Heterogeneous Development Graphs and Heterogeneous Borrowing

Till Mossakowski

BISS, Dept. of Computer Science, University of Bremen
till@tzi.de

Abstract. Development graphs are a tool for dealing with structured specifications in a formal program development in order to ease the management of change and reusing proofs. Often, different aspects of a software system have to be specified in different logics, since the construction of a huge logic covering all needed features would be too complex to be feasible. Therefore, we introduce *heterogeneous development graphs* as a means to cope with heterogeneous specifications.
We cover both the semantics and the proof theory of heterogeneous development graphs. A proof calculus can be obtained either by combining proof calculi for the individual logics, or by representing these in some "universal" logic like higher-order logic in a coherent way and then "borrowing" its calculus for the heterogeneous language.

1 Introduction

In an evolutionary software development process using formal specifications, typically not only implementations evolve over time, but during attempts to prove correctness (of implementations), also the *specifications* may turn out to be incorrect and therefore have to be revised. Development graphs [3] with hiding [13] have been introduced as a tool for dealing with the necessary management of change in structured formal specifications, with the goal of re-using proofs as much as possible. In this work, we extend these to deal with *heterogeneous* specifications, consisting of parts written in different logics. This is needed, since complex problems have different aspects that are best specified in different logics, while a combination of all these would become too complex in many cases. Moreover, we also aim at formal interoperability among different *tools*.

Consider the following sample specification, written in CASL-LTL [18], an extension of the Common algebraic specification language CASL [17] with a labeled transition logic. The behaviour of a buffer and a user writing into and reading from that buffer is described at a very abstract requirements level using temporal logic (cf. [2], chapter 13).

%CASL-LTL
spec SYSTEM = BUFFER **and** USER
then **dsort** *system*
free **types** $system ::= \_\_ || \_\_ (buffer;\ user)$
$lab\_system ::= START \mid OK \mid ERROR \mid tau$

M. Nielsen and U. Engberg (Eds.): Fossacs 2002, LNCS 2303, pp. 326–341, 2002.

∀s,s':SYSTEM
• $\nabla(s, \Box\langle\lambda l.\neg l = ERROR\rangle)$
%% there is always a possible correct behaviour
• $s \xrightarrow{START} s' \Rightarrow \nabla(s', \diamond(\langle OK\rangle \vee \langle ERROR\rangle))$
%% after starting, always the system will eventually send out either OK or ERROR
• $(\ s \xrightarrow{OK} s' \ \vee \ s \xrightarrow{ERROR} s' \Rightarrow \nabla(s', \Box\langle\lambda l.\neg l = OK \vee l = ERROR\rangle))$
%% OK and ERROR are sent at most once, and it cannot happen that both are sent
...

Typically, within the development process, one will need to refine the abstract requirements and pass over to a concrete design. This could, for example, be expressed in SB-CASL [5], a state based extension of CASL following the paradigm of abstract state machines. Here, it is possible to use assignments and sequential and parallel composition, such that the passage to an imperative program is no longer a big step. In order to be able to interface the SB-CASL specification with the CASL-LTL specification above, we here use labeled transition system signature as above, which is projected from CASL-LTL to CASL with **hide** *LTL-keep*, which means that the labeled transition system is kept (while the temporal logic formulae are dropped). Thus, we get a heterogeneous specification, consisting of parts written in different logics, and of inter-logic translations and inter-logic projections (we here denote logic translations by **with** and logic projections by **hide**, in analogy to CASL's translations and hidings *within* one logic).

%SB-CASL
System BIT
use VALUE **then**
{ **%CASL-LTL**
dsort *system*
free types $system ::= \_\_ || \_\_(buffer;\ user)$
$lab\_system ::= START \mid OK \mid ERROR \mid tau$ **}**
hide *LTL-keep*
dynamic func $Buf\_Cont : buffer;\ User\_State : user\_state;$
proc $proc\_START;\ proc\_OK;\ proc\_ERROR;\ proc\_tau;$
• $proc\_START =$ **seq** $User\_State := Putting\_0;\ Buf\_Cont := Empty$ **end**
• $proc\_tau =$ *if* $User\_State = Putting\_0$ *then*
seq $User\_State := Putting\_1;$
$Buf\_Cont := Put(0, Buf\_Cont')$ **end**
elseif ...
post $proc\_tau : (Buf\_Cont||User\_State) \xrightarrow{tau} (Buf\_Cont'||User\_State')$

%% Specify the generated LTL relation

We then have that BIT actually is a refinement of SYSTEM, which can be expressed in the following form in SB-CASL:

view V : **{** SYSTEM **hide** LTL-keep **with** SB-CASL **} to** BIT **end**
%% BIT is a possible run of SYSTEM

The first specification within this refinement is again heterogeneous: SYSTEM is projected from LTL-CASL to CASL (using **hide** LTL-keep) and then translated from CASL to SB-CASL (using **with** SB-CASL).

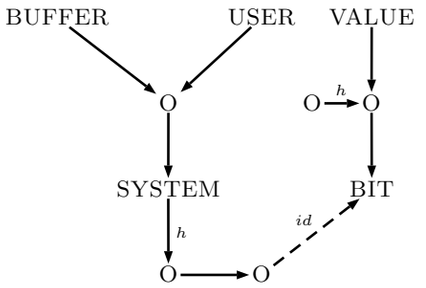

Fig. 1. Development graph for the refinement of SYSTEM into BIT

In Fig. 1 we present the development graph expressing the above refinement. Development graphs concentrate on the (homogeneous and heterogeneous) structuring of specifications and proofs, independently of the particular structuring or module-building constructs of the input language. In order to be able to use heterogeneous development graphs for performing proofs, we introduce the notion of *heterogeneous borrowing*, which is a generalization to the heterogeneous case of the notion of borrowing [8], which allows to re-use theorem provers. As an application, we will sketch how to represent various extensions of CASL within higher-order logic in a coherent way, such that heterogeneous borrowing becomes applicable. This means that we can re-use any theorem prover for higher-order logic to do theorem proving in the heterogeneous logic consisting of CASL and some of its extensions.

