ is a functor such that, for each $\Sigma \in |Sign^{\mathcal{PL}}|$, returns the set of expressions built of propositional symbols of Σ, closed by the typical boolean connectives. Given a morphism $\varphi : \Sigma \to \Sigma'$ in $Sign^{\mathcal{PL}}$:*

$$\begin{array}{lcl} Sen^{\mathcal{PL}}(\varphi)(p) & = & \varphi(p) \\ Sen^{\mathcal{PL}}(\varphi)(\rho \Rightarrow \rho') & = & Sen^{\mathcal{PL}}(\varphi)(\rho) \Rightarrow Sen^{\mathcal{PL}}(\varphi)(\rho') \\ Sen^{\mathcal{PL}}(\varphi)(\neg\rho) & = & \neg Sen^{\mathcal{PL}}(\varphi)(\rho) \end{array}$$

Models. *$Mod^{\mathcal{PL}}$ is a functor that, for each $\Sigma \in |Sign^{\mathcal{PL}}|$, gives the category generated by the powerset of propositional symbols P ordered by the set inclusion. In other words, models are the sets of holding propositional symbols in Σ; and the corresponding morphisms are the inclusions between those models. Given a signature morphism $\varphi : \Sigma \to \Sigma'$ in $Sign^{\mathcal{PL}}$, the reduct of a model $M \in$*

$|Mod^{\mathcal{PL}}(\Sigma')|$ through the morphism φ is defined as follows : For each $p \in \Sigma$, $p \in M\restriction_\varphi$ iff $\varphi(p) \in M$.

Satisfaction. *$\models^{\mathcal{PL}}$ is the satisfaction relation such that, given any $\Sigma \in |Sign^{\mathcal{PL}}|$ and $M \in |Mod^{\mathcal{PL}}(\Sigma)|$:*

$$\begin{array}{lll} M \models^{\mathcal{PL}}_\Sigma \rho \Rightarrow \rho' & \textit{iff} & M \models^{\mathcal{PL}}_\Sigma \rho' \textit{ whenever } M \models^{\mathcal{PL}} \rho \\ M \models^{\mathcal{PL}}_\Sigma \neg\rho & \textit{iff} & M \not\models^{\mathcal{PL}}_\Sigma \rho \\ M \models^{\mathcal{PL}}_\Sigma p & \textit{iff} & p \in M \end{array}$$

Satisfaction condition. *The proof of the satisfaction condition, follows by induction. Given any morphism $\varphi : \Sigma \to \Sigma'$ in $Sign^{\mathcal{PL}}$, a model $M' \in |Mod^{\mathcal{PL}}(\Sigma')|$ and a sentence $\rho \in Sen^{\mathcal{PL}}(\Sigma)$:*

When $\rho := p$,

$$M' \restriction_\varphi \models^{\mathcal{PL}}_\Sigma p$$

$\Leftrightarrow$ *{Satisfaction definition }*

$$p \in M' \restriction_\varphi$$

$\Leftrightarrow$ *{Reduct definition }*

$$\varphi(p) \in M'$$

$\Leftrightarrow$ *{$Sen^{\mathcal{PL}}$ definition }*

$$Sen^{\mathcal{PL}}(\varphi)(p) \in M'$$

$\Leftrightarrow$ *{Satisfaction definition }*

$$M' \models^{\mathcal{PL}}_{\Sigma'} Sen^{\mathcal{PL}}(\varphi)(p)$$

When there is an implication:

$$M' \restriction_\varphi \models^{\mathcal{PL}}_\Sigma \rho \Rightarrow \rho'$$

$\Leftrightarrow$ *{Satisfaction definition }*

$$M' \restriction_\varphi \models^{\mathcal{PL}}_{\Sigma} \rho' \text{ whenever } M' \restriction_\varphi \models^{\mathcal{PL}}_{\Sigma} \rho$$

$\Leftrightarrow$ *{Induction hypothesis, 2× }*

$$M' \models^{\mathcal{PL}}_{\Sigma'} Sen^{\mathcal{PL}}(\varphi)(\rho') \text{ whenever } M' \models^{\mathcal{PL}}_{\Sigma'} Sen^{\mathcal{PL}}(\varphi)(\rho)$$

$\Leftrightarrow$ *{Satisfaction definition }*

$$M' \models^{\mathcal{PL}}_{\Sigma'} Sen^{\mathcal{PL}}(\varphi)(\rho) \Rightarrow Sen^{\mathcal{PL}}(\varphi)(\rho')$$

$\Leftrightarrow$ *{$Sen^{\mathcal{PL}}$ definition }*

$$M' \models^{\mathcal{PL}}_{\Sigma'} Sen^{\mathcal{PL}}(\rho \Rightarrow \rho')$$

The case for negation is similar to the above. □

Naturally, one of the most used logics, first order logic, can also be defined in institutional terms:

Example 2.2.3. Many Sorted First Order logic

Signatures. *$Sign^{\mathcal{FOL}}$ 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 $f \in F_{w \to s}$, then $\varphi_{op}(f) \in F'_{\varphi_{st}(w) \to \varphi_{st}(s)}$, and if $p \in P_w$ then $\varphi_{rl}(p) \in P'_{\varphi_{st}(w)}$.*

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

$t \approx t'$,	*for* $t, t' \in terms$
$p(t_1, \ldots, t_n)$,	*for* $t_1, \ldots, t_n \in terms$ *and* $p \in P_w$
$\neg\rho$,	*for* $\rho \in Sen^{\mathcal{FOL}}((S, F, P))$
$\rho \Rightarrow \rho'$,	$\rho, \rho' \in Sen^{\mathcal{FOL}}((S, F, P))$
$\forall x : s \,.\, \rho$,	$s \in S$, $\rho \in Sen^{\mathcal{FOL}}((S, F \uplus \{x\}_{\to s}, P))$. *We simply represent the morphism* $(S, F, P) \to (S, F \uplus \{x\}_{\to s}, P)$ *as* x.

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* $Sign^{\mathcal{FOL}}$, *sentences are mapped in the following way:*

$Sen^{\mathcal{FOL}}(\varphi)(t \approx t')$	=	$terms(\varphi)(t) \approx terms(\varphi)(t')$
$Sen^{\mathcal{FOL}}(\varphi)(p(t_1, \ldots, t_n))$	=	$\varphi_{rl}(p)(terms(\varphi)(t_1), \ldots, terms(\varphi)(t_n))$
$Sen^{\mathcal{FOL}}(\varphi)(\neg\rho)$	=	$\neg Sen^{\mathcal{FOL}}(\varphi)(\rho)$
$Sen^{\mathcal{FOL}}(\varphi)(\rho \Rightarrow \rho')$	=	$Sen^{\mathcal{FOL}}(\varphi)(\rho) \Rightarrow Sen^{\mathcal{FOL}}(\varphi)(\rho')$
$Sen^{\mathcal{FOL}}(\varphi)(\forall x : s \,.\, \rho)$	=	$\forall x : \varphi_{st}(s) \,.\, Sen^{\mathcal{FOL}}(\varphi')(\rho)$, *where* φ' *canonically extends* φ *with* $\varphi'_{op}(x) = x$

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

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

- *for any* $s \in S$, $|(M' \restriction_\varphi)_s| = |M'_{\varphi_{st}(s)}|$
- *for any* $f \in F_{w \to s}$, $(M' \restriction_\varphi)_f = M'_{\varphi_{op}(f)}$
- *for any* $p \in P_w$, $(M' \restriction_\varphi)_p = M'_{\varphi_{rl}(p)}$

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

$$
\begin{array}{lll}
M \models^{\mathcal{FOL}}_{\Sigma} t \approx t' & \textit{iff} & M_t = M_{t'} \\
M \models^{\mathcal{FOL}}_{\Sigma} p(t_1, \ldots, t_n) & \textit{iff} & (M_{t_1}, \ldots, M_{t_n}) \in M_p \\
M \models^{\mathcal{FOL}}_{\Sigma} \neg\rho & \textit{iff} & M \not\models^{\mathcal{FOL}}_{\Sigma} \rho \\
M \models^{\mathcal{FOL}}_{\Sigma} \rho \Rightarrow \rho' & \textit{iff} & M \models^{\mathcal{FOL}}_{\Sigma} \rho' \textit{ whenever } M \models^{\mathcal{FOL}}_{\Sigma} \rho \\
M \models^{\mathcal{FOL}}_{\Sigma} \forall x : s \,.\, \rho & \textit{iff} & \textit{for any model expansion } M' \textit{ along the inclusion morphism } x \\
 & & \textit{previously defined in the functor } Sen^{\mathcal{FOL}}, M' \models^{\mathcal{FOL}}_{\Sigma'} \rho
\end{array}
$$

Lemma 2.2.1. *The following diagram is a strong amalgamation square for* $Mod^{\mathcal{FOL}}$

$$
\begin{array}{ccc}
\Sigma & \xrightarrow{\varphi} & \Sigma_2 \\
{\scriptstyle x}\big\downarrow & & \big\downarrow{\scriptstyle x^{\varphi}} \\
\Sigma_1 & \xrightarrow[\varphi']{} & \Sigma'
\end{array}
$$

where φ' *canonically extends* φ *with* $\varphi'_{op}(x) = x$, *and* x, x^{φ} *are inclusion morphisms. I.e. both add* x *(which is a constant function* $\{x\}_{\to s}$*) as a fresh symbol to the respective signatures.*

Proof. *From* $Mod^{\mathcal{FOL}}(x)(M_1) = Mod^{\mathcal{FOL}}(\varphi)(M_2)$, *we know that:*

$$
\begin{array}{c}
\textit{For any: } s \in S_{\Sigma},\ |M_{1s}| = |M_{2\varphi_{st}(s)}|; \\
f \in (F_{w \to s})_{\Sigma},\ M_{1f} = M_{2\varphi_{op}(f)}; \\
p \in (P_w)_{\Sigma},\ M_{1p} = M_{2\varphi_{rl}(p)}.
\end{array}
$$

$\Rightarrow$ *{*M_2 *is the* x^{φ} *reduct of* M'*; transitivity }*

For all models expansions M' *of* M_2, *by the inclusion morphism* x^{φ}*: given any,*

$$
\begin{array}{c}
s \in S_{\Sigma},\ |M_{1s}| = |M'_{\varphi_{st}(s)}|; \\
f \in (F_{w \to s})_{\Sigma},\ M_{1f} = M'_{\varphi_{op}(f)}; \\
p \in (P_w)_{\Sigma},\ M_{1p} = M'_{\varphi_{rl}(p)}.
\end{array}
$$

$\Rightarrow$ *{*$|M_{1s}| = |M'_{\varphi_{st}(s)}|$*, for any* $s \in S_{\Sigma}$*; inclusion morphism definition }*

There is exactly one model expansion of M_2 by x^φ, denoted as M', such that
$$M'_x = M_{1x}, \text{ entailing that for any : } s \in S_\Sigma, |M_{1s}| = |M'_{\varphi_{st}(s)}|;$$
$$f \in (F_{w\to s})_\Sigma, M_{1f} = M'_{\varphi_{op}(f)};$$
$$p \in (P_w)_\Sigma, M_{1p} = M'_{\varphi_{rl}(p)};$$
$$M'_x = M_{1x}.$$

$\Leftrightarrow$ *{Inclusion morphism x, definition }*

There is exactly one model expansion of M_2 by x^φ, denoted as M', such that, for
$$\text{any : } s \in S_{\Sigma_1}, |M_{1s}| = |M'_{\varphi_{st}(s)}|;$$
$$f \in (F_{w\to s})_\Sigma, M_{1f} = M'_{\varphi_{op}(f)};$$
$$p \in (P_w)_{\Sigma_1}, M_{1p} = M'_{\varphi_{rl}(p)};$$
$$M'_x = M_{1x}.$$

$\Leftrightarrow$ *{φ' definition }*

There is exactly one model expansion of M_2 by x^φ, denoted as M', such that for
$$\text{any : } s \in S_{\Sigma_1}, |M_{1s}| = |M'_{\varphi'_{st}(s)}|;$$
$$f \in (F_{w\to s})_{\Sigma_1}, M_{1f} = M'_{\varphi'_{op}(f)};$$
$$p \in (P_w)_{\Sigma_1}, M_{1p} = M'_{\varphi'_{rl}(p)}.$$

$\Leftrightarrow$ *{reduct definition }*

There is exactly one model expansion of M_2 by x^φ, denoted as M', such that
$$M_1 = Mod^{\mathcal{FOL}}(\varphi')(M')$$

□

Lemma 2.2.2. *For any signature morphism $\varphi : \Sigma \to \Sigma'$, a Σ'–model M', and a term t, built from Σ, $(M' \restriction_\varphi)_t = M'_{terms(\varphi)(t)}$.*

Proof. *When $t := \sigma$, for $\sigma \in (F_{\to s})_\Sigma$ and $s \in S_\Sigma$,*

$$(M' \restriction_\varphi)_\sigma$$

$\Leftrightarrow$ *{Reduct definition }*

$$M'_{\varphi_{op}(\sigma)}$$

$\Leftrightarrow$ *{terms(φ) definition }*

$$M'_{terms(\varphi)(\sigma)}$$

When $t := \sigma(t_1, \ldots, t_n)$*, for* $\sigma \in (F_{w \to s})_\Sigma$ *with* $w, s \in (S_\Sigma)^*$ *and* $t_1, \ldots, t_n \in \Sigma$*–terms,*

$$(M' \restriction_\varphi)_{\sigma(t_1,\ldots,t_n)}$$

$\Leftrightarrow$ *{Reduct definition, and I.H. for each term }*

$$M'_{\varphi_f(\sigma)((terms(\varphi)(t_1),\ldots,terms(\varphi)(t_n))}$$

$\Leftrightarrow$ *{terms(φ) definition }*

$$M'_{terms(\varphi)(\sigma(t_1,\ldots,t_n))}$$

□

Satisfaction condition. *The proof for the satisfaction condition, follows by induction. Given a morphism* $\varphi : \Sigma \to \Sigma'$ *in* $Sign^{\mathcal{FOL}}$*, a model* $M' \in |Mod^{\mathcal{FOL}}(\Sigma')|$ *and a sentence* $\rho \in Sen^{\mathcal{FOL}}(\Sigma)$*,*

When $\rho := t \approx t'$*, for* t, t' Σ*–terms*

$$M' \restriction_\varphi \models^{\mathcal{FOL}}_\Sigma t \approx t'$$

$\Leftrightarrow$ *{Satisfaction definition }*

$$(M' \restriction_\varphi)_t = (M' \restriction_\varphi)_{t'}$$

$\Leftrightarrow$ *{Lemma 2.2.2 }*

$$M'_{terms(\varphi)(t)} = M'_{terms(\varphi)(t')}$$

$\Leftrightarrow$ *{Satisfaction definition }*

$$M' \models^{\mathcal{FOL}}_{\Sigma'} terms(\varphi)(t) \approx terms(\varphi)(t')$$

$\Leftrightarrow$ *{$Sen^{\mathcal{FOL}}(\varphi)$ definition }*

$$M' \models^{\mathcal{FOL}}_{\Sigma'} Sen^{\mathcal{FOL}}(\varphi)(t \approx t')$$

The case where $\rho = p(t_1, \ldots, t_n)$ for $p \in (P_w)_\Sigma$ and $w \in (S_\Sigma)^*$ comes out straightforwardly, by using the Lemma 2.2.2 and the reduct/satisfaction definitions.

The implication and negation cases are proved analogously, to what was done in the $\mathcal{PL}$ institution (cf. example 2.2.2).

When $\rho := \forall x : s \,.\, \rho'$, for $s \in S_\Sigma$ and ρ' a Σ–sentence,

$$M' \restriction_\varphi \models^{\mathcal{FOL}}_{\Sigma} \forall x : s \,.\, \rho'$$

$\Leftrightarrow$ *{Satisfaction definition }*

For all model expansions of $M' \restriction_\varphi$, denoted as $(M' \restriction_\varphi)'$, by the inclusion morphism x previously defined on $Sen^{\mathcal{FOL}}$, $(M' \restriction_\varphi)' \models^{\mathcal{FOL}}_{\Sigma'} \rho'$

$\Leftrightarrow$ *{Lemma 2.2.1 and I.H. }*

For all model expansions of M', denoted as M'', by the inclusion morphism x^φ, previously defined on $Sen^{\mathcal{FOL}}$, $M'' \models^{\mathcal{FOL}}_{\Sigma'^x} Sen^{\mathcal{FOL}}(\varphi')(\rho')$

$\Leftrightarrow$ *{Satisfaction definition }*

$$M' \models^{\mathcal{FOL}}_{\Sigma'} \forall x : \varphi_{st}(s) \,.\, Sen^{\mathcal{FOL}}(\varphi')(\rho)$$

$\Leftrightarrow$ $\{Sen^{\mathcal{FOL}}(\varphi)$ *definition* $\}$

$$M' \models^{\mathcal{FOL}}_{\Sigma'} Sen^{\mathcal{FOL}}(\varphi)(\forall x : s \,.\, \rho)$$

□

Example 2.2.4. First order Equational logic ($\mathcal{FEQ}$) *is the institution of $\mathcal{FOL}$, without the relational symbols and the corresponding interpretations in models.*

□

Example 2.2.5. First order Partial logic ($\mathcal{FP}$) *as an institution, extends $\mathcal{FEQ}$ as follows: Signature become tuples (S, TF, PF), where TF is the family of (total) functions, and PF the family of partial functions indexed by their arity. As expected we assume that $TF_{w\to s} \cap PF_{w\to s} = \emptyset$. Signature morphisms are defined in the usual way.*

Models have to interpret partial functions, meaning that for some arguments the latter may not be defined.

Three kinds of atoms are added to sentences – definedness (def(_)), existential equality ($\overset{e}{=}$) and strong equality $(=)$. Actually the semantics of the latter is equivalent to the typical equality, as we will see next.

Regarding satisfaction, for any valid terms t, t', def(t) holds iff t is defined in the corresponding model. $t \overset{e}{=} t'$ holds iff t, t' are defined and equal. Finally $t = t'$ holds iff both terms are undefined or both are defined and equal.

Example 2.2.6. Equational logic ($\mathcal{EQ}$) *is the institution of $\mathcal{FEQ}$, where the sentences are restricted to universally quantified equations – those of the form $\forall X \,.\, t = t'$.*

□

Jumping to more "exotic" logics, the multi-valued ones replace the two-elements set of truth values $\{true, false\}$, structured as a Boolean

algebra, by other sets structured as *complete residuated lattices* (cf. [Got01] for an overview).

Multi-valued logics were first formalised as institutions in [ACEGG90], being [Dia13] a recent reference. We follow the latter in formalisation below.

Example 2.2.7. Multi-valued and Fuzzy Logics.

A residuated lattice is a structure $L = (\mathbf{L}, \leq, \wedge, \vee, \top, \bot, \star)$*, where*

- $(\mathbf{L}, \wedge, \vee, \top, \bot)$*, has an alphabet* $\mathbf{L}$*, (binary) infimum and supremum,* $\wedge$ *and* $\vee$*, and biggest and smallest elements,* $\top$ *and* $\bot$*. Elements of* $\mathbf{L}$ *are thus ordered by* $\leq$*.*
- $\star$ *is an associative and commutative binary operation such that, for any elements* $x, y, z \in \mathbf{L}$*:*
  - $x \star \top = \top \star x = x$*;*
  - $y \leq z$ *implies that* $(x \star y) \leq (x \star z)$*;*
  - *there exists an element* $x \Rightarrow z$ *such that*

$$y \leq (x \Rightarrow z) \text{ iff } x \star y \leq z.$$

  The residuated lattice L *is complete if any subset* $S \subseteq \mathbf{L}$ *has infimum and supremum denoted by* $\bigwedge S$ *and* $\bigvee S$*, respectively.*

Given a complete residuated lattice L*, we define the institution* $\mathcal{MVL}_L$ *as follows.*

- $Sig^{\mathcal{MVL}_L} = Sig^{\mathcal{FOL}}$*;*
- *Sentences of* $Sen^{\mathcal{MVL}_L}(S, F, P)$ *are pairs* (ρ, p) *where*
  - p *is an element of* $\mathbf{L}$*, and*
  - ρ *is a sentence generated from relational atoms of* $\pi(t_1, \ldots, t_n)$*, for* $\pi \in P_{s_1 \ldots s_n}$ *and* t_i *an algebraic term of sort* s_i*, by the (extended) set of connectives* $\{\Rightarrow \wedge, \vee, \top, \bot, \star\}$ *and quantifiers* $(\forall X)$ *and* $(\exists X)$ *for* X *a finite set of variables.*

- *a model* $M \in |Mod^{\mathcal{MVL}_L}(S,F,P)|$ *consists of*
  - *an algebra* (S,F),
  - *for each* $\pi \in P_{s_1 \ldots s_n}$, *a function* $M_\pi : M_{s_1 \ldots s_n} \to \mathbf{L}$.

  Morphisms between models M *and* N *are algebra homorphisms such that for any* $\pi \in P_{s_1 \ldots s_n}$, $M_\pi(t_1,\ldots,t_n) \leq N_\pi(h_{s_1}(t_1),\ldots,h_{s_n}(t_n))$.

- *For any* $M \in Mod^{MVL_L}(S,F,P)$ *and for any* $(\rho,p) \in Sen^{\mathcal{MVL}_L}(S,F,P)$ *the satisfaction relation is given by*

$$M \models^{\mathcal{MVL}_L}_{(S,F,P)} (\rho,p) \text{ iff } p \leq (M \models \rho)$$

  where $M \models \rho$ *is inductively defined as follows:*
  - *for any relational atom* $\pi(t_1,\ldots,t_n)$, $(M \models \pi(t_1,\ldots,t_n)) = M_\pi(M_{t_1},\ldots,M_{t_n})$;
  - $(M \models \top) = \top$;
  - $(M \models \bot) = \bot$;
  - $(M \models \rho_1 \odot \rho_2) = (M \models \rho_1) \odot (M \models \rho_2)$, *for* $\odot \in \{\wedge, \vee, \Rightarrow, \star\}$;
  - $(M \models (\forall X)\rho) = \bigwedge \{M' \models \rho | M' \restriction_{(S,F,P)} = M\}$;
  - $(M \models (\exists X)\rho) = \bigvee \{M' \models \rho | M' \restriction_{(S,F,P)} = M\}$.

This institution captures a number of multi-valued logics in the literature. For instance,

Example 2.2.8. Fuzzy logic ($\mathcal{FZL}$) *is the institution of* $\mathcal{MVL}$, *where* L *is the Łukasiewicz arithmetic lattice over the closed interval* $[0,1]$, *where* $x \star y = 1 - max\{0, x+y-1)\}$ *(and* $x \Rightarrow y = min\{1, 1-x+y\}$*).*

Example 2.2.9. 3–valued logic ($3\mathcal{VL}$), *is the institution of* $\mathcal{MVL}$, *where the alphabet of* L *has 3 elements.*

2.2.2 Morphisms

There are two types of arrows that can be defined to relate institutions, morphisms and comorphisms. The former are adequate for expressing "forgetful" operations from a "more complex" institution to a structurally "simpler" one. The latter "embed" a "simpler" institution into a more "complex" one.

One motto of category theory is that "arrows is what matters", a fact founded on the notion that any category can be defined just in terms of morphisms [2]. Also supporting this motto, suitable comorphisms in the category of institutions, have been successfully used for providing proof support to different logics. Such is possible because conservative comorphisms allow to "borrow" proof support, by "transporting" specific theories in a given logic, into another that, in principle, has better proof support.

A comorphism in the category of institutions, is formally defined as follows:

Definition 2.2.3. *Consider two institutions* $\mathcal{I} = (Sign^{\mathcal{I}}, Sen^{\mathcal{I}}, Mod^{\mathcal{I}} \models^{\mathcal{I}})$ *and* $\mathcal{I}' = (Sign^{\mathcal{I}'}, Sen^{\mathcal{I}'}, Mod^{\mathcal{I}'}, \models^{\mathcal{I}'})$. *A comorphism,* $\mathcal{I} \to \mathcal{I}'$ *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*:* $Mod^{\mathcal{I}'}.\Phi^{op} \Rightarrow Mod^{\mathcal{I}}$*, where for all* $\Sigma \in |Sign^{\mathcal{I}}|$, $\Phi(\Sigma)$*-Models M' and* Σ*-sentences* ρ *:*

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

Definition 2.2.4. *On the conditions above, the following commuting diagram of model transformations,*

[2] Objects are the identity arrow.

$$\begin{array}{ccc} Mod^{\mathcal{I}}(\Sigma) & \xleftarrow{\beta_{\Sigma}} & Mod^{\mathcal{I}'}(\Phi(\Sigma)) \\ Mod^{\mathcal{I}}(\varphi) \uparrow & & \uparrow Mod^{\mathcal{I}'}(\varphi') \\ Mod^{\mathcal{I}}(\Sigma') & \xleftarrow[\beta_{\Sigma'}]{} & Mod^{\mathcal{I}'}(\Phi(\Sigma')) \end{array}$$

is a weak amalgamation square, if and only if, 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.*

Note that the definition above is a special case of the Definiton 2.2.2.

Definition 2.2.5. *A comorphism is conservative whenever for any* $\Sigma \in |Sign^{\mathcal{I}}|$, β_{Σ} *is surjective. I.e., for any* Σ*–model* M, *there is a* $\Phi(\Sigma)$*–model* M', *such that* $\beta_{\Sigma}(M') = M$.

A conservative comorphism may be used as a sound way of "borrowing" proof support from other logics. Actually, this is method that we chose for giving proof support to hybrid logics.

There is also a special kind of comorphism (typically called an encoding) devised for when the source institution is "too complex" to be "embedded" into the target one. In an encoding such structural complexity is shifted to the mapping Φ on signatures, by switching the target institution with its presentation:

Definition 2.2.6. *For any institution* $\mathcal{I}$, *its presentation* $\mathcal{I}^{pres}$ *is also an institution. In the latter, signatures* $\Sigma \in |Sign^{\mathcal{I}}|$, *are extended to pairs* (Σ, Γ), *such that* $\Gamma \subseteq Sen^{\mathcal{I}}(\Sigma)$. *A presentation morphism* $\varphi : (\Sigma, \Gamma) \to (\Sigma', \Gamma')$ *is a signature morphism* $\varphi : \Sigma \to \Sigma'$ *such that* $\Gamma' \models Sign^{\mathcal{I}}(\varphi)(\Gamma)$. *Models are restricted to be the ones of* $Mod^{\mathcal{I}}(\Sigma)$, *where* Γ *is satisfied, i.e.,* $M \models^{\mathcal{I}}_{\Sigma} \Gamma$ *for* $M \in |Mod^{\mathcal{I}}(\Sigma)|$

2.2.3 The hybridisation method

Software products are built and maintained, with respect to requirements of different nature. Moreover, if one decides to formalise them using

specification logics, will probably find the need for different ones, suitable for whatever type of requirement that may appear.

