Interpreting Heterogeneous Theory Graphs in a Logical Framework

Florian Rabe

Jacobs University Bremen

Abstract

Theory graphs provide a formalism for developing and relating structured specifications over a logic. In the heterogeneous case, different logics can be used in the same specification if appropriate logic translations are given. But checking the syntactic and semantic requirements necessary for the correctness of such a graph is non-trivial. We extend OMDoc to use it as a generic representation language for heterogeneous theory graphs and describe how to interpret such a graph in the logical framework LF.

Keywords: logical framework, theory graph, OMDoc, heterogeneous specification

1 Introduction and Related Work

The logical framwork LF ([HHP93]) and its implementation Twelf ([PS99]) have been used extensively to encode logics (e.g., [HST94,Pfe00,PSK+06]). However, applications of logic encodings in LF suffer from the lack of high-level services for LF such as a module system, a scalable library architecture, library-wide structural search, management of change, and visual presentation tools.

Providing these services is the strength of the knowledge management language OMDoc ([Koh06]). It is an XML language for mathematical documents incorporating OpenMath ([BCC+]) to represent mathematical objects. Like LF, it provides logic-independent concrete syntax to represent the theories of a logic and the translations within a logic and between logics. But contrary to LF, it ignores type-checking issues and focuses on the above-mentioned advanced management services, which to a large extent are independent of whether the theories they operate on are type-correct. Its module system gives a powerful but simple standardized syntax; every document can be referenced by any other document using URIs; and OMDoc can be easily parsed and generated on all platforms.

The basic concept to represent theories in OMDoc is a theory graph, whose formal semantics is given by the model-theoretic framework of institutions ([GB92]). A theory over an institution is a pair (Σ, Δ) where Σ is a signature and Δ is a set

of sentences over Σ. For example, in the theory of monoids in the institution for first-order logic, Σ would consist of the symbols *unit* and *comp* and their arities, and Δ would contain the axioms for associativity and neutral element. A theory morphism from (Σ_1, Δ_1) to (Σ_2, Δ_2) is a signature morphism, which we can think of here as a mapping from the syntax over Σ_1 to the syntax over Σ_2, such that the axioms in Δ_1 are mapped to theorems of the theory Δ_2.

Development graphs ([MAH06],[AHMS02]) permit the hierarchic structured representation of complex theories via acyclic multi-graphs. In a development graph over an institution, every node is labeled with a theory (Σ, Δ), and every edge from $T_1 = (\Sigma_1, \Delta_1)$ to $T_2 = (\Sigma_2, \Delta_2)$ is labeled with a definitional theory morphism $\sigma : T_1 \to T_2$. If σ is the only edge of the graph, this means that the node T_2 represents the theory $(\Sigma_2, \Delta_2 \cup \sigma(\Delta_1)$, where $\sigma(\Delta_1)$ is the element-wise translation of the axioms of T_1. More generally, every node represents the theory arising by recursively importing all axioms from all incoming edges.

Development graphs may also have postulated edges, which represent theory morphisms that have to be established by proving the consequence requirements. One strength of development graphs is that postulated edges between complex theories may be decomposed into simpler edges that can be proved efficiently by calling an automated prover for the underlying institution. In heterogeneous development graphs ([Mos00],[MMLW06]), the nodes are triples $(\mathbb{I}, \Sigma, \Delta)$ where $\mathbb{I}$ is an institution, and the edges from $(\mathbb{I}_1, \Sigma_1, \Delta_1)$ to $(\mathbb{I}_2, \Sigma_2, \Delta_2)$ are pairs (μ, σ) where μ is an institution translation from $\mathbb{I}_1$ to $\mathbb{I}_2$, and σ is a theory morphism in $\mathbb{I}_2$ from $(\mu(\Sigma_1), \mu(\Delta_1))$ to (Σ_2, Δ_2).

However, currently, signatures, signature morphisms, and institution translations have to be implemented ad hoc for every institution. In [Rab07], we formalized how LF can be used as a meta-language in which these can be specified. Our current motivation is to use OMDoc as a high-level interface language mediating between tools maintaining development graphs, Twelf, and other services. Here, as a first step, we describe how certain OMDoc theory graphs can be interpreted in LF so that LF type-checking validates a theory graph.