2 Preliminaries

When studying heterogeneous development graphs, we want to focus on the structuring and abstract from the details of the underlying logical systems. Therefore, we recall the abstract notion of institution [10].

Definition 1. *An* institution $I = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ *consists of*

- *a category* $\mathbf{Sign}$ *of* signatures,
- *a functor* $\mathbf{Sen}\colon \mathbf{Sign} \longrightarrow \mathbf{Set}$ *giving the set of* sentences $\mathbf{Sen}(\Sigma)$ *over each signature* Σ*, and for each signature morphism* $\sigma\colon \Sigma \longrightarrow \Sigma'$*, the sentence translation map* $\mathbf{Sen}(\sigma)\colon \mathbf{Sen}(\Sigma) \longrightarrow \mathbf{Sen}(\Sigma')$*, where often* $\mathbf{Sen}(\sigma)(\varphi)$ *is written as* $\sigma(\varphi)$*,*

- *a functor* $\mathbf{Mod}\colon \mathbf{Sign}^{op} \longrightarrow \mathcal{CAT}$ *giving the category of* models *over a given signature, and for each signature morphism* $\sigma\colon \Sigma \longrightarrow \Sigma'$*, the* reduct functor $\mathbf{Mod}(\sigma)\colon \mathbf{Mod}(\Sigma') \longrightarrow \mathbf{Mod}(\Sigma)$*, where often* $\mathbf{Mod}(\sigma)(M')$ *is written as* $M'|_\sigma$*,*
- *a satisfaction relation* $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ *for each* $\Sigma \in \mathbf{Sign}$*,*

such that for each $\sigma\colon \Sigma \longrightarrow \Sigma'$ *in* **Sign***,*

$$M' \models_{\Sigma'} \sigma(\varphi) \Leftrightarrow M'|_\sigma \models_\Sigma \varphi$$

holds for each $M' \in \mathbf{Mod}(\Sigma')$ *and each* $\varphi \in \mathbf{Sen}(\Sigma)$ *(*satisfaction condition*).*

A *logic* is an institution equipped with an *entailment system* consisting of an entailment relation $\vdash_\Sigma \subseteq |\mathbf{Sen}(\Sigma)| \times \mathbf{Sen}(\Sigma)$, for each $\Sigma \in |\mathbf{Sign}|$, such that the following conditions are satisfied:

1. *reflexivity:* for any $\varphi \in \mathbf{Sen}(\Sigma)$, $\{\varphi\} \vdash_\Sigma \varphi$,
2. *monotonicity:* if $\Gamma \vdash_\Sigma \varphi$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash_\Sigma \varphi$,
3. *transitivity:* if $\Gamma \vdash_\Sigma \varphi_i$, for $i \in I$, and $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_\Sigma \psi$, then $\Gamma \vdash_\Sigma \psi$,
4. $\vdash$*-translation:* if $\Gamma \vdash_\Sigma \varphi$, then for any $\sigma\colon \Sigma \longrightarrow \Sigma'$ in **Sign**, $\sigma[\Gamma] \vdash_{\Sigma'} \sigma(\varphi)$,
5. *soundness:* for any $\Sigma \in |\mathbf{Sign}|$, $\Gamma \subseteq \mathbf{Sen}(\mathbf{Sign})$ and $\varphi \in \mathbf{Sen}(\Sigma)$,

$$\Gamma \vdash_\Sigma \varphi \text{ implies } \Gamma \models_\Sigma \varphi.$$

A logic will be called *complete* if, in addition, the converse of the soundness implication holds.

We will at times need the assumption that a given institution $I = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ has *composable signatures*, i.e. **Sign** has finite colimits, and moreover, I admits *weak amalgamation*, i.e. **Mod** maps finite colimits to weak limits. Informally, this means that if a diagram of signatures is glued together, then it is also possible to glue together families of models that are compatible w.r.t. the morphisms in the diagram.

Examples of logics that can be formalized in this sense are many-sorted equational logic, many-sorted first-order logic, higher-order logic, various lambda calculi, various modal, temporal, and object-oriented logics etc.

3 Development Graphs with Hiding

Development graphs, as introduced in [3], represent the actual state of a formal program development. They are used to encode the structured specifications in various phases of the development and make them amenable to theorem proving. Roughly speaking, each node of such a graph represents a theory such as BIT in the above example. The links of the graph define how theories can make use of other theories. Leaves in the graph correspond to basic specifications, which do not make use of other theories (e.g. VALUE). Inner nodes correspond to structured specifications which define theories importing other theories (e.g. BIT using VALUE). The corresponding links in the graph are called *definition*

links. If only part of a theory shall be imported, one can use a *hiding link*. With these kinds of links, one can express a wide variety of formalisms for structured specification. E.g., there is a translation from CASL structured specifications (as used in the examples) to development graphs [3].

Fix an arbitrary institution $I = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$.

Definition 2. *A* development graph *is an acyclic directed graph* $\mathcal{S} = \langle \mathcal{N}, \mathcal{L} \rangle$.