To "connect" requirements of different families in a specification, it may be of the interest of the user to formally combine the logics found more suitable to deal with them. Actually this conforms with what Goguen and Meseguer beautifully put in [GM87]:

"The right way to combine various programming paradigms is to discover their underlying logics, combine them, and then base a language upon the combined logic."

Currently, sundry results on the subject of combining logics at the institutional level, exist. From such, one may mention:

1. The addition of a linear temporal logic layer (*cf.* [ST12]) to a given institution. Sentences are extended with the expected machinery, and the resulting models are linear sequences of the base institution's models.

2. The modalisation process [DS07], as the name suggests, brings the modal machinery to an institution. As such, the hybridisation method underlying this dissertation [MMDB11] extends this work.

3. [Dia13] presents a very recent result, which turns an institution into a many-valued one.

As already mentioned, the hybridisation process [Mad13] is an extension of the modalisation one, which endows the hybrid properties to a given logic, while retaining the latter's original properties. Thus, hybridised logics may be used to support the following methodology :

Whenever reconfigurable systems are in order, chose a logic to talk about the configurations. Then, hybridise it for dealing with the reconfigurations.

In a more technical perspective, the hybridisation process acts on the category of institutions: For an object $\mathcal{I}$ in this category, the hybridisation

gives an object $\mathcal{HI}$ in the same category, by extending the former logic with the hybrid machinery.

Formally

Definition 2.2.7. *Given an institution,* $\mathcal{I} = (Sign^{\mathcal{I}}, Sen^{\mathcal{I}}, Mod^{\mathcal{I}}, (\models^{\mathcal{I}}_{\Sigma})_{\Sigma \in |Sign^{\mathcal{I}}|})$, *the hybridisation method produces another institution,*
$\mathcal{HI} = (Sign^{\mathcal{HI}}, Sen^{\mathcal{HI}}, Mod^{\mathcal{HI}}, (\models^{\mathcal{HI}}_{\Sigma})_{\Sigma \in |Sign^{\mathcal{HI}}|})$, *such that:*

- $Sign^{\mathcal{HI}} = Sign^{\mathcal{I}} \times Sign^{\mathcal{H}}$. *Objects in* $Sign^{\mathcal{H}}$ *are tuples* $(Noms, \Lambda)$, *where* $Noms$ *denote a set of nominals, and* Λ *the set of modalities. Signature morphisms for* $Sign^{\mathcal{H}}$ *are defined as expected.*

- *Given a signature* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, *with* $\Delta = (Noms, \Lambda)$, $Sen^{\mathcal{HI}}((\Sigma, \Delta))$ *is the least set such that :*

  - $Noms \subseteq Sen^{\mathcal{HI}}((\Sigma, \Delta))$;
  - $Sen^{\mathcal{I}}(\Sigma) \subseteq Sen^{\mathcal{HI}}((\Sigma, \Delta))$;
  - $\rho \Rightarrow \rho' \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$, *for* $\rho, \rho' \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$;
  - $\neg\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$, *for* $\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$;
  - $@_i\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$, *for* $\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$ *and* $i \in Noms$;
  - $[m]\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$, *for* $\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$ *and* $m \in \Lambda$;
  - $\textcircled{\forall} x \,.\, \rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$, *for* $\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta'))$, *with* $\Delta' = (Noms \uplus \{x\}, \Lambda)$.

  For any morphism in the $Sign^{\mathcal{HI}}$ *category,* $(\varphi, \theta) : (\Sigma, \Delta) \to (\Sigma', \Delta')$ *with* $\Delta = (Noms, \Lambda)$, *sentences are mapped in the following way :*

$$
\begin{array}{lcl}
Sen^{\mathcal{HI}}((\varphi,\theta))(n) & = & \theta_{Noms}(n), \text{ for } n \in Noms \\
Sen^{\mathcal{HI}}((\varphi,\theta))(\delta) & = & Sen^{\mathcal{I}}(\varphi)(\delta), \text{ for } \delta \in Sen^{\mathcal{I}}(\Sigma) \\
Sen^{\mathcal{HI}}((\varphi,\theta))(\rho \Rightarrow \rho') & = & Sen^{\mathcal{HI}}((\varphi,\theta))(\rho) \Rightarrow Sen^{\mathcal{HI}}((\varphi,\theta))(\rho') \\
Sen^{\mathcal{HI}}((\varphi,\theta))(\neg\rho) & = & \neg Sen^{\mathcal{HI}}((\varphi,\theta))(\rho) \\
Sen^{\mathcal{HI}}((\varphi,\theta))(@_i\rho) & = & @_{\theta_{Noms}(i)} Sen^{\mathcal{HI}}((\varphi,\theta))(\rho) \\
Sen^{\mathcal{HI}}((\varphi,\theta))([m]\rho) & = & [\theta_{\Lambda}(m)] Sen^{\mathcal{HI}}((\varphi,\theta))(\rho) \\
Sen^{\mathcal{HI}}((\varphi,\theta))(\textcircled{\forall}\, x \,.\, \rho) & = & \textcircled{\forall} x \,.\, Sen^{\mathcal{HI}}((\varphi,\theta'))(\rho), \text{ where} \\
 & & \theta' \text{ canonically extends } \theta \text{ with } \theta'_{Noms}(x) = x
\end{array}
$$

- *For any signature* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$*, a model* $M \in |Mod^{\mathcal{HI}}((\Sigma, \Delta))|$ *is a tuple* (f, R)*, such that :*

  - $R \in |Mod^{\mathcal{H}}(\Delta)|$*. A model in* $Mod^{\mathcal{H}}(\Delta)$ *has a carrier set* $|R|$ *defining the possible worlds; for each* $n \in Noms_{\Delta}$*,* R_n *is interpreted as the world pointed by* n*, and for each modality symbol* m *in* Δ*, a binary relation* $R_m \subseteq |R| \times |R|$ *defining the corresponding accessibility relation. In other words, such models are Kripke frames with a naming function.*

    Given a model $R' \in |Mod^{\mathcal{H}}(\Delta')|$*, its reduct by* $\theta : \Delta \to \Delta'$*, denoted as* $R' \restriction_{\theta}$*, is defined in the following way:*

    - $|R'| = |R' \restriction_{\varphi}|$,
    - *for any* $n \in Noms_{\Delta}$, $R_n = R'_{\theta_{Noms}(n)'}$
    - *for any* $m \in \Lambda_{\Delta}$, $R_m = R'_{\theta_{\Lambda}(m)}$

  - f *is a function such that* $f : |R| \to |Mod^{\mathcal{I}}(\Sigma)|$*. I.e., each world of the hybrid component points to a model in the base institution. Notice that all models in the codomain, share a unique signature. This is a known limitation of the hybridisation method.*

  The reduct (f, R)*, of a model* $(f', R') \in |Mod^{\mathcal{HI}}((\Sigma', \Delta'))|$*, by the signature morphism* $(\varphi, \theta) : (\Sigma, \Delta) \to (\Sigma', \Delta')$*, is defined as :*

$$(f, R) = (Mod^{\mathcal{I}}(\varphi).f', Mod^{\mathcal{H}}(\theta)(R'))$$

- *For any signature,* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$*, a model* $(f, R) \in |Mod^{\mathcal{HI}}((\Sigma, \Delta))|$*, a world* $w \in |R|$*, and a sentence* $\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta))$*, the respective satisfaction relation is defined in the following way :*

$$
\begin{array}{lll}
(f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} n & \textit{iff} & R_n = w, \textit{for } n \in Noms_\Delta \\
(f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \delta & \textit{iff} & f(w) \models^{\mathcal{I}}_{\Sigma} \delta,\ \delta \in Sen^{\mathcal{I}}(\Sigma) \\
(f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho \Rightarrow \rho' & \textit{iff} & (f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho' \textit{ whenever } (f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho \\
(f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \neg\rho & \textit{iff} & (f,R),w \not\models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho \\
(f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} @_i\rho & \textit{iff} & (f,R),R_i \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho \\
(f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} [m]\rho & \textit{iff} & \textit{for all } (w,w') \in R_m,\ (f,R),w' \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho \\
(f,R),w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \textcircled{\forall} x\,.\,\rho & \textit{iff} & \textit{for all model expansions } R' \textit{ of } R \textit{ along the} \\
 & & \quad \textit{inclusion morphism previously defined in } Sen^{\mathcal{HI}}, \\
 & & (f,R'),R'_x \models^{\mathcal{HI}}_{(\Sigma,\Delta')} \rho
\end{array}
$$

Lemma 2.2.3. *On the conditions above, the following diagram is a strong amalgamation square for* $Mod^{\mathcal{HI}}$,

$$
\begin{array}{ccc}
(\Sigma,\Delta) & \xrightarrow{(\varphi,\theta)} & (\Sigma_2,\Delta_2) \\
{\scriptstyle (id,x)}\big\downarrow & & \big\downarrow{\scriptstyle (id,x^{(\varphi,\theta)})} \\
(\Sigma,\Delta_1) & \xrightarrow[(\varphi,\theta')]{} & (\Sigma_2,\Delta')
\end{array}
$$

where θ' *canonically extends* θ *with* $\theta'_{Noms}(x) = x$, *and* (id,x) *and* $(id,x^{(\varphi,\theta)})$ *are inclusion morphisms adding a fresh nominal* x *to the respective signatures.*

Proof. *From* $Mod^{\mathcal{HI}}((id,x))(M_1) = Mod^{\mathcal{HI}}((id,x^{(\varphi,\theta)}))(M_2)$ *we know that:*

$$
\begin{array}{c}
|R_1| = |R_2|, \\
\textit{for any } n \in Noms_\Delta\ R_{1n} = R_{2\theta_{Noms}(n)}, \\
\textit{for any } m \in Mods_\Delta\ R_{1m} = R_{2\theta_{Mods}(m)}
\end{array}
$$

$\Rightarrow$ *{definition of inclusion morphism; transitivity }*

For all model expansions M' *of* M_2 *by the inclusion morphism* $(id,x^{(\varphi,\theta)})$:

$$
\begin{array}{c}
|R_1| = |R'|, \\
\textit{for any } n \in Noms_\Delta\ R_{1n} = R'_{\theta_{Noms}(n)}, \\
\textit{for any } m \in Mods_\Delta\ R_{1m} = R'_{\theta_{Mods}(m)}
\end{array}
$$

$\Rightarrow$ *{$|R_1| = |R'|$; inclusion morphism $(id, x^{(\varphi,\theta)})$ definition }*

There is exactly one model expansion M' of M_2 by the inclusion morphism $(id, x^{(\varphi,\theta)})$ such that:

$$|R_1| = |R'|,$$
$$\text{for any } n \in Noms_\Delta \; R_{1n} = R'_{\theta_{Noms}(n)},$$
$$\text{for any } m \in Mods_\Delta \; R_{1m} = R'_{\theta_{Mods}(m)},$$
$$\text{and } R_{1x} = R'_x$$

$\Leftrightarrow$ *{definition of the inclusion morphism (id, x); definition of θ' }*

There is exactly one model expansion M' of M_2 by the inclusion morphism $(id, x^{(\varphi,\theta)})$, such that:

$$|R_1| = |R'|,$$
$$\text{for any } n \in Noms_{\Delta_1} \; R_{1n} = R'_{\theta'_{Noms}(n)},$$
$$\text{for any } m \in Mods_{\Delta_1} \; R_{1m} = R'_{\theta'_{Mods}(m)}$$

$\Leftrightarrow$ *{ M_1 has exactly the same underlying models of $\mathcal{I}$ than M. The same happens between M' and M_2. }*

There is exactly one model expansion of M_2 by the inclusion morphism $(id, x^{(\varphi,\theta)})$ such that

$$|R_1| = |R'|,$$
$$\text{for any } n \in Noms_{\Delta_1} \; R_{1n} = R'_{\theta'_{Noms}(n)},$$
$$\text{for any } m \in Mods_{\Delta_1} \; R_{1m} = R'_{\theta'_{Mods}(m)},$$
$$\text{and } f_1 = Mod^{\mathcal{I}}(\varphi).f'$$

$\Leftrightarrow$ *{$Mod^{\mathcal{HI}}$ reduct definition }*

There is exactly one model expansion M' of M_2 by the inclusion morphism $(id, x^{(\varphi,\theta)})$, such that:

$$(f_1, R_1) = Mod^{\mathcal{HI}}((\varphi, \theta'))(f', R')$$

□

Satisfaction proof. Given a signature $(\Sigma', \Delta') \in |Sign^{\mathcal{HI}}|$, a signature morphism $(\varphi, \theta) : (\Sigma, \Delta) \to (\Sigma', \Delta')$, a (Σ', Δ')–model M', and a (Σ, Δ)–sentence ρ, for any world, $w \in M'_{|R'|}$:

When $\rho := n, n \in Noms_\Delta$:

$$M' \restriction_{(\varphi,\theta)}, w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} n$$

$\Leftrightarrow$ {Satisfaction definition }

$$(M' \restriction_{(\varphi,\theta)})_{R_n} = w$$

$\Leftrightarrow$ {Reduct definition }

$$M'_{R_{\theta_{Noms}(n)}} = w$$

$\Leftrightarrow$ {Satisfaction definition }

$$M', w \models^{\mathcal{HI}}_{(\Sigma',\Delta')} \theta_{Noms}(n)$$

$\Leftrightarrow$ {$Sen^{\mathcal{HI}}$ definition }

$$M', w \models^{\mathcal{HI}}_{(\Sigma',\Delta')} Sen^{\mathcal{HI}}((\varphi, \theta))(n)$$

When $\rho \in Sen^{\mathcal{I}}(\Sigma)$:

$$M' \restriction_{(\varphi,\theta)}, w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho$$

$\Leftrightarrow$ {Satisfaction and reduct definition }

$$(M' \restriction_{(\varphi,\theta)})_{Mod^{\mathcal{I}}(\varphi).f'(w)} \models^{\mathcal{I}}_{\Sigma} \rho$$

$\Leftrightarrow$ {The base logic is an institution }

$$(M' \restriction_{(\varphi,\theta)})_{f'(w)} \models^{\mathcal{I}}_{\Sigma'} Sen^{\mathcal{I}}(\varphi)(\rho)$$

$\Leftrightarrow$ {Reduct definition }

$$M'_{f'(w)} \models^{\mathcal{I}}_{\Sigma'} Sen^{\mathcal{I}}(\varphi)(\rho)$$

$\Leftrightarrow$ {Satisfaction definition }

$$M', w \models^{\mathcal{HI}}_{(\Sigma',\Delta')} Sen^{\mathcal{I}}(\varphi)(\rho)$$

$\Leftrightarrow$ {$Sen^{\mathcal{HI}}$ definition }

$$M', w \models^{\mathcal{HI}}_{(\Sigma',\Delta')} Sen^{\mathcal{I}}((\varphi,\theta))(\rho)$$

Proofs for the next two cases are analogous to the same ones in $\mathcal{FOL}$. When $\rho := @_i\rho$,

$$M' \restriction_{(\varphi,\theta)}, w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} @_i\rho$$

$\Leftrightarrow$ {Satisfaction definition }

$$M' \restriction_{(\varphi,\theta)}, R_i \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho$$

$\Leftrightarrow$ {Induction hypothesis }

$$M', R_i \models^{\mathcal{HI}}_{(\Sigma',\Delta')} Sen^{\mathcal{HI}}((\varphi,\theta))(\rho)$$

$\Leftrightarrow$ {Reduct definition }

$$M', R'_{\theta_{Noms}(i)} \models^{\mathcal{HI}}_{(\Sigma',\Delta')} Sen^{\mathcal{HI}}((\varphi,\theta))(\rho)$$

$\Leftrightarrow$ {Satisfaction definition }

$$M', w \models^{\mathcal{HI}}_{(\Sigma',\Delta')} @_{\theta_{Noms}(i)} Sen^{\mathcal{HI}}((\varphi, \theta))(\rho)$$

$\Leftrightarrow$ $\{Sen^{\mathcal{HI}}$ definition $\}$

$$M', w \models^{\mathcal{HI}}_{(\Sigma',\Delta')} Sen^{\mathcal{HI}}((\varphi, \theta))(@_i \rho)$$

Proof for the modality case is analogous to the above.

Proof for the quantification case is analogous to the quantification case in $\mathcal{FOL}$, when replacing the employed amalgamation lemma, by Lemma 2.2.3. □

Concluding this section, it should be noted that the notion of hybridisation just given, is less powerful than the ones presented in [MMDB11] and [Mad13]. For instance, we do not deal with quantification spaces (*cf.* [Dia10]) which are essential to define which kinds of quantification are supported at the level of the hybrid component. Actually, we are interested in "pushing" quantification over sorts of the base logic (in the case of such machinery existing) to the hybrid component, as it provides interesting ways to write statements. The "push" may be achieved by restricting the worlds of the underlying Kripke frame to have exactly the same carrier sets, as described in the more powerful version of the hybridisation notion. We make the restriction of the common realisation of carrier sets to the hybridised logics presented in this dissertation.

2.3 Proof support for hybrid logics

2.3.1 Conventional proof strategies

Considerable work has been made on proof theory for hybrid logics since the 90's. In particular proof theory for $\mathcal{HPL}$ and $\mathcal{HFOL}$ has been notably extended in [Bra11], paving the way for a number of proof-tool implementations. However, to the best of our knowledge there is not any assisted prover for $\mathcal{HFOL}$.

Several proof methods, considered suitable for tool implementation, have been developed for $\mathcal{HPL}$. Resolution and tableau are among the most successful ones on this category, and as such, they were consistently relied on for the implementation of several dedicated $\mathcal{HPL}$ proof tools. In the next section, some of them are briefly described, to contextualise a comparison between them and the tools developed in the context of this dissertation. Naturally, such comparison can only be made for $\mathcal{HPL}$, due to the absence of proof tools for other hybrid logics.

Translation is another proof strategy, suitable for implementation. It works by "borrowing" the proof support from another logic. A legitimate choice for the logic being targeted in such translation, is $\mathcal{FOL}$, as it is endowed with several forms of proof methods and an impressive number of proof tools.

As it should be expected, the translation strategy has its own limitations: When translating into another logic, one may be forced to enter into undecidability, or at least increased complexity.

The translation method, is used in this dissertation to at some extent, adding proof support for hybridised logics [MMDB11]. The following section, describes this technique from a technical perspective.

2.3.2 Proofs with comorphisms

We have presented hybridised logics as suitable candidates when the specification of reconfigurable systems is in order. However, their potentialities for specification shall always be limited if assisted reasoning is not supported.

Reference [MMDB11] describes how a comorphism from a base institution $\mathcal{I}$ to $\mathcal{FOL}$, can be lifted into another one from $\mathcal{HI}$ to $\mathcal{FOL}$, while in the well–behaved cases retaining conservativeness. In such cases, this provides means to "borrow" the proof environment from $\mathcal{FOL}$, and use it on $\mathcal{HI}$. Formally speaking,

Definition 2.3.1. *Given a comorphism,* $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{FOL}$,
the lifting, $(\Phi', \alpha', \beta') : \mathcal{HI} \to \mathcal{FOL}$ *is defined in the following way:*

- *For a signature* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, $\Phi'((\Sigma, \Delta)) = (S', F', P')$, *such that for* $\Phi(\Sigma) = (S, F, P)$*:*
  - $S' = S \uplus \{World\}$,
  - $F' = (F_{World, w \to s} | F_{w \to s} \in F) \uplus (n | n \in \Delta_{Noms})_{\to World}$, *for* $w, s \in S^*$,
  - $P' = (P_{World,w} | P_w) \uplus (m | m \in \Delta_\Lambda)_{World,World}$, *for* $w \in S^*$;
- *Given a signature* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$ *and a* (Σ, Δ)*–sentence* ρ, α' *is defined as follows:*

$$\alpha'_{(\Sigma,\Delta)}(\rho) = (\forall x : World)\, \eta^x_{(\Sigma,\Delta)}(\rho), \text{ where}$$

$$\begin{array}{lcl}
\eta^x_{(\Sigma,\Delta)}(n) & = & x \approx n, \text{ for } n \in Noms_\Delta \\
\eta^x_{(\Sigma,\Delta)}(\rho) & = & (\alpha^x_\Sigma . \alpha_\Sigma)(\rho), \text{ for } \rho \in Sen^{\mathcal{I}}(\Sigma) \\
\eta^x_{(\Sigma,\Delta)}(\rho \Rightarrow \rho') & = & \eta^x_{(\Sigma,\Delta)}(\rho) \Rightarrow \eta^x_{(\Sigma,\Delta)}(\rho') \\
\eta^x_{(\Sigma,\Delta)}(\neg\rho) & = & \neg\eta^x_{(\Sigma,\Delta)}(\rho) \\
\eta^x_{(\Sigma,\Delta)}(@_i\rho) & = & \eta^i_{(\Sigma,\Delta)}(\rho) \\
\eta^x_{(\Sigma,\Delta)}([m]\rho) & = & (\forall y : World)\, m(x,y) \Rightarrow \eta^y_{(\Sigma,\Delta)}(\rho) \\
\eta^x_{(\Sigma,\Delta)}(\textcircled{\forall} x' . \rho) & = & (\forall x' : World)\, \eta^x_{(\Sigma,\Delta')}(\rho), \text{ where } \Delta' = (\Delta_{Noms} \uplus \{x'\}, \Delta_\Lambda)
\end{array}$$

where,

$$\begin{array}{lcl}
\alpha^x_\Sigma(f(t_1, \ldots, t_n)) & = & f(x, \alpha^x_\Sigma(t_1), \ldots, \alpha^x_\Sigma(t_n)), \text{ for } f \in F_\Sigma \\
\alpha^x_\Sigma(t \approx t') & = & \alpha^x_\Sigma(t) \approx \alpha^x_\Sigma(t') \\
\alpha^x_\Sigma(p(t_1, \ldots, t_n)) & = & p(x, \alpha^x_\Sigma(t_1), \ldots, \alpha^x_\Sigma(t_n)), \text{ for } p \in P_\Sigma \\
\alpha^x_\Sigma(\rho \Rightarrow \rho') & = & \alpha^x_\Sigma(\rho) \Rightarrow \alpha^x_\Sigma(\rho') \\
\alpha^x(\neg\rho) & = & \neg\alpha^x_\Sigma(\rho) \\
\alpha^x_\Sigma(\forall y : s . \rho) & = & \forall y : s . \alpha^x_{\Sigma'}(\rho), \text{ where } \Sigma' = (S_\Sigma, F_\Sigma \uplus \{y\}_{\to s}, P_\Sigma)
\end{array}$$

- *Given a signature* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, *and a model* $\Phi'(\Sigma, \Delta)$*–model* M, $\beta'_{(\Sigma,\Delta)}(M) = (f, R)$ *such that :*
  - $|R| = |M_{World}|$,
  - *For each symbol* $n \in \Delta_{Noms}$, $R_n = M_n$,
  - *For each symbol* $m \in \Delta_\Lambda$, $R_m = M_m$,

- *For each $w \in |R|$, $f(w) = \beta_\Sigma(M')$, with M' being defined as :*
  - *For each sort $s \in (S_\Sigma \backslash World)$, $|M_s| = |M'_s|$*[3]
  - *For each $\sigma \in (F_{a \to s})_\Sigma \backslash (F_{\to World})_\Sigma$, $M_{\sigma_w} = M'_\sigma$, where $a, s \in S^*_\Sigma$ and σ_w denotes the function resulting from applying the argument w to σ;*
  - *For each $p \in (P_a)_\Sigma \backslash (P_{World,World})_\Sigma$, $M_{p_w} = M'_p$, where $a \in S^*_\Sigma$ and p_w denotes the predicate resulting from applying the argument w to predicate p.*

2.3.3 Dedicated provers for the propositional case

There are several dedicated provers for $\mathcal{HPL}$. Among them, HyLoRes[AH02] and HTab[HA09] distinguish themselves by their availability, stability, documentation and respective features, although others should also be mentioned, namely, HyLoTab [vE02], Pilate and Herod [CMC10], Hydra [BBW01] and Spartacus [GKS10].

Since one of the contributions of this dissertation is to "freely" provide hybridised logics (integrated in HETS) with the respective tool support, it is natural to have some kind of comparison against the dedicated provers. Therefore, follows a succinct description of HyLoRes and HTab, in order to have a more clear context for this comparison to be made in chapter 3.

HyLoRes is the first and the only prover, to our knowledge, implementing the $\mathcal{HPL}$ resolution method. Unfortunately, accordingly to its repositories, development has stopped in 2009. HyLoRes supports $\mathcal{HPL}$, with the binder and the global operator, entailing that termination is not guaranteed. Overall it is considered to be a successful prover, and is often involved in comparisons between $\mathcal{HPL}$ provers.

[3]Note that we may not have conservativeness for β', since we are assuming that these models (M') have the same carrier sets, which is not necessarily true and thus possible models are not being considered. The way to guarantee conservativeness, is to make suitable restrictions to the hybridised logic in question in order to remove the unconsidered models. Regarding the hybridised logics involved in this dissertation using proof support, the only restriction needed is that the states of the underlying Kripke frame must have the same realisation of carrier sets.

HTab, as its name suggests, uses the $\mathcal{HPL}$ tableau method, which is considered to be the most successful when dealing with modal logics. Like HyLoRes, supports $\mathcal{HPL}$ with the binder and the global operator, once again not guaranteeing termination. Additionally, HTab provides prototype mechanisms to build specifications driven by the user. To the best of our knowledge, HTab is the fastest dedicated $\mathcal{HPL}$ prover.

The algorithms behind these provers work in a similar way: The sentence being evaluated is negated, and a model that satisfies it is searched for. The search is made by removing possible hypothesis by means of contradictions. If in the end all hypothesis have been removed, *i.e.*, if no model exists, the sentence can be taken as valid.

Ending this section, two minor contributions resulting from the analysis of the described provers, are mentioned:

- Changes on the compiler used by HyLoRes, turned impossible its compilation on recent systems. However, in the context of this dissertation HyLoRes's code was updated and put available online [4]. As such, this prover be now used in the newest systems, and moreover, along the update process, a deeper perspective of HyLoRes's inner workings was obtained.

- In order to avoid the hassle of obtaining and compiling each prover separately, a package was made that provides an easier way of installing several of these systems without the need for compiling. More concretely, a Mac Os package including HyLoRes, HTab and Spartacus binaries, was built and put available online[5]. Hopefully this will provide an easier way to use such systems, thus widening their use within the scientific community.

[4]https://github.com/nevrenato/HyLoRes_Source
[5]https://github.com/nevrenato/Hybrid_package

Chapter 3

Hybridisation in HETS

The core of this dissertation is the integration of the hybridisation method into HETS. This work, was framed by two milestones. In the first one, a single language was to be hybridised, and the respective proof support (as described in section 2.3.2) provided. Integration of these into HETS, would then follow. The second, aimed at a more generic hybridisation, one that potentially gives to the end user an hybridised version of each logic already available in HETS.