2 Theory Graphs in OMDoc

To make the notation more compact, we will condense some of the XML markup of OMDoc. A *theory graph* is an OMDoc document consisting of theory and theory-inclusion elements. A *theory* T has the form

```
<theory xml:id="T" meta="M">
  symbols, definitions, axioms, imports
</theory>
```

where the optional M refers to another theory, called the *meta-theory* of T. A *logic* L is a theory that does not declare axioms and additionally singles out a valid object over L, which we denote by L^*. Objects represent terms, types, etc.; they are defined below.

A *symbol* is an element with an attribute giving its name and a child giving its *type*; the type must be a valid object over T. A *definition* has an attribute referring to a symbol of T and a child giving the definiens for that symbol as a valid object over T. An *axiom* has an attribute giving its name, an attribute giving a theory morphism into T, and a child giving the axiom's formula as a valid object over the source of that theory morphism.

An *import imp* from the theory S (into T) has the following form

```
<imports xml:id="imp" from="S" meta="m">
  <morphism>[O ↦ O']*</morphism>
</imports>
```

where m is a theory morphism (defined below) from the meta-theory of S into the meta-theory of T, called the meta-morphism of *imp*. (m is omitted if S has no meta-theory.) $[O \mapsto O']^*$ abbreviates markup specifying that *imp* maps some elementary objects O of S to valid objects O' over T. Allowed as O are symbols, imported symbols, and theory morphisms. The intuition is that symbols O in the domain of the map are instantiated with their images O' whereas the unmapped symbols of S are imported into T as new symbols. We require that O' is a theory morphism iff O is; and if so, that they have the same domain and that the codomain of O' is T. If *imp* maps an import imp' into S to some import into T, it defines maps for all symbols imported via imp', which represents an (asymmetric) sharing specification.

A *theory-inclusion inc* from S to S' is of the form

```
<theory-inclusion xml:id="inc" from="S" to=S' meta="m">
  <morphism>[O ↦ O']*</morphism>
  obligations
</theory-inclusion>
```

where m is as for imports. The morphism child of a theory-inclusion element must be as for an imports element except that it must map every (local or imported) symbol of S. A *logic translation* is a theory-inclusion between logics that additionally singles out a valid object over S', which we denote by inc^*. We will explain the obligations below.

Then it remains to define what *valid objects* and *theory morphisms* are.

- If n is a local symbol of T, then <OMS cd="T" name="n"/> is valid over T.
- For a name v, the variable <OMV name="v"/> is valid over T.
- If O is valid over the meta-theory of T, it is valid over T.
- If O and $O_1, \ldots, O_n$ are valid over T, then so is the application of O to $O_1, \ldots, O_n$ written <OMA>O O_1 ... O_n</OMA>.
- If O_1, O_2, O_3 are valid over T, then so is the binding with binder O_1, variable type O_2, and expression O_3 under the binder. (We omit the XML syntax.)
- If O is valid over S and <OMM via="*via*"/> is a theory morphism from S to T, then <OMM via="*via*">O</OMM> is valid over T.
- <OMM via=""/> is a theory morphism from any theory into itself.

- If <OMM via="*via*"/> is a theory morphism from R into S, and m refers to an imports or a theory-inclusion from S into T, then <OMM via="*via m*"/> is a theory morphism from R into T.

Except during this inductive definition, we require that no variable occurs freely in a valid object. We will use *via* to abbreviate <OMM via="*via*"/>.

Thus an OMDoc theory is a list of typed, possibly defined symbols, axioms, and imports of symbols and axioms from other theories. And every theory morphism *via* from S to T induces a mapping which maps a valid object O over S to the valid object <OMM via="*via*">*O*</OMM> over T. This map is the composition of the natural extensions of the maps specified by the imports or theory-inclusion elements in *via*. In particular, if an import *imp* imports a symbol s into T, T can refer to this imported symbol as <OMM via="*imp*">s</OMM>. And <OMM via="*via*"><OMM via="*via′*"/></OMM> represents the composition $via \circ via'$.

imports elements correspond to definitional theory morphisms of a development graph, but they are more general because they permit to import not only axioms but also symbols. A theory-inclusion *inc* corresponds to a postulated theory morphism: For every axiom with formula A of the source theory, the target theory must have a proof of <OMM via="*inc*">A</OMM>, to which the *obligation* elements refer. We do not give the full OMDoc syntax for *proofs* and instead give a proof step as an example:

```
<derive xml:id="id">
  <method><OMOBJ>R</OMOBJ></method>
  parameters, premises
  <OMOBJ>F</OMOBJ>
</derive>
```

A derive element represents a node of a proof tree over a theory T. F is the formula at the node, R is the proof rule used to establish F, typically a symbol in the meta-language of T, parameters are objects over T with which R is instantiated, and the premise elements refer to the children of the node. The leaves are given by references to axioms of T or to *hypotheses* introduced in the proof. The latter permits OMDoc to support hypothetical reasoning as in natural deduction proofs. Furthermore, OMDoc supports incomplete proofs by so-called *gap* steps.