$\mathcal{N}$ is a set of nodes. Each node $N \in \mathcal{N}$ is a tuple (Σ^N, Γ^N) such that Σ^N is a signature and $\Gamma^N \subseteq \mathbf{Sen}(\Sigma^N)$ is the set of **local axioms** *of N.*

$\mathcal{L}$ is a set of directed links, so-called **definition links***, between elements of $\mathcal{N}$. Each definition link from a node M to a node N is either*

- **global**[1] *(denoted $M \xrightarrow{\sigma} N$), annotated with a signature morphism $\sigma : \Sigma^M \to \Sigma^N$, or*
- **hiding** *(denoted $M \xrightarrow[h]{\sigma} N$), annotated with a signature morphism $\sigma : \Sigma^N \to \Sigma^M$* going against the direction of the link. *Typically, σ will be an inclusion, and the symbols of Σ^M not in Σ^N will be hidden.*

Definition 3. *Given a node $N \in \mathcal{N}$, its associated class $\mathbf{Mod}_S(N)$ of models (or N-models for short) consists of those Σ^N-models n for which*

- *n satisfies the local axioms Γ^N,*
- *for each $K \xrightarrow{\sigma} N \in \mathcal{S}$, $n|_\sigma$ is a K-model,*
- *for each $K \xrightarrow[h]{\sigma} N \in \mathcal{S}$, n has a σ-expansion k (i.e. $k|_\sigma = n$) which is a K-model.*

Complementary to definition and hiding links, which *define* the theories of related nodes, we introduce the notion of a *theorem link* with the help of which we are able to *postulate* relations between different theories. (Global) theorem links (denoted by $N \overset{\sigma}{\dashrightarrow} M$) are the central data structure to represent proof obligations arising in formal developments. The semantics of theorem links is given by the next definition.

Definition 4. *Let $\mathcal{S}$ be a development graph and N, M be nodes in $\mathcal{S}$. $\mathcal{S}$* **implies** *a global theorem link $N \overset{\sigma}{\dashrightarrow} M$ (denoted $\mathcal{S} \models N \overset{\sigma}{\dashrightarrow} M$) iff for all $m \in \mathbf{Mod}_S(M)$, $m|_\sigma \in \mathbf{Mod}_S(N)$.*

A sound and complete (relative to an oracle for proving conservative extension) set of proof rules for deriving entailments of form

$$\mathcal{S} \vdash N \overset{\sigma}{-\!\!\to} M$$

has been introduced in [13] (based on the assumption that the underlying logic is complete and two further technical assumptions, namely composable signatures and weak amalgamation).

Based on this, there is a development graph tool MAYA [4], keeping track of specifications, proof goals and proofs, and also supporting a management of

[1] There are also local links, which are omitted since they are not so essential here.

change. The tool is parameterized over a tool for the entailment system of the underlying logic, and only assumes the abstract properties of entailment systems given in their definition above.

E.g. consider the development graph of the running example (cf. Fig. 1): The theorem link from SYSTEM to BIT expresses the postulation that the latter is a refinement of the former. Note that this development graph is heterogeneous because it involves different logics – hence it goes beyond the above formalization of development graphs over *one* arbitrary but fixed logic. The main goal of this paper is hence to provide a formal basis for *heterogeneous development graphs*. But before coming to this, let us first examine how to use translations between logics to *prove* theorem links.

4 Borrowing

Often, for a given logic, there is no direct proof support available. Then, a way to obtain proof support is to *encode* the logic into another logic that has good tool support. For encoding logics, we use the notion of *institution representation*.

Definition 5. *Given institutions I and J, a* simple institution representation *[20] (also called* simple map of institutions *[11]) $\mu = (\Phi, \alpha, \beta): I \longrightarrow J$ consists of*

- *a functor $\Phi: \mathbf{Sign}^I \longrightarrow \mathbf{Pres}^J$*[2]*,*
- *a natural transformation $\alpha: \mathbf{Sen}^I \longrightarrow \mathbf{Sen}^J \circ \Phi$,*
- *a natural transformation $\beta: \mathbf{Mod}^J \circ \Phi^{op} \longrightarrow \mathbf{Mod}^I$*

such that the following representation condition *is satisfied for all $\Sigma \in \mathbf{Sign}^I$, $M' \in \mathbf{Mod}^J(\Phi(\Sigma))$ and $\varphi \in \mathbf{Sen}^I(\Sigma)$:*

$$M' \models^J_{Sig[\Phi(\Sigma)]} \alpha_\Sigma(\varphi) \Leftrightarrow \beta_\Sigma(M') \models^I_\Sigma \varphi.$$

In more detail, this means that each signature $\Sigma \in \mathbf{Sign}^I$ is translated to a presentation $\Phi(\Sigma) \in \mathbf{Pres}^J$, and each signature morphism $\sigma: \Sigma \longrightarrow \Sigma' \in \mathbf{Sign}^I$ is translated to a presentation morphism $\Phi(\sigma): \Phi(\Sigma) \longrightarrow \Phi(\Sigma') \in \mathbf{Pres}^J$. Moreover, for each signature $\Sigma \in \mathbf{Sign}^I$, we have a sentence translation map $\alpha_\Sigma: \mathbf{Sen}^I(\Sigma) \longrightarrow \mathbf{Sen}^J(\Phi(\Sigma))$ and a model translation functor $\beta_\Sigma : \mathbf{Mod}^J(\Phi(\Sigma)) \longrightarrow \mathbf{Mod}^I(\Sigma)$. Naturality of α and β means that for any signature morphism

[2] A *presentation* $P = \langle \Sigma, \Gamma \rangle \in \mathbf{Pres}$ consists of a signature Σ and a finite set of sentences $\Gamma \subseteq \mathbf{Sen}(\Sigma)$ (we set $Sig[P] = \Sigma$ and $Ax[P] = \Gamma$). *Presentation morphisms* are those signature morphisms that map axioms to logical consequences.

