Development of Structured Ontologies in CASL

Klaus Lüttich

Kumulative Dissertation
zur Erlangung des Grades eines Doktors der Ingenieurwissenschaften
– Dr.-Ing. –

Vorgelegt im Fachbereich 3 (Mathematik und Informatik),
Universität Bremen

3. Mai 2007

Abstract

Ontologies are differentiated in *foundational* and *domain* ontologies. As both kinds of ontologies are developed under different considerations such as expressivity or efficiency, usually different formal languages are used. Foundational ontologies are typically specified in first-order logic or modal logic, since precise definitions are considered more important than efficiency. Since domain ontologies are applied to large knowledge bases, efficiency is considered the most important factor. This initiated intensive investigation into description logics, a less expressive logical formalisms aiming at tractable and efficient reasoning.

In the cumulative thesis at hand the algebraic specification language CASL is studied for its applicability as an ontology specification language. Both kinds of ontologies, foundational and domain ontologies, are considered. For facilitating the development of ontologies the notions of *strongly typed* and *structured ontologies* are introduced and elaborated by using CASL as an example. CASL is extended through the support of description logics. This enables the investigation of the relation between first-order logic and description logic within structured CASL. Additionally, a knowledge compilation method — preserving basic properties — for the approximation of first-order logic theories with description logic theories is described, that allows for the development of efficiently usable domain ontologies based on foundational ontologies.

Moreover, the Heterogeneous Tool Set (HETS), a computer program for the analysis of CASL specifications, is described and extended to be applied as an ontology development tool. Particularly, the connection of automated theorem proving systems to HETS is described by the author, thus easing potentially tedious work on proofs.

The applicability of CASL as ontology specification language is demonstrated by providing various examples in the context of spatial cognition. In particular, the development methods of strongly typed and structured ontologies are applied to the analysis of the WWW navigation metaphor and the representation of route knowledge.

Zusammenfassung

Ontologien werden unterschieden in *grundlegende* und *anwendungsspezifische* Ontologien. Da diese beiden Arten von Ontologien unter verschiedenen Gesichtspunkten wie z.B. Ausdrucksmächtigkeit oder Effizienz entwickelt werden, kommen in der Regel unterschiedliche formale Sprachen zum Einsatz. Die grundlegenden Ontologien werden vorwiegend mittels Prädikatenlogik erster Stufe oder Modallogik spezifiziert, wobei mehr Wert auf die präzise Definition als auf Effizienz gelegt wird. Da anwendungsspezifische Ontologien häufig mit großen Wissensbasen eingesetzt werden, gilt Effizienz als der wichtigste Faktor. Dies initiierte intensive Forschung an Beschreibungslogiken (ausdrucksschwache Formalismen) mit dem Ziel, berechenbares und effizientes Schließen zu ermöglichen.

In der vorliegenden kumulativen Dissertation wird die algebraische Spezifikationssprache CASL auf ihre Einsatzfähigkeit als Spezifikationssprache für Ontologien untersucht. Hierbei werden grundlegende und anwendungsspezifische Ontologien betrachtet. Um die Entwicklung von Ontologien zu erleichtern, werden die Begriffe *streng getypte* und *strukturierte* Ontologie eingeführt und am Beispiel von CASL verdeutlicht. CASL wird um die Unterstützung von Beschreibungslogiken erweitert. Dadurch läßt sich in strukturiertem CASL das Verhältnis von Ontologien in Prädikatenlogik erster Stufe und Beschreibungslogik zueinander untersuchen. Zusätzlich wird eine approximierende Übersetzung von Prädikatenlogik in Beschreibungslogik, welche grundlegende Eigenschaften bewahrt, dargestellt, um die Entwicklung von effizient einsetzbaren Anwendungsontologien aus grundlegenden Ontologien zu ermöglichen.

Weiterhin wird das Heterogeneous Tool Set (HETS), ein Computerprogramm zur Analyse von CASL-Spezifikationen, beschrieben und für den Einsatz als Ontologie-Entwicklungswerkzeug erweitert. Insbesondere die Anbindung von automatischen Theorembeweissystemen wird vom Autor beschrieben. Die Anbindung von automatischen Beweissystemen dient der Erleichterung von langwierigen Beweisen.

Die Einsatzfähigkeit von CASL als Spezifikationssprache für Ontologien wird an Hand von mehreren Beispielen aus dem Kontext Raumkognition demonstriert. Hierbei werden insbesondere die Entwicklungsmethoden für streng getypte und strukturierte Ontologien eingesetzt, um die WWW-Navigationsmetapher zu analysieren und um Routenwissen zu repräsentieren.

Contents

List of Figures

List of Tables

1 Introduction

Formalised ontologies are applied to achieve the interoperability and integration of different information systems. These systems are often developed collaboratively, which involves the negotiation of shared knowledge, which is captured by an ontology. Husserl introduced a distinction between *formal* and *material* ontology (cf. [63]). Foundational and general "laws" applicable to different domains are captured by formal ontology, whereas material ontologies deal with specific kinds of entities. This led to the dichotomy of *foundational* and *domain* ontologies [49]. Foundational ontologies describe knowledge at an abstract level by defining general concepts (such as event, object, or quality) and relations (such as parthood, constitution, or integrity), whereas domain ontologies define material knowledge in detail, such as the structure of a particular organisation or company.

Since domain ontologies emerging from different sources most likely have very different approaches in defining their ontologies — alignment or integration may be needed if such ontologies are to be combined for a broader coverage. The knowledge described by foundational ontologies is often too abstract for a direct application and is supposed to be refined for different domains. The term *application* ontology is also used for domain ontologies, since these are more often directly applied than foundational ontologies.

These two kinds of ontologies are defined in logical formalisms with rather different characteristics in terms of expressiveness, tool support, or computational efficiency. Foundational ontologies focus on precise definitions using expressive logics such as first-order logic (FOL) or modal logic. However, tool support and computational aspects play only a minor role in the field of foundational ontology. The opposite approach is taken when domain ontologies are specified. Since they are intended for computational applications dealing with large knowledge bases in the first place, they are specified in logical formalisms such as description logics (DL), of which the computational aspects have been studied intensively. These studies allowed the development of efficient reasoning tools capable of dealing with large knowledge bases.

Up to now, the concepts of foundational and domain ontologies have been studied independently of each other, since no commonly used logical language has been established for the definition of foundational ontologies. Additionally, the availability of tools dealing with FOL and DL languages simultaneously is very limited. Thus the relation between foundational and domain ontologies was only studied informally. Since both kinds of ontologies are specified in logical languages, the term logical theory is also used for ontologies throughout this thesis.

Various logical languages have been introduced for the specification of ontologies. Most of them suffer from certain defects such as no formal support for the combination of independently developed theories, or the lack of types for the relations which are enforced early enough for the detection of unintended application of the relations. The need for modularisation of ontologies emerged in the first Workshop on Modular Ontologies (WoMO 2006)[1], which brought together ontology developers with very different backgrounds contributing rather orthogonal aspects of ontology modularisation. Most ontology specifications include some kind of typing for relations, usually formulated as axioms; thus the type information of relations is only used at the level of reasoning, but is not available for type checking, that might be performed prior to any reasoning.

The Common Algebraic Specification Language (CASL) provides a many-sorted FOL with powerful, logic independent structuring constructs. It is proposed by this thesis as an ontology specification language. CASL is suitable for both kinds of ontologies, since the possibility of extensions towards other logical systems has been a design principle of the CASL language family. Moreover, the Heterogeneous Tool Set (HETS) provides formal analysis functions for CASL at the structured and at the logical level, which enables computationally supported theorem proving within foundational ontologies and the establishment of formal relations between foundational and domain ontologies given in different logical formalisms.

[1]Hosted by the International Semantic Web Conference (ISWC 2006).