Finally, the graph induced by the references to theories, symbols, imports, and theory-inclusions must be acyclic. Without loss of generality, we require the stronger condition that all these references go against the order in which the elements occur in the OMDoc document. (Note that this does not exclude mutual recursion: Only the types of two symbols may not be mutually recursive, their definitions can be.)

It is not easy to define directly which maps are allowed in an imports or theory-inclusion element: We need to expand all imports elements recursively to find out what the symbols that can (or in the case of theory-inclusion: must) be mapped are. Furthermore, since images may be specified both for an imported symbol and for the imports importing it, there may be conflicting images. But these requirements can be expressed in a straightforward but technical way on the OMDoc level.

However, the following questions relevant for the correctness of a theory graph cannot be answered on the OMDoc level: Which objects are eligible as the type and definiens of a symbol, or as axioms? Which definiens-type combinations are permitted? Which imports and theory-inclusion elements preserve types? (This depends on subtle typing conditions such as: If a sort a is mapped to a', a constant of sort a must be mapped to a term of sort a'.) When can an imports imp map a theory morphism m to m'? (This depends on even subtler conditions regarding the equality of imp composed with m and m' for certain arguments.) In the next section, we define and limit attention to a special case for which LF can be used to give the answers.

3 Interpreting a Theory Graph in Twelf

We restrict theory graphs as follows: Firstly, every theory graph starts with a theory $twelf$ declaring the (untyped) symbols `type`, λ, Π, as well as symbols for the implicit versions of the binders. And secondly, all logics must have $twelf$ and all other theories must have a logic as their meta-theory. And imports or theory-inclusion elements between logics must be logic translations with the identity of $twelf$ as their meta-morphism.

Intuitively, the logics declare, e.g., connectives, quantifiers, and proof rules, essentially mirroring the syntax used for Twelf logic encodings. And the other theories declare, e.g., sorts, function and predicate symbols, and axioms using both the symbols of Twelf and of their logic. For example, assume that we have a theory PL for propositional logic:

```
<theory xml:id="PL" meta="twelf">
  <symbol name="o">
    <type><OMOBJ><OMS cd="twelf" name="type"/></OMOBJ></type>
  </symbol>  <!-- other symbols omitted -->
</theory>
```

Here a symbol o is declared for the type of formulas, its OMDoc type is given in the syntax of the meta-language Twelf. Similarly, the connectives and proof rules can be defined. A theory FOL for first-order logic could start with a declaration <imports xml:id="$pl-fol$" from="PL"/> and would refer to the imported type of formulas as <OMM via="$pl-fol$"><OMS cd="PL" name="o"/></OMM>. Similarly, a theory ML for modal logic could be defined. Then a theory for first-order modal logic would look like this:

```
<theory xml:id="FOML" meta="twelf">
  <imports xml:id="fol-foml" from="FOL"/>
  <imports xml:id="ml-foml" from="ML">
    <morphism><OMM via="pl-ml/> ↦ <OMM via="pl-fol fol-foml"/></morphism>
  </imports>
</theory>
```

The object L^* of a logic L is interpreted as a type family over a certain type t, intuitively the truth judgment over formulas. Only objects type-checking against t may be axioms in theories having L as their meta-theory. The need for inc^* in a logic translation inc is less obvious and detailed in [Rab07]. Assume we want to

represent as a logic translation $ml-fol$ from ML to FOL the standard relational translation from modal to first-order logic by making worlds explicit ([SH06]). This will be a theory-inclusion specifying maps for all local and imported symbols of ML; in particular, the Twelf type <OMM via="$pl-ml$"><OMS cd="PL" name="o"/></OMM> of ML-formulas is translated to the Twelf type of FOL-formulas in one free variable (where the free variable expresses the world-dependency of truth). It is easy to see that this translation can be defined as a syntax-directed recursive function and thus as a theory morphism, e.g., $\Box F \mapsto \forall x\ (acc(x,y) \to F'[y])$ if $F \mapsto F'[x]$. But in order to make statements about the preservation of the semantics, a one-time top-level step is needed going from $F'[x]$ to $\forall x F'[x]$; because of this top-level step, the translation as a whole cannot be represented by a theory morphism. Therefore, inc^* is used to express this last step as a λ term.