$\sigma\colon \Sigma \longrightarrow \Sigma' \in \mathbf{Sign}^I$,

$$\begin{array}{ccc}
\mathbf{Sen}^I(\Sigma) & \xrightarrow{\alpha_\Sigma} & \mathbf{Sen}^J(\Phi(\Sigma)) \\
\Big\downarrow{\scriptstyle \mathbf{Sen}^I(\sigma)} & & \Big\downarrow{\scriptstyle \mathbf{Sen}^J(\Phi(\sigma))} \\
\mathbf{Sen}^I(\Sigma') & \xrightarrow{\alpha_{\Sigma'}} & \mathbf{Sen}^J(\Phi(\Sigma'))
\end{array}$$

and

$$\begin{array}{ccc}
\mathbf{Mod}^I(\Sigma) & \xleftarrow{\beta_\Sigma} & \mathbf{Mod}^J(\Phi(\Sigma)) \\
\Big\uparrow{\scriptstyle \mathbf{Mod}^I(\sigma)} & & \Big\uparrow{\scriptstyle \mathbf{Mod}^J(\Phi(\sigma))} \\
\mathbf{Mod}^I(\Sigma') & \xleftarrow{\beta_{\Sigma'}} & \mathbf{Mod}^J(\Phi(\Sigma'))
\end{array}$$

commute.

Example 1. The logic of CASL (subsorted partial first-order logic with sort generation constraints) can be encoded with a simple institution representation into second-order logic [12]. This representation can be described as a composite of three representations: The first one encodes partiality via error elements living in a supersort, the second one encodes subsorting via injections, and the third one encodes sort generation constraints via second-order induction axioms. The details can be found in [12].

Definition 6. *Given a simple institution representation $\mu = (\Phi, \alpha, \beta)\colon I \longrightarrow J$, we can extend it to a translation $\hat{\mu}$ of development graphs with hiding as follows: Given a development graph $\mathcal{S}$, $\hat{\mu}(\mathcal{S})$ has the same structure as $\mathcal{S}$, but the signature of a node N is changed from Σ^N to $Sig[\Phi(\Sigma^N)]$, the set of local axioms is changed from Γ^N to $\alpha_{\Sigma^N}(\Gamma^N) \cup Ax[\Phi(\Sigma^N)]$. Moreover, a signature morphism σ occurring in a link (of any type) is replaced by $\Phi(\sigma)$. Note that $\Phi(\sigma)$ as used above is a morphism between presentations, but as such it also is a signature morphism.*

An important use of (simple) institution representations is the re-use (also called borrowing) of entailment systems along the lines of [8,6]. Therefore, we need two preparatory notions.

A simple institution representation $(\Phi, \alpha, \beta)\colon I \longrightarrow J$ *admits model expansion* if β is pointwise surjective on objects (i.e., each β_Σ is surjective on objects). Informally, this means that each model of the source institution has a model representing it in the target institution.

Let a class $\mathcal{D}$ of signature morphisms in I be given. An institution representation $\mu\colon I \longrightarrow J$ admits *weak $\mathcal{D}$-amalgamation*, if for signature morphisms

in $\mathcal{D}$, the β-naturality diagrams shown above are weak pullbacks. Informally, this means that each representation of a reduct of a model can be extended to a representation of the whole model (for reducts along signature morphisms in $\mathcal{D}$).

Theorem 1 (Local borrowing [8]). *Let $\mu = (\Phi, \alpha, \beta)\colon I \longrightarrow J$ be a simple institution representation admitting model expansion into a (complete) logic J. Then we can turn I into a (complete) logic by putting*

$$\Gamma \vdash_\Sigma \varphi \quad \textit{iff} \quad Ax[\Phi(\Sigma)] \cup \alpha_\Sigma[\Gamma] \vdash_{Sign(\Phi(\Sigma))} \alpha_\Sigma(\varphi).$$

Theorem 2 (Global borrowing). *Let $\mu = (\Phi, \alpha, \beta)\colon I \longrightarrow J$ be an institution representation admitting model expansion and weak $\mathcal{D}$-amalgamation, and let J be a complete logic. Then μ admits global borrowing, i.e. if we put*

$$\mathcal{S} \vdash M \xrightarrow{\sigma} N \quad \textit{iff} \quad \hat{\mu}(\mathcal{S}) \vdash M \xrightarrow{\Phi(\sigma)} N.$$

we get an entailment relation which is sound and complete (relative to an oracle for proving conservative extension) for a subset of development graphs over I (namely the set of all those development graphs with all hiding links along signature morphisms in $\mathcal{D}$).

Example 2. The institution representation from Example 1 admits model expansion and weak (injective)-amalgamation. Hence, it admits global borrowing for development graphs over the CASL logic that contain hiding links only along injective signature morphisms. That is, to prove a theorem link of such a development graph, it suffices to prove the translation of the link in the translated development graph over second-order logic.

5 Logic Morphisms and a CoFI Logic Graph

How can we give a precise semantics to the development graph in Fig. 1? As a prerequisite, we need to relate the underlying institutions somehow. In the previous section, we have introduced institution representations serving the purpose of *encoding* an institution within another one. But they are not so appropriate for dealing with heterogeneity (a motivation for this is given in [21]). Rather, we need *institution morphisms* [10], expressing the fact that a "larger" institution *is built upon* a "smaller" institution by *projecting* the "larger" institution onto the "smaller" one.