Structure of This Thesis. Section 2 discusses methodological considerations by presenting Casl, introducing the notion of strongly typed ontology, and elaborating the structuring facilities of Casl for the specification of ontologies. Section 3 focuses on the relation of ontologies written in FOL and DL: a Casl extension towards DL enables formal analysis of domain ontologies within Casl, and knowledge approximation allows the transformation of precise definitions given in FOL by foundational ontologies into computationally efficient DL theories. Section 4 introduces Hets, which provides various analysis functions for Casl, and the integration of automated theorem proving (ATP) systems into Hets. This helps the ontology developer, working with Casl and Hets, to carry out the simple proofs using ATP systems, while she can restrict interactive proofs within semi-automated theorem proving systems to interesting and hard theorems. Section 5 presents various ontological specifications within the context of spatial cognition and spatial representation, where the concepts of the preceding sections are applied to the development of ontologies. Furthermore, Section 6.1 on page 44 provides "Bibliographical Remarks" that detail the contributions by the author of this thesis to collaboratively authored publications. Publications by the author are marked differently from other bibliographical references, by e.g. [KL-1].

2 Methodology: How to Develop Ontologies in Casl

Contributions to the methodology of developing formal ontologies are made in [KL-8, KL-7, KL-1]. The author has introduced the advantages of algebraic specification with Casl for the development of formal ontologies by presentations at the conference Formal Ontology in Information Systems (FOIS). The benefits of powerful structuring facilities for easier reuse and easier understanding of large ontologies are demonstrated. Moreover, the notion of strongly typed ontologies is established. This methodology is developed in collaboration with LOA, Trento, Italy and is also applied to the development of the foundational ontology Dolce.

2.1 Origin of Casl

In 1995 the Common Framework Initiative (CoFI) [55] started a unification process for algebraic specification languages and formal software development. Its result is the language Casl (the Common Algebraic Specification Language) which was subsequently approved by IFIP[2] WG 1.3 (Foundations of System Specification) and serves as a de-facto standard for algebraic specification. The aim of CoFI has been to develop an algebraic specification framework for common usage within the already collaborating algebraic specification community, since many diverse approaches have been developed using different languages and tools. The goals for this framework included the specification of functional requirements using formal development simultaneously focusing on libraries, reuse, evolution and tool interoperability.

The concrete and abstract syntax and formal semantics of Casl, including also libraries of basic datatypes and relations, have been completely described in the Reference Manual [10] and illustrated in the User Manual [5]. Casl features a critical selection of known syntactical constructs from previous algebraic specification contexts. The syntax is expressive, simple and pragmatic for specifying many-sorted first-order logic (FOL) theories, and it resembles mathematical notation. The restriction to less expressive sublanguages and the extension, towards e.g. higher-order logic and modal logic, have been a central point in the discussion of the language design as a family of languages. Casl subsumes the efforts of various other algebraic specification approaches regarding modularisation of software and specification of abstract requirements combined with a relatively straightforward semantics.

Casl uses the category theory notions of institution and comorphism [25] to formally grasp and relate sublogics of Casl's many-sorted first-order logic language and its logical extensions. A basic specification is defined by a signature and a set of sentences over this signature. A signature is basically a declaration of sorts, predicates and (total or partial) functions, including type profiles of predicates and functions. A sentence is either an axiom or a theorem. The semantics of a basic specification is given by a class of models satisfying the set axioms.

The structuring facilities of Casl are independent of the underlying logic and can be used for arbitrary logics formalised in terms of institutions. The following section discusses Casl's basic specification constructs in the light of ontology specification, and Section 2.3 deals with the structuring constructs of Casl.

2.2 Strongly Typed Ontologies

Commonly used ontology languages such as KIF [22] or plain mathematical LaTeX formulas as used for Dolce [49] do not provide explicit typing of predicates. Thus argument restriction axioms are used instead. If a predicate is used apart from its intended usage, a theorem prover is needed to discover such errors. OWL [12] provides explicit argument restrictions for properties, but as they are not required, they are treated as axioms only, which are not checked statically. All approaches to ontology formalisation so far allow the introduction

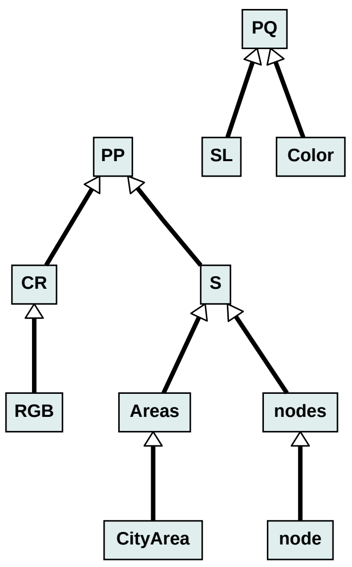
[2]The International Federation for Information Processing (`http://www.ifip.org`)

of new symbols at any point, which makes it hard to detect mistyped symbols. The OWL community tries to circumvent these defects by providing graphical ontology engineering tools like Protégé [37], which allow only the usage of declared symbols within axioms, but argument restrictions are still not enforced and are only checked by connected DL reasoners. The developers of DOLCE extensively used LaTeX-macros in order to avoid mistyping. However, there is no tool support for checking or proving formulas written in LaTeX.

CASL encourages the usage of many-sorted FOL with subsorts, where each symbol has to be declared explicitly before it is used in axioms. Unlike plain FOL (e.g. as available in KIF [22]) every relation and function has a type profile, which is statically checked when a relation or function is applied to its arguments. Sorts are used instead of unary predicates for dividing the domain of interest. This has two advantages: (1) Sorts that are not in subsort relation are disjoint with respect to usage in quantification;[3] (2) sentences are easier to read, if the types of variables are introduced during quantification instead of writing implications with possibly long premises in front of each formula. A minor drawback — from the point of view of ontology engineering — is that the carrier set of a sort is non-empty in CASL and hence sorts cannot be used for concepts like *Nothing* in OWL [12]. Consequently, unary predicates are used to model concepts which might have no extension.

```
spec TypedRelations =
     sorts CR, S < PP;
           RGB < CR;
           Areas, nodes < S;
           node < nodes;
           CityArea < Areas;
           Color, SL < PQ
     preds pos : RGB × Color;
           pos : node × SL;
           pos : CityArea × SL
     ∀ x : SL; y, y' : node; z, z' : CityArea
     • pos(z, x) ∧ pos(z', x) ⇒ z = z'        %(correct 1)%
     • pos(y, x) ∧ pos(y', x) ⇒ y = y'        %(correct 2)%
then pred pos : S × SL
     ∀ x : SL • ∃ a : S • pos(a, x);          %(correct 3)%
     ∀ y : SL; z, z' : node
     • pos(y, z) ∧ pos(z', y) ⇒ z = z'      %(no typing 1)%
     ∀ x : RGB; y, y' : SL
     • pos(y, x) ∧ pos(y', x) ⇒ y = y'      %(no typing 2)%
```

(a)



(b)

Figure 1: A strongly typed ontology

Figure 1 models the functional relation *pos* that provides the positions of physical qualities (subsorts of sort *PQ*) in some quality spaces (subsorts of sort *PP*). For each spatial location (*SL*) at least one position in a spatial quality space (suborts of *S*) must be provided %(correct 3)%. The general idea is to model representations for different information via qualities which are attached to the objects (which are not specified in TypedRelations). In fact this is an almost unstructured extract of the QualitySpace specification as instantiated by ApplicationQualities in [KL-1, Section 5].

Symbols of predicates and functions might be used with different type profiles. They are in overloading relation if the sorts of the profiles are in subsort relation, otherwise such symbols stand for different relations or functions. To illustrate the power of strong typing Figure 1a shows the declaration of two subsort hierarchies (graphs shown in Figure 1b) and the declaration of the predicate *pos* with different profiles. In fact only two relations are specified: (1) $pos : RGB \times Color$; (2) $pos : S \times SL$. However, in the first part of the

[3]In the models of the theory the carrier sets of sorts might overlap.

specification two profiles for *pos* are declared with sort *SL* and they are used to specify that each element in *SL* can only be related to exactly one element in sort *node* and one in *CityArea*, respectively, (cf. axioms: %(correct 1)% and %(correct 2)%). After the "**then**" the predicate *pos* is overloaded with respect to *S* and its transitive subsorts *node* and *CityArea*; axiom %(correct 3)% applies via embedding into the supersort *S*, also to sorts *node* and *CityArea*. Formula %(no typing 1)% is ill typed because the typing of arguments in predicate application (predication) $pos(y, z)$ is wrong. Although sorts *RGB* and *S* have a common supersort *PP*, there is no type profile in the specification which would allow a typing of the predications in the last formula. Hence any overloading must be introduced explicitly.