The present chapter describes the results achieved in this dissertation with respect to both milestones.

3.1 Background

3.1.1 HETS as the implementation framework

The Heterogeneous Tool Set (HETS), based on the theory of institutions, follows the multi logic approach by providing the means to integrate a number of institutions, comorphisms, and respective tools within a single framework. Therefore, one can see HETS as a graph whose nodes are individual logics (institutions), and the edges are translations (comorphisms) between them, providing a method to "transport" properties and proofs inside the network. Additionally we have extra an dimension over the latter, relating nodes and compatible reasoning tools.

Such tools range from easy-to-use automatic theorem provers (as DARWIN [BFT06], SPASS [WDF+09], VAMPIRE [RV02], EPROVER [Sch02]) to interactive ones (such as ISABELLE [NWP02], VSE [AHL+00] or LEO II [BTPF08]).

HETS is targeted as an implementation framework for the hybridisation process, given its institutional theoretical foundation which fits well this approach. Furthermore, the community supporting HETS, is increasing and actively developing improvements, extensions, and fixing bugs. Therefore, at this level, the hybridisation method has fertile grounds to grow.

3.1.2 CASL

The Common Algebraic Specification Language [MHST03], also known as CASL, was developed within the CoFI initiative, with purpose of creating a sort of an universal language for specifying requirements and design conventional software packages. The resulting product is a language extending $\mathcal{FOL}$ with:

1. Subsorting;
2. Generated types, *i.e.*, types whose all elements are exclusively built by constructors;
3. Free types, *i.e.*, types that are generated, and whose different constructors terms must denote different elements;
4. Partial functions.

From an institutional perspective, a CASL signature is a tuple (S, TF, PF, P), where S is the set of sort symbols; TF the set of function symbols; PF the set of partial function symbols and P the set of relational symbols. As expected, $TF \cap PF \subseteq \emptyset$. If subsorting functionality is intended, then the previous signature is extended with a partial order relation $\leq$ on sorts.

Sentences extend $\mathcal{FOL}$'s ones in order to accommodate the functionalities previously described.

Currently CASL is regarded as the *de facto* standard language for algebraic specifications. Furthermore it is integrated into HETS, along with many of its expansions, as depicted by figure 3.1. One such expansion, particularly relevant to the work presented here, is MCASL [Mos04]. The latter adds to CASL the ability to deal with modalities, therefore turning it into a modal logic whose underlying worlds, correspond to CASL models. Naturally, we are dealing with an expansion similar to the hybridised version of CASL (which is introduced in the present chapter). Actually MCASL was of utmost importance as an object of analysis, used to understand how to implement a CASL extension in HETS.

Other relevant expansions of CASL, currently implemented in HETS are:

- CoCASL [MRRS06] which introduces coalgebraic types. It should be interesting to explore in which ways $\mathcal{H}$CASL relates to CoCASL, since a modal logic can be generalised into a coalgebraic one [CKP$^+$08];

- CspCASL [Rog03] which combines the process algebra CSP with CASL;

- HasCASL [SM09] which extends CASLwith the implementation of a partial λ–calculus. This provides functional programming and specification within a single language, consequently narrowing the gap between both. Furthermore, portions of HasCASL's hybrid[1] models can actually be executed, since a comorphism from HasCASL to Haskell[2] is implemented. It should be very interesting to ascertain how the hybridised version of HasCASL, behaves in terms of state–based programming.

[1]Specifications in HasCASL are hybrid, in the sense that code and specification are mixed together.

[2]Haskell is also captured as an institution, meaning that HETS can be embedded to itself!

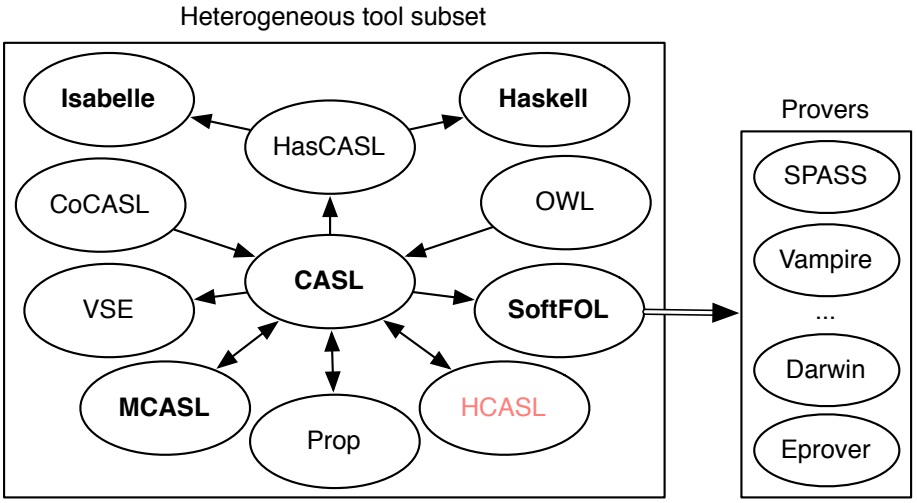


Figure 3.1: A subset of HETS logic graph

3.2 On applying the hybridisation to CASL

3.2.1 $\mathcal{H}$CASL in HETS

The first step of integrating the hybridisation process into HETS, consisted of the hybridisation of CASL, the almost *lingua franca* of HETS, and its posterior implementation in the framework. Then, resorting to the technique presented in section 2.3.2, we provided proof support for it.

Recalling that the only hybrid logic with tool support was $\mathcal{HPL}$, this first result is interesting because we gave tool support to a powerful hybridised logic. Furthermore, since CASL harbours may sub–logics, its hybridisation can also accommodate the hybrid versions of those. Thus, this result actually gives tool support to a number of hybridised logics. Examples of such are : $\mathcal{HPL}$, $\mathcal{HEQ}$, $\mathcal{HFOL}$, $\mathcal{HFP}$ ...

From a more technical perspective, applying the hybridisation method to CASL results in a new logic (henceforth named $\mathcal{H}$CASL) which extends the former with the expected hybrid machinery. In particular, $\mathcal{H}$CASL extends CASL signatures, by supporting the declaration of nominals and modalities. In addition, sentences are extended as follows:

$$\rho := n \mid \delta \mid \rho \Rightarrow \rho' \mid \neg\rho \mid @_i\,\rho \mid [m]\rho \mid \textcircled{\forall} x\,.\,\rho$$

for n a nominal, δ a CASL sentence, ρ, ρ' $\mathcal{H}$CASL sentences, and m a modality.

Notice that, there are repeated operators in the grammar above: Both $\neg$ and $\Rightarrow$ are defined at the hybrid level, although, they were already present in CASL. Although repeated operators may cause ambiguities, such is not the case here since the repeated ones ($\neg$, $\Rightarrow$) can be "collapsed" into the correspondent originals, as they semantically coincide (proof in [Mad13]).

$\mathcal{H}$CASL's underlying models are Kripke frames whose worlds point to CASL models obliged to share a given signature, as imposed by the hybridisation method. In addition, we need to restrict the pointed models to have the same realisation of carrier sets. Otherwise, there will be $\mathcal{H}$CASL's models not reachable by β, thus conservativeness is lost (*cf.* section 2.3.2). Such restriction provides another benefit: Since all words share the same carrier sets, quantification in CASL may be "pushed" to the level of the hybrid component (*cf.* section 2.2.3). Consequently, we gain useful, new ways of writing statements as we will see later on.

Given the details of $\mathcal{H}$CASL as an hybridised logic we are ready to go over its integration into HETS.

One fundamental step in the process of integrating $\mathcal{H}$CASL into HETS, was the reverse engineering of other CASL extensions already there implemented. This so that we could have a better understanding of the framework implementation at a low level. In particular, MCASL was notably useful for such, since it is a similar logic to $\mathcal{H}$CASL. From analysing such extensions, we have learned that CASL acts as a boilerplate [3], making easier and rather mechanical to create new extensions. Actually to concretise a CASL extension, one just needs to,

1. instantiate the datatype encoding CASL signatures, with the one defining the signatures of the extension;

[3] In fact, CASL is built in HETS as a generic data structure with "holes" to be filled by each extension.

2. do the same thing with respect to sentences;

3. extend the CASL's parser in order to support the new functionalities;

4. finally, extend the CASL's semantics analyser accordingly.

Following this procedure, CASL's signature datatype and the corresponding parser were enriched in order to support the declaration of nominals and modalities, as described by the following grammar:

Nominals-decl := `nominals` **Ids**
Modalities-decl := `modalities` **Ids**
Ids := **Id**(,**Id**)*

In addition, CASL's sentences were extended as follows :

CSen := **HSen** | **CSen** $\wedge$ **CSen** | $\neg$ **CSen** | **Term** $\approx$ **Term** | **...**
HSen := **@** ***Id*** **CSen** | $\langle$**Id**$\rangle$ **CSen** | [**Id**] **CSen** | $\langle$**Id**$\rangle$'' **CSen** |
`Here` ***Id*** | ! **Id CSen** | ? ***Id*** **CSen** | `true` | `false`

As already mentioned, we added a new constructor (**HSen**) to the one syntactically defining CASL's sentences (**CSen**) [4], thus, extending the latter with the typical hybrid machinery. Furthermore one may notice that there are "extra" (*i.e.* not included in the hybridisation method) alternatives in the **HSen** constructor. Actually, such alternatives are macros defined as follows:

- The second one is the dual of the modality operator;

- The fourth one corresponds to patterns of the kind $[r]\rho \wedge \langle r\rangle\rho$ ($\langle r\rangle''\rho \equiv [r]\rho \wedge \langle r\rangle\rho$). This pattern was found to be commonly used on specifications involving hybridised logics, hence its definition as a macro;

- The seventh one, is the dual of the universal quantification of nominals, which is denoted by the exclamation point;

[4] **CSen** is a generic datatype, which we are instantiating with the **HSen** datatype.

- Finally the last two, define the typical $true/false$ macros.

An additional macro was implemented regarding rigid designators: Whenever the keyword `rigid` is put behind a declaration of predicates or operations, they will behave in a rigid way, as section 2.1.3 describes.

Notice also the fifth alternative, denoting the case of when a nominal is taken for a sentence – The keyword `Here` is necessary, so that the CASL's parser does not wrongly sees the ensuing nominal as the typical proposition.

Finally, sentences in **HSen** obey the left precedence approach. For instance, $\mathcal{H}$CASL sees the sentence $@_n p \wedge q \wedge r$, as $((@_n p) \wedge q) \wedge r$.

We have just described how $\mathcal{H}$CASL extends CASLregarding syntax. On the side of semantics, a number of features were added in order to correctly analyse $\mathcal{H}$CASL's specifications. The resulting analyser inherits the features from the CASL's analyser, therefore can detect invalid specifications with respect to the CASL component. In addition, the features added provide means to detect invalid specifications regarding the hybrid component. In this context, they are invalid whenever one of the following is true:

- Repeated declarations of nominals/modalities;
- Quantified nominals sharing symbols with declared nominals. For instance, a specification with a declared nominal, n, and a quantified nominal, n, is not valid;
- Sentences with nominals/modalities that were not previously declared or introduced by nominal quantification.

We have also mentioned that $\mathcal{H}$CASL's specifications can be validated resorting to the proof method provided. Such comes from the lifting introduced in section 2.3.2, and is currently integrated into HETS, thus providing to the end user an impressive number of proof tools. More concretely the user has at its disposal any proof tool already made available to CASL.

The proof method achieved for $\mathcal{H}$CASL (*i.e.* its translation targeting $\mathcal{FOL}$) is conservative due to the restriction applied to $\mathcal{H}$CASL which states that local models must have a common realisation of the carrier sets. Thus, we have a sound method to generate proofs for $\mathcal{H}$CASL's specifications.

Lets us give an example of this translation at work. Consider an $\mathcal{H}$CASL signature $\Sigma = ((S, F, P), (Noms, \Lambda))$:

$S = \{A\}$,

$F = \{\}$,

$P = \{R\}_{A \times A}$,

$Noms = \{N\}$,

$\Lambda = \{M\}$.

Translating the signature above to $\mathcal{FOL}$, results in,

$S = \{A, World\}$,

$F = \{N\}_{\rightarrow World}$,

$P = \{R\}_{World \times A \times A} \cup \{M\}_{World \times World}$.

Now consider the following sentences built from Σ:

$@_N \langle M \rangle N$;

$\forall a, a' : A \,.\, @_N R(a, a') \Rightarrow @_N [M] R(a', a)$.

The respective translation returns sentences equivalent to,

$M(N, N)$;

$\forall a, a' : A \,.\, R(N, a, a') \Rightarrow (\forall w : World \,.\, M(N, w) \Rightarrow R(w, a', a))$.

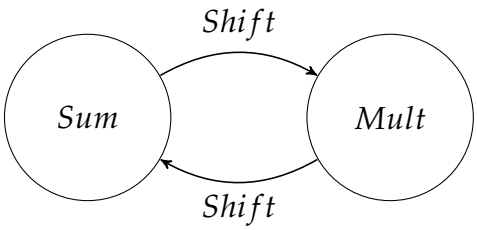
3.2.2 $\mathcal{H}$CASL at work – A case study

$\mathcal{H}$CASL provides means to model in a natural way reconfigurable systems whose modes of operation can be suitably specified with CASL. The objective of this section is to illustrate $\mathcal{H}$CASL at work, through a small case study. We present below the key points of the corresponding specification, and divided into the following points:

1. The specification of the underlying Kripke frame, *i.e.*, identification of the possible modes of operation and transitions between them;
2. Identification of the "global" signature used by the "local" configurations;
3. "Global" property definition, *i.e.*, definition of properties common to all configurations;
4. "Local" property definition, *i.e.*, definition of properties in the context of each local configuration.

The full specification can be seen in appendix A.

Example 3.2.1. *The swinging calculator, is a reconfigurable system whose only operation changes its behaviour along two possible states. In one state it behaves like the addition over naturals, in the other like multiplication. One switches between these two modes through the Shift command, as depicted below.*



Let us then start by focusing on the underlying Kripke frame of this example.

The swinging calculator was described to have two possible operating modes – *Sum* and *Mult*. Additionally, such modes are reachable from one another. One possible formulation for this, is as follows:

modalities *Shift*
nominals *Sum*, *Mult*

$@Sum\ \neg\ Mult$
$Sum \vee\ Mult$
$@Sum\ \langle Shift\rangle"\ Mult$
$@Mult\ (\langle Shift\rangle\ \ Sum \wedge [Shift]\ \ Sum)$

The purpose of the first two sentences, is to define the possible modes of operation. In particular, the first one avoids models where *Sum* and *Mult* are collapsed into each other. While the second one limits models to have at maximum two possible states, *Sum* and *Mult*. Thus all valid Kripke frames for this example will have precisely the two desired modes of operation.

The possible reconfigurations are defined by the last two sentences, each "create and cut" the relevant relations, accordingly to the example's description. Actually, the macro $\langle r\rangle"\rho$ (defined in section 3.2.1) , is useful in these situations as it provides means to "create and cut" in "one step". This can be seen in the last two sentences, where one applies the macro and the other does not, although it could.

The specification of the "global" signature of the swinging calculator, is as follows,

op $\_\#\_ : Nat \times Nat \rightarrow Nat$

with # denoting the operation that behaves as the addition or multiplication, depending on the current state.

Let us now focus on the specification of the "global" properties of this system. Recalling that operation # behaves as either addition or multiplication, we have the commutative and associative laws as possible "global" properties:

$\forall\ n, m, p : Nat$
• $n \# m = m \# n$
• $(n \# m) \# p = n \# (m \# p)$

Finally, let us focus on the "local" properties. They are the ones defining the operation # relative to each possible modes of operation, *i.e.*, Sum and $Mult$. For both states, the operation # was defined accordingly to Peano's arithmetical definitions, as shown below:

$\forall\ n, m : Nat$
• $@Sum\ n \# 0 = n$
• $@Sum\ n \# suc(m) = suc(n \# m)$
• $@Mult\ n \# 0 = 0$
• $\exists\ p, q : Nat$
  • $@Mult\ n \# suc(m) = p \wedge @Sum\ n \# q = p \wedge @Mult\ n \# m = q$

Completed the specification the next natural step is to check for properties. The following are simple examples of properties that one may want to prove.

$\forall\ n, m, r : Nat$
• $@Sum\ (n < m \Rightarrow n < m \# r)$ %(lemma1)%
• $@Mult\ (m = 0 \vee m = suc(0) \Rightarrow n \# m <= n)$ %(lemma6)%
• $\exists\ p : Nat \bullet @Sum\ n \# n = p \Rightarrow @Mult\ n \# suc(suc(0)) = p$ %(DoubleDef)%
• $\exists\ p, q : Nat$
  • $m <= suc(0) \Rightarrow @Sum\ n \# m = p \wedge @Mult\ n \# m = q \Rightarrow p >= q$
  %(CasesSumBiggerMult)%

All of the properties presented above were easily proved by one of the reasoners connected to HETS, namely SPASS.

To conclude this case study, figure 3.2 registers an HETS session relative to this example showing the proof window, part of the model theory, and the specification graph.

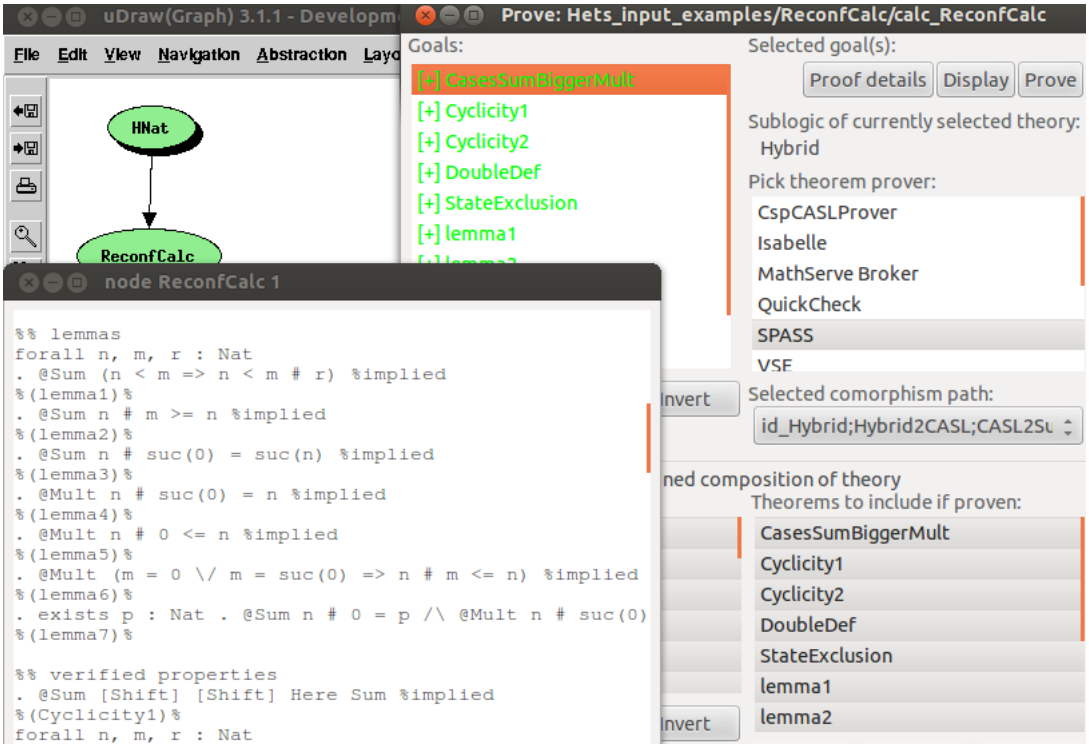


Figure 3.2: HETS session respective to the swinging calculator example

3.2.3 $\mathcal{H}$CASL as an $\mathcal{HPL}$ prover

The integration of $\mathcal{H}$CASL (jointly with its encoding to $\mathcal{FOL}$) into HETS, provides new proof tools to a significant number of hybridised logics. In this context, it should be interesting (at least from an academic point of view) to have a comparison, for each one of these logics, of such "old" tools with the ones provided in this dissertation. As already mentioned, prior to the work relative to this dissertation the only hybridised language with tool support was the $\mathcal{HPL}$, therefore the envisaged comparison is only possible for this case.

The comparison consists of feeding a set of sentences to both classes of tools and analysing the respective outputs to provide conclusions with respect to performance. However, not being one of the goals of this dissertation, the comparison to be presented is not at all complete. Thus, any conclusion taken from it, should only be taken as indicative.

In section 2.3.3, several dedicated $\mathcal{HPL}$ provers were mentioned, and two (HTab and HyLoRes) of them distinguished due to their performance and documentation, among other things. They make one side of the comparison.

On the other side, we have two state-of-the-art $\mathcal{FOL}$ theorem provers, currently connected to HETS – SPASS and Darwin, although in the interest of being practical, the latter is only used on the most interesting cases.

The set of input formulas used for the comparison, was obtained from HTab[5] and HyLoRes[6] online repositories, and filtered in order to make this comparison as balanced as possible. As such, the resulting set contains sundry input formulas ranging from ones with singular clauses to others with hundreds.

For automation purposes and to pave the way for a more thorough comparison, a small application with two functionalities was developed:

- the first one picks specifications in HTab's format, and makes them

[5]http://hylo.loria.fr/intohylo/htab.php
[6]http://hylo.loria.fr/intohylo/hylores.php

compatible to $\mathcal{H}$CASL;

- the second one does the same with HyLoRes as the source format.

This application is available online[7].

Tables 3.1 and 3.2, register the outputs obtained from this comparison, in terms of time. We use centiseconds as the unit of measure, and any result equal or above 18000 (3 minutes) is classified as a timeout. Notice that while the table 3.1 concerns unsatisfiable formulas, table 3.2 goes over satisfiable ones. The latter are usually harder to check, thus we give more focus to them. Its also worth noticing that the same tables, use the column Id to uniquely identify the formulas within the input set [8].

The results shown provide interesting information with respect to the performance of the provers being tested. The most obvious, is that HTab is an impressive tool, even when compared against the other dedicated tool, HyLoRes. Both for simple formulas, are consistently faster than the method implemented in this dissertation, which is not surprising since they are dedicated. Yet, when looking to the results of the more complex formulas, things are not so obvious.

It is interesting to see that when formulas start to get harder, the gap between the "old" and "new" tools closes substantially; there are even cases where the "new" performs better. This is probably due to world class provers (where SPASS and Darwin fit) integrating heavy optimizations to tackle hard formulas.

Whenever complex formulas are taken into consideration one can clearly see that HyLoRes performs badly against our proposal. On the other hand, HTab is notably faster – both SPASS and Darwin rarely reach similar or better times. However, notice the results of the test formula, D_tab, defined as $\textcircled{\forall} x \,.\, @_x(\textcircled{\exists} y \,.\, y \wedge \langle m \rangle \neg y)$. From analysing the underlying algorithms of both HTab and HyLoRes, it seems that formulas akin to the latter induce an infinite loop on these systems. In order to understand why this happens, we need to know two particular rules of these

[7] https://github.com/nevrenato/HTab2HCASL

[8] Online available at: https://github.com/nevrenato/HTab2HCASL

algorithms: Whenever verifying a given sentence,

1. if an instance of the universal nominal quantifier appears, all existing nominals must be checked to see if they conform with the statement associated with that instance;

2. if an instance of the existential nominal quantifier appears, a fresh nominal is introduced.

Now, notice that the formula, $\textcircled{\forall} x \,.\, @_x(\textcircled{\exists} y \,.\, y \wedge \langle m \rangle \neg y)$, has an existential nominal quantifier "inside" a universally quantified statement. Due to this, whenever rule 1 is applied, a new nominal is created in the process (rule 2); then, since a new nominal was introduced, the algorithm is obliged to use rule 1 again. However, by applying rule 1 we create yet another nominal and so we have an infinite loop.

To sum up, the dedicated $\mathcal{HPL}$ tools involved in this comparison (*i.e.* HTab and HyLoRes), seem to be consistently better in terms of performance when compared to our proposal. This is expected as such tools are dedicated $\mathcal{HPL}$ provers, while on the other side we have non dedicated ones (*i.e.* SPASS and Darwin) accessed through HETS. However, when the input formulas get increasingly harder, the difference between them blurs to a point where it is not easy to see which side best performs. Typically, it is the hard formulas and not the simple ones that may cause problems. Thus, taking into account what was said about the performance of the "new" tools, we think that they present an interesting alternative even for property verification in the $\mathcal{HPL}$ domain. Moreover for non propositional hybridised logics the "new" tools are the only option available.

As already mentioned, one must be careful about further conclusions taken from the comparison presented as it is incomplete in many ways. For instance, the set of input formulas used is very small; instead, a generator of random $\mathcal{HPL}$ formulas should be put to use. Moreover only two of the many provers connected to HETS were used. Indeed with enough time, one could resort to other world class provers, such as Vam-

Id	HTab	HyLoRes	SPASS	Darwin
01	1	1	10	
02	1	1	12	
03	1	1	9	
04	1	1	14	
05	2	1	12	
06	1	1	12	
07	1	1	14	
11	1	1	14	
12	1	1	13	
13	2	1	8	
29	1	8	15	
32	1	1	9	
34	1	1	8	
35	1	1	5	
37	1	1	6	
39	11	1774	1070	70
40	15	timeout	3500	2120

Table 3.1: Unsatisfiable formulas with timeout being at 18000 csec

pire, MathServe or EProver[9]. The results obtained, even if indicative are useful to illustrate our point.

3.3 Hybridisation in HETS – going generic

In broad terms, the previous sections of this chapter introduced and explored $\mathcal{H}$CASL along with the respective proof support. This relates to the first milestone of the core of this dissertation, *i.e.*, the hybridisation of a specific logic, as the first step to integrating the hybridisation process into HETS. The current section presents the second step, *i.e.*, the incorporation of the hybridisation process into HETS, in a generic way. In theory, such provides by request the hybridised version of a given logic available in HETS; which includes already hybridised logics.

[9]Actually, as new provers are constantly being added into HETS, this task would never end.

Id	HTab	HyLoRes	SPASS	Darwin
02	1	1	12	
03	1	1	12	
04	1	1	9	
05	2	1	13	
06	1	1	11	
07	1	1	8	
13	1	1	17	
17	1	1	13	
18	2	1	16	
19	1	8	17	
21	1	1	18	
30	1	1	6	
31	1	1	9	
34	1	1	11	
35	11	1774	10	
39	40	timeout	1600	timeout
41	11	timeout	146	timeout
42	140	timeout	262	50
43	45	timeout	289	290
44	46	timeout	702	660
D_tab	timeout	timeout	13	< 1
hard	1800	timeout	3300	timeout

Table 3.2: Satisfiable formulas with timeout being at 18000 csec

In practice, we extended HETS with a *quasi* generic hybridisation proces. It is *quasi* in the sense that some manual intervation is in order for each first hybridisation of a logic in HETS: The sentence's parser and analyser of the logic to be hybridised need to be presented to the hybridisation framework. This is made at code level. Once done, the logic in question can be hybridised an arbitrary number of times. Actually, the following logics already meet the referred requirements to be hybridised in this way: CASL, PROPOSITIONAL and CoCASL.