Given institutions I and J, an *institution morphism* [10] $\mu = (\Phi, \alpha, \beta)\colon I \longrightarrow J$ consists of

- a functor $\Phi\colon \mathbf{Sign}^I \longrightarrow \mathbf{Sign}^J$,
- a natural transformation $\alpha\colon \mathbf{Sen}^J \circ \Phi \longrightarrow \mathbf{Sen}^I$ and
- a natural transformation $\beta\colon \mathbf{Mod}^I \longrightarrow \mathbf{Mod}^J \circ \Phi^{op}$,

such that the following *satisfaction condition* is satisfied for all $\Sigma \in \mathbf{Sign}^I$, $M \in \mathbf{Mod}^I(\Sigma)$ and $\varphi' \in \mathbf{Sen}^J(\Phi(\Sigma))$:

$$M \models^I_\Sigma \alpha_\Sigma(\varphi') \Leftrightarrow \beta_\Sigma(M) \models^J_{\Phi(\Sigma)} \varphi'.$$

If I and J are also logics, an institution morphism $\mu: I \longrightarrow J$ is called a *logic morphism*, if for any $\Sigma \in \mathbf{Sign}^I$, and $\{\varphi\}, \Gamma \subseteq \mathbf{Sen}^J(\Phi(\Sigma))$,

$$\Gamma \vdash^J_{\Phi(\Sigma)} \varphi \text{ implies } \alpha_\Sigma[\Gamma] \vdash^I_\Sigma \alpha_\Sigma(\varphi).$$

Note that this condition always holds if I is complete.

This leads to a category $\underline{Log}$ of logics and logic morphisms.

As an example, consider the graph of logics and logic morphisms shown in Fig. 2. CASL extends first-order logic with partiality, subsorting and generation constraints (some form of induction). CASL-LTL [18] is an extension of CASL with a CTL-like labeled transition logic. LB-CASL [1] extends CASL with late binding, and SB-CASL [5] is an extension of CASL following the abstract state machine paradigm, where states correspond to algebras. HO-CASL [14] extends CASL with higher-order functions, and HasCASL [19] further adds shallow polymorphism and type constructors. (We here use the extensional variants ExtHO-CASL and ExtHasCASL of these logics, in order to be able to embed them into classical higher-order logic later on.) $FOL^=$ is the restriction of CASL to first-order logic, $SubPHorn^=$ [12] the restriction to Horn logic, and $Horn^=$ is the intersection of both restrictions. The definition of the logic morphisms is quite straightforward, except that for projecting CASL-LTL and LB-CASL onto CASL, we have two choices: since the dynamic structure can be represented in CASL itself, we either can choose to keep it or to drop it. Note that in Example 1 we have chosen the keep-morphism for going from CASL-LTL to CASL in order to be able to keep the labeled transition system also in SB-CASL and thus provide a true interaction between the two worlds.

6 Heterogeneity through Grothendieck Logics

With a given (arbitrary but fixed) graph of logics and morphisms as exhibited in the last section, we are now able to define heterogeneous development graphs. We could introduce new types of definition link to capture the heterogeneity, similarly to [21]. However, a more elegant way is to flatten the graph of logics, and then use the usual constructions for the thus obtained logic. This leads to the notion of Grothendieck logic, extending Diaconescu's Grothendieck institutions [9].

Definition 7. *An* indexed logic *is a functor* $\mathcal{L}: Ind^{op} \longrightarrow \underline{Log}$ *into the category of logics and logic morphisms.*

For example, the graph of logics from Fig. 2 can be easily considered to be an indexed logic. (Any graph of logics can be extended to an indexed logic by taking Ind^{op} to be the free category over the graph, basically consisting of paths.)

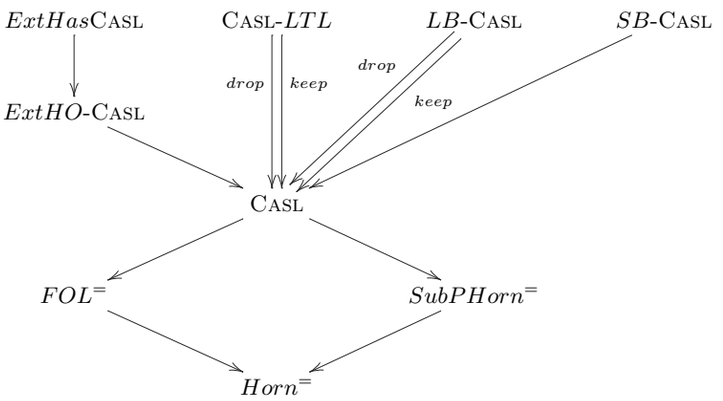

Fig. 2. A first CoFI logic graph

An indexed logic can now be flattened, using the so-called *Grothendieck construction*. The basic idea here is that all signatures of all logics are put side by side, and a signature morphism in this large realm of signatures consists of an intra-logic signature morphism plus an inter-logic translation (along some logic morphism from the graph). The other components are then defined in a straightforward way.

Definition 8. *Given an* indexed logic $\mathcal{L}\colon Ind^{op} \longrightarrow \underline{Log}$*, define the* Grothendieck logic $\mathcal{L}^{\#}$ *as follows:*

- *signatures in* $\mathcal{L}^{\#}$ *are pairs* (Σ, i)*, where* $i \in |Ind|$ *and* Σ *a signature in the logic* $\mathcal{L}(i)$*,*
- *signature morphisms* $(\sigma, d)\colon (\Sigma_1, i) \longrightarrow (\Sigma_2, j)$ *consist of a morphism* $d\colon i \longrightarrow j \in Ind$ *and a signature morphism* $\sigma\colon \Sigma_1 \longrightarrow \Phi^{\mathcal{L}(d)}(\Sigma_2)$ *(here,* $\mathcal{L}(d)\colon \mathcal{L}(j) \longrightarrow \mathcal{L}(i)$ *is the logic morphism corresponding to the arrow* $d\colon i \longrightarrow j$ *in the logic graph, and* $\Phi^{\mathcal{L}(d)}$ *is its signature translation component),*
- *the* (Σ, i)*-sentences are the* Σ*-sentences in* $\mathcal{L}(i)$*, and sentence translation along* (σ, d) *is the composition of sentence translation along* σ *with sentence translation along* $\mathcal{L}(d)$*,*
- *the* (Σ, i)*-models are the* Σ*-models in* $\mathcal{L}(i)$*, and model reduction along* (σ, d) *is the composition of model translation along* $\mathcal{L}(d)$ *with model reduction along* σ*, and*
- *satisfaction (resp. entailment) w.r.t.* (Σ, i) *is satisfaction (resp. entailment) w.r.t.* Σ *in* $\mathcal{L}(i)$*.*