Another advantage of strongly typed ontologies is a reduced search space for theorem proving systems which use a typed logic, because all the clumsy premises with unary predicate assertions are "hidden" into quantifiers which introduce typed variables. For the translation of CASL into logics of automated theorem proving systems see Section 4.2. Although a strongly typed ontology is mathematically equivalent to an untyped version, the typed version contains more information.

Other concepts built into CASL are *generated* and *freely generated* sorts (denoted as generated and free type) which are used in the context of software specification for the precise definition of datatypes. This is achieved by defining constructor functions and/or a partition into subsorts, where the generatedness restricts the data values to those generated by constructor terms or steming from the declared subsorts; freeness adds pairwise disjointness between the ranges of the constructor functions and/or the declared subsorts and injectivity of the constructors. In the context of ontology development these constructs are useful for the compact definition of concept partitions with possibly overlapping or disjoint carrier sets. Furthermore, concepts where all individuals are known and form a set of pairwise distinct elements can be concisely defined with a free type. Although it is also possible to define such a finite disjoint partition just with first-order axioms, the use of freely generated sets is less error-prone and more readable if these axioms are hidden into a type definition and only introduced by tools for proofs. The complete definition of a class by enumerating individuals is also possible in OWL DL; however, the distinctness of the individuals has to be specified separately.

2.3 Structuring of Ontologies

At the Workshop on Modular Ontologies (WoMO 2006)[4] the author has presented an internationally reviewed paper in collaboration with Claudio Masolo and Stefano Borgo [KL-7]. Aim of the workshop was to bring together various approaches to modular ontologies. Since 2003 we collaboratively develop DOLCE as a structured ontology in CASL. [KL-7] has been the first one reporting the benefits of CASL's structuring facilities for the development of formal ontologies, where an example much more light-weight than DOLCE has been introduced. Also [KL-1] emphases the benefits of structured ontologies especially for spatial cognition applications like wayfinding.

The development of ontologies evolved since the emergence of the Semantic Web towards a new discipline: *ontology engineering*. Furthermore, the topic of reuse and sharing of ontologies has led to the discussion of *modular* ontologies following similar trends in software engineering and other areas. Modularisation techniques are introduced to enhance the reusability, understandability and scalability, to aid the maintenance of ontologies, and to improve reasoning with large ontologies. Ontologies are treated as logical theories throughout this document, hence the term theory is used interchangeably with ontology. In the field of ontologies the definition of the term module is itself problematic. Upcoming questions concern e.g. conceptual relevance, integrity and independence of a logical theory. [35] considers modules as theories which formalise specific domains completely or serve specific purposes. But modules can also be understood as general theories which can be composed, specialised and used in different contexts and domains. In the area of knowledge engineering

[4]Hosted by the International Semantic Web Conference (ISWC 2006).

such modules are called components [16, 9]. These differing views of modules led also to the distinction of *material* and *formal* ontology, where the former applies to an ontology formalising a domain completely. The formal ontology approach is taken here, because CASL offers techniques for the combination and specialisation of general theories. Of course it is also possible to completely specify a domain in CASL, but the reuse of such modules is very limited, because they are much more tailored to one specific task or purpose. A discussion of principles regarding how many symbol declarations and/or axioms are needed to call a logical theory a module are beyond the scope of this thesis.

Typical languages used for ontologies provide only a very limited set of import mechanisms; the Knowledge Interchange Format (KIF) [21] even lacks any import mechanism at all. The authors of e.g. SUMO[5] introduced a modularisation via comments, which proves the need for modularisation of ontologies. OWL [12] offers simple import mechanisms, even with cyclic imports, where all imported symbols are always visible within the importing ontology. It is not possible to form the union of two ontologies with an identification of common symbols, because all symbols defined in one OWL ontology carry fixed, globally unique names. The import facility of OWL acts much like a textual expansion of the imported ontology. It is only possible to define, via axioms, that certain concepts are equivalent or that some individuals are equal, but all symbols are still present.

In contrast to the structuring mechanism found in OWL, CASL [10] provides powerful structuring facilities. Each file organises a library of named specifications. Specifications are imported by name from other (predefined) libraries. A named specification is composed by basic specifications or imported named specifications. A basic specification consists of a declaration of symbols and a list of FOL sentences. If a basic specification extends a previous specification (cf. **then** in Figure 1a), the signature part may be empty, because it is inherited from the specification being extended.

```
spec PreOrder =
     sort  Elem
     pred  __≤__ : Elem × Elem
     ∀ x, y, z : Elem
     • x ≤ x                                          %(refl)%
     • x ≤ z if x ≤ y ∧ y ≤ z                         %(trans)%

spec PartialOrder =
     PreOrder
then ∀ x, y : Elem • x = y if x ≤ y ∧ y ≤ x           %(antisym)%
```

Figure 2: Extension of a specification

Within the definition of a named specification, new specifications are composed by extension and union, and in both cases symbols common to both specifications are identified. Figure 2 shows such a named specification PARTIALORDER extending PREORDER with an antisymmetry axiom (specifications taken from library CITYEXAMPLE [KL-1]). Renaming is used to instantiate a specification for different sorts or to give some predicate a more suitable name. For example, parthood known from Mereology [49, KL-1] has the same axioms as PARTIALORDER and hence is formed by importing PARTIALORDER and renaming predicate __≤__ into P as presented in Figure 3b. Mereology is closely related to Cohn's Region Connection Calculus [60, 2] and represents "wholes" in terms of their parts (parthood), e.g. atomic parts, extensionality, sum, and difference are typical axioms and definitions. In DOLCE [49] a mereological theory is used for e.g. physical endurants (objects in time), space and time, but without providing any relation to a concrete representation, whereas [74] discusses a concrete representation of spatial regions with a parthood relation by open discs in the Euclidian plane (interpreted as metric space).

[5]The Suggested Upper Merged Ontology [56] developed in SUO-KIF (a simplified version of KIF).

Argument Restrictions
(Ad1) $\mathsf{P}(x,y) \rightarrow (AB(x) \vee PD(x)) \wedge (AB(y) \vee PD(y))$
(Ad2) $\mathsf{P}(x,y) \rightarrow (PD(x) \leftrightarrow PD(y))$
(Ad3) $\mathsf{P}(x,y) \rightarrow (AB(x) \leftrightarrow AB(y))$
Ground Axioms
(Ad5) $(AB(x) \vee PD(x)) \rightarrow \mathsf{P}(x,x)$
(Ad6) $\mathsf{P}(x,y) \wedge \mathsf{P}(y,x) \rightarrow x = y$
(Ad7) $(\mathsf{P}(x,y) \wedge \mathsf{P}(y,z) \rightarrow \mathsf{P}(x,z)$

(a)

```
spec PARTHOOD =
     PARTIALORDER with __≤__ ↦ P
```

(b)

```
     ...
     PARTHOOD with s ↦ AB
and  PARTHOOD with s ↦ PD
     ...
```

(c)

Figure 3: Parthood of *AB* and *PD* in untyped FOL and structured CASL

The axioms (Ad1) to (Ad7) from [49] shown in Figure 3a are represented by the importing of the specification PARTHOOD twice with renaming. The argument restriction axioms (Ad1) to (Ad3) correspond to the declaration of the sorts *AB* and *PD* and of the predicate *P*. The ground axioms are given for each sort. Such a specification style helps to define strongly typed ontologies, although the concepts of strongly typed and structured ontologies are orthogonal and need not to be combined. Named specifications are reused and symbols are renamed to fit the needs of the new specification by means of the structuring language of CASL instead of error-prone copy-and-paste techniques. Additionally, a specification built from smaller, carefully crafted specifications is likely to have less errors than a specification, where all the basics of i.e. relations or set theory are repeated over and over. Recently, Barry Smith also pointed out the importance of reuse and consideration of previous work by contrasting ISO Standard 15926 with commonly known mathematical and ontological principles [64].

```
spec QUALITYSPACE[sorts s, q] =
     pred pos : s × q                                          %(A6)%
     ∀ x, x′ : s; y : q • pos(x, y) ∧ pos(x′, y) ⇒ x = x′      %(A7-A8)%

spec APPLICATIONQUALITIES =
     ...
and  QUALITYSPACE [sorts node[sub], SL fit q ↦ SL, s ↦ node[sub]]
and  QUALITYSPACE [sorts node[city], SL fit q ↦ SL, s ↦ node[city]]
and  QUALITYSPACE [sorts CityArea[Building], SL
                   fit q ↦ SL, s ↦ CityArea[Building]]
and  QUALITYSPACE [sorts CityArea[Area], SL
                   fit q ↦ SL, s ↦ CityArea[Area]]
then ...
```