On a more technical perspective, the hybridisation framework is implemented in HETS as an "incomplete" logic, *i.e.*, we do not know at the full extent which are its possible signatures and sentences. We complete it when parsing the input specification by checking which base logic the user has chosen. Then, whenever necessary, the hybridisation framework resorts to the tools specific to that logic. HETS is not prepared for this kind of procedure and moreover does not know all the tools respective to a given logic. This justifies the condition mentioned above, needed to hybridise a logic in HETS.

A challenge that emerges from the pursue of generic hybridisation, is to avoid ambiguities arising from combining the hybrid component with the base logic. Actually they may have symbols in common which do not necessarily have the same semantical meaning. For instance, take an $\mathcal{H}$CASL specification whose signature has a nominal p and a predicate p. It is not clear if the sentence $p \wedge p$, is referring to the nominal p, the predicate p, or both. In a first instance this was resolved by enforcing the use of the keyword `Here` before a nominal. However, in the context of a generic hybridisation such technique does not remove all ambiguities. To see why this is true, consider an $\mathcal{H}^2\mathcal{PL}$ specification, that on the top level has a nominal n, and on the lower one has another nominal n. A sentence such as `Here` $n \wedge$ `Here` n, clearly does not carry enough information for us to know which nominals are being referred, even when using the previous technique. Note that $\mathcal{H}^2\mathcal{PL}$ is the logic resulting from the hybridisation of $\mathcal{HPL}$.

In order to overcome the ambiguity problem once and for all, a syn-

tactical constraint was introduced: Sentences corresponding to the base logic, must be wrapped within brackets. Thus, in the $\mathcal{H}^2\mathcal{PL}$ specification above, the sentence $\texttt{Here}\, n \wedge \texttt{Here}\, n$ is changed to $n \wedge \{n\}$. It becomes clear that the left side refers to nominal n of the top level, while the right side corresponds to nominal n of the base logic (*i.e.* $\mathcal{HPL}$). Notice also that with this approach, the sentence $n \wedge \{n \wedge m\}$ in $\mathcal{H}^2\mathcal{PL}$, is equivalent to $n \wedge \{n\} \wedge \{m\}$, as the operators denoted by $\wedge$, semantically coincide.

Let us now present the generic hybridisation framework at work, through an illustrative example

logic *Hybrid*
baselogic *Hybrid*
Basic_Spec {
    baselogic *Propositional*
    Basic_Spec { **props** *p* }
    Nominals *Portugal*, *England*, *Canada*
    Modalities *Car*
}
Nominals *Europe*, *America*
Modalities *Plane*
• *Europe* ⇒ { (*Portugal* ∨ *England*) };
• @ *Europe* ⟨*Plane*⟩ *America*;
• @ *Europe* { @ *Portugal* ⟨*Car*⟩ *England* }

Its corresponding logic is a double hybridisation of the PROPOSITIONAL language and gives geographic information [10]. In particular, it describes routes in a map linked by some means of transport (the modalities) between different places (identified by nominals). One level of hybridisation corresponds to countries; the second one to continents. Clearly, in this case nominals can be ordered with respect to the order used to build an

[10] Contrary to the usual, we do not impose the restriction regarding the common realisation of carrier sets to $\mathcal{H}^2\mathcal{PL}$ (*cf.* section 2.2.3). Otherwise, the intended hierarchy would be lost.

hierarchy of countries and continents. Note however, that is impossible to define connections between countries of different continents, as the available transitions may access only one level of hierarchy. The next chapter explores and gives more detail about specifications written in $\mathcal{H}^2\mathcal{PL}$.

Chapter 4

Case study : The insulin infusion pump

The motivation for the incorporation of the hybridisation process into HETS, a main objective of this dissertation, was to contribute with effective tool support for a methodology for the suitable specification of reconfigurable systems, by expressing them as structural automata, whose states define configurations, and transitions reconfigurations. The theoretical foundations for this, were laid out in [MMDB11] with the introduction of the hybridisation method. The corresponding practical side is the focus of this dissertation.

In chapter 3, we integrated the hybridisation method into HETS, or in other words, we brought the hybridisation to the working software engineer. Such provides a concrete framework for the specification of reconfigurable systems with a suitable logic which combines the hybrid component with any logic that the user may see suitable to formalise the configurations of a particular system.

This chapter illustrates the whole methodology at work. The case study, from the medical domain, is the partial specification of a medical device that has been helping millions to lead a normal life – the insulin infusion pump (henceforth called IIP).

Diabetes mellitus is an incurable disease affecting the lives of millions

around the world. In the most severe forms, diabetes is controlled by insulin injections, a treatment that hinders a normal life for the patient in many ways.

The insulin infusion pump was created with the purpose to replace the referred treatment. It works by injecting insulin trough a subcutaneous catheter, at different rates along time, in order to compensate the occurring insulin disequilibriums. An increasing number of patients is turning to this kind of devices due to the numerous benefits that come with them, namely increased convenience and flexibility, as well as greater dose precision and quicker dosing adjustments.

Even though the IIP acts as a key contributor for patients to lead a normal life, it does imposes risks (*cf.* [ZJJ10]). Risks come from the complexity rooted in the infusion pump technology, and may cause severe harm, or even death. On the software side, formal specification methods may help on mitigating such dangers.

The application of (formal) specification techniques to design and develop IIPs, is an ongoing concern of the scientific community, which has already tackled this subject with several approaches. For example, [AJJ$^+$07] uses reference models (*i.e.* sets of communicating state machines) which are translated into UPAAL for verification. Also, the Msc. thesis [XM12], takes Event-B as specification language to model a version of state machines oriented to timing issues. In both cases a specific logic is chosen based on the perspective from which the IIP is seen. Our approach differs here: Instead of fixing a particular logic, we chose a suitable one whenever a requirement of a different nature appears. Since the IIP is a complex and heterogeneous device, with requirements of different kinds, it should be appropriate to take a number of logics for the complete specification. Also, independently of the logics being chosen, one wants to retain the automata specification capabilities, hence their hybridised versions are in order. Tool support for this comes at some extent with the new version of HETS introduced in chapter 3.

Concerning the IIPs safe behaviour, Zhang *et al*, give important contributions on this subject (*cf.* [ZJJ10, ZJJR11]) by eliciting an impressive

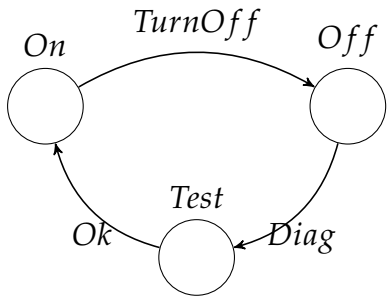
number of safety requirements. We base on them to illustrate the methodology proposed for specifying heterogeneous systems.

The remaining of this chapter introduces the specification of the infusion pump whose requirements are collected in appendix B. They are divided in different tables organised by the (hybridised) logic[1] that seems more suitable to express them. Actually this case study aims at illustrating the expressive power of this specification methodology in which different logics (and their hybridised versions) can be simultaneously used to tackle complex problems. Our integration of the hybridisation process in HETS, makes possible the offer of suitable proof support to reason and verify the corresponding properties.

4.1 On specifying the IIP – reconfigurations

The specifications described in this section, formalise the requirements in table B.1. The latter leads us to take the IIP system as a simple state machine where configurations are points, *i.e.*, they carry no information. Reconfigurations thus, are the only concern at this level, and a logic as simple as $\mathcal{H}1$ suffices. HETS however, does not embeds it; therefore to have suitable tool support, $\mathcal{HPL}$ is used instead.

The formalisation of first three requirements of table B.1, gives an automata as presented below:



[1]Recall that we always restrict the hybridised logics to follow the condition, mentioned in section 2.3.2, for "pushing" a possible quantification support to the hybrid component.

Let us then analyse each of the three requirements and the corresponding formalisations. The first requirement describes the possible modes of operation which we can simply formulate as:

$$On \vee Test \vee Off$$

Requirement 2 is more complex. It says that there is a path from state Off to On, with $Test$ being a middle point. Moreover, it imposes that On can only be directly reached from $Test$, and even then, only through a modality denoting that all diagnostic tests went ok. To formalise this requirement, firstly one "creates" the referred path where $Test$ is in the middle:

$$@_{Off}\langle Diag\rangle Test \wedge @_{Test}\langle Ok\rangle On$$

Then, the statement that On is only directly reachable from $Test$ through the modality Ok, ensues. This is done by forcing just $Test$ to be reached through the modality Ok, and prohibiting any transition to On, through modalities $TurnOff$ and $Diag$:

$$\langle Ok\rangle true \Rightarrow Test \wedge \neg(\langle TurnOff\rangle On \vee \langle Diag\rangle On)$$

Requirement 3 simply states that when On, the pump can be turned Off:

$$@_{On}\langle TurnOff\rangle Off$$

Such requirement may be however not "strong" enough. For instance, it allows the pump to go from On to $Test$ through $TurnOff$, which does not make much sense. It is possible to forbid this case by changing the previous sentence to,

$$@_{On}\langle TurnOff\rangle Off \wedge @_{On}[TurnOff]Off$$

which is equivalent to[2],

$$@_{On}\langle TurnOff\rangle "Off$$

[2] $\langle r\rangle "\rho \equiv \langle r\rangle\rho \wedge [r]\rho$, for ρ a sentence of the respective hybrid logic, and r a modality

These first three requirements already give a minimal automata, which can be validated resorting to the provers connected to HETS. For instance, we can prove properties regarding cyclic paths, such as

$$@_{On}\langle TurnOff\rangle\langle Diag\rangle\langle Ok\rangle On$$

The complete specification of this fragment of the problem is available in appendix C (specification IIP_FST).

Requirement 4 on table B.1 leads us to a second stage in this specification. Its description suggests a refinement of the problem by stating that two states are particular cases of another one. Such is not possible to specify with $\mathcal{HPL}$, thus a change of logic is in order.

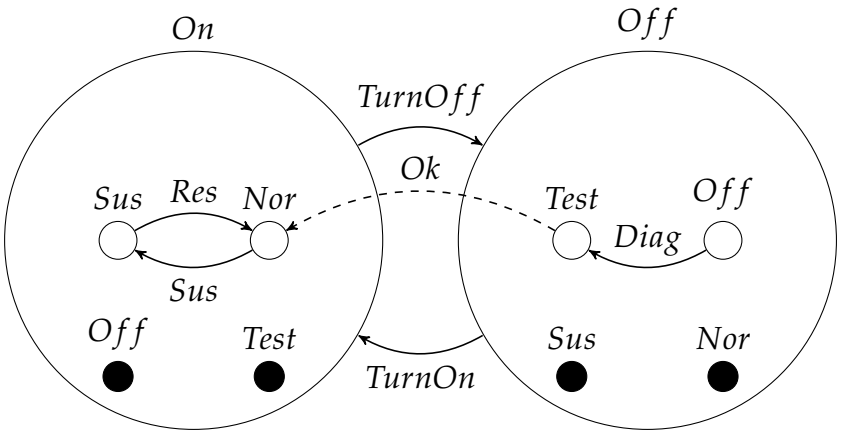
In this new stage the previous configurations cease to be just points. They become themselves state machines. We are therefore dealing with hierarchic state machines.

What is the suitable logic to specify hierarchic state machines ?

In the context of the hybridisation process, the following line of thought may be legitimately used: $\mathcal{H}1$ deals with state machines, whose states are points. Hybridising $\mathcal{H}1$ again, results in a logic able to express state machines whose states are the underlying models of $\mathcal{H}1$, *i.e.*, state machines with states as points. $\mathcal{HH}1$ (denoted as $\mathcal{H}^2 1$) is thus one possible answer. However, as done before in order to guarantee tool support we use $\mathcal{H}^2\mathcal{PL}$ instead [3].

$\mathcal{H}^2\mathcal{PL}$ is used to specify the last three requirements (4, 5 and 6) of table B.1. The corresponding specification defines the following hierarchic state machine:

[3]Contrary to the usual, we do not impose the restriction regarding the common realisation of carrier sets to $\mathcal{H}^2\mathcal{PL}$ (*cf.* section 2.2.3). Otherwise, the intended hierarchy would be lost.



Before going into the details of each requirement, notice the black states within On and Off in the above figure. Although unnecessary, they cannot be removed due to a limitation (tackled in chapter 6) of the hybridisation method: Configurations must share a given signature. In the context of $\mathcal{H}^2\mathcal{PL}$, this rule limits all supernominals (nominals carrying other nominals) to carry the same set of states. Hence, supernominals can become "overpopulated". One way to deal with this is to turn the unwanted states inaccessible. For instance, $Test$ and Off are unwanted within On. Then:

$$@_{On}\{\neg(\langle Res\rangle(Off \vee Test) \vee \langle Sus\rangle(Off \vee Test) \vee \langle Diag\rangle(Off \vee Test))\}$$

Note that the length of the above sentence increases proportionally to the number of declared modalities. The implication of this is that formulas akin to the ones above, may become cumbersome to write when many modalities are declared. One way to avoid this would be with modality quantification: The latter sentence would simply become,

$$@_{On}\{\neg\exists m : \Lambda \,.\, \langle m\rangle(Off \vee Test)\}$$

Unfortunately, quantification over modalities is not included in the hybridisation method described in this dissertation. However, [Mad13] presents the hybridisation process

Consider now requirement 4. It states that when the pump is On, there are two possible modes of operation: $Suspension$ and $Normal$. With $\mathcal{H}^2\mathcal{PL}$ this becomes:

$$@_{On}\{Sus \vee Nor\}$$

Requirement 5, defines the reconfigurations between states *Suspension* and *Normal*, which can be formalised as:

$$@_{On}\{@_{Sus}\langle Res\rangle''Nor \wedge @_{Nor}\langle Sus\rangle''Sus\}$$

Finally, we have requirement 6 (denoted by the dashed arrow in the corresponding state machine) addressing refinement in a clear way. It announces a transition *Ok*, from Off to *On*. But in particular, from *Test* (within Off) to *Nor* (within *On*). This is not possible to formulate as the classes of transitions available in $\mathcal{H}^2\mathcal{PL}$ cannot interact with super-nominals and nominals at the same time. In other words, transitions are limited to their corresponding hybrid components.

The full specification of this fragment of the problem is available in appendix C (specification IIP_DUPL).

4.2 On specifying the IIP – configurations

In this section we concentrate on the possible modes of insulin infusion. We regard the latter as the possible configurations of an IIP system which has different insulin outputs throughout the day. This is critical for the patient since inaccurate insulin quantities may cause harm, or even death.

To better contextualise the specifications, let us first analyse the different insulin infusion modes:

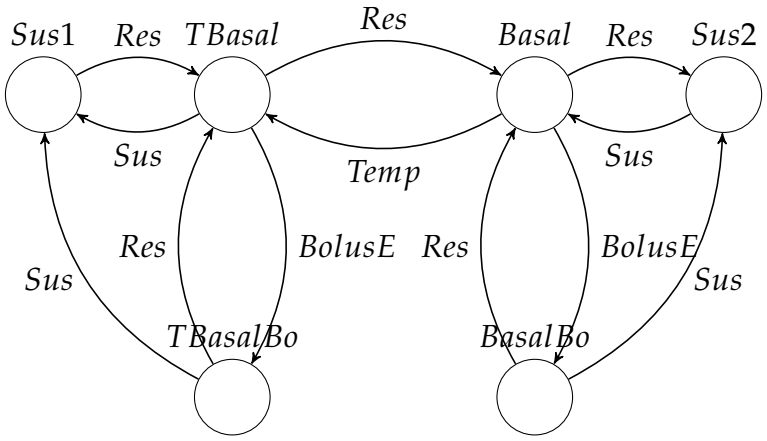
- Normal basal – The profile followed by default; it can be defined to allow the delivery of different quantities of insulin throughout the day;
- Temporary basal – Is used as a temporary measure when the normal basal mode is not suitable for a particular situation;
- Instantaneous bolus – Is a one time insulin delivery, typically used to cover food intake or similar situations. It adds up to one of the basal modes;

- Extended bolus – Serves the same purpose of the instantaneous bolus, but instead delivers the bolus at a constant rate along an interval of time.

Each mode characterises the insulin infusion rate, thus entailing that the configurations of the underlying Kripke models cannot be just points. Instead they must be instances of an algebra. Furthermore, the kind of algebra (or base logic) to employ is strongly related to the nature of the requirements to be specified. As such, since in the insulin infusion rate characterisations, requirements of different nature abound, several hybridised logics (each with its own underlying algebra) are used for specifying the insulin infusion rates.

As already mentioned, our concern at this stage is not on the transition system, but rather on "what is inside" the configurations. In this sense, the transition structure was defined once (based on the requirements in table B.2) and is the frame for the different models, emerging from the requirements relative to the insulin infusion rates. In other words, along the different models presented below, the transition structure remains the same; the configuration's algebra (*i.e.* the base logic), is what changes.

Let us start by analysing the transition structure, given by the formalisation of the requirements in table B.2:



From the figure above, is possible to see that the reconfigurations include the main modes of insulin infusion, basal (*Basal*) and temporary basal

($TBasal$); but also the bolus modes denoted as $TBasalBo$ and $BasalBo$. In addition, it is possible to see that from any mode of infusion the pump can be suspended ($Sus1, Sus2$) as the requirement 4 of table B.2 states. The reader may wonder about what is the reason to have two states denoting the suspension mode. The reason is that hybrid logics do not have memory of the previous state. Thus, if the pump is suspended and wants to resume it cannot know to which state should return to. If each relevant mode has its own suspended state, then when resuming, there is only one possible mode to return to.

The complete specification regarding the formalisation of the transition system given above, is available in appendix C (specification IIP_low_KripkeF).

The specification of the configurations (*i.e.* the insulin infusion modes), brings again the need to choose a suitable base logic to use. Clearly $\mathcal{PL}$ or $\mathcal{HPL}$ are no more suitable logics, since the requirements (from tables B.3,B.4) concerned with insulin infusion rates, require quantification, equality statements and relations. Table B.3 only requires the first two, therefore $\mathcal{FEQ}$ is a suitable logic to specify the respective requirements. Let us further analyse them.

Requirement 1 of this table details about the maximum insulin flow allowed in the different modes. In particular, it specifies that the maximum insulin flow must be 0 when the pump is suspended. The formalisation of this, entails first of all, the need to declare a constant function representing the maximum insulin flow:

$$maxFlow : Nat$$

Naturally, its value may change along the possible modes of operation. In other words, its value is only constant locally. Note also, that we chose the natural numbers to model the insulin flow. This is a clear abstraction from the physical world and may not be sufficiently precisely in more demanding contexts. But it is enough for the illustrative character of this chapter.

Declared a function denoting the maximum insulin flow, saying that its value is 0 in the suspended modes is straightforward:

$$(Sus1 \vee Sus2) \Rightarrow maxFlow = 0$$

The insulin flow at *Basal* and *TBasal* modes is defined by requirements 2, 3. They state that the insulin infusion rate must conform with the profiles corresponding to these modes. To model this, firstly three functions need to be declared. One to represent the current insulin flow, and two to represent the basal and temporary basal profiles. With respect to the latter, their values cannot change along the possible modes (requirement 6), *i.e.*, they must be rigid (*cf.* section 2.1.3):

$$\begin{array}{c} \textbf{\textit{rigid}} \;\; basal, tbasal : Time \rightarrow Nat \\ curFlow : Time \rightarrow Nat \end{array}$$

Note that we introduced a new sort (*Time*) in the specification with the purpose of having more intuitive descriptions for names. Actually, *Time* is just a type synonym for *Nat*.

With these functions declared, let us formalise requirements 2,3:

$$\begin{array}{c} @_{Basal} \forall t : Time \,.\, curFlow(t) = basal(t) \\ @_{TBasal} \forall t : Time \,.\, curFlow(t) = tbasal(t) \end{array}$$

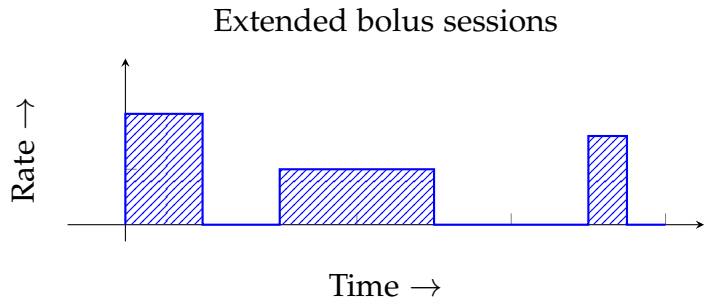
Requirement 4 is similar to 2 and 3. It states that in the modes where a bolus is being given, the current flow is the result of the corresponding bolus and the active profile. Once again, a new rigid function needs to be declared to denote the profile corresponding to the extended bolus:

$$\textbf{\textit{rigid}} \;\; extBolus : Time \rightarrow Nat$$

Then requirement 4 can be formalised:

$$\begin{array}{c} @_{BasalBo} \forall t : Time \,.\, curFlow(t) = basal(t) + extBolus(t) \\ @_{TBasalBo} \forall t : Time \,.\, curFlow(t) = tbasal(t) + extBolus(t) \end{array}$$

Requirement 5 distinguishes from the others because it refers to the behaviour of the extended bolus profile. In particular, it says that in any extended bolus session, the corresponding rates must be constant. However it does not define what a session is, giving room for different interpretations. Ours is that is a segment of the extended bolus profile whose insulin infusion rates never fall to 0 and whose adjacent points (*i.e.* time instants) always return 0. To exemplify this interpretation, the graphic below portrays extended bolus sessions denoted by the area filled with diagonal lines:



Given the definition of an extended bolus session, the formalisation of requirement 5 becomes:

$$\begin{gathered}\forall t : Time \,.\, extBolus(t) \neq 0 \Rightarrow \\ (extBolus(t-1) \neq 0 \Rightarrow extBolus(t) = extBolus(t-1)) \wedge \\ (extBolus(t+1) \neq 0 \Rightarrow extBolus(t) = extBolus(t+1))\end{gathered}$$

Since $\mathcal{HFEQ}$ is a sublogic of $\mathcal{H}$CASL, properties regarding the formulation of these requirements can be automatically verified. For instance, we can state that:

$$\neg(\forall t : Time \,.\, curFlow(t) = basal(t)) \Rightarrow \neg Basal$$

or,

$$\begin{gathered}(\forall t : Time \,.\, extBolus(t) = 0) \Rightarrow \\ \forall t : Time \,.\, \exists n : Nat \,.\, @_{Basal} curFlow(t) = n \wedge @_{BasalBo} curFlow(t) = n\end{gathered}$$

The formulation of the requirements on table B.2 reflects one of the limitations (which we tackle in chapter 6) of the hybridisation method – that all configurations must share a given signature. This can be seen in the respective specification since the infusion profiles are defined for all configurations, which is a too strong assumption. To see why consider the suspended mode where clearly the infusion profiles do not need to be defined there.

One way to tackle the overrestriction described is by employing the logic $\mathcal{HFP}$ instead of $\mathcal{HFEQ}$. Although it is still not possible to define a different signature for each configuration, one can however define at which states the infusion profiles are defined using partial functions embedded in $\mathcal{HFP}$. To do this firstly the profiles need to be declared as partial functions:

$$basal, tbasal, extBolus : Time \to? Nat$$

Then, we need to say at which states they are defined. For example, in the *basal* case the procedure should be as follows:

$$(Basal \vee BasalBo) \Rightarrow \forall t : Time \,.\, df(basal(t))$$
$$\neg(Basal \vee BasalBo) \Rightarrow \forall t : Time \,.\, \neg df(basal(t))$$

Note that, rigidity needs to be retained in order to conform with the requirements described. Naturally, for partial functions such condition can be only applied at the configurations in which they are defined. Let us exemplify how it may be formulated for the *basal* profile:

$$\forall t : Time \,.\, \exists n : Nat \,.\, @_{Basal} basal(t) = n \Leftrightarrow @_{BasalBo} basal(t) = n$$

Alternatively, by resorting to nominal quantification and the properties of the strong equation, the latter sentence can also be formulated in as follows:

$$\textcircled{\forall} w, w' \,.\, \forall t : Time \,.\, \exists n : Nat \,.\, @_w basal(t) = n \Rightarrow @_{w'} basal(t) = n$$

The sentence above is interesting because we do not have to explicitly refer the relevant states, thus we can apply it in a generic way to the other

infusion profiles. However, this approach is not used in this specification as nominal quantification increases complexity.

The next set of requirements to be formalised is from table B.4. Actually, just one requirement is present, but it is perhaps one of the most critical. It states that independently of the mode of operation, the current flow can never surpass the maximum insulin flow established. The formulation of this typically involves the use of relations. However, these are not built-in $\mathcal{HFEQ}$ or $\mathcal{HFP}$. Therefore a sensible step to take is to increase the power of the logics being used, accordingly. To do this we switch to $\mathcal{HFOL}$ which treats relations as first-class citizens. Then we proceed to formalise the requirement in question:

$$\forall t : Time \,.\, curFlow(t) \leq maxFlow$$

With $\mathcal{HFOL}$ interesting properties of this specification can be elaborated. Fortunately these can also be proved with the help of HETS as $\mathcal{HFOL}$ is a sub–logic of $\mathcal{H}$CASL. For example the following is easily proved:

$$@_{Basal} \forall t : Time \,.\, basal(t) \leq maxFlow$$

The above is interesting in the sense that it proves that any requirement concerning the validation of the basal profile with respect to the maximum flow, is redundant when the requirement 1 of table B.4 is present.

Another interesting (proved) property, states that the insulin flow in the suspended mode is not bigger than in any another modes:

$$\forall t : Time \,.\, \forall n : Nat \,.\, (@_{Sus1} curFlow(t) = n) \Rightarrow curFlow(t) \geq n$$

A last example of a proved property, is the following:

$$\forall t : Time \,.\, \exists n, n' : Nat \,.$$
$$@_{Basal} curFlow(t) = n \wedge @_{BasalBo} curFlow(t) = n' \wedge n \leq n'$$

It simply states that the insulin flow in *Basal* mode is not bigger than in *BasalBo*. This makes sense since the *BasalBo* mode has an insulin flow equal to the one in *Basal* mode plus any bolus currently active.

The complete specifications regarding the logics $\mathcal{HFEQ}$, $\mathcal{HFP}$ and $\mathcal{HFOL}$, are available in appendix C (specifications IIP_LOW and IIP_LOWPAR).

Up until now, the focus of this section was on the specification of different insulin infusion modes. For this, we resort to the hybridised versions of a number of conventional logics, namely: $\mathcal{PL}$, $\mathcal{FEQ}$, $\mathcal{FP}$ and $\mathcal{FOL}$. As a last stage in this case study, we focus on yet another IIP's critical component – the insulin reservoir.