Now we can just define heterogeneous development graphs over $\mathcal{L}$ to be usual development graphs over the logic $\mathcal{L}^{\#}$. Hence, the graph shown in Fig 1 now becomes a development graph in the formal sense. Proving in such heterogeneous development graphs is then heterogeneous proving: the goal of deriving a global theorem link is decomposed (using the proof rules from [13]) into local goals

that refer to the entailment relation of the Grothendieck logic, which in turn is defined in terms of the entailment relations of the individual logics.

However, there is one obstacle with this approach: in order to be able to use the calculus for the development graphs with hiding in a sound and relatively complete way, one needs weak amalgamation for the Grothendieck logic $\mathcal{L}^{\#}$ (note that with the calculus for structured specifications given in [6], one even needs Craig interpolation). Diaconescu [9] gives necessary and sufficient conditions for this. However, he points out that in many cases not all of these conditions will be satisfied (and this also is the case for our graph of logics). Therefore, we also will pursue a different way of obtaining proof support for heterogeneous logics.

7 Grothendieck Representations and Heterogeneous Borrowing

Often, heterogeneous proving in the Grothendieck logic is not feasible. One problem is a possible lack of weak amalgamation as indicated in the previous section. Another problem with a logic graph covering a variety of logics is that one needs to implement a proof tool for each individual logic. Therefore, we now show how to translate heterogeneous proof goals into homogeneous ones (i.e. over *one* logic), using heterogeneous borrowing.

To this end, we need the notion of institution representation map [20,16]. Fix a logic $U = (\mathbf{USign}, \mathbf{USen}, \mathbf{UMod}, \models, \vdash)$ which we will very informally view as a "universal" logic (with sufficient expressiveness to represent many logics, and with suitable tool support). We will also denote the institution $(\mathbf{USign}, \mathbf{USen}, \mathbf{UMod}, \models)$ by U.

Definition 9. *Let $I = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ and $I' = (\mathbf{Sign}', \mathbf{Sen}', \mathbf{Mod}', \models')$ be institutions and $\rho = (\Phi, \alpha, \beta)\colon I \to U$ and $\rho' = (\Phi', \alpha', \beta')\colon I' \to U$ be their representations in U. A* representation map *from ρ to ρ' consists of:*

- *an institution morphism $\mu = (\bar{\Phi}, \bar{\alpha}, \bar{\beta})\colon I \to I'$, and*
- *a natural transformation $\theta\colon \Phi' \circ \bar{\Phi} \to \Phi$,*

such that

- $\alpha \circ \bar{\alpha} = (\mathbf{USen} \cdot \theta) \circ (\alpha' \cdot \bar{\Phi})$,
- $\bar{\beta} \cdot \beta = (\beta' \cdot \bar{\Phi}^{op}) \circ (\mathbf{UMod} \cdot \theta^{op})$ *, i.e., that for each signature $\Sigma \in |\mathbf{Sign}|$ the following diagram commutes:*

$$\begin{array}{ccc} \mathbf{Mod}(\Sigma) & \xleftarrow{\beta_\Sigma} & \mathbf{UMod}(\Phi(\Sigma)) \\ {\scriptstyle \bar{\beta}_\Sigma}\downarrow & & \downarrow{\scriptstyle \mathbf{UMod}(\theta_\Sigma)} \\ \mathbf{Mod}'(\bar{\Phi}(\Sigma)) & \xleftarrow{\beta'_{\bar{\Phi}(\Sigma)}} & \mathbf{UMod}(\Phi'(\bar{\Phi}(\Sigma))) \end{array}$$

Moreover, we say that (μ, θ) admits weak amalgamation*, if for each signature $\Sigma \in |\mathbf{Sign}|$, the above diagram is a weak pullback in $\mathcal{CAT}$.*

With an obvious composition, this gives us a category $Repr(U)$ of institution representations into U and representation maps. An indexed representation *then is just a functor $\mathcal{R}: Ind^{op} \longrightarrow Repr(U)$. We now have:*

Theorem 3. *Given an indexed institution representation $\mathcal{R}: Ind^{op} \longrightarrow Repr(U)$, we can form its* Grothendieck representation *$\mathcal{R}^{\#}: (\Pi_1 \circ \mathcal{R})^{\#} \longrightarrow U$, which is a representation of the Grothendieck institution of the indexed institution $\Pi_1 \circ \mathcal{R}$ formed from the source institutions and morphisms involved in $\mathcal{R}$.*

Proposition 1. *Given an indexed representation $\mathcal{R}: Ind^{op} \longrightarrow Repr(U)$, if all the individual representations $\mathcal{R}(i)$ admit model expansion, then also $\mathcal{R}^{\#}$ admits model expansion.*

Definition 10. *Given an indexed representation $\mathcal{R}: Ind^{op} \longrightarrow Repr(U)$ and a class of signature morphisms $\mathcal{D}$ in $(\Pi_1 \circ \mathcal{R})^{\#}$, $\mathcal{R}$ is said to* admit weak $\mathcal{D}$-amalgamation *if*