Figure 4: Generic specification and instantiation example

Generic specifications allow an even more succinct reuse of specifications than importing with renaming. A generic specification takes specifications as parameters. These specifications might be very simple, such as declarations of symbols or specifications with axioms as in a specification of ordered lists of elements with a total order relation. The import of a generic specification is called instantiation and often involves an explicit fitting symbol map which associates symbols of the formal parameter specification to symbols of the actual parameter specification used in the instantiation. Figure 4 shows a generic specification of quality spaces for different representations of space (example taken from [KL-1]) where a fitting symbol map is used. Note that by declaring the sort *SL* in all formal parameters and taking the union of these four instantiations, the resulting theory has only one sort *SL* thanks to the "same name same thing"-principle. However, the four type profiles of *pos* are not in overloading relation, because the four sorts used as spaces are not related via subsort relations.

The instantiation of a generic specification, requires a proof that the actual parameter fits the argument specification. In the case of ordered lists, an instantiation of such lists with integers yields a proof obligation that the specification of integers provides a total order relation. In the case of simple parameter specifications as in Figure 4 these proof obligations can be and are actually discharged automatically, since only the compatibility of the signatures has to be checked (for tool support see also Section 4 and [KL-11]). *Views* are named theory morphisms showing entailment relations between specifications explicitly. A definition of a view is shown in Figure 5 (the referenced specification PartialOrder is presented in Figure 2). A view provides a signature morphism which maps the symbols in the first specification to symbols in the second specification. For the accompanied proof all sentences of the first specification are translated along the signature morphism and serve as theorems entailed by the second specification. Hence the model theoretic semantics of the view in Figure 5 is

$$\mathbf{Mod}(\{\textsc{Connection}\ \mathbf{then} \ldots\}) \subseteq \mathbf{Mod}(\sigma(\textsc{PartialOrder})).$$

In order to show this relation of the models the entailment below needs to be proved:

$$\{\textsc{Connection}\ \mathbf{then} \ldots\} \models \sigma(\textsc{PartialOrder})$$

Here σ is the signature morphism between the two specifications given by the symbol map of the view. More specificaly, the view Connection_induces_PartialOrder yields proof obligations which show that the relation defined by axiom %(<= def)% actually is a partial-order relation (see [8, KL-1] for details).

Views can be used instead of specifications for the actual parameter of an instantiation, e.g. a view showing that a total order relation is entailed by an integer specification might be used as actual parameter for the instantiation of ordered lists with integers. However, the signature morphism introduced by a view must exactly match the signatures of the formal and actual parameter specifications, because a further renaming of symbols when using a view is not possible. Also, views are useful for just stating the relation of specifications e.g. showing an approximation (see Section 3.3) or refinement without usage of the view for instantiations.

```
spec Connection =
     sort  Elem
     pred  __connected__ : Elem × Elem
     ∀ r, s : Elem
     • r connected r                                          %(refl_connected)%
     • s connected r ⇒ r connected s                           %(sym_connected)%
     • s = r ⇔ ∀ z : Elem • z connected r ⇔ z connected s      %(equal regions)%

view Connection_induces_PartialOrder :
     PartialOrder to
     {Connection
      then pred  __≤__(x, y : Elem) ⇔
                 ∀ z : Elem • z connected x ⇒ z connected y        %(<= def)%
     } =
     Elem ↦ Elem, __≤__ ↦ __≤__
```

Figure 5: Connection induces partial order

Semantic annotations allow the distinction of theorems among the sentences of a theory and indicate the kind of an extension. Theorems are either marked individually with

"**%implied**", or a whole extension is indicated to consist of theorems by "**then %implies**". The latter annotation disallows signature declarations. Section 4.2 discusses proving of such theorems with tool support. An import of a theory with implied sentences provides these sentences as additional axioms. Furthermore, annotations can mark an extension as conservative (**%cons**), monomorphic (**%mono**), or definitional (**%def**), where the indicating annotation always follows immediately after the keyword **then**. The ordering of the annotations listed above is by increasing strength, and an extension marked with **%implies** is strictly stronger than a definitional extension (see [10, pp. 110–111, 209] for more details). These semantic annotations are provided to express the intended kind of the extension, and a tool like HETS [KL-11] generates appropriate proof obligations for checking this intention with a suitable theorem prover.

In the field of ontology modularisation, conservative extensions have recently been discussed in the context of description logics (DL) such as $\mathcal{ALCQI}$ [45] and $\mathcal{SHOIQ}$ (related to OWL DL) [27], where syntactical and logical detection algorithms are discussed.

2.4 Summary

The field of ontology development is brought forward by the powerful and expressive language CASL developed with extensibility in mind. The original purpose of CASL has been algebraic software specification — the author of this thesis shows its applicability to ontology development. CASL provides many-sorted first-order logic paired with logic independent, powerful structuring constructs. CASL allows the specification of abstract requirements that is particularly useful for knowledge representation where the representation of concrete data is often less important than the relations among abstract individuals modelled as constant functions. At the same time, CASL is not restricted to description logics like OWL DL but offers the full expressivity of FOL with a concrete syntax designed towards commonly known mathematical notation. Hence neither a specialised tool is needed to write a CASL specification nor does the user have to adopt a hard to remember and hard to read "prefix language" like KIF. Despite the readable syntax, non-trivial tool support is available for parsing and manipulation of CASL specifications (see Section 4).

While rewriting DOLCE [49] as a structured ontology in CASL, the generation of predicate profiles and associated axioms has been used extensively. Additionally, alternative specifications of e.g. Mereology [8] have been provided which can be interchanged for different DOLCE variants sharing the same signature. A simplification and revision of DOLCE's treatment of physical qualities has been published in [KL-1]; some condensed extract is illustrated in Figure 4.

3 Relation of Ontologies in FOL and DL

Ontologies are mostly applied in the form of Description Logic (DL) in the context of the Semantic Web [4, 17], and the Web Ontology Language (OWL) [12] has been recommended as the standard interchange format. OWL DL is the most prominent variant of OWL and it combines good computational behaviour with a relative high degree of expressiveness. Furthermore, OWL DL aims at knowledge bases with thousands of facts. In the light of these discussions, OWL DL has been chosen for the connection of CASL with DL. On the one hand the author has developed a restricted version of CASL tailored towards OWL called CASL-DL [KL-10], and on the other hand an algorithm approximating FOL theories as DL theories has been introduced [KL-4, KL-5].

3.1 CASL-DL — A Description Logic CASL Extension

Providing a direct formal connection of CASL via CASL-DL to OWL DL has several major advantages:

- many ontologies for the Semantic Web will be written in OWL DL and a translation of OWL DL into CASL makes these ontologies accessible to ontologies written in CASL and CASL-DL;
- for DLs like OWL DL specialised efficient reasoners exist which are exploitable via translations of CASL sublanguages into CASL-DL;
- the integration and alignment of ontologies in DL via rich foundational ontologies in FOL are formally expressible in CASL with translations of OWL DL into CASL.