The insulin reservoir has a sensor attached that is able to tell the amount of insulin remaining. However, this sensor (like all others) is not perfect in the sense may not be completely exactly. Therefore, these degrees of uncertainty should be taken into consideration, when specifying the IIP system. Many–valued or probabilistic logics, are the typical candidates for this kind of specification. But at the same time reconfigurability is in order, thus we need the hybridised versions of these logics.

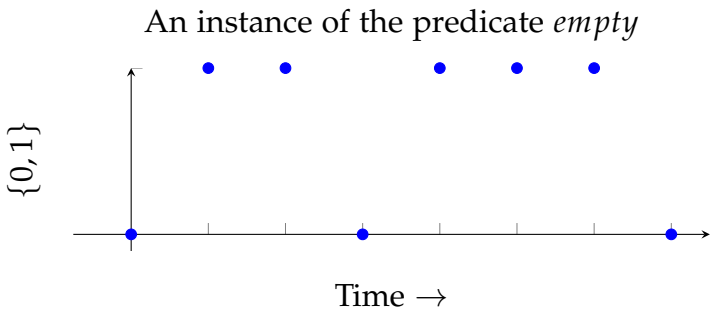
To illustrate what can be expressed by such logics, let us assume a predicate *empty*, that given an instant of time says if the reservoir is empty or not. In addition consider another predicate (*alarm*), that for an instant of time tells if the alarm is active:

$$empty, alarm : Time$$

In this context there could be a requirement stating: *Whenever the reservoir is found to be empty, in the next instant of time the alarm must be activated.* With $\mathcal{HFOL}$ this is formulated as:

$$\forall t : Time \,.\, empty(t) \Rightarrow alarm(t+1)$$

with a valid instance of the predicate *empty* being as follows:



However this sort of ideal situation typically does not exist in the "real world".

In practice, we may consider that there are cases where the value of predicate *empty* is unknown. For this, we can use $\mathcal{H}3\mathcal{V}\mathcal{L}$ since it has the truth value space of $\{0, u, 1\}$; with u denoting the unknown value.

Now, let us analyse a possible requirement : "In basal mode, at any instant of time, if one knows for sure that the reservoir is not empty, and in the next instant that becomes unclear, then it is also unclear if the alarm must be activated."

$$@_{Basal} \forall t : Time \,.\, \neg(empty(t), u) \wedge (empty(t+1), u) \Rightarrow (alarm(t+2), u)$$

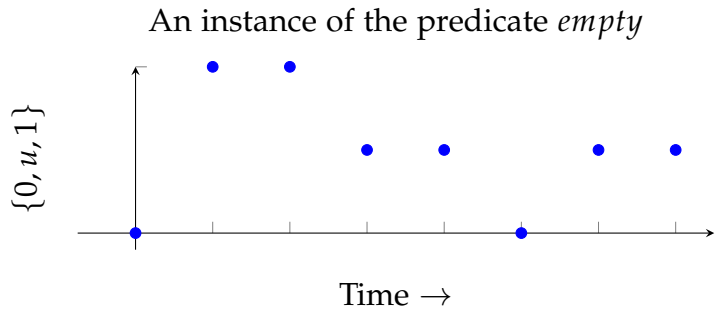
Continuing with the latter requirement, in basal with bolus mode, the insulin flow is usually bigger. As such, in the described situation we want to make sure that the alarm is activated:

$$@_{BasalBo} \forall t : Time \,.\, \neg(empty(t), u) \wedge (empty(t+1), u) \Rightarrow (alarm(t+2), 1)$$

On the other hand, since in the suspended mode there is no insulin flow, the alarm does not need to be activated:

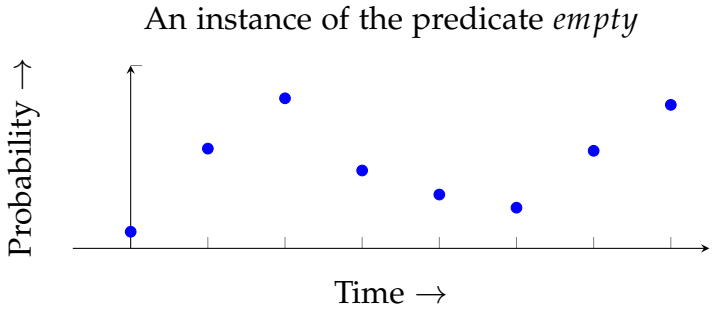
$$(Sus1 \vee Sus2) \Rightarrow$$
$$\forall t : Time \,.\, \neg(empty(t), u) \wedge (empty(t+1), u) \Rightarrow (alarm(t+2), 0)$$

The following graphic portrays the kind of uncertainty obtained, with predicate *empty* defined in the truth value space of $\mathcal{H}3\mathcal{V}\mathcal{L}$.



To deal with uncertainty, one can also specify a probabilistic behaviour to the sensor, *i.e.*, rather than saying if deposit is full or not, a level of confidence is given regarding that information. $\mathcal{F}\mathcal{Z}\mathcal{L}$ (a probabilistic logic),

takes the closed interval $[0,1]$ as the truth value space. The granularity of uncertainty obtained is very impressive and can be used to model systems more realistically. The codomain of predicate *empty* becomes a $\mathbb{R}$–valued interval as exemplified in the following graph:



As usual, to employ $\mathcal{FZL}$ in the reservoir case, we need its hybridised version, $\mathcal{HFZL}$. Then we may refine the latter requirement as follows: *In basal mode, at any instant of time, if one knows with a degree of confidence x that the reservoir is empty, and in the next instant of time that confidence becomes y, then the alarm becomes active with a confidence of z.*

$$@_{Basal}\forall t : Time\,.\,(empty(t), x) \wedge (empty(t+1), y) \Rightarrow (alarm(t+2), z)$$

We can also adapt this requirement to the other configurations as we did when illustrating the $\mathcal{H}3\mathcal{VL}$ case. For instance, we can say that when this situation occurs in the basal with bolus mode, the alarm must be activated:

$$@_{BasalBo}\forall t : Time\,.\,(empty(t), x) \wedge (empty(t+1), y) \Rightarrow (alarm(t+2), 1)$$

On the other hand, in the suspended mode the alarm does not need to be active:

$$(Sus1 \vee Sus2) \Rightarrow$$
$$\forall t : Time\,.\,(empty(t), x) \wedge (empty(t+1), y) \Rightarrow (alarm(t+2), 0)$$

Let us take yet another approach. Systems whose configurations evolve temporally can be captured by the hybridisation method when taking a

linear discretisation of time as the reconfiguration space. Configurations are then time instants and a modality $after$ relates them chronologically. Nominals refer to such instants.

Adding an additional hybrid layer to $\mathcal{HFZL}$ [4] with the intent to capture time, provides us with the option to discard the sort *Time* and the universal quantification when formalising the latter requirement. It then becomes:

$$\{ @_{Basal}(empty, x)\} \wedge \langle next \rangle \{@_{Basal}(empty, y)\} \Rightarrow \\ \langle after \rangle \{@_{Basal}(alarm, z)\}$$

[4]Contrary to the usual, we do not impose the restriction regarding the common realisation of carrier sets to $\mathcal{H}^2\mathcal{FZL}$ (*cf.* section 2.2.3). Otherwise, the intended hierarchy would be lost.

Chapter 5

Case study: Hybridising Alloy

Lightweight formal methods combine mathematical rigour with simple notations and ease-of-use support platforms. Alloy [Jac06], 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. [MC12, UGGT12, AKMR03]). The

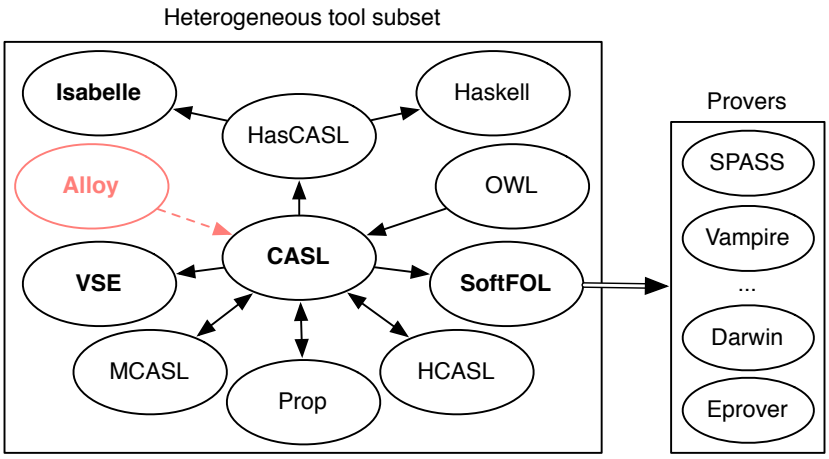


Figure 5.1: "Plugging" Alloy into the Hets network

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, [UGGT12], one of the most recent proposals in this trend, translates models into a first-order dialect supported by the KEY theorem prover. Our proposal in this chapter goes one step further by "plugging" Alloy into the HETS network, as depicted in figure 5.1.

Plugging Alloy to HETS brings for free the power of several provers and model checkers connected into the network, including, for instance, Vampire, SPASS, EPROVER, Darwin, Isabelle, among many others. Experiments can then be carried out in different tools, typically tuned to specific application areas. Moreover, Alloy models can also be translated into a number of languages available in HETS, including CASL, HasCASL, or even Haskell. The price for this greater level of generalisation (when compared to related work), is to capture Alloy as an institution and to define a conservative comorphism, targeting one of the central languages in HETS.

The other obvious advantage is that by capturing Alloy as an institution, its hybridised version ($\mathcal{H}$Alloy) is provided "for free". Thus

this chapter, expands the hybridisation's field of action to the domain of lightweight formal tools. As expected, tool support for $\mathcal{H}$ALLOY comes from the lifting presented in section 2.3.2, applied to the comorphism ALLOY $\rightarrow$ CASL introduced in this chapter.

Taking into account the motivations just described, the present chapter formalises ALLOY as an institution and defines an encoding leading to a broad proof support for it. This chapter's contribution is illustrated through a case study whose typical proof support is restricted to bounded domains. In a second moment, this exercise is extended by making use of the hybridisation method.

5.1 Alloy in the category of institutions

5.1.1 Alloy as an institution

ALLOY [Jac06] 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 turned hierarchical 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[1]. 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.

The ALLOY institution $(Sign^{\mathcal{A}}, Sen^{\mathcal{A}}, Mod^{\mathcal{A}}, \models^{\mathcal{A}})$, is defined 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} represents all kinds, S_{Abs} represents the abstract ones, S_{Som} represents the non-empty ones, and S_{One} represents the kinds containing exactly one element. Clearly, 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 singleton 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 $\varphi = (\varphi_s, \varphi_r, \varphi_v)$ such that:

[1] In the corresponding encoding an extra relation is added to each binary one as the transitive closure of the latter.

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

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

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

- $\varphi_v : X \to X'$ is a function.

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

p,	$p \in (S_\Sigma)_{All} \cup (R_\Sigma)_w \cup X_\Sigma$
$\hat{}r$,	$r \in (R_\Sigma)_w$ and $\lvert w\rvert = 2$
$\sim e$,	$e \in Exp(\Sigma)$
$e \mathrel{->} e'$,	$e, e' \in Exp(\Sigma)$
$e \odot e'$,	$e, e' \in Exp(\Sigma)$, $\lvert e\rvert = \lvert e'\rvert$, and $\odot \in \{+, -, \&\}$
$e \,.\, e'$,	$e, e' \in Exp(\Sigma)$, and $\lvert e\rvert + \lvert e'\rvert > 2$

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

$\lvert r\rvert$	=	$\lvert w\rvert$, for $r \in (R_\Sigma)_w$
$\lvert s\rvert$	=	1, for $s \in (S_\Sigma)_{All}$
$\lvert x\rvert$	=	1, for $x \in X_\Sigma$
$\lvert \hat{}r\rvert$	=	$\lvert r\rvert$
$\lvert \sim e\rvert$	=	$\lvert e\rvert$
$\lvert e \odot e'\rvert$	=	$\lvert e\rvert$, for $\odot \in \{+, -, \&\}$
$\lvert e \,.\, e'\rvert$	=	$(\lvert e\rvert + \lvert e'\rvert) - 2$
$\lvert e \mathrel{->} e'\rvert$	=	$\lvert e\rvert + \lvert e'\rvert$

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

e `in` e'	$e, e' \in Exp(\Sigma)$, and $\lvert e\rvert = \lvert e'\rvert$
`not` ρ	$\rho \in Sen^{\mathcal{A}}(\Sigma)$
ρ `implies` ρ'	$\rho, \rho' \in Sen^{\mathcal{A}}(\Sigma)$
(`all` $x : e$) ρ	$e \in Exp(\Sigma)$, $\lvert e\rvert = 1$, and $\rho \in Sen^{\mathcal{A}}(\Sigma')$, where $\Sigma' = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma \uplus \{x\})$

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

1. A carrier set $|M|$;
2. An unary relation $M_s \subseteq |M|$, for each $s \in S_{All}$;
3. A relation $M_r \subseteq M_w$, for each $r \in R_w$ where $w \in (S_{All})^+$;
4. A singleton (*i.e.* one element) relation, $M_x \subseteq |M|$, for each $x \in X$.

satisfying the following axioms for any $s, s' \in S_{All}$,

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

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

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

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:

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

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

$$\begin{array}{lll} M \models^{\mathcal{A}}_{\Sigma} e \texttt{ in } e' & \text{iff} & M_e \subseteq M_{e'} \\ M \models^{\mathcal{A}}_{\Sigma} \texttt{not } \rho & \text{iff} & M \not\models^{\mathcal{A}}_{\Sigma} \rho \\ M \models^{\mathcal{A}}_{\Sigma} \rho \texttt{ implies } \rho' & \text{iff} & M \models^{\mathcal{A}}_{\Sigma} \rho' \text{ whenever } M \models^{\mathcal{A}}_{\Sigma} \rho \\ M \models^{\mathcal{A}}_{\Sigma} (\texttt{all } x : e)\rho & \text{iff} & M' \models^{\mathcal{A}}_{\Sigma'} (x \texttt{ in } e) \texttt{ implies } \rho, \\ & & \text{for all model expansions } M' \text{ of } M, \text{ along the} \\ & & \text{inclusion morphism } x, \text{ previously defined} \\ & & \text{in } Sen^{\mathcal{A}}. \end{array}$$

Lemma 5.1.1. *The following diagram is a strong amalgamation square,*

$$\begin{array}{ccc} \Sigma & \xrightarrow{\varphi} & \Sigma_2 \\ {\scriptstyle x}\downarrow & & \downarrow{\scriptstyle x^{\varphi}} \\ \Sigma_1 & \xrightarrow[\varphi']{} & \Sigma' \end{array}$$

where φ' canonically extends φ with $\varphi'_v(x) = x$, and x, x^{φ} are inclusion morphisms. I.e. both just add x to X_Σ, as a fresh symbol to the respective signatures.

Proof. *From $Mod^{\mathcal{A}}(x)(M_1) = Mod^{\mathcal{A}}(\varphi)(M_2)$, we know that:*

$$\begin{array}{c} \text{For any: } s \in (S_\Sigma)_{All},\ M_{1s} = M_{2\varphi_s(s)}; \\ r \in (R_\Sigma)_w,\ M_{1r} = M_{2\varphi_r(r)}; \\ y \in X_\Sigma,\ M_{1y} = M_{2\varphi_x(y)}; \\ |M_1| = |M_2|. \end{array}$$

$\Rightarrow$ *{M_2 is the x^φ reduct of M'; transitivity }*

For all models expansions M' of M_2, by the inclusion morphism x^φ: given any,
$$s \in (S_\Sigma)_{All},\ M_{1s} = M'_{\varphi_s(s)};$$
$$r \in (R_\Sigma)_w,\ M_{1r} = M'_{\varphi_r(r)};$$
$$y \in X_\Sigma,\ M_{1y} = M'_{\varphi_v(y)};$$
$$|M_1| = |M'|.$$

$\Rightarrow$ *{$|M_1| = |M'|$; inclusion morphism definition }*

There is exactly one model expansion of M_2 by x^φ, denoted as M', such that $M'_x = M_{1x}$, entailing that for any: $s \in (S_\Sigma)_{All}$, $M_{1s} = M'_{\varphi_s(s)}$;
$$r \in (R_\Sigma)_w,\ M_{1r} = M'_{\varphi_r(r)};$$
$$y \in X_\Sigma,\ M_{1y} = M'_{\varphi_v(y)};$$
$$|M_1| = |M'|;$$
$$M_{1x} = M'_x.$$

$\Leftrightarrow$ *{inclusion morphism x, definition }*

There is exactly one model expansion of M_2 by x^φ, denoted as M', such that, for any: $s \in (S_{\Sigma_1})_{All}$, $M_{1s} = M'_{\varphi_s(s)}$;
$$r \in (R_{\Sigma_1})_w,\ M_{1r} = M'_{\varphi_r(r)};$$
$$y \in X_\Sigma,\ M_{1y} = M'_{\varphi_v(y)};$$
$$|M_1| = |M'|;$$
$$M_{1x} = M'_x.$$

$\Leftrightarrow$ *{φ' definition }*

There is exactly one model expansion of M_2 by x^φ, denoted as M', such that for any : $s \in (S_{\Sigma_1})_{All}$, $M_{1s} = M'_{\varphi_s(s)}$;
$$r \in (R_{\Sigma_1})_w,\ M_{1r} = M'_{\varphi_r(r)};$$
$$y \in X_{\Sigma_1},\ M_{1y} = M'_{\varphi_v(y)};$$
$$|M_1| = |M'|.$$

$\Leftrightarrow$ *{reduct definition }*

There is exactly one model expansion of M_2 *by* x^{φ}*, denoted as* M'*, such that*
$$M_1 = Mod^{\mathcal{A}}(\varphi')(M')$$

□

Lemma 5.1.2. *For any signature morphism,* $\varphi : \Sigma \to \Sigma'$*, an expression* e *in* Σ*, and a* Σ'*–model,* $(M' \restriction_{\varphi})_e = M'_{Exp^{\mathcal{A}}(\varphi)(e)}$.

Proof.

When $e := r, r \in (R_{\Sigma})_w$:

$$(M' \restriction_{\varphi})_r$$

$=$ *{Reduct definition }*

$$M'_{\varphi_r(r)}$$

$=$ *{*$Exp^{\mathcal{A}}$ *definition }*

$$M'_{Exp^{\mathcal{A}}(\varphi)(r)}$$

The case for when $e = p$*, for* $p \in (S_{\Sigma})_{All} \cup X_{\Sigma}$*, is analogous.*

When $e := e + e'$:

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

$=$ *{Expression evaluation }*

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

$=$ *{Induction hypothesis }*

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

$=$ *{Expression evaluation }*

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

$=$ *{$Exp(\varphi)$ definition }*

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

For the remaining operators the proof is analogous. □

Satisfaction condition. Given any $Sign^{\mathcal{A}}$ morphism $\varphi : \Sigma \to \Sigma'$, a Σ–sentence ρ, and a Σ'–model:

When $\rho := e \texttt{ in } e'$,

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

$\Leftrightarrow$ {Satisfaction definition }

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

$\Leftrightarrow$ {Lemma 5.1.2, $2\times$ }

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

$\Leftrightarrow$ {Satisfaction definition }

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

$\Leftrightarrow$ {$Sen^{\mathcal{A}}$ definition }

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

The cases of negation and implication are analogous to the ones presented in the satisfaction condition proof concerning $\mathcal{FOL}$.

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

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

$\Leftrightarrow$ {Satisfaction definition }

For all model expansions $(M' \restriction_\varphi)'$ of $M' \restriction_\varphi$, along the inclusion morphism x previously defined in $Sen^{\mathcal{A}}$, $(M' \restriction_\varphi)' \models^{\mathcal{A}}_{\Sigma^x} (x\ \texttt{in}\ e)\ \texttt{implies}\ \rho$

$\Leftrightarrow$ {Lemma 5.1.1; I.H. }

For all model expansions M'' of M', along the inclusion morphism x previously defined in $Sen^{\mathcal{A}}$, $M'' \models^{\mathcal{A}}_{\Sigma'^x} Sen^{\mathcal{A}}(\varphi')((x\ \texttt{in}\ e)\ \texttt{implies}\ \rho)$

$\Leftrightarrow$ {Satisfaction definition }

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

$\Leftrightarrow$ {$Sen^{\mathcal{A}}$ definition }

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

□

5.1.2 From Alloy to CASL

This section characterises a conservative *comorphism* from ALLOY to the presentation of CASL. The latter needs to be a presentation to deal appropriately with ALLOY implicit rules over kinds and the transitive closure. Both features will be encoded into Γ, therefore restricting the models available. Recall that an object in the category $Sign^{\mathcal{C}}$ of CASL signatures is a tuple (S, TF, PF, P) where S is the set of sorts, TF a family of

function symbols indexed by their arity; PF a family of partial function symbols indexed by their arity; and finally P is a family of relational symbols also indexed by their arity. Then, we define

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

$$
\begin{array}{lcl}
S' & = & \{U, Nat\} \\
PF & = & \varnothing \\
F_w = & & \begin{cases} \{x | x \in X\} & \text{if } w = U \\ \{0\} & \text{if } w = Nat \\ \{suc\} & \text{if } w = Nat, Nat \\ \varnothing & \text{for the other cases} \end{cases} \\
P_w = & & \begin{cases} \{s | s \in S_{All}\} \cup \{r | r \in R_s, s \in S_{All}\} & \text{if } w = U \\ \{r | r \in R_{s_1 \ldots s_n}, s_i \in S_{All}\} & \text{if } w = U^n, n > 1 \\ \{t_r | r \in R_{s,s}, s \in S_{All}\} & \text{if } w = Nat, U, U \\ \varnothing & \text{for the other cases} \end{cases}
\end{array}
$$

and Γ is the least set containing the following axioms:

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

8. $\{(\forall u, v : U)\ t_r(0, u, v) \Leftrightarrow r(u, v) \wedge$
$(\forall n : Nat)\ t_r(suc(n), u, v) \Leftrightarrow (\exists x : U)\ t_r(0, u, x) \wedge t_r(n, x, v) | r \in$
$R_{s_1,s_2}\}$

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

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

where η is defined as follows :

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

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

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

Lemma 3. For any $\Sigma \in |Sign^{\mathcal{A}}|$, the following diagram commutes:

$$\begin{array}{ccc} \Sigma & \xleftarrow{\beta_\Sigma} & \Phi(\Sigma) \\ {\scriptstyle x^\beta}\downarrow & & \downarrow{\scriptstyle x} \\ \Sigma^x & \xleftarrow[\beta_{\Sigma^x}]{} & \Phi(\Sigma)^x \end{array}$$

Stating that, given a Σ^x–model M^x and a $\Phi(\Sigma)$–model N, such that $Mod^{\mathcal{A}}(x^\beta)(M^x) = \beta_\Sigma(N)$, there is exactly one model N^x such that $Mod^{\mathcal{C}}(x)(N^x) = N$ and $M^x = \beta_{\Sigma^x}(N^x)$.

Proof.

For any $p \in ((S_\Sigma)_{All} \cup (R_\Sigma)_{All} \cup X_\Sigma)$, $N_p = M^x_p$ and $|N| = |M^x|$

$\Rightarrow$ {N is the x reduct of N^x; transitivity }

For all model expansions N^x of N along the inclusion morphism x, for any $p \in ((S_\Sigma)_{All} \cup (R_\Sigma)_{All} \cup X_\Sigma)$, $N^x_p = M^x_p$ and $|N^x| = |M^x|$

$\Rightarrow$ {$|M^x| = |N^x|$; definition of the inclusion morphism x }

There is exactly one model N^x such that $N^x_x = M^x_x$, entailing that : $|N^x| = |M^x|$, for any $p \in ((S_\Sigma)_{All} \cup (R_\Sigma)_{All} \cup X_\Sigma)$, $N^x_p = M^x_p$ and $N^x_x = M^x_x$

$\Leftrightarrow$ {Definition of the inclusion morphism x^β }

$|N^x| = |M^x|$, for any $p \in ((S_{\Sigma^x})_{All} \cup (R_{\Sigma^x})_{All} \cup X_{\Sigma^x})$, $N^x_p = M^x_p$

$\Leftrightarrow$ {β definition }

$$M^x = \beta_{\Sigma^x}(N^x)$$

□

Lemma 4. Given any $\Phi(\Sigma)$-model M', where $\Sigma = (S_\Sigma, m_\Sigma, R_\Sigma, X_\Sigma)$, and expression e:

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

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

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \eta_{(v_1,\ldots,v_n)}(p)$$

$\Leftrightarrow$ {α definition }

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} p(v_1, \ldots, v_n)$$

$\Leftrightarrow$ {Satisfaction definition }

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

$\Leftrightarrow$ {v_i elements are constants }

$$M'_{(v_1 \times \cdots \times v_n)} \subseteq M'_p$$

$\Leftrightarrow$ {β definition }

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

$\Leftrightarrow$ {Expression evaluation and satisfaction definition }

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

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

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \eta_v(x)$$

$\Leftrightarrow$ {α definition }

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} v = x$$

$\Leftrightarrow$ {Satisfaction definition }

$$M'_v = M'_x$$

$\Leftrightarrow$ {v and x are constants }

$$M'_v \subseteq M'_x$$

$\Leftrightarrow$ {β definition }

$$\beta_\Sigma(M')_v \subseteq \beta_\Sigma(M')_x$$

$\Leftrightarrow$ {Satisfaction definition }

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

When $e := {\wedge}r, r \in R_w$, such that $|w| = 2$:

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \eta_{(v_1,v_2)}({\wedge}r)$$

$\Leftrightarrow$ {α definition }

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} (\exists n : Nat)\, tr(n, v_1, v_2)$$

$\Leftrightarrow$ {Satisfaction definition and definition of tr by Γ }

$$M'_{(v_1,v_2)} \in M'_{(r^+)}$$

$\Leftrightarrow$ {v_1, v_2 are constants and β definition }

$$\beta_\Sigma(M')_{(v_1 \times v_2)} \subseteq \beta_\Sigma(M')_{(r^+)}$$

$\Leftrightarrow$ {Expression evaluation and satisfaction definition }

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

When $e := \sim e$:

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

$\Leftrightarrow$ {α definition }

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \eta_{(v_n,\ldots,v_1)}(e)$$

$\Leftrightarrow$ {I.H }

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

$\Leftrightarrow$ {Galois connection }

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

When $e := e + e'$:

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \eta_{(v_1,\ldots,v_n)}(e + e')$$

$\Leftrightarrow$ {α definition, satisfaction definition }

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \eta_{(v_1,\ldots,v_n)}(e) \text{ or } M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \eta_{(v_1,\ldots,v_n)}(e')$$

$\Leftrightarrow$ {I.H }

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

$\Leftrightarrow$ {Satisfaction and sum definition }

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

Proof of the remaining cases is analogous.