- *for each $i \in Ind$, $\mathcal{R}(i)$ admits weak $\mathcal{C}$-amalgamation, where $\mathcal{C} = \{\sigma \mid (\sigma, d) \in \mathcal{D}$ for some $d: i \longrightarrow j \in Ind\}$, and*
- *for each $d \in Ind$ such that $(\sigma, d) \in \mathcal{D}$ for some σ, $\mathcal{R}(d)$ admits weak amalgamation.*

Proposition 2. *Given an indexed representation $\mathcal{R}: Ind^{op} \longrightarrow Repr(U)$ and a class of signature morphisms $\mathcal{D}$ in $(\Pi_1 \circ \mathcal{R})^{\#}$, if $\mathcal{R}$ admits weak $\mathcal{D}$-amalgamation, then also $\mathcal{R}^{\#}$ admits weak $\mathcal{D}$-amalgamation.*

Corollary 1. *Given an indexed representation $\mathcal{R}: Ind^{op} \longrightarrow Repr(U)$, if all the individual representations $\mathcal{R}(i)$ admit model expansion, and moreover $\mathcal{R}$ admits weak $\mathcal{D}$-amalgamation, then $\mathcal{R}^{\#}$ admits global borrowing for development graphs containing hiding links only along signature morphisms in $\mathcal{D}$.*

This means that global theorem links in heterogeneous development graphs can be derived using only the entailment relation of U (and the proof rules for development graphs from [13]).

Theorem 4. *The underlying indexed institution from Fig. 2 can be extended to an indexed representation $\mathcal{COFI}: Ind^{op} \longrightarrow Repr(HOL)$, as indicated in Fig. 3. Moreover, if $\mathcal{INJ}$ is the set of all signature morphisms that are injective in the first (intra-institution) component, then $\mathcal{COFI}$ admits weak $\mathcal{INJ}$-amalgamation.*

Proof. (Sketch) CASL is represented in HOL by Example 1. CASL-LTL and LB-CASL are represented in CASL by construction, hence we can compose this with the representation of CASL in HOL. ExtHO-CASL can be represented in HOL by an easy extension of the representation of CASL in HOL (for representing

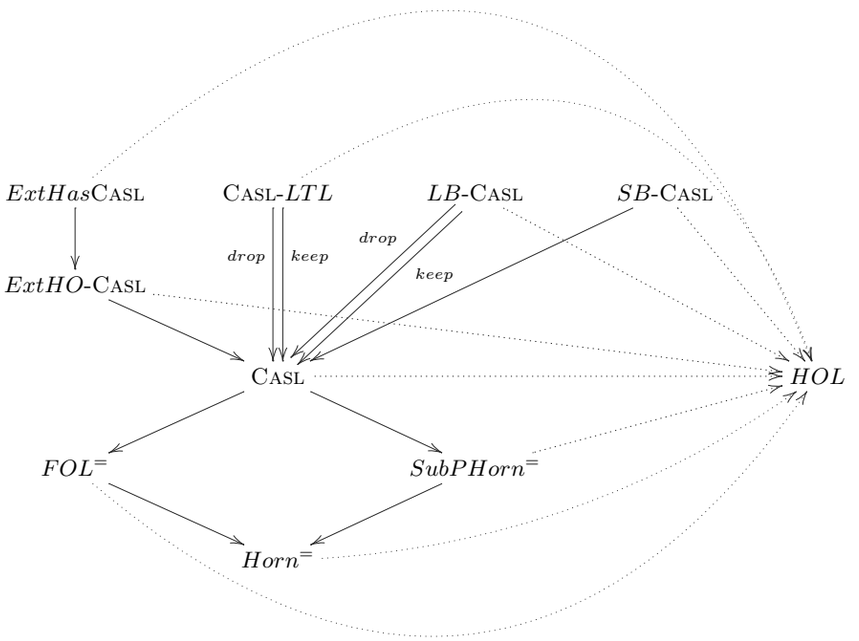

Fig. 3. The embedding to the CoFI logic graph into HOL

ExtHasCASL, we need a variant of HOL supporting type constructors and shallow polymorphism). The most complicated representation is that of SB-CASL: we have to represent the static part as with the standard CASL representation in HOL in order to get a representation map, while we need a set-theoretic representation for dynamic part (which involves whole algebras as states). The link between the static and the dynamic part is done by lifting functions which lift the static part into set theory. □

By Corollary 1, $\mathcal{COFI}^{\#}$ admits global borrowing for development graphs containing hiding links only along signature morphisms in $\mathcal{INJ}$.

8 Conclusion and Related Work

Multi-logic systems can be studied in the context of an arbitrary but fixed graph of logics and logic morphisms (formalized as an indexed logic). In such a setting, we have generalized development graphs with hiding [13] to the heterogeneous case, using the Grothendieck construction of Diaconescu [9]. We then have extended the Grothendieck construction form institutions (and morphisms) to institution representations (and representation maps). We also have studied conditions under which Grothendieck representations admit the re-use ("borrowing") of theorem provers for proving global theorem links in heterogeneous

development graphs. Our work is related to the introduction of heterogeneous notions in [21], which here occur more naturally through the Grothendieck construction.

As a first application, we have sketched a graph of institutions consisting of institutions for the Common Algebraic Specification Language CASL and some of its extensions and sublanguages. We also have extended this to a graph of representations and representation maps in higher-order logic. By the construction of the associated Grothendieck representation, any theorem prover for higher-order logic, together with the development graph tool based on the calculus for development graphs with hiding introduced in [13], can be used for theorem proving in the heterogeneous language over the above mentioned institution graph. As a first practical step, the development graph tool MAYA has been connected with the Isabelle/HOL theorem prover [4]. Other provers will follow, with the possibility of multi-prover proofs. We also have made the static analysis of CASL in-the-large institution independent [15], which is a step towards an analysis tool for the heterogeneous input language.