Furthermore, given that efficient reasoners exist for description logics like $\mathcal{SHIQ}$ or $\mathcal{SHIN}$ (fragments of OWL DL and CASL-DL), these description logics provide a reasonable target language for approximations of first-order logic ontologies. This allows us to benefit from ontologies defined by elaborated axioms like DOLCE within the context of the Semantic Web and other application scenarios where short answering times play an important role, which cannot be provided by general purpose FOL theorem proving systems. Another possible application scenario for CASL-DL is a validation of domain ontologies as approximations, where an OWL DL ontology is considered as approximation of a foundational ontology written in CASL, with a proof of the entailment shown as a view in Figure 6 (see Section 3.3 for more details on approximations).

```
view VALIDATION_OF_OWL_DL_ONTOLOGY :
     OWL_DL_ONTOLOGY to FOL_ONTOLOGY
```

Figure 6: A view representing a validation of an OWL DL ontology as approximation of a FOL ontology

The Web Ontology Language (OWL) has been established as a W3C recommendation in 2004 [12, 59] and provides three sublanguages presented in decreasing expressiveness:

OWL full provides unlimited usage of symbols like RDF Schema [7] and corresponds to some untyped higher-order logic;

OWL DL corresponds to $\mathcal{SHOIN}$(D), some fragment of FOL; it reaches the limits of DLs in terms of efficient reasoning and expressivity;

OWL Lite limits the DL further, it is equivalent to $\mathcal{SHIF}$(D) where constructs like nominals are disallowed for improving efficiency.

$\mathcal{SHOIN}$(D)	OWL DL	CASL-DL
concept	class	sort or unary predicate
subconcept	subclass	subsort
number restriction	cardinality	special predicate equivalent to an instantiation of a generic specification
role	property	binary predicate
role inclusion	subproperty	implication
individual	individual	constant operation
$\top$	class Thing	maximal super sort *Thing*
$\bot$	class Nothing	empty unary predicate *Nothing*
datatype	XML datatype	predefined datatypes

Table 1: Correspondences between $\mathcal{SHOIN}$(D), CASL-DL and OWL DL

The variant OWL DL has been chosen as a starting point for the development of CASL-DL, since OWL full has a very complicated semantics which lacks any tool support.

In a description logic knowledge is represented by a concept hierarchy with a maximal concept subsuming all other concepts. Binary roles are used to relate individuals and to define the coverage of concepts. Only a limited set of axioms is possible for defining characteristics of roles. Datatypes are predefined and not subject to axiomatisation beyond relations to individuals.

$\mathcal{SHOIN}$(D), equivalent to OWL DL [33], provides a reasonable number of logical constructs for dealing with ontologies aiming at efficient reasoning. Although OWL DL and ordinary DLs [1] use different terminologies for their constructs, the underlying logic is the same. Table 1 provides an overview of the terms used for some constructs in the different languages presented in this section. The combination of letters forming a DL language name immediately gives rise to the expressiveness, e.g. the letter 'D' denotes the availability of predefined datatypes like numbers and strings for the object position or filler of roles (i.e. the second position of a binary predicate). The detailed correspondences of the other letters to DL constructs has been laid out in [KL-10] following the notation presented in [1, pp. 494–495]. Still, Table 2 gives a short overview of which DL constructs are abbreviated by which letter, where the DL constructs are shown in Figure 7 and differentiated from FOL below.

Compared to first-order logic, a description logic like $\mathcal{SHOIN}$(D) allows only a limited form of quantification, where exactly two variables are simultaneously available and they are connected via a role application; e.g. the schematic formula $\phi(x) \equiv \forall y \bullet R(x,y) \Rightarrow \psi(y)$ shows

$\mathcal{S}$	extends $\mathcal{ALC}$ by transitive roles
$\mathcal{H}$	role hierarchies
$\mathcal{O}$	nominals
$\mathcal{I}$	inverse roles
$\mathcal{N}$	unqualified number restrictions
$\mathcal{Q}$	qualified number restrictions
$\mathcal{F}$	functional roles (only number restriction $\leqslant 1R$)
$\mathcal{AL}$	atomic concepts; top and bottom concept; intersection and value restriction of arbitrary concepts; existential quantification limited to top concept
$\mathcal{C}$	negation and union of arbitrary concepts
D	availability of datatypes

Table 2: DL letters and corresponding constructs

$$\begin{array}{rlll}
C, D ::= & A & \text{(atomic concept)} \\
\mid & \top & \text{(universal concept)} \\
\mid & \bot & \text{(bottom concept)} \\
\mid & \neg A & \text{(atomic negation)} \\
\mid & C \sqcap D & \text{(intersection)} \\
\mid & C \sqcup D & \text{(union)} \\
\mid & \{o_1, ...\} & \text{(nominals)} \\
\mid & \forall R.C & \text{(value restriction)} \\
\mid & \exists R.C & \text{(existential quantification)} \\
\mid & R : o & \text{(has value restriction)} \\
\mid & \geqslant n\ R & \text{(Unqualified} \\
\mid & \leqslant n\ R & \text{ number} \\
\mid & = n\ R & \text{ restriction)}
\end{array}$$

(a) Concept Descriptions

$$\begin{array}{rlll}
R, S ::= & S & \text{(atomic role)} \\
\mid & R^- & \text{(inverse role)} \\
\mid & R^+ & \text{(transitive role)}
\end{array}$$

(b) Role Descriptions

$$\begin{array}{rl}
C \sqsubseteq D & \text{(concept inclusion)} \\
R \sqsubseteq S & \text{(role inclusion)} \\
C \equiv D & \text{(concept equality)} \\
R \equiv S & \text{(role equality)} \\
 & \\
C(a) & \text{(concept assertion)} \\
R(a, b) & \text{(role assertion)}
\end{array}$$

(c) Terminological and Assertional Axioms

Figure 7: Syntax of $\mathcal{SHOIN}$

the translation of a value restriction into FOL. The syntax presented in Figure 7 essentially hides variables and restricts implication. Also, only individuals are available for assertions, and free variables are disallowed. Furthermore, unqualified number restrictions are added to FOL, which limit or demand the number of relations asserted for each individual in the described concept. Note that no concept description is allowed for the second argument of the relation, hence the name unqualified. There are no functions and predicates over the predefined datatypes available in $\mathcal{SHOIN}$(D), and it is not recommended to define further datatypes since they will not be recognised by other tools [12, Section 6.1]. So, merely data literals with built-in equality are supported by OWL DL, $\mathcal{SHOIN}$(D) and also $\mathcal{SHIF}$(D), thus the relation of individuals with data literals is a possible usage (the only one).

Typically, DLs are not statically typed such that axioms are necessary to restrict roles to certain concepts. Axioms for roles relating concepts with datatypes are called *datatype properties* in OWL DL; their axioms are limited to role inclusion, domain and range definitions. Roles relating concepts are called *object properties* in OWL DL; the same axioms as for datatype properties are available plus axioms stating functionality, inverse functionality, symmetry, transitivity and axioms defining a property to be the inverse of a role. While ordinary DLs do not distinguish these kinds of roles explicitly, OWL DL has different declaration constructs. Furthermore, not all combinations of axioms for roles are allowed, e.g. a role may not be simultaneously defined as transitive and used in cardinality constraints; this applies also to functional roles (global cardinality constraints).

The DL axioms described so far form the TBox of an ontology. They are accompanied by axioms for the ABox providing concept assertions and relations of individuals. The TBox defines the terminology and the ABox deals with facts about the individuals in the knowledge base. With CASL-DL the author focuses on the analysis and definition of TBoxes where the usage of strongly typed ontologies is inherited from CASL (see Section 2.2 for the notion of strongly typed ontologies). The definition of large ABoxes has not been an aim of CASL-DL. This would not be very feasible, since writing assertion axioms in FOL is a tedious and error-prone process — databases are more suitable for such a task. In an application, an OWL DL ontology generated from a CASL-DL specification would be used as TBox, and the individuals and their relations would be kept in a database for persistence and short answering times. For testing an ontology it is still possible to easily define ABoxes in CASL-DL. Protégé [23] is a tool aiming at the definition of knowledge bases[6] with a large number of facts by providing easily definable input forms for facts.

CASL-DL provides the same FOL notation as CASL, instead of DL notations, although the hiding of variable introduction used by DLs might lead to an easier analysis of axioms.