To interpret a theory graph in Twelf, we will process its elements in document order according to the following algorithm. This will generate a sequence of Twelf declarations for every theory and every theory-inclusion. We will write $\overline{O}$ for the OMDoc object O printed in Twelf syntax as defined below. And we will use the character –, assuming it does not occur in identifiers in the theory graph, to generate fresh unique Twelf identifiers.

Firstly, the theory $twelf$ is ignored. Secondly, for a theory T, process in document order all its symbols and imports children as follows:

- For a symbol with name n, type T, and (if present) definiens D, generate $T–n : \overline{T} = \overline{D}$.
- For an imports imp from S, do the following:

(i) For every Twelf declaration $m_r–\ldots–m_1–R–n : \overline{T} = \overline{D}$ generated by interpreting S (including the cases where there is no D), compute the map of $m_r–\ldots m_1–R–n$ by imp by

(a) If imp specifies a map <OMM via="$m_1\ \ldots\ m_r$"><OMS cd="R" name="n"/></OMM>$\mapsto$ O', put $M := O'$ and go to Step (ii). Otherwise put $i = 1$.

(b) If imp specifies a map <OMM via="$m_i\ \ldots\ m_r$"/> $\mapsto$ <OMM via="via"/>, put $M :=$ <OMM via="$m_1\ \ldots\ m_{i-1}\ via$"><OMS cd="$R$" name="$n$"/></OMM> and go to Step (ii). Otherwise, if $i = r$, go to Step (ii); otherwise, increment i and repeat this step.

(ii) If M was defined, generate

· $imp–m_r–\ldots–m_1–T–n : \overline{imp(T)} = \overline{M}$. Here $imp(O)$ abbreviates the object <OMM via="imp">O</OMM>.

· And if D was present, also generate declarations that succeed iff $\overline{imp(D)}$ is definitionally equal to $\overline{M}$. (It is actually not obvious how to express definitional equality in Twelf, but it is possible.)

(iii) If M was not defined, generate $imp–m_r–\ldots m_1–T–n : id(T) = id(D)$ (and accordingly if D was not present).

- For every axiom of T with name n, theory morphism $m_1\ \ldots\ m_r$, and formula A, generate the declaration $T–m : \overline{L^*}\ \overline{t_n(\ldots(t_1(A))\ldots)}$. Here L is the meta-theory of T, μ_i is the meta-morphism of m_i, and $t_i(O)$ is <OMA>$\mu_i^*\ m_i(O)$</OMA>.

And finally, process a theory-inclusion inc from S to S' as follows. For every

declaration of a Twelf symbol generated when processing S except for those stemming from axioms, generate a Twelf declaration in the same way as in step 3 above with *inc* instead of *imp*. And for those declarations $c : A$ stemming from an axiom, generate a Twelf declaration $inc{-}c : \overline{A'} = P$ where μ is the meta-morphism of *inc*, $A' =$<OMA>μ^* *inc*(A)</OMA> and P is the rendering of the proof referred to by the corresponding obligation child of *inc* (defined below).

It remains to define how to print an OMDoc object O in Twelf syntax. This is straightforward except that we have to eliminate OMM elements. In a first step, all OMM elements are pushed to the inside of O while merging consecutive OMM elements. Then references to imports elements are replaced with their meta-morphism if they are applied to a symbol of the meta-theory. And then a reference to a theory-inclusion elements is eliminated by replacing a symbol with its map. This is repeated until O has been transformed to an object which only contains OMM elements of the form <OMM via="imp_1 ... imp_r"><OMS cd="T" name="n"/></OMM> such that all imp_i refer to imports elements and the source of imp_1 is T; then a Twelf declaration for the name $imp_m - \ldots imp_1 - T - n$ exists. Thus all OMDoc objects can be printed.