We have also presented a sample heterogeneous specification involving a refinement of an abstract requirement specification in CASL-LTL using temporal logic to a design specification in SB-CASL following paradigm of abstract state machines. With this, we have indicated that heterogeneous development graphs are indeed useful for dealing with interaction of different formalisms. It should be noted though that this interaction is possible because we use an institution morphism form CASL-LTL to CASL that keeps the dynamic structure. In order to have an interaction also in cases where it is not possible to keep some structure, one has to use parchments [16,7], which allow a true "interleaved" feature interaction. However, the theory of parchments is not ripe yet to deal with a combination of all the logics in our sample institution graph.

In the future, one should, of course, further explore the applicability of this approach. Concerning tool support, we have presented two extremes: either each logic comes with individual proof support, or all logics are encoded into one "universal" logic like HOL. Possibly it will be desirable to find some way between these extremes by allowing a large variety of input logics which is then mapped into a smaller variety of proof logics (for example, MAYA already supports both HOL and HOL plus TLA). The details of how to construct a logic representation between the induced Grothendieck logics need to be worked out yet. Another line of future work concerns the application of the ideas presented here to *programming* languages, which can also be considered to be institutions [20].

Acknowledgments

This work came out of both a cooperation with Andrzej Tarlecki on the semantic aspects and with Serge Autexier and Dieter Hutter on the proof theoretic aspects of specifications. This work has been supported by the *Deutsche Forschungsgemeinschaft* under Grant KR 1191/5-1.

References

1. D. Ancona, M. Cerioli, and E. Zucca. Extending CASL by late binding. In C. Choppy, D. Bert, and P. Mosses, editors, *Recent Trends in Algebraic Development Techniques, 14th International Workshop, WADT'99, Bonas, France*, volume 1827 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.
2. E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner. *Algebraic Foundations of Systems Specification*. Springer, 1999.
3. S. Autexier, D. Hutter, H. Mantel, and A. Schairer. Towards an evolutionary formal software-development using CASL. In C. Choppy and D. Bert, editors, *Recent Trends in Algebraic Development Techniques, 14th International Workshop, WADT'99, Bonas, France*, volume 1827 of *Lecture Notes in Computer Science*, pages 73–88. Springer-Verlag, 2000.
4. S. Autexier and T. Mossakowski. Integrating HOL-CASL into the development graph manager MAYA. FroCoS 2002, to appear.
5. H. Baumeister and A. Zamulin. State-based extension of CASL. In *Proceedings IFM 2000*, volume 1945 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.
6. T. Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, to appear.
7. C.Caleiro, P.Mateus, J.Ramos, and A.Sernadas. Combining logics: Parchments revisited. In M. Cerioli and G. Reggio, editors, *Recent Trends in Algebraic Development Techniques, 15th International Workshop, WADT'01, Genova, Italy*, Lecture Notes in Computer Science. Springer-Verlag. To appear.
8. M. Cerioli and J. Meseguer. May I borrow your logic? (transporting logical structures along maps). *Theoretical Computer Science*, 173:311–347, 1997.
9. R. Diaconescu. Grothendieck institutions. *Applied categorical structures*. to appear.
10. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. Predecessor in: LNCS 164, 221–256, 1984.
11. J. Meseguer. General logics. In *Logic Colloquium 87*, pages 275–329. North Holland, 1989.
12. T. Mossakowski. Relating CASL with other specification languages: the institution level. *Theoretical Computer Science*. To appear.
13. T. Mossakowski, S. Autexier, and D. Hutter. Extending development graphs with hiding. In H. Hußmann, editor, *Fundamental Approaches to Software Engineering*, volume 2029 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 2001.
14. T. Mossakowski, A. Haxthausen, and B. Krieg-Brückner. Subsorted partial higher-order logic as an extension of CASL. In C. Choppy, D. Bert, and P. Mosses, editors, *Recent Trends in Algebraic Development Techniques, 14th International Workshop, WADT'99, Bonas, France*, volume 1827 of *Lecture Notes in Computer Science*, pages 126–145. Springer-Verlag, 2000.
15. T. Mossakowski and B. Klin. Institution independent static analysis for CASL. In M. Ceriolo and G. Reggio, editors, *Recent Trends in Algebraic Development Techniques, 15th International Workshop, WADT'01, Genova, Italy*, Lecture Notes in Computer Science. Springer-Verlag. To appear.
16. T. Mossakowski, A. Tarlecki, and W. Pawłowski. Combining and representing logical systems using model-theoretic parchments. In F. Parisi Presicce, editor,

Recent trends in algebraic development techniques. Proc. 12th International Workshop, volume 1376 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 1998.

17. P. D. Mosses. CoFI: The Common Framework Initiative for Algebraic Specification and Development. In *TAPSOFT '97, Proc. Intl. Symp. on Theory and Practice of Software Development*, volume 1214 of *LNCS*, pages 115–137. Springer-Verlag, 1997.
18. G. Reggio and L. Repetto. CASL-CHART: a combination of statecharts and of the algebraic specification language CASL. In *Proc. AMAST 2000*, volume 1816 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.
19. L. Schröder and T. Mossakowski. HasCASL: Towards integrated specification and development of Haskell programs. Submitted.
20. A. Tarlecki. Moving between logical systems. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specifications. 11th Workshop on Specification of Abstract Data Types*, volume 1130 of *Lecture Notes in Computer Science*, pages 478–502. Springer Verlag, 1996.
21. A. Tarlecki. Towards heterogeneous specifications. In D. Gabbay and M. d. Rijke, editors, *Frontiers of Combining Systems 2, 1998*, Studies in Logic and Computation, pages 337–360. Research Studies Press, 2000.