[6]A knowledge base may be exported into OWL DL or other knowledge representation languages.

```
spec GenCardinality [sorts Subject < Thing
                          pred predicate : Subject × Thing] =
  {Set[sort Thing fit sort Elem ↦ Thing]
   reveal Set[Thing], ♯__, __ϵ__, Nat, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, __@@__,
          __≥__, __≤__
   then
     op    toSet : Subject → Set[Thing]
     ∀ x : Subject; y : Thing • predicate(x, y) ⇔ y ϵ toSet(x);
     preds minCardinality[predicate](s : Subject; n : Nat) ⇔ ♯ toSet(s) ≥ n;
           maxCardinality[predicate](s : Subject; n : Nat) ⇔ ♯ toSet(s) ≤ n;
           cardinality[predicate](s : Subject; n : Nat) ⇔ ♯ toSet(s) = n;
  } hide Pos, toSet, Set[Thing], ♯__, __ϵ__, __≤__, __≥__

spec PredefinedConcepts =
  sort  Thing
  pred  Nothing : Thing
  ∀ x: Thing • ¬ Nothing(x)                          %(empty_concept_Nothing)%
```

Figure 8: Proposed Casl-DL prelude

Only special cardinality constructs are added to Casl which provide easier access to number restrictions than the originally proposed instantiation of a generic specification. Figure 8 shows the GenCardinality specification which has been introduced as "Casl-DL Prelude" in [KL-10]. However, Casl-DL specifications are easier to comprehend, when the predicates for the number restriction are part of the language. Furthermore, the connection of DL reasoners for Casl-DL is easier to implement when cardinality restrictions are predefined symbols instead of user defined. So, the predicates *minCardinality*, *maxCardinality* and *cardinality* are built-in constructs of Casl-DL, and the predicate given in square brackets needs to be declared with this schematic profile "$s \times Thing$" where s is either *Thing* or a subsort of *Thing*. In summary, Casl-DL is equivalent to $\mathcal{SHOIN}$(D) — it is a restriction of Casl to $\mathcal{SHOI}$(D) extended by those cardinality predicates ($\mathcal{N}$).

A translation of Casl-DL into Casl preserving structuring still needs to introduce instantiations of the GenCardinality specification for predicates which are used in number restrictions. In contrast, a translation of a flattened Casl-DL specification just adds the required cardinality predicates with their definition without hiding the correspondence to sets. Furthermore, the concepts **sort** *Thing* and **pred** *Nothing* and the sentence, labelled with %(empty_concept_Nothing)%, are always present in a Casl-DL theory and need not be introduced by import. Again, the translation of Casl-DL theories into Casl adds the sentence %(empty_concept_Nothing)% from PredefinedConcepts (Figure 8).

Since the structuring elements of Casl are directly available for Casl-DL, all the advantages described in Section 2.3 apply to Casl-DL, too. Nonetheless, the translation of Casl-DL into OWL DL needs to take care of expanding hidden parts of the theory for a complete translation. Where possible the structuring via simple imports and instantiations might be translated into OWL DL import statements, but in general the easiest translation of a Casl-DL specification flattens the structuring elements and forms a single theory prior to the translation. If the OWL DL specification is only supposed to be loaded into a DL reasoner, the latter approach is preferable since the DL reasoner integrates all sentences (including the imported ones) into one knowledge base anyway. Thus providing only one OWL DL file speeds up parsing and analysis of the ontology. However, a structure preserving translation of Casl-DL specifications into a set of OWL DL ontologies with equivalent import statements, where each ontology corresponds directly to one named Casl-DL specification, is more appropriate when the ontology specifications are supposed to be reused separately.

```
from BASIC/NUMBERS get INT

view INT_IMPLEMENTS_INTEGERLITERAL :
      INTEGERLITERAL to
      {INT
       then %def
            sorts  $nonPositiveInteger = \{i : Int \bullet i \leq 0\}$;
                   $negativeInteger = \{i : nonPositiveInteger \bullet i < 0\}$
      } =
      $integer \mapsto Int$, $positiveInteger \mapsto Pos$, $nonNegativeInteger \mapsto Nat$
end
```

Figure 9: Mapping of INTEGERLITERAL to INT

The predefined XML Schema datatypes [6] of OWL DL are provided as predefined signature elements bootstrapped from the CASL library CASL_DL_DATATYPES presented in Appendix A. Again, the translation of CASL-DL into CASL uses an import of the specification DL_LITERAL for providing the needed signature. Only the integer numbers (**sort** *integer* and subsorts) are mapped to subsorts of *Int* (specified in INT of library BASIC/NUMBERS [10, pp. 379–385]) with the mapping and definitions shown as a view in Figure 9. This is necessary because the generic specification GENCARDINALITY uses ordering predicates on natural numbers for the definition of the cardinality predicates which are not defined by INTEGERLITERAL. Figure 10 shows the currently implemented datatypes of CASL-DL with the same names as in [12, Section 6] (this is just the minimal recommended support). *Char* is the only datatype not provided by the XML Schema datatypes and is not available for CASL-DL specifications; it is only needed for CASL-DL to achieve support for string literals. A char is formed as a single element string in XML Schema datatypes and CASL-DL.

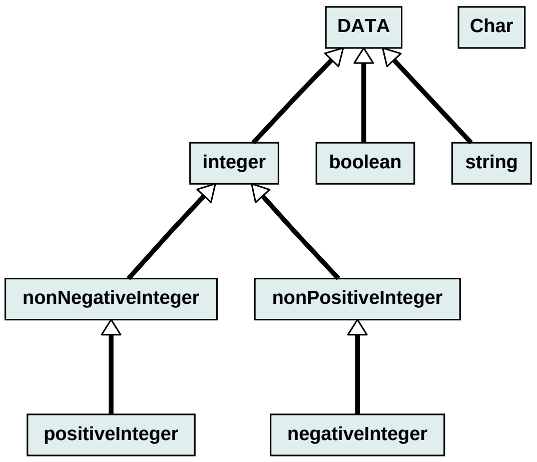


Figure 10: Taxonomy of predefined datatypes in CASL-DL

3.2 Comorphisms Relating OWL DL and CASL-DL

In [KL-10] translations of CASL-DL into OWL DL and vice versa are introduced as institution comorphisms [25]. Institutions formalise the notion of logical system. An institution has to provide definitions of the concepts signature, sentence, model and satisfaction. Since only very little is assumed about the nature of these concepts, the notion of institution is rather general and covers a great variety of logics. These concepts are then used to formally describe mappings between logics called comorphisms. The comorphisms make use of annotations on the CASL-DL and OWL DL side. Classes in OWL DL can be marked as "sorts" such that they result in sorts after a translation into CASL-DL instead of unary predicates.

This is a slight change in the handling of OWL DL classes in contrast to [KL-10] where we planned to translate all classes into sorts except those marked with an annotation. But the former proposed treatment of classes might result in inconsistent theories on the CASL-DL side, because sorts have to have non-empty carrier sets and OWL DL classes do not necessarily obey this restriction. Furthermore, all symbols in OWL DL ontologies are URIs [3] which normally share a long prefix like `http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#ConsumableThing`. The comorphism outlined here substitutes the common prefix (usually up to "#") with a globally unique name space qualifier, e.g. "food" in this case. The mapping of name space qualifiers to URI prefixes is kept in special CASL annotations for a translation back to an equivalent OWL DL theory. Also, the names of CASL-DL specifications resulting from translations of OWL DL ontologies are derived from the URI. Usually dropping of the prefix up to the last slash "/" is sufficient, but depending on the imported ontologies it might be necessary to keep more of the URI for a proper disambiguation. Nevertheless, the dropped prefix is stored in a CASL annotation, too. Circular imports on the OWL DL side are treated as a union of all ontologies which are part of the circle, e.g. the wine and food ontologies (examples in [65]) import each other and result in only one joint CASL-DL specification where the name is derived from the URIs of both ontologies. A translation back to several OWL DL ontologies equivalent to each of the originals is not planned, since each sentence in the CASL-DL theory would need to be tagged with its originating ontology for separation, not only the symbols. Figure 11 provides an overview of the comorphisms represented as solid lines and the approximation of CASL theories with CASL-DL theories depicted as a dashed line (see Section 3.3).