Satisfaction condition. For any signature $\Sigma \in |Sign^{\mathcal{A}}|$ a $\Phi(\Sigma)$-model M' a Σ-sentence ρ :

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

When $\rho := e\ \texttt{in}\ e'$:

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \alpha_\Sigma(e\ \texttt{in}\ e')$$

$\Leftrightarrow$ {α definition }

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} (\forall (v_1, \ldots, v_n) : U) \eta_{(v_1,\ldots,v_n)}(e) \Rightarrow \eta_{(v_1,\ldots,v_n)}(e')$$

$\Leftrightarrow$ {Satisfaction definition, $2\times$ }

for all model expansions M'' of M' defined in the expected way,
$M'' \models^{\mathcal{A}}_{\Phi(\Sigma)'} \eta_{(v_1,\ldots,v_n)}(e')$ whenever $M'' \models^{\mathcal{A}}_{\Phi(\Sigma)'} \eta_{(v_1,\ldots,v_n)}(e)$

$\Leftrightarrow$ {Lemma 4 and satisfaction definiton }

for all model expansions M'' of M' defined in the expected way,
$\beta_{\Sigma'}(M'') \models^{\mathcal{A}}_{\Sigma^x} (v_1 \texttt{->} \ldots \texttt{->} v_n)\ \texttt{in}\ e \Rightarrow (v_1 \texttt{->} \ldots \texttt{->} v_n)\ \texttt{in}\ e'$

$\Leftrightarrow$ {Inclusion definition, Lemma 3 }

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

When $\rho := \neg\rho$:

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \alpha_\Sigma(\neg\rho)$$

$\Leftrightarrow$ {α definition }

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \neg \alpha_\Sigma(\rho)$$

$\Leftrightarrow$ {Satisfaction definition }

$$M' \not\models^{\mathcal{C}}_{\Phi(\Sigma)} \alpha_\Sigma(\rho)$$

$\Leftrightarrow$ {I.H. }

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

$\Leftrightarrow$ {Satisfaction definition }

$$\beta_\Sigma(M') \models^{\mathcal{A}}_{\Sigma} \neg\rho$$

The implication case is analogous to the one above.

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

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} \alpha_\Sigma((\texttt{all}\ x : e)\ \rho)$$

$\Leftrightarrow$ {α definition }

$$M' \models^{\mathcal{C}}_{\Phi(\Sigma)} (\forall x : U)\alpha_\Sigma((x\ \texttt{in}\ e)\ \texttt{implies}\ \rho)$$

$\Leftrightarrow$ {Satisfaction definition }

for all model expansions M'' of M', defined in the expected way,
$$M'' \models^{\mathcal{C}}_{\Phi(\Sigma)^x} \alpha_{\Sigma^x}((x\ \texttt{in}\ e)\ \texttt{implies}\ \rho)$$

$\Leftrightarrow$ {I.H. }

for all model expansions M'' of M', defined in the expected way,
$$\beta_{\Sigma^x}(M'') \models^{\mathcal{A}}_{\Sigma^x} (x\ \texttt{in}\ e)\ \texttt{implies}\ \rho$$

$\Leftrightarrow$ {Lemma 3 }

for all model expansions $\beta_{\Sigma^x}(M'')$ of $\beta_{\Sigma}(M')$, defined in the expected way, $\beta_{\Sigma^x}(M'') \models^{\mathcal{A}}_{\Sigma^x} (x \,\texttt{in}\, e)\ \texttt{implies}\, \rho$

$\Leftrightarrow$ {Satisfaction definition }

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

□

Lemma 5.1.3. *The above comorphism is conservative.*

Proof.The proof that the above comorphism is conservative may be achieved in the following way : Given a (S, m, R, X)-model M, build a $\Phi((S, m, R, X))$-model M' such that, $\beta_{(S,m,R,X)}(M') = M$. One way to build such model may be :

(a) M' has a carrier set, $|M'_U|$, such that $|M'_U| = |M|$;

(b) $M'_p = M_p$, for any $p \in (S_{All} \cup R_w \cup X)$;

(c) $M'_{Nat} = \mathbb{N}at$;

(d) For any r in R_{s_1,s_2}, M' has a relation, t_r, defining the transitive closure of r.

M' satisfies the rules 1 to 6 in Γ, due to (b). The rules 7,8 are satisfied by (c) and (d) respectively.

□

5.2 DCR graphs with ($\mathcal{H}$)Alloy

5.2.1 DCR graphs – Introductory concepts

DCR graphs, short for Distributed Condition Response Graphs, were introduced in [HM10] to specify workflow models in an implicit way, through a number of conditions. A functional style and precise semantics make DCR graphs excellent candidates for modelling critical workflows.

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

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

A mark or execution state, in a DCR graph G, is a tuple $(Ex, Res) \in \mathbb{P}(E) \times \mathbb{P}(E)$, where Ex is the set of the events that have 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. $\{e' \mid condition(e', e)\} \subseteq Ex$,
2. $Ex' = Ex \cup \{e\}$,
3. $Res' = (Res \backslash \{e\}) \cup \{e' \mid response(e, e')\}$.

Proof support for DCR graphs is discussed by Mukkamala [Muk12] who suggests a translation to PROMELA so that the specification of workflows can be checked with the SPIN [Hol03] model checker. The encoding, however, is not easy. For example, the target language has only arrays as a basic data structure, thus events and relations have to be encoded as arrays, with relations becoming two-dimensional bit arrays. Moreover, SPIN based verification is limited by possible state explosion, an inevitability in model checking. 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 mentioned above, ALLOY plugged into the HETS family offers sundry unbounded verifiers that complement the ALLOY analyser.

To demonstrate our proposal for proof support for DCR graphs, let us encode their canonical definition into ALLOY:

$$
\begin{array}{l}
\mathit{abstract\ sig\ Event}\ \{ \\
\quad \mathit{condition} : \mathit{set\ Event}, \\
\quad \mathit{response} : \mathit{set\ Event} \\
\} \\
\\
\quad \mathit{sig\ Mark}\ \{ \\
\quad\quad \mathit{executed} : \mathit{set\ Event}, \\
\quad\quad \mathit{toBeExecuted} : \mathit{set\ Event}, \\
\quad\quad \mathit{action} : \mathit{set\ Mark} \mathrel{->} \mathit{set\ Event} \\
\quad \} \\
\\
\quad \mathit{fact}\ \{ \\
\quad\quad \mathit{all}\ m, m' : \mathit{Mark}, e : \mathit{Event} \mid \\
\quad\quad\quad (m \mathrel{->} m' \mathrel{->} e)\ \mathit{in\ action} \Leftrightarrow \\
\quad\quad\quad\quad (\mathit{condition}.e\ \mathit{in}\ m.\mathit{executed} \wedge m'.\mathit{executed} = m.\mathit{executed} + e \wedge \\
\quad\quad\quad\quad m'.\mathit{toBeExecuted} = (m.\mathit{toBeExecuted} - e) + e.\mathit{response}\ ) \\
\quad \}
\end{array}
$$

Such encoding includes the declaration of two kinds (denoted as *sig*), one of events and another one 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 $\mathit{Mark} \times \mathit{Mark} \times \mathit{Event}$. Finally note how the invariant for valid execution steps is directly captured in the fact statement above.

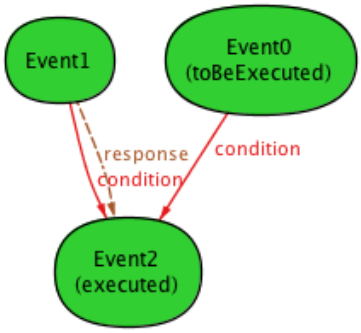


Figure 5.2: An instance of a generic DCR graph

The formalisation of DCR graphs in ALLOY already provides the conditions to see instances of them with the ALLOY analyser. Figure 5.2 gives one such instance.

Properties regarding generic DCR graphs can also be directly checked in ALLOY. For example,

$$all\ m, m' : Mark, e : Event \mid$$
$$(m \text{ -> } m' \text{ -> } e)\ in\ action \wedge e\ in\ m'.toBeExecuted \Rightarrow e\ in\ e.response$$

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, one can resort to HETS, provided that ALLOY is already plugged into the respective wider network. The comorphism defined in this chapter, may thus be employed by translating the latter sentence into the following one (after a few reduction steps),

$$\forall m, m', e : U\ .\ Mark(m) \wedge Mark(m') \wedge Event(e) \Rightarrow$$
$$action(m, m', e) \wedge toBeExecuted(m', e) \Rightarrow response(e, e))$$

which, can be verified with a theorem prover connected to HETS, such as SPASS or VAMPIRE. This approach allows us to have a viable alternative for verification in ALLOY.

5.2.2 The oncology workflow – A (small) case study

One interesting case study explored by Mukkamala in his P.h.D thesis [Muk12], concerns medicine administration in an oncological context. In this study DCR graphs are employed in order to define valid medicine administration workflows. Such is established in the expected way, *i.e.*, by stating conditions that all valid workflows must follow. For instance, no medicine can be given without being prescribed first.

To illustrate our approach regarding tool support for DCR graphs (but also more generally, for unbounded proof support for ALLOY), a fragment of this case study was encoded into ALLOY, generating the graphical representation of the encoded specification, show in figure 5.3.

The next sensible step is thus to check for interesting properties. However, recall that ALLOY may give a false sense of security as the scope set for a simulated session may not be wide enough to produce a counter example. For instance, consider the following property in which we assume that $transRun = \hat{}(action.Event)$. In English it reads: *starting with an empty mark (∅, ∅), 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,

$$all\ m, m' : Mark \mid (\ no\ m.(executed + toBeExecuted) \wedge m'\ in\ m.transRun \wedge SecEffect\ in\ m'.executed \wedge no\ m'.toBeExecuted) \Rightarrow no\ m'.executed$$

A quick analysis of the workflow diagram shows that the property is false. Actually, if the left side of the implication is true, it must 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 (the default scope is 3). The problem 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 a reasoner in the HETS network (e.g. VAMPIRE), the result pops out in a few seconds.

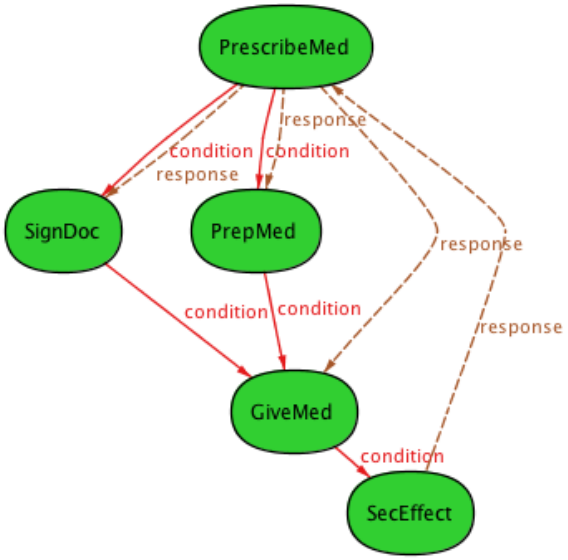


Figure 5.3: A medical workflow

Let us now focus (again) on the characteristics of DCR graphs. They are known to be a flexible way of specifying workflows, due to their declarative nature. Actually, they are much more flexible than the more conventional approaches (namely the ones based on UML) for this kind of specifications, as they allow many possible workflows. However, notice that the conditions imposed by them are made "once and for all" situations, which may be considered an over restriction. Taking as an example the condition stating that a signature of a doctor is needed to give a medicine, it may be true that such does not apply in the context of an emergency.

The hybridisation of ALLOY may then be used for the example just described, by considering nominals as the possible contexts and modalities as the events changing between them. In this situation, the example above should be extended in order to contemplate the different contexts and events:

Noms = { $Normal, Emergency$ }

$\Lambda = \{ \mathit{EmergencyHappens} \}$

Then, we use $\mathcal{H}$ALLOY as the typical hybrid logic. *I.e.*, we state that from a normal situation we can transit into an emergency,

$$@_{Normal}\langle \mathit{EmergencyHappens}\rangle \mathit{Emergency}$$

and after that, the conditions relative to the different contexts are specified:

$$@_{Normal}\, \mathit{condition.GiveMed} = \mathit{SignDoc} + \mathit{PrepMed}$$
$$@_{Emergency}\, \mathit{condition.GiveMed} = \mathit{PrepMed}$$

In summary, this chapter has shown how ALLOY (and its connection to HETS) may be a viable option for giving tool support to DCR graphs. Furthermore, through this last example, it also illustrated a logic (with proof support) for specifying reconfigurable workflows.

Chapter 6

On empowering the hybridisation method

This chapter contains two contributions of a more theoretical flavour. Both propose extensions to the hybridisation process discussed along the previous chapters. In particular,

- the first one extends the hybridisation process to morphisms between institutions, and characterises it as a functor;
- while the second is a (partial) solution for the problem of fixed interfaces mentioned before as a limitation of the hybridisation method. This contribution was recently published in [MNMB13].

6.1 Hybridisation is an (endo)functor

Despite being often criticized ("too abstract", some say), it is a fact that category theory has been a particularly good "friend" to computer science. Notably, many aspects of the latter rely on and can be represented in categorical terms at a great extent – examples range from programming languages to type and proof theories, along with many others. Ironically, this impressive reach of category theory, is partially explained by the "criticism" that is too abstract as a field.

Joseph Goguen, in the paper titled "A Categorical Manifesto" ([Gog91]), summarises the many ways in which category theory may contribute to computer science. We recall here the following ones, as they are significantly relevant to the work presented in this chapter:

1. *"Dealing with abstraction and representation independence"*,

2. *"Formulating conjectures and research directions"*.

In the light of the above, when one switches to the categorical perspective, the typical case is that theories are in some way empowered, turned more general and elegant. Actually, such was the case of institutions themselves.

Accordingly, the natural research direction for when a theory akin to the hybridisation appears, is to ascertain in what ways is related to categorical terms, or rather, in which categorical concepts may fit. The objective of this section is to do just this kind of research. More concretely we aim at exploring hybridisation as a functor between suitable categories.

In section 2.2.3 the hybridisation method was defined as a function on the category of institutions, one that given an object of this category, returns the respective hybridised version (also an institution). Therefore in its essence, is just an incomplete mapping between two (equal) categories, as objects are mapped but not morphisms. The question that naturally emerges, is if the hybridisation process can be extended to morphisms; and if so, then does the extension preserves (co)domains, identity and composition. If indeed they are preserved, then one may call the enriched mapping a functor (or more precisely, an endofunctor).

We answer these questions in the following order: Firstly the hybridisation is endowed with the ability to map morphisms[1]. Again the interest of this theoretical research is to grant new forms of proof support to hybridised logics, as we will see further on.

[1]Since we are interested in the category of institutions with reversed arrows, morphisms here, are in fact the comorphisms between institutions, defined in the last chapter.

Secondly, conservation with respect to (co)domains, identity and composition is proved. Once proved, the enriched version of the hybridisation meets the properties needed to be called an (endo)functor. Consequently, all theory developed around the concept of functor, may be used upon the hybridisation method.

Let us start by recalling definition of functor :

Definition 6.1.1. *Let C, D be categories. A functor F is a mapping, associating to each object $O \in |C|$, an object $F(O) \in |D|$. In addition, F maps each morphism $f : A \to B$ in C, to a morphism $F(f) : F(A) \to F(B)$ in D, such that the following rules hold:*

1. *Given any object $O \in |C|$, $F(id_O) = id_{F(O)}$;*
2. *For all morphisms $f : A \to A'$, $g : A' \to B$ in C*
   $F(f \,.\, g) = F(f) \,.\, F(g)$.

Definition 6.1.2. *Let C, D be categories. A functor $F : C \to D$, is an endofunctor whenever the categories being mapped, are the same, i.e., $C = D$.*

We have previously mentioned that the first step in the quest to explore hybridisation as a functor, is to enrich it with morphism mapping. Furthermore, the definition of functor asserts that such map must preserve (co)domains, as presented by the following commuting square :

$$\begin{array}{ccc} \mathcal{I} & \xrightarrow{\quad f \quad} & \mathcal{I}' \\ {\scriptstyle \mathcal{H}}\downarrow & & \downarrow{\scriptstyle \mathcal{H}} \\ \mathcal{H}\mathcal{I} & \xrightarrow[\mathcal{H}(f)]{} & \mathcal{H}\mathcal{I}' \end{array}$$

where $\mathcal{I}, \mathcal{I}'$ are institutions, f a morphism, and $\mathcal{H}$ the mapping corresponding to the hybridisation method.

Let $f = (\Phi, \alpha, \beta)$, $\mathcal{H}((\Phi, \alpha, \beta))$ be defined as follows:

Definition 6.1.3. *Let $\mathcal{H}((\Phi, \alpha, \beta)) = (\Phi', \alpha', \beta')$.*

- $\Phi' = \Phi \times id$;
- *For any* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, *given a* (Σ, Δ)*–sentence* ρ*:*

$$\begin{array}{lcl}
\alpha'_{(\Sigma,\Delta)}(n) & = & n,\ n \in \Delta_{Noms} \\
\alpha'_{(\Sigma,\Delta)}(\rho) & = & \alpha_\Sigma(\rho),\ \rho \in Sen^{\mathcal{I}}(\Sigma) \\
\alpha'_{(\Sigma,\Delta)}(\rho \Rightarrow \rho') & = & \alpha'_{(\Sigma,\Delta)}(\rho) \Rightarrow \alpha'_{(\Sigma,\Delta)}(\rho'),\ \rho, \rho' \in Sen^{\mathcal{HI}}((\Sigma, \Delta)) \\
\alpha'_{(\Sigma,\Delta)}(\neg\rho) & = & \neg\alpha'_{(\Sigma,\Delta)}(\rho),\ \rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta)) \\
\alpha'_{(\Sigma,\Delta)}(@_i\rho) & = & @_i\alpha'_{(\Sigma,\Delta)}(\rho),\ i \in \Delta_{Noms} \text{ and } \rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta)) \\
\alpha'_{(\Sigma,\Delta)}([m]\rho) & = & [m]\alpha'_{(\Sigma,\Delta)}(\rho),\ m \in \Delta_{Mods} \text{ and } \rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta)) \\
\alpha'_{(\Sigma,\Delta)}(\textcircled{\triangledown} x\, \rho) & = & \textcircled{\triangledown} x\, \alpha'_{(\Sigma,\Delta')}(\rho),\ \Delta' = (Noms_\Delta \uplus \{x\}, Mods_\Delta)
\end{array}$$

and $\rho \in Sen^{\mathcal{HI}}((\Sigma, \Delta'))$

- *For any signature object* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, *and a* $\Phi'((\Sigma, \Delta))$*–model* (f, R), $\beta'_{(\Sigma,\Delta)}((f, R)) = (\beta_\Sigma \,.\, f, R)$

Lemma 6.1.1. *Recall Definition 2.2.4. For any signature* $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, *the following diagram defines a strong amalgamation square,*

$$\begin{array}{ccc}
Mod^{\mathcal{HI}}((\Sigma, \Delta)) & \xleftarrow{\beta'_{(\Sigma,\Delta)}} & Mod^{\mathcal{HI}}((\Phi(\Sigma), \Delta)) \\
\Big\uparrow {\scriptstyle Mod^{\mathcal{HI}}((id,x))} & & \Big\uparrow {\scriptstyle Mod^{\mathcal{HI}'}((id,x))} \\
Mod^{\mathcal{HI}}((\Sigma, \Delta')) & \xleftarrow[\beta'_{(\Sigma,\Delta')}]{} & Mod^{\mathcal{HI}'}((\Phi(\Sigma), \Delta'))
\end{array}$$

where (id, x),(id, x) *are inclusion morphisms adding a nominal* x *to the corresponding signatures.*

Proof. Let $M' = (f', R')$ and $M_\Phi = (f_\Phi, R_\Phi)$.

$$\beta'_{(\Sigma,\Delta)}(M_\Phi) = Mod^{\mathcal{HI}}((id, x))(M')$$

$\Leftrightarrow$ $\{Mod^{\mathcal{H}}$,$Mod^{\mathcal{I}'}$ definitions $\}$

$$(\beta_\Sigma \,.\, f_\Phi, R_\Phi) = (f', Mod^{\mathcal{H}}(x)(R'))$$

$\Leftrightarrow$ {inclusion morphism (id, x) definition }

For all model expansions (f'_Φ, R'_Φ) of M_Φ along the inclusion morphism $(id, x) : f'_\Phi = f_\Phi$ and $Mod^{\mathcal{H}}(x)(R'_\Phi) = R_\Phi$.

$\Rightarrow$ {Transitivity }

For all model expansions (f'_Φ, R'_Φ) of M_Φ along the inclusion morphism $(id, x) : (\beta_\Sigma \,.\, f'_\Phi, Mod^{\mathcal{H}}(x)(R'_\Phi)) = (f', Mod^{\mathcal{H}}(x)(R'))$

$\Rightarrow$ {$|R'_\Phi| = |R'|$, by reduct definition }

There is exactly one model expansion (f'_Φ, R'_Φ) of M_Φ along the inclusion morphism (id, x), such that $R'_{\Phi n(x)} = R'_{n(x)}$. Consequently for that expansion, $(\beta_\Sigma \,.\, f'_\Phi, R'_\Phi) = (f', R')$, and therefore $\beta'_{(\Sigma,\Delta')}((R'_\Phi, f'_\Phi)) = M'$.

□

Follows the proof that (co)domains are preserved, with respect to morphism mapping. *I.e* that the comorphism is valid:

Satisfaction condition. For any $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, a $\Phi'((\Sigma, \Delta))$–model M' ($M' = (f, R)$), a world $w \in |R|$, and a (Σ, Δ)–sentence ρ :

When $\rho := n$, for $n \in \Delta_{Noms}$,

$$\beta'_{(\Sigma,\Delta)}(M'), w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} n$$

$\Leftrightarrow$ {Satisfaction, and $\beta'_{(\Sigma,\Delta)}$ definitions }

$$R_n = w$$

$\Leftrightarrow$ {$\alpha'_{(\Sigma,\Delta)}$ definition }

$$R_{\alpha'_{(\Sigma,\Delta)}(n)} = w$$

$\Leftrightarrow$ {Satisfaction definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(n)$$

When $\rho \in Sen^{\mathcal{I}}(\Sigma)$,

$$\beta'_{(\Sigma,\Delta)}(M'), w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho$$

$\Leftrightarrow$ {Satisfaction, and $\beta'_{(\Sigma,\Delta)}$ definitions }

$$\beta_{\Sigma}.f(w) \models^{\mathcal{I}}_{\Sigma} \rho$$

$\Leftrightarrow$ {$(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$, is a morphism }

$$f(w) \models^{\mathcal{I}'}_{\Phi(\Sigma)} \alpha_{\Sigma}(\rho)$$

$\Leftrightarrow$ {Satisfaction definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha_{\Sigma}(\rho)$$

$\Leftrightarrow$ {$\alpha'_{(\Sigma,\Delta)}$ definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(\rho)$$

The implication case,

$$\beta'_{(\Sigma,\Delta)}(M'), w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho \Rightarrow \rho'$$

$\Leftrightarrow$ {Satisfaction definition }

$$\beta'_{(\Sigma,\Delta)}(M'), w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho' \text{ whenever } \beta'_{(\Sigma,\Delta)}(M'), w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho$$

$\Leftrightarrow$ {Induction hypothesis }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(\rho') \text{ whenever } M', w \models^{\mathcal{HI}}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(\rho)$$

$\Leftrightarrow$ {Satisfaction definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(\rho) \Rightarrow \alpha'_{(\Sigma,\Delta)}(\rho')$$

$\Leftrightarrow$ {$\alpha'_{(\Sigma,\Delta)}$ definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(\rho \Rightarrow \rho')$$

The negation case is analogous to the above.

When $\rho := @_i\rho$,

$$\beta'_{(\Sigma,\Delta)}(M'), w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} @_i\rho$$

$\Leftrightarrow$ {Satisfaction, and $\beta'_{(\Sigma,\Delta)}$ definitions }

$$\beta'_{(\Sigma,\Delta)}(M'), R_i \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \rho$$

$\Leftrightarrow$ {Induction hypothesis }

$$M', R_i \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(\rho)$$

$\Leftrightarrow$ {Satisfaction definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} @_i\alpha'_{(\Sigma,\Delta)}(\rho)$$

$\Leftrightarrow$ {$\alpha'_{(\Sigma,\Delta)}$ definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(@_i\rho)$$

The modality case is analogous to the above.

The universal quantification case,

$$\beta'_{(\Sigma,\Delta)}(M'), w \models^{\mathcal{HI}}_{(\Sigma,\Delta)} \forall x\, \rho$$

$\Leftrightarrow$ {Satisfaction definition }

For all model expansions $(\beta'_{(\Sigma,\Delta)}(M'))'$ of $\beta'_{(\Sigma,\Delta)}(M')$, by the previously defined inclusion morphism (id, x),
$$(\beta'_{(\Sigma,\Delta)}(M'))', w \models^{\mathcal{HI}}_{(\Sigma,\Delta')} \rho$$

$\Leftrightarrow$ {Lemma 6.1.1 }

For all model expansions M'' of M', by the previously defined inclusion morphism (id, x), $M'', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta'))} \alpha'_{(\Sigma,\Delta')}(\rho)$

$\Leftrightarrow$ {Satisfaction definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \forall x\, \alpha'_{(\Sigma,\Delta')}(\rho)$$

$\Leftrightarrow$ {$\alpha'_{(\Sigma,\Delta)}$ definition }

$$M', w \models^{\mathcal{HI}'}_{\Phi'((\Sigma,\Delta))} \alpha'_{(\Sigma,\Delta)}(\forall x\, \rho)$$

□

Theorem 6.1.1. *Given a morphism (Φ, α, β), let $\mathcal{H}((\Phi, \alpha, \beta)) = (\Phi', \alpha', \beta')$. The latter is a conservative morphism if and only if (Φ, α, β) is a conservative morphism.*

Proof. We want to prove that for any signature $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, $\beta'_{(\Sigma,\Delta)}$ is surjective. In other words, that for any model $(f, R) \in |Mod^{\mathcal{HI}}((\Sigma, \Delta))|$ there is a model $(f', R') \in |Mod^{\mathcal{HI}'}(\Phi'((\Sigma, \Delta)))|$, such that $\beta'_{(\Sigma,\Delta)}((f', R')) = (f, R)$,where $\beta'_{(\Sigma,\Delta)} = (\beta_\Sigma\, .\,) \times id$.

A model $(f', R') \in |Mod^{\mathcal{HI}'}(\Phi'((\Sigma, \Delta)))|$, is a tuple with two components, hence this proof is divided also in two parts, one relative to the left component and the other to the right one.

1. By β' definition $f = \beta_\Sigma \,.\, f'$, thus in the "worst" case, $img(f) = |Mod^{\mathcal{I}}((\Sigma, \Delta))|$, entailing that $img(\beta_\Sigma) = |Mod^{\mathcal{I}}((\Sigma, \Delta))|$. The latter is the definition of surjectivity on β_Σ.