The printing of proofs as Twelf terms is done by first transforming them to an OMDoc object which is then printed as above. This is straightforward since the OMDoc proof structure already corresponds to the syntax tree of a proof term: Every proof step is transformed to an OMA element that applies the proof rule to the list of parameters and the objects generated by transforming the premises. Only gap steps, hypotheses, and implicit arguments require some attention. If a gap step is encountered, a proof rule filling the gap is added before generating the declaration that checks the proof; and this new proof rule is shadowed by some irrelevant declaration immediately afterwards so that the additional proof rule is not used anywhere else. If a proof declares local hypotheses, we wrap it in the corresponding λ binders. And if a proof rule R takes n implicit arguments, its OMDoc type must start with n binders using the symbol for the implicit Π; this type is inspected to obtain n, and then the first n parameters of every proof step using R are omitted.

We say that a theory T of a theory graph is correct if the following Twelf signature type-checks: The signature formed by taking the result of processing the meta-theory of T and appending the result of processing T itself. And we say that a theory-inclusion *inc* from S to T of a theory graph is correct if the following Twelf signature type-checks: The signature formed by joining the result of processing the meta-theory of T, the result of processing T, and the result of processing *inc*. Then we have the following result: A heterogeneous theory graph is correct if every theory and every theory-inclusion in it is correct.

4 Conclusion and Future Work

We have described the current state of ongoing work to establish and tighten the connection between high-level structured specification languages and the logical

framework LF by using OMDoc as an interface language and Twelf as a type-checking backend. This combination also opens the door to add the advanced services of OMDoc on top of Twelf. For example, the conservative LF module system designed by Watkins and Pfenning (unpublished work) is a special case of the OMDoc module system because a modular LF signature can be transformed naturally into an OMDoc theory graph, which can then be interpreted in non-modular LF. It is also promising to use OMDoc as a library backend for Twelf: The library would store logics, theories, and translations and produce the corresponding Twelf files transparently upon request. This would permit to build the Logosphere project ([PSK+06]) on top of a powerful database.

We are currently implementing the interpretation. The interpretation of proofs and theory-inclusions is still missing, but there are no problems left, and and we expect the first release version within weeks. Our future work will consist of specifying the logics and translations used in tools like Hets in OMDoc so that they can generate Twelf files via OMDoc. For heterogeneous proving, it will be crucial to add proof steps to OMDoc consisting of translating the proof goal into a different logic. This does not present a major problem.

References

[AHMS02] S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The Development Graph Manager Maya (System Description). In H. Kirchner and C. Ringeissen, editors, *Algebraic Methods and Software Technology, 9th International Conference*, pages 495–502. Springer, 2002.

[BCC+] S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaëtano, and M. Kohlhase. The openmath standard. See http://www.openmath.org/cocoon/openmath/standard/.

[GB92] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[HST94] R. Harper, D. Sannella, and A. Tarlecki. Structured presentations and logic representations. *Annals of Pure and Applied Logic*, 67:113–160, 1994.

[Koh06] M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer, 2006.

[MAH06] T. Mossakowski, S. Autexier, and D. Hutter. Development graphs - Proof management for structured specifications. *J. Log. Algebr. Program*, 67(1–2):114–145, 2006.

[MMLW06] T. Mossakowski, C. Maeder, K. Lüttich, and S. Wölfl. The Heterogeneous Tool Set, 2006. To appear.

[Mos00] T. Mossakowski. Heterogeneous development graphs and heterogeneous borrowing. In M. Nielsen and U. Engberg, editors, *Foundations of Software Science and Computation Structures*, pages 326–341. Springer, 2000.

[Pfe00] F. Pfenning. Structural cut elimination: I. intuitionistic and classical logic. *Information and Computation*, 157(1-2):84–141, 2000.

[PS99] F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. *Lecture Notes in Computer Science*, 1632:202–206, 1999.

[PSK+06] F. Pfenning, C. Schürmann, M. Kohlhase, N. Shankar, and S. Owre. The Logosphere Project, 2006. See http://www.logosphere.org/.

[Rab07] F. Rabe. Institutions with Proofs and their Representation in a Logical Framework. Submitted, see http://kwarc.eecs.iu-bremen.de/frabe/Research/rabe_instlf_07.pdf, 2007.

[SH06] R. Schmidt and U. Hustadt. First-Order Resolution Methods for Modal Logics, 2006. To appear in Volume in memoriam of Harald Ganzinger.