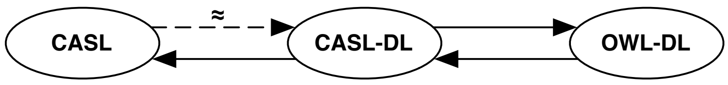


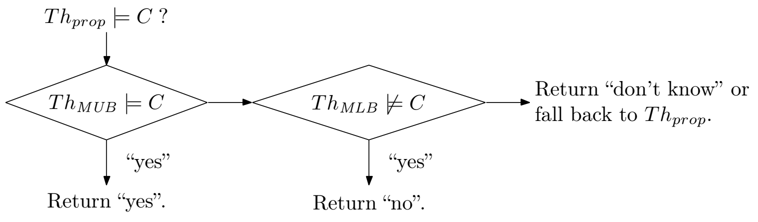
Figure 11: Comorphisms and approximation related to CASL-DL

The comorphism of CASL-DL into CASL is almost an embedding since CASL-DL is an extension of a sublanguage of CASL. Only the datatypes and cardinality restrictions need a special treatment as described above. Of course, certain sublanguages of CASL can be translated back to CASL-DL without loss of information, but a translation of these CASL-DL specifications into OWL DL would result in different ontologies from the originals in a round trip from OWL DL over CASL-DL to CASL and back to OWL DL using again CASL-DL as intermediate step with editing the CASL specification in between; e.g. after a renaming of a cardinality predicate, the translation cannot keep track of the name change. Also, all other renamings must be reflected into the name space annotations. If a CASL specification SP_1, which is generated from a CASL-DL specification, is extended by a sentence not obeying the restrictions of CASL-DL, the translation of the extended specification SP_1 back to CASL-DL will fail. The following section presents a theory approximation of FOL theories with DL theories explained along the languages CASL and CASL-DL.

3.3 Approximation of FOL Theories by Description Logic Theories

The algorithm for approximation of first-order logic theories by description logic aims at the compilation of foundational ontologies provided in FOL into ontologies efficiently processable with DL reasoners. The resulting DL theory may be judged according to different criteria, which need to be balanced against each other: on the one hand readability of the DL theory by a human recipient, and on the other hand the amount of knowledge compiled into the result with respect to the original theory. The latter criterion is discussed with respect to related work, but is not the main goal of this section. The main focus of this section is a human readable format for the development of domain ontologies on the basis of compiled foundational ontologies. Furthermore, the discussion of approximations may also lead to a validation method for domain ontologies against foundational ontologies where a view from an application CASL-DL ontology (possibly translated from OWL DL along the comorphism presented in Section 3.1) to a foundational CASL ontology is formulated in CASL and checked with some automated theorem proving system. Even if not all axioms of the domain ontology are entailed by the foundational ontology, certain aspects such as properties of spatial or time relations might be validated against the foundational ontology. In fact, the approximation algorithm presented here compiles into $\mathcal{SHIF}$(D), a sublogic of CASL-DL (and $\mathcal{SHOIN}$(D)). Note, although $\mathcal{SHIF}$ is not tractable[7], it is still computationally efficient enough for practical purposes [34, 69]; i.e. there exist specialised DL reasoners such as Racer [28] and FaCT++ [70, 32]. The approximation of FOL theories by DL theories has been presented at the doctoral colloquium of CALCO 2005 (Conference on Algebra and Coalgebra in Computer Science) [KL-4] and has been published after revision in the proceedings of the FOIS 2006 Conference (Formal Ontologies in Information Systems) [KL-5].

Related work. In [62] Selman and Kautz describe algorithms to derive tractable theories from intractable ones. The main algorithm is a *theory approximation* of Horn clauses from arbitrary propositional-logic theories. This method has also been called *knowledge compilation* and is used to obtain a tractable form of the theory for automated reasoning. Their algorithm yields two sets of Horn clauses, one with weaker and one with stronger Horn clauses. These sets are derived from the original set of propositional clauses. Additionally, sketches of various other approximations are presented in [62], such as FOL with Horn clauses or $\mathcal{FL}$ with $\mathcal{FL}^-$ (two DLs).



where Th_{prop} is the propositional theory; C is the query in propositional logic; Th_{MUB} and Th_{MLB} are Horn approximations.

Figure 12: Schematic representation of query answering with Horn approximations[8]

The quality of an approximation is determined by how close the approximating theories are to the original theory. Selman and Kautz [62] define a weaker approximation as an upper-bound which has more models than the original theory, i.e. their definition is in terms of model classes for an approximation of FOL with DL: $\mathbf{Mod}(Th_{FOL}) \subseteq \mathbf{Mod}(Th_{DL})$. At best an approximation algorithm would find a minimal upper-bound[9] (MUB) which is in the weaker language and there exists no DL theory Th' with $\mathbf{Mod}(Th_{FOL}) \subseteq \mathbf{Mod}(Th') \subset \mathbf{Mod}(Th_{DL})$. Note that more than one MUB might exist in case of DL approximations since different sets of role axioms are valid as approximation. In a similar way, lower-bounds and maximal lower-bounds (MLB) of a theory are defined. In [62] a Horn approximate compilation of propositional logic is presented where propositional queries are answered with Horn theories that are approximations from above and below of a propositional theory. Figure 12 shows the general approach of answering queries with compiled Horn theories. The Horn

[7] $\mathcal{SHIF}$ has EXPTIME complexity.

[8] Figure 12 resembles Figure 2 in [62, p. 197]. Note that in the original figure the abbreviations LUB and GLB were used wrongly instead of MUB and MLB.

[9] In [62] the same definition is given, but called wrongly least upper bound (LUB).

approximations Th_{MUB} and Th_{MLB} fulfil the following model theoretic properties w.r.t. the original propositional theory Th_{prop}: $\mathbf{Mod}(Th_{MLB}) \subseteq \mathbf{Mod}(Th_{prop}) \subseteq \mathbf{Mod}(Th_{MUB})$.

A more general algorithm for FOL knowledge compilation into various sublanguages of FOL has been introduced by Alvaro del Val [13, 14]; the main target languages are again variants of Horn logic. However, one important finding of del Val for the approximation of FOL theories has been the need of only a weaker theory (which is maximally strong) if only the target language is used for queries to the compiled theory [13].

Overview of the algorithm. Figure 13 shows an overview of the theory approximation described in this section where the initial reification step has not been included in [KL-4, KL-5], because it is a transformation preserving the semantics, although the comprehensibility and readability of the theory is decreased. All arrows transport a whole theory (rectangular boxes) except where marked differently; the ellipses depict the functions needed for the approximation.

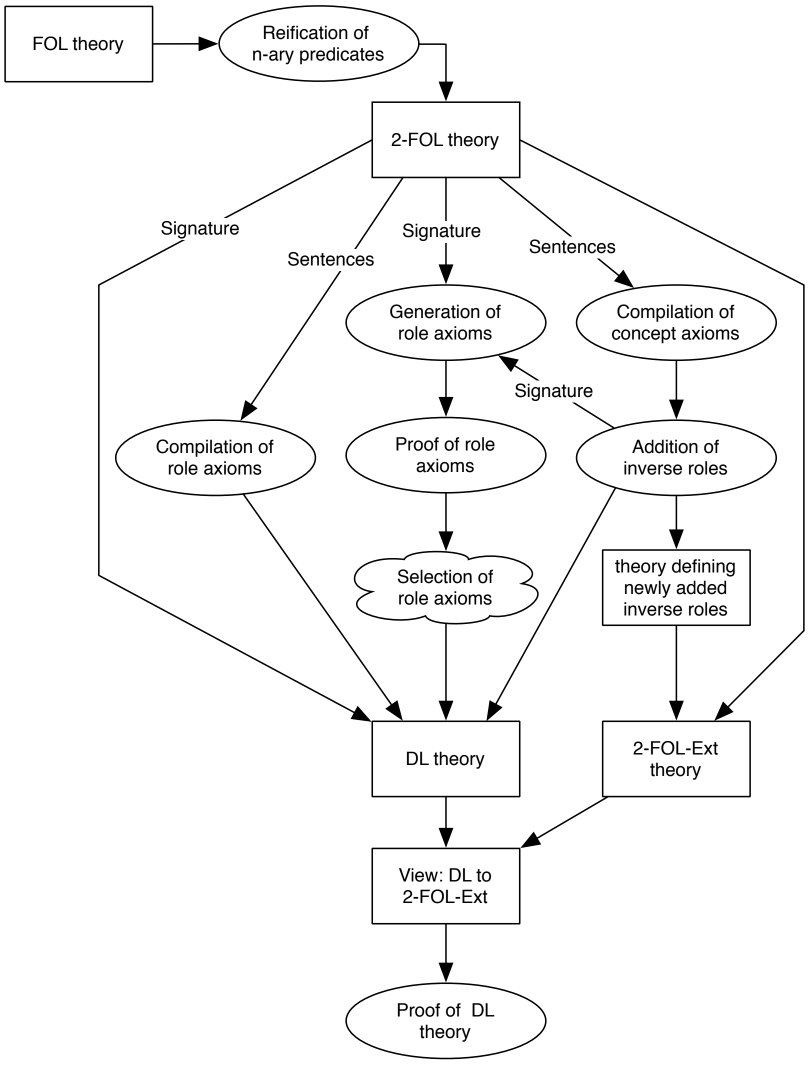

FOL theory
Reification of n-ary predicates
2-FOL theory
Signature
Sentences
Signature
Sentences
Generation of role axioms
Compilation of concept axioms
Signature
Compilation of role axioms
Proof of role axioms
Addition of inverse roles
Selection of role axioms
theory defining newly added inverse roles
DL theory
2-FOL-Ext theory
View: DL to 2-FOL-Ext
Proof of DL theory