2. The id function, is surjective.

□

We have enriched the hybridisation with the ability to map morphisms, and proved that (co)domains are preserved. The next step is to prove that identity is also preserved :

Proof. Consider a morphism $(id, id, id) : \mathcal{I} \to \mathcal{I}$,
$\mathcal{H}((id, id, id)) : \mathcal{HI} \to \mathcal{HI}$, preserves identity, *i.e.*, $\mathcal{H}((id, id, id)) = (id, id, id)$.

Let $\mathcal{H}((id, id, id)) = (\Phi', \alpha', \beta')$.

- We know that : $\Phi' = (\Phi, id)$.

$$\Phi' = (\Phi, id)$$

$$\Leftrightarrow \qquad \{\Phi = id\ \}$$

$$\Phi' = (id, id)$$

- The proof that for any $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$ and (Σ, Δ)–sentence ρ, $\alpha'_{(\Sigma,\Delta)}(\rho) = \rho$, follows from induction. With the base cases:

  - When $\rho := n, n \in \Delta_{Noms}$; $\alpha'_{(\Sigma,\Delta)}(\rho) = \rho$, by $\alpha'_{(\Sigma,\Delta)}$ definition.
  - When $\rho \in Sen^{\mathcal{HI}}(\Sigma)$; $\alpha'_{(\Sigma\Delta)}(\rho) = \alpha_\Sigma(\rho)$, by $\alpha'_{(\Sigma,\Delta)}$ definition. But $\alpha_\Sigma = id$, and therefore $\alpha'_{(\Sigma,\Delta)}(\rho) = \rho$.

  The proof for the other cases is a trivial application of the induction hypothesis.

- Recall β' definition : For any signature object $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, and a $\Phi'((\Sigma, \Delta))$–model (f, R), $\beta'_{(\Sigma,\Delta)}((f, R)) = (\beta_\Sigma \,.\, f, R)$.

  Thus $\beta'_{(\Sigma,\Delta)} = (\beta_\Sigma \,.\,) \times id$.

$$\beta'_{(\Sigma,\Delta)} = (\beta_\Sigma \,.\,) \times id$$

$\Leftrightarrow$ $\{\beta_\Sigma = id\}$

$$\beta'_{(\Sigma,\Delta)} = (id \,.\,) \times id$$

$\Leftrightarrow$ {For any relation R, $(id \,.)\, R = R$ }

$$\beta'_{(\Sigma,\Delta)} = id \times id$$

□

We have proved that identity is preserved, the next step is to prove that composition is also preserved.

Proof. For any morphism (Φ, α, β), let $\mathcal{H}((\Phi\, \alpha, \beta)) = (\varphi(\Phi), \varphi(\alpha), \varphi(\beta))$ (*). We want prove that for any composable morphisms $(\Phi_a, \alpha_a, \beta_a)$ and $(\Phi_b, \alpha_b, \beta_b)$,$\mathcal{H}((\Phi_a, \alpha_a, \beta_a) \,.\, (\Phi_b, \alpha_b, \beta_b)) = \mathcal{H}((\Phi_a, \alpha_a, \beta_a)) \,.\, \mathcal{H}((\Phi_b, \alpha_b, \beta_b))$

$$\mathcal{H}((\Phi_a, \alpha_a, \beta_a) \,.\, (\Phi_b, \alpha_b, \beta_b)) = \mathcal{H}((\Phi_a, \alpha_a, \beta_a)) \,.\, \mathcal{H}((\Phi_b, \alpha_b, \beta_b))$$

$\Leftrightarrow$ {Functor definition }

$$\mathcal{H}((\Phi_a \,.\, \Phi_b, \alpha_a \,.\, \alpha_b, \beta_a \,.\, \beta_b)) = \mathcal{H}((\Phi_a, \alpha_a, \beta_a)) \,.\, \mathcal{H}((\Phi_b, \alpha_b, \beta_b))$$

$\Leftrightarrow$ {(*), 2× }

$$(\varphi(\Phi_a \,.\, \Phi_b), \varphi(\alpha_a \,.\, \alpha_b), \varphi(\beta_a \,.\, \beta_b)) = (\varphi(\Phi_a), \varphi(\alpha_a), \varphi(\beta_a)) \,.\, (\varphi(\Phi_b), \varphi(\alpha_b), \varphi(\beta_b))$$

$\Leftrightarrow$ {Functor definition }

$$(\varphi(\Phi_a \,.\, \Phi_b), \varphi(\alpha_a \,.\, \alpha_b), \varphi(\beta_a \,.\, \beta_b)) = (\varphi(\Phi_a) \,.\, \varphi(\Phi_b), \varphi(\alpha_a) \,.\, \varphi(\alpha_b), \varphi(\beta_a) \,.\, \varphi(\beta_b))$$

$\Leftrightarrow$ {pointwise tuple equality }

$$\begin{cases} \varphi(\Phi_a \,.\, \Phi_b) = \varphi(\Phi_a) \,.\, \varphi(\Phi_b) & (1) \\ \varphi(\alpha_a \,.\, \alpha_b) = \varphi(\alpha_a) \,.\, \varphi(\alpha_b) & (2) \\ \varphi(\beta_a \,.\, \beta_b) = \varphi(\beta_a) \,.\, \varphi(\beta_b) & (3) \end{cases}$$

- Proving (1) :

$$\varphi((\Phi_a \,.\, \Phi_b))$$

$\Leftrightarrow$ {$\mathcal{H}$ definition }

$$(\Phi_a \,.\, \Phi_b, id)$$

$\Leftrightarrow$ {$id \,.\, id = id$ }

$$(\Phi_a \,.\, \Phi_b, id \,.\, id)$$

$\Leftrightarrow$ {Tuple definition }

$$(\Phi_a, id) \,.\, (\Phi_b, id)$$

$\Leftrightarrow$ {$\mathcal{H}$ definition }

$$\varphi(\Phi_a) \,.\, \varphi(\Phi_b)$$

- Proving (2) : Proof for this case relies on induction, with the expected base cases. For any signature object $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$, and any (Σ, Δ)–sentence ρ; let $\varphi((\alpha_a \,.\, \alpha_b))_{(\Sigma,\Delta)} = \varphi((\eta_a \,.\, \eta_b))$, $\varphi(\alpha_a)_{\varphi(\Phi_b)(\Sigma)} = \varphi(\eta_a)$, $\varphi(\alpha_b)_{\Sigma} = \varphi(\eta_b)$, $\alpha_{a\Phi_b(\Sigma)} = \eta_a$ and $\alpha_{b\Sigma} = \eta_b$. Then :

  - When $\rho := n, n \in \Delta_{Noms}$,

    $$\varphi((\eta_a \,.\, \eta_b))(n) = n$$

    $\Leftrightarrow$ {$\mathcal{H}$ definition }

    $$\varphi((\eta_a \,.\, \eta_b))(n) = \varphi(\eta_a)(n)$$

    $\Leftrightarrow$ {$\mathcal{H}$ definition }

    $$\varphi((\eta_a \,.\, \eta_b))(n) = \varphi(\eta_a)(\varphi(\eta_b)(n))$$

    $\Leftrightarrow$ {Function composition }

    $$\varphi((\eta_a \,.\, \eta_b))(n) = (\varphi(\eta_a) \,.\, \varphi(\eta))(n)$$

  - When $\rho \in Sen^{\mathcal{HI}}(\Sigma)$,

    $$\varphi((\eta_a \,.\, \eta_b))(\rho) = (\eta_a \,.\, \eta_b)(\rho)$$

    $\Leftrightarrow$ {$\mathcal{H}$ definition }

    $$\varphi((\eta_a \,.\, \eta_b))(\rho) = (\varphi(\eta_a) \,.\, \eta_b)(\rho)$$

    $\Leftrightarrow$ {$\mathcal{H}$ definition }

    $$\varphi((\eta_a \,.\, \eta_b))(\rho) = (\varphi(\eta_a) \,.\, \varphi(\eta_b))(\rho)$$

  Proof for the other cases is given by the induction hypothesis.

- Proving (3) : For any signature object $(\Sigma, \Delta) \in |Sign^{\mathcal{HI}}|$; let $\varphi((\beta_a \,.\, \beta_b))_{(\Sigma,\Delta)} = \varphi((\eta_a \,.\, \eta_b))$, $\varphi(\beta_a)_{\mathcal{H}(\Phi_b)(\Sigma)} = \varphi(\eta_a)$, $\varphi(\beta_b)_\Sigma = \varphi(\eta_b)$, $\beta_{a\Phi_b(\Sigma)} = \eta_a$ and $\beta_{b\Sigma} = \eta_b$. Then :

$$\varphi((\eta_a \,.\, \eta_b))$$

$\Leftrightarrow$ $\{\mathcal{H}$ definition $\}$

$$((\eta_a \,.\, \eta_b) \,.\,) \times id$$

$\Leftrightarrow$ $\{((g \,.\, f) \,.\,) \equiv (g \,.\,) \,.\, (f \,.\,)\}$

$$((\eta_a \,.\,) \,.\, (\eta_b \,.\,)) \times id$$

$\Leftrightarrow$ $\{id = id \,.\, id\}$

$$((\eta_a \,.\,) \,.\, (\eta_b \,.\,)) \times (id \,.\, id)$$

$\Leftrightarrow$ {Product definition }

$$((\eta_a \,.) \times id) \,.\, ((\eta_b \,.) \times id)$$

$\Leftrightarrow$ $\{\mathcal{H}$ definition, $2\times\}$

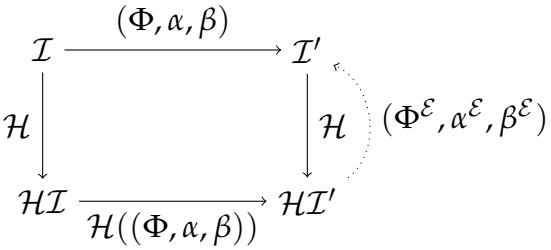
$$\varphi(\eta_a) \,.\, \varphi(\eta_b)$$

□

We have proved that composition is preserved, and thus completed the last step in classifying the hybridisation process as an (endo)functor.

The current method for obtaining proof support for hybridised logics, relies on the lifting targeting $\mathcal{FOL}$ (*cf.* section 2.3.2). The enrichment of the hybridisation with morphism mapping, points to a direction where

a broader proof support for hybridised logics may be achieved: Given an embedding morphism $(\Phi^{\mathcal{E}}, \alpha^{\mathcal{E}}, \beta^{\mathcal{E}}) : \mathcal{H}\mathcal{I}' \to \mathcal{I}'$, any morphism of the type $\mathcal{I} \to \mathcal{I}'$ may be systematically lifted into one with type $\mathcal{H}\mathcal{I} \to \mathcal{I}'$[2] as presented by the following commuting square:



Notice that in the above diagram, the classical method (the lifting as recalled in section 2.3.2) seems to be a particular instance which "compresses" the mapped and embedding morphisms into a single one.

The process of classifying the enriched version of the hybridisation process as an (endo)functor, also brings to light new research directions that should be considered for future work. In detail,

1. such method may equally be applied to operations akin to the hybridisation, potentially bringing to them some of the results obtained in the present section, namely in what concerns extra proof support.

2. Then, common features between these operations (the ones that are found to be functors) should be singled out, and from them, a general technique for adding new features to logics, envisaged. This may act like a boilerplate aiming at accommodating any new similar operation that may come up. Thus, the process of creating them and adding the respective proof support may become systematic.

3. Finally, category theory points to yet another research direction:

   "... if you have found an interesting functor, you might be well advised to investigate its adjoints." [Gog91].

[2]Note that $\mathcal{I}'$ is no more restricted to the $\mathcal{FOL}$, which broads the scope of application of the method.

6.2 Evolving interfaces

The approach to the specification of reconfigurable systems discussed up to now in this dissertation, assumes that all configurations share the same signature, i.e., that the interface provided at any local state is fixed. Or, to put it in yet another way, that the system's interface is invariant with respect to the reconfiguration process. In practice, however, this may be a too strong assumption. Actually, not only the realisation of a service may change from a configuration to another, but also the set of services provided may itself vary. In other words, sometimes in reconfigurable systems, local interfaces may evolve as well.

Although taking up this challenge in a completely general setting would require a substantial review of the method, a partial answer can be obtained by exploring the generated (hybrid) languages. This section proposes a technique, published in [MNMB13], to deal with interface reconfiguration whenever the local specifications are given in *equational logic* ($\mathcal{EQ}$). We want to allow not only a possibly different algebra in each state, but also different algebras over different signatures. Technically, this is achieved through the introduction of (hybrid) *partial algebra*-specifications to "simulate" the intended, independent (hybrid) *equational* ones. Note, however, that, even resorting to *partial* specifications, models will always be (total) algebras with respect to the corresponding local interface.

Switching to a more technical perspective, let us consider

- a set of relevant configurations named by the set of nominals Nom;
- and a family of modalities Λ to trigger reconfigurations.

Suppose, however, that in the place of a unique (static) interface (S, F), we consider

- a family $(S^i, F^i)_{i \in Nom}$ of local signatures, indexed by the set of nominals.

The technique proceeds by building a presentation

$$\big(\big((S, TF, PF), Nom, \Lambda\big), \Gamma\big) \in |Sign^{\mathcal{HFP}pres}|$$

in the institution $\mathcal{HFP}^{pres}$ of presentations over $\mathcal{HFP}$, where all this information can be considered, and the hybridisation applied.

The first step is to define a signature (S, TF, PF) in $\mathcal{FP}$ able to capture all the possible interfaces. Thus, services are split into the ones which are globally defined (*i.e.*, present in any (S^i, F^i), for $i \in Nom$) and those only concerning a specific state. These two sets of operations define a (global) $\mathcal{HFP}$-signature :

$S = \bigcup_{i \in Nom} S^i$

$TF_{ar \to s} = \{\sigma | \sigma \in \bigcap_{i \in Nom} F^i_{ar \to s}\}$

$PF_{ar \to s} = \{\sigma | \sigma \in (\bigcup_{i \in Nom} F^i_{ar \to s}) \setminus TF_{ar \to s}\}$

Hence, recovering an unique base signature to proceed with the specification. However, the information about the definition of those operations, respective to specific states, has yet to be considered. Such is put by the following axioms:

$$\begin{aligned} \Gamma = \quad & (\bigcup_{i \in Nom} \{@_i(\forall X)\, df(\sigma(X)) | \sigma \in PF_{ar \to s} \cap F^i{}_{ar \to s}\}) \cup \\ & (\bigcup_{i \in Nom} \{\neg @_i(\exists X)\, df(\sigma(X)) | \sigma \in PF_{ar \to s} \setminus F^i{}_{ar \to s}\}) \end{aligned}$$

Consequently, one ends up with a presentation

$$\big(\big((S, TF, PF), Nom, \Lambda\big), \Gamma\big)$$

collecting all the intended information on the interfaces. The specification method can be safely applied from this point on.

In broad terms, we are going to simulate *local, total* functions with *global, partial* ones. This entails the need for adopting strong equality to specify the "global properties" of a given operation, defined in (but, not all) specific configurations. Conversely, using the existential equation

$t \stackrel{e}{=} t'$, would necessarily bring inconsistency, since it fails on configurations where the involved operations are not defined. Of course, when existential equations are prefixed by reference operators, i.e., sentences of the form $@_i(t \stackrel{e}{=} t')$, the same does not apply.

Example 6.2.1. *Suppose that, in the context of a client server architecture, a buffering component is required to store and manage incoming messages from different clients. Depending on the server's execution mode, i.e., on its current configuration, issues like the order in which calls have arrived or the number of repeated messages may, or may not, be relevant.*

A model for this component comprises four kinds of configurations endowed with, respectively,

1. an algebra of *sequences* (for configurations where both order and multiplicities are relevant issues),
2. an algebra of *multi sets* (when the order may be left out),
3. an algebra of *sets* (when the application may abstract over order and repetitions), and finally
4. an algebra of *repetition free sequences* (to cater only for the messages' order).

Going from one configuration to another involves not only a change in the way a service is realised (e.g., insertion clearly differs from one state to the other), but also a change at the *interface* level. For instance, an operation to count the number of replicated messages, does not make sense if *sets* are used as a local model.

We start by defining a set $Nom = \{OM, Om, oM, om\}$ of nominals, where the capitalised letters correspond to the relevance of *order* and *multiplicity* issues (e.g., Om refers to a configuration where order, but not multiplicity, is relevant). Then, follows the definition of reconfiguration events :

$$Mod = \{goto\_OM, goto\_Om, goto\_oM, goto\_om\}.$$

Consider now the local interfaces. For (S^{om}, F^{om}) choose the usual signature of *Sets* comprising the set of sorts $S^{om} = \{Elem, Store, Bool\}$ and operation symbols ${F^{om}}_{Store} = \{empty\}$, ${F^{om}}_{Elem \times Store \to Bool} = \{is\_in\}$, ${F^{om}}_{Elem \times Store \to Store} = \{insert\}$ and ${F^{om}}_{ar \to s} = \emptyset$ for the other arities. Clearly, $(S^{Om}, F^{Om}) = (S^{om}, F^{om})$. The remaining cases need to deal with multiplicities, and therefore signatures have to be enriched with new operations. Hence, (S^{oM}, F^{oM}) can be defined as $S^{oM} = S^{om} \uplus \{Nat\}$, ${F^{oM}}_{Elem \times Store \to Nat} = \{mult\}$ and ${F^{oM}}_{ar \to s} = {F^{om}}_{ar \to s}$ for the other arities. Again, $(S^{OM}, F^{OM}) = (S^{oM}, F^{oM})$.

On the conditions above, the following "global" partial signature is defined as : $\big(((S, TF, PF), Nom, Mod), \Gamma\big)$ taking,

$S = \bigcup_{i \in Nom} S^i = S^{OM}$

$TF = F^{om}$

$PF_{Elem \times Store \to Nat} = \{mult\}$, and $PF_{ar \to s} = \emptyset$ for the other arities.

On its turn, Γ is equal to the union of the sentences :

$@_i(\forall s)(\forall e)\, df(mult(e, s))$, for $i \in \{oM, OM\}$

$\neg @_i(\forall s)(\forall e)\, df(mult(e, s))$, for $i \in \{om, Om\}$.

In this setting, we may now proceed with the specification of the global properties, say,

$(\forall e : elem)\, is\_in(e, empty) = False$

For the local properties one resorts to the hybrid reference operator. This allows, for instance, to record the fact that ordering and the multiple insertion are irrelevant for the configuration om :

$@_{om}(\forall e, e')(\forall s) insert(e', insert(e, s)) = insert(e, insert(e', s))$

$@_{om}(\forall e)(\forall s)\, insert(e, insert(e, s)) = insert(e, s)$

On the other hand, the specification of $mult$ in configuration oM is introduced as :

$@_{oM}(\forall e, e')(\forall s) \neg e = e' \Rightarrow mult(e, insert(e', s)) = mult(e, s)$

$@_{oM}(\forall e) mult(e, empty) = 0.$

Finally, we have to specify the possible reconfigurations. For this, one may use sentences as direct as,

$@_{om}\langle goto\_OM\rangle OM$

stating that a reconfiguration from om to OM is possible, or opt for more elaborated forms :

$(\forall e, e')(\forall s)\, insert(e', insert(e, s)) \;=\; insert(e, insert(e', s)) \Rightarrow \langle goto\_Om\rangle Om.$

The latter states the system can evolve to configuration Om (through the event $goto\_Om$) from any other configuration where the order of insertion is irrelevant.

We conclude here the illustration of the specification method extended to accommodate the presence of different interfaces (i.e., algebraic signatures) in different configuration states. Notice, however, that a number of details were not considered; for example, a definition of the natural numbers and the booleans should be included (and all signatures extended accordingly).

Chapter 7

Conclusions

7.1 What was achieved

The widening of the class of software systems which are simultaneously *reconfigurable* and critical, is leading to the research of formal verification techniques that in one way or another may avoid incorrect or unexpected behaviour. This is important, because in such class of systems, a potential fault is a potential tragedy.

In the context of the research group in which this dissertation was made, a methodology was proposed in [MFMB11] and its theoretical foundations established in [MMDB11]. The idea is to build an *hybrid ad-hoc* logic by taking into account the particular characteristics of each *reconfigurable* system to be specified. Thus we referred this construction technique as the *hybridisation* process.

Throughout the present dissertation, we were devoted on bringing the *hybridisation* to the working software engineer, supporting the motto *build the right tool for each job*. This was accomplished at some extent, by incorporating it into HETS, hence providing *hybridised* versions of logics integrated in the HETS network. Furthermore, we illustrated the methodology at work by exploring a case study of a critical medical device (the *insulin infusion pump*) where several *hybridised* logics were employed. These two lines of work correspond to what was defined as the goal of this

dissertation from the outset.

As additional contributions, we extended the theory supporting the *hybridisation* method, and in the process paved the way for broader means of validating specifications in *hybridised* logics, than the one proposed in [MMDB11]. In addition, we introduced a technique to tackle one of the limitations of the *hybridisation* process – the lack of machinery to capture systems whose interface also evolves. Finally, we extended the *hybridisation* to *lightweight formal methods* by providing the theory needed to *hybridise* one of the most famous of those (ALLOY), but also to validate specifications written in its *hybridised* or original version.

Another extra contribution, namely the comparison between $\mathcal{H}$CASL and $\mathcal{HPL}$ provers, lead to extra (small) contributions that may be useful in related work. They are:

1. The refactoring of HYLORES (detailed in section 2.3.3) in order to be compatible with the most recent versions of the respective compiler, *i.e.*, GLASGOW HASKELL COMPILER. The new version of HYLORES is available online [1].

2. The creation of a MACOS package (detailed in section 2.3.3) that provides an easy way to install several dedicated $\mathcal{HPL}$ provers. It is also available online [2].

3. The development of an application (used in section 3.2.3) that changes the format of a specification written in HTAB or HYLORES, to the format of $\mathcal{H}$CASL. Once again, it is available online[3].

As a final note we would like to stress that most of the contributions of this dissertation have already been submitted and validated by the international scientific community. In detail,

1. the incorporation of the *hybridisation* method in the HETS framework (chapter 3), published in [NMMB13b]. Interestingly, the source

[1] http://github.com/nevrenato/HyLoRes_Source
[2] https://github.com/nevrenato/Hybrid_package
[3] https://github.com/nevrenato/HTab2HCASL

code respective to this work was copied (by the HETS' community) to the latest version of *standard* HETS, which can be downloaded from its official website [4]. This is another positive feedback of the scientific community regarding this contribution, and more generally the methodology explored in the present dissertation.

2. The hybridisation of ALLOY and its coordination with HETS (chapter 5) published in [NMMB13a], with an invited extended journal version to appear in [NMMBar].

3. The extension for evolving interfaces (chapter 6) published in [MNMB13].

7.2 Future work

In the overall, this dissertation promotes a rigorous but flexible method for specifying *reconfigurable* systems, by giving contributions both in the theoretical and in the practical side. However, several lines of work are yet to be pursued, and furthermore new ones have emerged:

- **Generic proof support in HETS**. The implementation of the *hybridisation* method into HETS provides means to *hybridise* logics already integrated there. The next sensible step is thus to give proof support to arbitrary *hybridised* logics by implementing the lifting defined in reference [MMDB11].

- **A framework for combining logics**. The implementation of the *hybridisation* method into HETS provided knowledge and experience. These may be put to use in order to achieve in HETS a framework harbouring operations akin to *hybridisation*. The benefit of this is obvious: the end user gains means to combine logics in more ways than just the *hybridisation*.

[4] `http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/hets/`

- **Industrial assessment**. Although explored through several case studies in this dissertation, the methodology needs yet to be assessed in an industrial context.

- **Generalizing the *hybridisation* process**. Even though *hybridised* logics can capture automata whose states can take different forms, their underlying transition structure (*i.e.* the *Kripke frame*) is restricted to one specific class of automata. Therefore, many interesting classes of automata are missed. An example is the class of probabilistic transition systems (*c.f* [Paz71]), which have recently emerged as a main challenge in software engineering.

  From noticing that many kind of automata can be generalised into coalgebras (as pointed out by [Cô6]), one way to approach this boils down to replacing *Kripke frames*, by coalgebras for suitable endofunctors that can be instantiated into different classes of automata.

- **Exploration of other logic combiners.** The research of the *hybridisation* process was intensive. From this resulted knowledge and experience that can be recycled for improving similar combiners. Naturally, new ones are also in order.

Bibliography

[ACEGG90] Jaume Agustí-Cullell, Francesc Esteva, Pere Garcia, and Lluis Godo. Formalizing multiple-valued logics as institutions. In Bernadette Bouchon-Meunier, Ronald R. Yager, and Lotfi A. Zadeh, editors, *Uncertainty in Knowledge Bases, 3rd International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU 90, Paris, France, July 2-6, 1990, Proceedings*, volume 521 of *Lecture Notes in Computer Science*, pages 269–278. Springer, 1990.

[AH02] Carlos Areces and Juan Heguiabehere. Hylores: A hybrid logic prover based on direct resolution. In *Proc. 8th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Computer Science*. Springer, 2002.

[AHL+00] Serge Autexier, Dieter Hutter, Bruno Langenstein, Heiko Mantel, Georg Rock, Axel Schairer, Werner Stephan, Roland Vogt, and Andreas Wolpers. Vse: formal methods meet industrial needs. *STTT*, 3(1):66–77, 2000.

[AJJ+07] D. Arney, R. Jetley, P. Jones, Insup Lee, and O. Sokolsky. Formal methods based development of a pca infusion pump reference model: Generic infusion pump (gip) project. In *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007. HCMDSS-MDPnP. Joint Workshop on*, pages 23–33, 2007.

[AKMR03] 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.

[AtC06] Carlos Areces and Balder ten Cate. Hybrid logics. In P. Blackburn, F. Wolter, and J. van Benthem, editors, *Handbook of Modal Logics*, pages 821–868. Elsevier, 2006.

[Awo10] Steve Awodey. *Category Theory (Oxford Logic Guides)*. Oxford University Press, USA, 2 edition, 2010.

[BBW01] Patrick Blackburn, A. Burchard, and S. Walter. Hydra: a tableaux-based prover for basic hybrid logic. In *Proceedings of Methods for Modalities 2*, Amsterdam, The Netherlands, November 2001.

[BBW06] Patrick Blackburn, Johan F. A. K. van Benthem, and Frank Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., New York, NY, USA, 2006.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Scie*. Cambridge University Press, Cambridge, 2001.

[BFT06] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the model evolution calculus. *International Journal on Artificial Intelligence Tools*, 15(1):21–52, 2006.

[Bla06] Patrick Blackburn. Arthur prior and hybrid logic. *Synthese*, 150(3):329–372, 2006.

[Bra11] T Braüner. *Proof-Theory of Propositional Hybrid Logic*. Hybrid Logic and its Proof-Theory, 2011.

[BTPF08] 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.

[Cî06] Corina Cîrstea. An institution of modal logics for coalgebras. *J. Log. Algebr. Program.*, 67(1-2):87–113, 2006.

[CKP+08] Corina Cîrstea, Alexander Kurz, Dirk Pattinson, Lutz Schröder, and Yde Venema. Modal logics are coalgebraic. In Samson Abramsky, Erol Gelenbe, and Vladimiro Sassone, editors, *Visions of Computer Science, BCS International Academic Research Conference (BCS 2008)*, pages 129–140. British Computer Society, 2008.

[CMC10] Marta Cialdea Mayer and Serenella Cerrito. Herod and pilate: two tableau provers for basic hybrid logic. In *Proceedings of the 5th international conference on Automated Reasoning*, IJCAR'10, pages 255–262, Berlin, Heidelberg, 2010. Springer-Verlag.

[Dia08] Răzvan Diaconescu. *Institution-independent Model Theory*. Birkhäuser Basel, 2008.

[Dia10] Răzvan Diaconescu. Quasi-boolean encodings and conditionals in algebraic specification. *J. Log. Algebr. Program.*, 79(2):174–188, 2010.

[Dia13] Răzvan Diaconescu. Institutional semantics for many-valued logics. *Fuzzy Sets Syst.*, 218:32–52, May 2013.

[DS07] Răzvan Diaconescu and Petros Stefaneas. Ultraproducts and possible worlds semantics in institutions. *Theor. Comput. Sci.*, 379(1-2):210–230, July 2007.

[GB92] Joseph A. Goguen and Rod M. Burstall. Institutions: abstract model theory for specification and programming. *J. ACM*, 39:95–146, January 1992.

[GG93] G. Gargov and V. Goranko. Modal logic with names. *Journal of Philosophical Logic*, 22(6):607–636, 1993.

[GKS10] Daniel Götzmann, Mark Kaminski, and Gert Smolka. Spartacus: A tableau prover for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 262:127–139, 2010.

[GM87] Joseph A. Goguen and José Meseguer. Models and equality for logical programming. In Hartmut Ehrig, Robert Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT '87*, volume 250 of *Lecture Notes in Computer Science*, pages 1–22. Springer Berlin Heidelberg, 1987.

[Gog91] Joseph A. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, 1991.

[Gor96] Valentin Goranko. Hierarchies of modal and temporal logics with reference pointers. *Journal of Logic, Language and Information*, 5(1):1–24, 1996.

[Got01] Siegfried Gottwald. A treatise on many-valued logics. In *Studies in Logic and Computation*. Press, 2001.

[HA09] Guillaume Hoffmann and Carlos Areces. Htab: a terminating tableaux system for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 231:3–19, 2009.

[HM10] 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.

[Hol03] Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[Mad13] Alexandre Madeira. *Foundations and techniques for software reconfigurability (An institution-independent approach to specifying and reasoning about reconfigurable systems)*. PhD thesis, Minho, Aveiro and Porto Universities (MAP-i Doctoral Programme), July 2013.

[MBMNed] Alexandre Madeira, Luis Barbosa, Manuel Martins, and Renato Neves. Evolving software and hybrid specifications (An educational perspective). To be published.

[MC12] Nuno Macedo and Alcino Cunha. Automatic unbounded verification of Alloy specifications with Prover9. *CoRR*, abs/1209.5773, 2012.

[MFMB11] 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.

[MHST03] 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.

[MMDB11] 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.

[MML07] 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.

[MNMB13] Alexandre Madeira, Renato Neves, Manuel A. Martins, and Luís S. Barbosa. When even the interface evolves... In Hai Wang and Richard Banach, editors, *Proceedings of 7th Internl Symp. on Theoretical Aspects of Software Engineering (TASE 2013)*, pages 79–82. IEEE Press, 2013.

[Mos04] Till Mossakowski. Modalcasl - specification with multimodal logics. language summary. Technical report, 2004.

[MRRS06] Till Mossakowski, Horst Reichel, Markus Roggenbach, and Lutz Schröder. Algebraic-coalgebraic specification in cocasl. *J. LOGIC ALGEBRAIC PROGRAMMING*, 67:2006, 2006.

[Muk12] R. R. Mukkamala. *A Formal Model For Declarative Workflows : Dynamic Condition Response Graphs*. PhD thesis, IT University of Copenhagen, 2012.

[NMMB13a] 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 in-*

formation reuse and intergration, pages 512–519. IEEE press, 2013.

[NMMB13b] Renato Neves, Alexandre Madeira, Manuel A. Martins, and Luís S. Barbosa. Hybridisation at work. In *CALCO TOOLS*, volume 8089 of *Lecture Notes in Computer Science*. Springer, 2013.

[NMMBar] Renato Neves, Alexandre Madeira, Manuel A. Martins, and Luís S. Barbosa. Unbounded verification with alloy : An institutional perspective. Advances in Intelligent and Soft Computing, 2013. (to appear).

[NWP02] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[Paz71] Azaria Paz. *Introduction to probabilistic automata (Computer science and applied mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1971.

[Rog03] Markus Roggenbach. Csp-casl – a new integration of process algebra and algebraic specification. *Theoretical Computer Science*, 354:2006, 2003.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *AI Commun.*, 15(2,3):91–110, August 2002.

[Sch02] Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2,3):111–126, August 2002.

[SM09] Lutz Schröder and Till Mossakowski. HasCasl: Integrated higher-order specification and program development. *Theor. Comput. Sci.*, 410(12-13):1217–1260, 2009.

[ST12] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. EATCS Monographs on theoretical computer science. Springer, 2012.

[UGGT12] 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.

[vE02] Jan van Eijck. Hylotab-tableau-based theorem proving for hybrid logics. Technical report, 2002.

[WDF+09] 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.

[XM12] Hao Xu and Tom Maibaum. An event-b approach to timing issues applied to the generic insulin infusion pump. In *Proceedings of the First international conference on Foundations of Health Informatics Engineering and Systems*, FHIES'11, pages 160–176, Berlin, Heidelberg, 2012. Springer-Verlag.

[ZJJ10] Yi Zhang, Paul L Jones, and Raoul Jetley. A hazard analysis for a generic insulin infusion pump. *J Diabetes Sci Technol*, 4(2):263–83, 2010.

[ZJJR11] Yi Zhang, Raoul Jetley, Paul L Jones, and Arnab Ray. Generic safety requirements for developing safe insulin pump software. *J Diabetes Sci Technol*, 5(6):1403–19, 2011.

Appendix A

Reconfigurable Calculator – Full specification

spec ReconfCalc =
HNat
then **modality** *Shift* {}
nominal *Sum*;
Mult
{}
op __#__ : $Nat \times Nat \to Nat$

%% global axioms
$\forall\ n, m, p : Nat$
• $n \# m = m \# n$
• $(n \# m) \# p = n \# (m \# p)$

%% axioms specific to Sum and Mult
$\forall\ n, m : Nat$
• $@Sum\ n \# 0 = n$
• $@Sum\ n \# suc(m) = suc(n \# m)$
• $@Mult\ n \# 0 = 0$
• $\exists\ p, q : Nat$

• $@Mult\ n\ \#\ suc(m) = p \wedge @Sum\ n\ \#\ q = p$
$\wedge\ @Mult\ n\ \#\ m = q$

%% axioms specific to the Kripke frame
• $Here\ Sum \vee Here\ Mult$
• $@Sum\ (< Shift > Here\ Mult \wedge [Shift]\ Here\ Mult)$
• $@Mult\ (< Shift > Here\ Sum \wedge [Shift]\ Here\ Sum)$

%% lt relation definition, using # op
$\forall\ n, m, r : Nat \bullet n <= m \Rightarrow n\ \#\ r <= m\ \#\ r$

%% lemmas
$\forall\ n, m, r : Nat$
• $@Sum\ (n < m \Rightarrow n < m\ \#\ r)$ **%implied**
%(lemma1)%
• $@Sum\ n\ \#\ m >= n$ **%implied**
%(lemma2)%
• $@Sum\ n\ \#\ suc(0) = suc(n)$ **%implied**
%(lemma3)%
• $@Mult\ n\ \#\ suc(0) = n$ **%implied**
%(lemma4)%
• $@Mult\ n\ \#\ 0 <= n$ **%implied**
%(lemma5)%
• $@Mult\ (m = 0 \vee m = suc(0) \Rightarrow n\ \#\ m <= n)$ **%implied**
%(lemma6)%
• $\exists\ p : Nat \bullet @Sum\ n\ \#\ 0 = p \wedge @Mult\ n\ \#\ suc(0) = p$
%implied
%(lemma7)%

%% verified properties
• $@Sum\ [Shift]\ [Shift]\ Here\ Sum$ **%implied**
%(Cyclicity1)%
$\forall\ n, m, r : Nat$

• $n \# 0 = 0 \Rightarrow < Shift > n \# 0 = n$ **%implied**

%(StateExclusion)%

• $\exists\, p : Nat$

• $@Sum\ n \# m = p \wedge @Sum < Shift > < Shift > n \# m = p$ **%implied**

%(Cyclicity2)%

• $\exists\, p : Nat \bullet @Sum\ n \# n = p \Rightarrow @Mult\ n \# suc(suc(0)) = p$ **%implied**

%(DoubleDef)%

• $\exists\, p, q : Nat$

• $m <= suc(0)$

$\Rightarrow @Sum\ n \# m = p \wedge @Mult\ n \# m = q \Rightarrow p >= q$ **%implied**

%(CasesSumBiggerMult)%

end

Appendix B

Requirements for the infusion pump

Number	Title	Description
1	Modes	The pump can be *on*, *off* or in *test* mode.
2	Diagnostic	When *off* the pump can be turned *on*, but first it must go through a series of diagnostics *tests*. Only if all are *ok*, the pump is turned *on*.
3	Going off	Only from mode *on*, the pump can be turned *off*.
4	Suspension	When *on*, the pump can be in two modes: *Suspension* or *Normal*.
5	Resuming	When running *normally* the pump may be *suspended*. Conversely, when *suspended*, resuming it is an option.
6	Default mode	When *diagnosing*, if all tests go *ok*, the pump goes by default to *Normal* mode

Table B.1: Requirements list for $\mathcal{HPL}$, $\mathcal{H}^2\mathcal{PL}$

number	title	description
1	Modes of operation	The modes of insulin infusion are *basal* and *temporary basal*.
2	Bolus	In either of the infusion modes, a *bolus* can be given. The latter can be *instantaneous* or *extended*.
3	Bolus extended	An *extended bolus* cannot be given while other is active.
4	Suspension	When *on* the pump can be *suspended*.
5	Resume	If the pump is *suspended* while a *bolus* is being given, when resumed, that same *bolus* should be forgotten.
6	Disjointness	The pump cannot be in *temporary basal* and *normal basal* at the same time.

Table B.2: Requirements list for $\mathcal{HPL}$

number	title	description
1	Max flow	When the pump is *suspended* the maximum flow must be 0. In all others modes the maximum flow is defined by the user.
2	Basal rates	When in the *basal normal* mode, the current flow must be equal to the value given by the basal profile.
3	Temporary basal rates	When in the *temporary basal* mode, the current flow must be equal to the value given by the temporary basal profile.
4	Bolus rates	When in a state where the *bolus* is active, the respective value must be added to the current flow.
5	Extended bolus	The insulin infusion rate for a particular extended bolus session, must be constant.
6	Static profiles	The *basal*, *temporary basal* and *extended bolus* profiles, cannot be changed while the pump is active.

Table B.3: Requirements list for $\mathcal{HFEQ}$

number	title	description
1	Max flow	Indepedently of the mode of operation, the flow rate cannot surpass the maximum flow established.

Table B.4: Requirements list for $\mathcal{HFOL}$

Appendix C

Infusion Pump – Full specification

library SOURCE_MODELS/IIP

logic HYBRID

spec IIP_FST =
nominal *On*;
Off;
Test
{}
modality *Turn_on*;
Turn_off;
AllOK
{}

%% the states are different from each other
• @*On* (¬ (*Here Off* ∨ *Here Test*))
• @*Off* (¬ (*Here On* ∨ *Here Test*))
• @*Test* (¬ (*Here On* ∨ *Here Off*))

• $Here\ On \vee Here\ Off \vee Here\ Test$

%% Restricing the relations between states

• $@On < Turn\_off > Here\ Off \wedge [Turn\_off]\ Here\ Off$

• $@Off < Turn\_on > Here\ Test \wedge [Turn\_on]\ Here\ Test$

• $@Test < AllOK > Here\ On \wedge [AllOK]\ Here\ On$

%% Giving to each modality the corresponding state

• $< Turn\_off > true \Rightarrow Here\ On$

• $< Turn\_on > true \Rightarrow Here\ Off$

• $< AllOK > true \Rightarrow Here\ Test$

%% properties

• $@On < Turn\_off > < Turn\_on > < AllOK > Here\ On$ **%implied**

%(cyclic)%

• $@On\ [Turn\_off]\ [Turn\_on]\ [AllOK]\ Here\ On$ **%implied**

%(deterministic)%

%% The On state is ONLY reachable when ALL diagnostic tests sucessfully passed

• $(< AllOK > Here\ On \Rightarrow Here\ Test)$
$\wedge \neg (< Turn\_off > Here\ On \vee < Turn\_on > Here\ On)$ **%implied**

%(OnOnlybyTest)%

end

spec IIP_LOW_KRIPKEF =

nominal *BnBo*;
BtBo;
Su1;
Su2;
Bn;
Bt

{}

modality *Sus*;
Res;
Go_Bo;
Go_Bt
{}

%% the states are different from each other, and are the only in the universe
• $@BnBo$
$(\neg\ (Here\ BtBo \vee Here\ Su1 \vee Here\ Su2 \vee Here\ Bn \vee Here\ Bt))$
• $@BtBo\ (\neg\ (Here\ Su1 \vee Here\ Su2 \vee Here\ Bn \vee Here\ Bt))$
• $@Su1\ (\neg\ (Here\ Su2 \vee Here\ Bn \vee Here\ Bt))$
• $@Su2\ (\neg\ (Here\ Bn \vee Here\ Bt))$
• $@Bn\ (\neg\ Here\ Bt)$
• $Here\ BnBo \vee Here\ BtBo \vee Here\ Su1 \vee Here\ Su2 \vee Here\ Bn$
$\vee\ Here\ Bt$

%% specifying relations between states
• $@BnBo\ (< Sus > Here\ Su1 \wedge < Res > Here\ Bn)$
• $@BtBo\ (< Sus > Here\ Su2 \wedge < Res > Here\ Bt)$
• $@Su1 < Res > Here\ Bn \wedge @Su2 < Res > Here\ Bt$
• $@Bn$
$(< Sus > Here\ Su1 \wedge < Go\_Bo > Here\ BnBo$
$\wedge < Go\_Bt > Here\ Bt)$
• $@Bt$
$(< Sus > Here\ Su2 \wedge < Res > Here\ Bn$
$\wedge < Go\_Bo > Here\ BtBo)$

%% Restrict origin.
• $< Go\_Bt > true \Rightarrow Here\ Bn$
%% From the origin restrict the destination
• $@Bn\ [Go\_Bt]\ Here\ Bt$

%% Folows an example, with shared modalities

%% Restricting the origin the destination in one sentence

- $(< Go\_Bo > true \Rightarrow Here\ Bn \vee Here\ Bt)$
  $\wedge\ @Bn\ [Go\_Bo]\ Here\ BnBo \wedge @Bt\ [Go\_Bo]\ Here\ BtBo$

%% Because they share a modality we also need to restrict the modality among them

- $(< Go\_Bo > Here\ BnBo \Rightarrow Here\ Bn)$
  $\wedge\ (< Go\_Bo > Here\ BtBo \Rightarrow Here\ Bt)$

- $< Sus > true$
  $\Rightarrow Here\ Bn \vee Here\ BnBo \vee Here\ Bt \vee Here\ BtBo$
- $@Bn\ [Sus]\ Here\ Su1 \wedge @BnBo\ [Sus]\ Here\ Su1$
  $\wedge\ @Bt\ [Sus]\ Here\ Su2 \wedge @BtBo\ [Sus]\ Here\ Su2$
- $(< Sus > Here\ Su1 \Rightarrow Here\ Bn \vee Here\ BnBo)$
  $\wedge\ (< Sus > Here\ Su2 \Rightarrow Here\ Bt \vee Here\ BtBo)$
- $< Res > true$
  $\Rightarrow Here\ Su1 \vee Here\ Su2 \vee Here\ BtBo \vee Here\ Bt \vee Here\ BnBo$
- $@Su1\ [Res]\ Here\ Bn \wedge @Su2\ [Res]\ Here\ Bt$
  $\wedge\ @BtBo\ [Res]\ Here\ Bt \wedge @Bt\ [Res]\ Here\ Bn$
  $\wedge\ @BnBo\ [Res]\ Here\ Bn$
- $(< Res > Here\ Bt \Rightarrow Here\ Su2 \vee Here\ BtBo)$
  $\wedge\ (< Res > Here\ Bn \Rightarrow Here\ Bt \vee Here\ BnBo \vee Here\ Su1)$

%% Properties (relations between states)

- $\neg\ (Here\ Su1 \vee Here\ Su2) \Rightarrow < Sus > (Here\ Su1 \vee Here\ Su2)$
  %implied

%(alwaysSuspendable)%

- $Here\ Su1 \vee Here\ Su2 \Rightarrow < Res > (Here\ Bn \vee Here\ Bt)$
  %implied

%(afterAnySuspensionBolusAreLost)%

- $Here\ BnBo \vee Here\ BtBo \Rightarrow \neg < Go\_Bo > true$ **%implied**

%(beingInBolusModeIsNotPossibleToActivateBolus)%

- $< Go\_Bo > true \Rightarrow < Go\_Bo > (Here\ BnBo \vee Here\ BtBo)$
  %implied

%(BolusModeIsOnlyAchiveableByRequestOfTheUser)%

spec IIP_LOW =
HNAT
then IIP_LOW_KRIPKEF
then **rigid ops**
max_flow : *Nat*;
%% max flow of insulin permitted
basal : *Nat* → *Nat*;
%% The basal application, in later instances we could turn it in a pred
tbasal : *Nat* → *Nat*;
%% The basal temporary, in later instances we could turn it in a pred
ext_bolus : *Nat* → *Nat*;
%% The extended bolus profile
ins_bolus : *Nat* → *Nat*
%% The instant bolus profile
op *cur_flow* : *Nat* → *Nat*
%% the current flow of insulin
∀ *t* : *Nat*
• @*Bn* *cur_flow*(*t*) = *basal*(*t*)
• @*BnBo* *cur_flow*(*t*) = *basal*(*t*) + *ext_bolus*(*t*)
• @*Bt* *cur_flow*(*t*) = *tbasal*(*t*)
• @*BtBo* *cur_flow*(*t*) = *tbasal*(*t*) + *ext_bolus*(*t*)
• @*Su1* *cur_flow*(*t*) = 0
• @*Su2* *cur_flow*(*t*) = 0
• ∀ *t* : *Nat* • *cur_flow*(*t*) <= *max_flow*

%% Properties
∀ *t* : *Nat*
• @*Bn* *cur_flow*(*t*) <= *max_flow* **%implied**
%(helper1)%
• @*Bn* *basal*(*t*) <= *max_flow* **%implied**
%(onlyBasal)%
• @*Bn* *cur_flow*(*t*) >= *basal*(*t*) **%implied**

%(atleastbasal)%

• $\exists\ n, n' : Nat$

  • $@Su1$

    $(cur\_flow(t) = n \wedge < Res > cur\_flow(t) = n' \wedge n' >= n)$

%implied

%(afterSus)%

• $\forall\ n : Nat \bullet @Su1\ cur\_flow(t) = n \Rightarrow cur\_flow(t) >= n$

%implied

%(theflowistheleastpossible)%

• $Here\ Bn \vee Here\ BnBo \Rightarrow cur\_flow(t) >= basal(t)$

%(helper2)%

• $Here\ Bt \vee Here\ BtBo \Rightarrow cur\_flow(t) >= tbasal(t)$

%(helper2a)%

• $@BnBo\ cur\_flow(t) >= basal(t)$ **%implied**

%(helper3)%

• $@Bt\ cur\_flow(t) >= tbasal(t)$ **%implied**

%(helper4)%

• $@BtBo\ cur\_flow(t) >= tbasal(t)$ **%implied**

%(helper5)%

• $\neg\ (Here\ Su1 \vee Here\ Su2)$

  $\Rightarrow cur\_flow(t) >= basal(t) \vee cur\_flow(t) >= tbasal(t)$

%implied

%(mustbeworking)%

• $\neg\ cur\_flow(t) = basal(t) \Rightarrow \neg\ Here\ Bn$ **%implied**

%(contradiction)%

• $(\forall\ t : Nat \bullet ext\_bolus(t) = 0)$

  $\Rightarrow @Bn\ cur\_flow(t) = basal(t)$

    $\wedge\ @BnBo\ cur\_flow(t) = basal(t)$

%implied

%(helper6)%

• $(\forall\ t : Nat \bullet ext\_bolus(t) = 0)$

  $\Rightarrow \forall\ t : Nat$

    • $\exists\ n : Nat$

• $@Bn\ cur\_flow(t) = n \wedge @BnBo\ cur\_flow(t) = n$

%implied

%(states_same_flow)%

• $\forall\ t : Nat$

• $\exists\ n, n' : Nat$

• $@Bn\ cur\_flow(t) = n \wedge @BnBo\ cur\_flow(t) = n'$
$\wedge\ n' >= n$

%implied

%(curflow_betw_states)%

end

spec IIP_LOWPAR =

HNAT

then IIP_LOW_KRIPKEF

then ops $cur\_flow : Nat \to Nat$;
$basal : Nat \to? Nat$;
$tbasal : Nat \to? Nat$;
$ext\_bolus : Nat \to? Nat$

%% presentation the axioms of defininig partiality

$\forall\ t : Nat$

• $Here\ BnBo$
$\Rightarrow basal(t) \stackrel{e}{=} basal(t) \wedge ext\_bolus(t) \stackrel{e}{=} ext\_bolus(t)$

• $Here\ BtBo$
$\Rightarrow tbasal(t) \stackrel{e}{=} tbasal(t) \wedge ext\_bolus(t) \stackrel{e}{=} ext\_bolus(t)$

• $\neg\ (Here\ BnBo \vee Here\ BtBo) \Rightarrow \neg\ ext\_bolus(t) \stackrel{e}{=} ext\_bolus(t)$

• $Here\ Bn \Rightarrow basal(t) \stackrel{e}{=} basal(t)$

• $Here\ Bt \Rightarrow tbasal(t) \stackrel{e}{=} tbasal(t)$

• $\neg\ (Here\ Bn \vee Here\ BnBo) \Rightarrow \neg\ basal(t) \stackrel{e}{=} basal(t)$

• $\neg\ (Here\ Bt \vee Here\ BtBo) \Rightarrow \neg\ tbasal(t) \stackrel{e}{=} tbasal(t)$

%% applying existential rigidification manually

$\forall\ t : Nat$

• $\exists\ n : Nat \bullet @Bn\ basal(t) = n \wedge @BnBo\ basal(t) = n$

• $\exists\ n : Nat$

  • $@BnBo\ ext\_bolus(t) = n \wedge @BtBo\ ext\_bolus(t) = n$

• $\exists\ n : Nat \bullet @Bt\ tbasal(t) = n \wedge @BtBo\ tbasal(t) = n$

%% characterising the cur_flow

$\forall\ t : Nat$

• $\neg\ ext\_bolus(t) \stackrel{e}{=} ext\_bolus(t) \wedge def\ basal(t)$
  $\Rightarrow cur\_flow(t) = basal(t)$

• $\neg\ ext\_bolus(t) \stackrel{e}{=} ext\_bolus(t) \wedge def\ tbasal(t)$
  $\Rightarrow cur\_flow(t) = tbasal(t)$

• $def\ basal(t) \wedge def\ ext\_bolus(t)$
  $\Rightarrow cur\_flow(t) = basal(t) + ext\_bolus(t)$

• $def\ tbasal(t) \wedge def\ ext\_bolus(t)$
  $\Rightarrow cur\_flow(t) = tbasal(t) + ext\_bolus(t)$

%% properties to prove

• $\forall\ t : Nat \bullet def\ ext\_bolus(t) \Rightarrow ext\_bolus(t) > 0$

• $(\neg\ \forall\ t : Nat \bullet cur\_flow(t) \stackrel{e}{=} basal(t)) \Leftrightarrow \neg\ Here\ Bn$ **%implied**

%(state equiv)%

• $\forall\ t : Nat \bullet basal(t) \stackrel{e}{=} basal(t) \Leftrightarrow Here\ Bn \vee Here\ BnBo$ **%implied**

%(state equiv2)%

• $Here\ Bn \wedge Here\ Bt$ **%implied**

%(incon)%

• $\forall\ t : Nat$

  • $\neg\ ext\_bolus(t) \stackrel{e}{=} ext\_bolus(t) \wedge def\ basal(t) \Leftrightarrow Here\ Bn$ **%implied**

%(test)%

end

library IIPDuplHyb

logic HYBRIDIZE

spec IIP_DUPL =
Nominals *On*, *Off*,
Modalities *TurnOn*, *TurnOff*, *Ok*
Under Spec {
Nominals *Test*, *Off*, *Nor*, *Sus*,
Modalities *Sus*, *Res*, *NotOk*, *Activate*
Under Spec { *p* **}**
%% The possible sub states
Test $\vee$ *Off* $\vee$ *Nor* $\vee$ *Sus*
}
%% The possible super states
On $\vee$ *Off*

%% Defining and restricting relations between the sub states in the super state On
@ *On* **{ @** *Nor* **(** **<***Sus***>** *Sus* $\wedge$ **[***Sus***]** *Sus* **)** $\wedge$ **@** *Sus* **(** **<***Res***>** *Nor* $\wedge$ **[***Res***]** *Nor* **) },**
@ *On* **{ (** **<***Sus***>** **True** $\Rightarrow$ *Nor* $\wedge$ **<***Res***>** **True** $\Rightarrow$ *Sus* **)** $\wedge$
$\neg$ **(** **<***NotOk***>** **True** $\vee$ **<***Activate***>** **True) }**

%% Defining and restricting relations between the sub states in the super state Off
@ *Off* **{ @** *Test* **(** **<***NotOk***>** *Off* $\wedge$ **[***NotOk***]** *Off* **)** $\wedge$
@ *Off* **(** **<***Activate***>** *Test* $\wedge$ **[***Activate***]** *Test* **) }**
@ *Off* **{ (** **<***NotOk***>** **True** $\Rightarrow$ *Test* $\wedge$ **<***Activate***>** **True** $\Rightarrow$ *Off* **)** $\wedge$
$\neg$ **(** **<***Res***>** **True** $\vee$ **<***Sus***>** **True) }**

end