Figure 13: Flow of data during the approximation

Reification of predicates. The initial step of the theory approximation has not been published yet and is called “Reification of n-ary predicates” in Figure 13, where $n > 2$. This is needed, because only unary and binary predicates are available in CASL-DL. A reification

```
spec Binary_Temporal_Dissective =
       Time_Mereology with sort T, pred P : T × T
then sorts  s1, s2
       pred  Rel : s1 × s2 × T
       ∀ x1 : s1; x2 : s2; t1, t2 : T
       • Rel(x1, x2, t1) ∧ P(t2, t1) ⇒ Rel(x1, x2, t2)

spec reified_Binary_Temporal_Dissective =
       Time_Mereology with sort T, pred P : T × T
then sorts  s1, s2
       sort  Rel
       ops    Rel_1 : Rel → s1;
              Rel_2 : Rel → s2;
              Rel_3 : Rel → T
       ∀ r1, r2 : Rel; x1 : s1; x2 : s2; t1, t2 : T
       • Rel_1(r1) = x1 ∧ Rel_2(r1) = x2 ∧ Rel_3(r1) = t1 ∧ P(t2, t1)
         ⇒ ∃ r2 : Rel • Rel_1(r2) = x1 ∧ Rel_2(r2) = x2 ∧ Rel_3(r2) = t2

view ReificationEntailsOriginal :
    Binary_Temporal_Dissective to
    {reified_Binary_Temporal_Dissective
     then %def
         pred  Rel : s1 × s2 × T
         ∀ x1 : s1; x2 : s2; t : T
         • Rel(x1, x2, t) ⇔
           ∃ r : Rel
           • Rel_1(r) = x1 ∧ Rel_2(r) = x2 ∧ Rel_3(r) = t        %(Rel_def)%
    }
```

Figure 14: Reification of a ternary predicate

in general uses something more concrete than the original, i.e. instead of a ternary predicate a concept together with three roles is used, where each instance (individual) of the concept represents one triple. Fig 14 shows a predicate taken from Dolce in Casl and its reification as a sort with three functions. This reification follows the method described in [1, p. 164] but it translates many-sorted FOL into 2-FOL instead of entity relationship diagrams into DL.

2-FOL is a restricted form of FOL allowing only unary and binary predicates and unary functions; it has been introduced to describe the approximation of FOL by DL in [KL-4, KL-5]. The view ReificationEntailsOriginal in Figure 14 shows that the reification of the predicate *Rel* with a sort and functions entails the sentence for the predicate *Rel* defined in Binary_Temporal_Dissective. The opposite entailment is also valid and is presented Appendix B.1. The naming scheme for the functions relating the reified n-tuple with its elements used in this example by adding the position to the predicate symbol can be refined to consider the sort names, too. However, n-ary predicates where all arguments have the same sort will still need the number representing the argument position. A combination of both, sort symbol and argument position, yields an injective mapping. Moreover, the reification algorithm still needs to check for newly introduced overloading of symbols and has to resolve these name clashes. The reification of functions with more than one argument works analogously to the reification of predicates.

Furthermore, Liem et.al. [43] have recently introduced a slightly different reification pattern, which yields reified classes which are easier to reuse. This is achieved by moving the actual argument restricting axioms into the significant class of the n-ary relation; e.g. the result type of a ternary sum relation determines its argument types and possible limiting axioms for the arguments.

Approximation algorithm. The approximation of a 2-FOL theory is performed in four steps (cf. Figure 13):

1. Unary predicates and the subsort taxonomy are just incorporated into the CASL-DL theory; if necessary a new maximal sort is introduced which subsumes all other sorts except those which are supposed to be mapped to datatypes;
2. compilation of concept axioms which include also sort membership assertions and unary predicates;
3. Generation of role axioms for binary predicates found in the signature and for inverse roles introduced by the previous step;
4. compilation of role axioms, which have only binary predicates.

After incorporating the whole 2-FOL signature into the DL theory, all possible DL axioms are generated for each role and each axiom is sent as a conjecture together with the 2-FOL theory to an Automated Theorem Proving (ATP) system. If the conjecture is proved, it is kept as an axiom for selection by the user. Those sentences of the 2-FOL theory which are implications or equivalences with binary predicates on both sides of the Boolean connective are used to generate role inclusions. Again, only those role inclusions entailed by the input 2-FOL theory are part of the approximation. If two roles are equivalent, i.e. role inclusions in both directions are valid, an equivalence axiom is incorporated into the approximation instead of two role inclusions.

Out of the generated role axioms entailed by the 2-FOL theory different axiom sets are formed, which conflict with respect to the limits that a role may be either transitive or functional, but cannot have both properties. Note each union of two generated sets of role axioms is not in $\mathcal{SHIF}$. The roles in each set are related via subrole or inverse role axioms, since the limit applies to all axioms related via these axioms. The axiom sets are presented in a way that the user can easily select the appropriate set by deciding if the transitivity or functionality of a role is more important for the further application of the compiled ontology. The compilation of unary functions is not explicitly described here or in [KL-4, KL-5], but it is easy to translate unary functions into equivalent binary predicates with an additional axiom defining the functionality of the predicate. Note that cardinality restriction axioms apart from global cardinality restrictions (used for functional roles) are not generated by the compilation, and hence there is no reason for presenting any concept description axioms to the user for choosing.

Moreover, formulas with unary predicates and sorts (concepts) are compiled by finding certain patterns which are derived from the legal concept descriptions of $\mathcal{SHIF}$(D). When concept descriptions with roles are approximated, inverse roles are often needed to remain within the limits of DL. The axioms in Figure 15 taken from the example in [KL-5] show the approximation of mereological definitions found in DOLCE, e.g. a rewriting of the definition %(Dd3_Atom)% with the definition %(Dd3_Atom_rw)% uses *PP_ i*, the inverse role of *PP*. The complete example including structuring is presented in Appendix B.2. The patterns of concept descriptions are at first searched for implications where the premise is a named concept (unary predicate application or a sort membership assertion) and equivalences where one side is again a named concept. Detailed descriptions of the approximation methods depicted may be obtained from [KL-5].

At the end of the approximation, the $\mathcal{SHIF}$(D) theory is formed by taking the union of the results of all the knowledge compilation methods described above, and the entailment shown in Figure 16 is proved automatically by some ATP system validating the result. The view in Figure 16 describes the relation between a 2-FOL theory and its DL approximation formally and yields proof obligations for the following theory entailment:

$$Th_{\text{2-FOL\_SPECIFICATION\_EXT}} \models Th_{\text{APPROXIMATION\_IN\_DL}}$$

The author has proved such an entailment in [KL-4, KL-5] for the binary relations of Mereology specified in DOLCE automatically with SPASS. Figure 15 presents an extract of the

```
spec GenMereology[sort s] =
       GenParthood[sort s] with pred P
then preds PP(x, y : s) ⇔ P(x, y) ∧ ¬ P(y, x);       %(Dd1_Proper_Part)%
           O(x, y : s) ⇔ ∃ z : s • P(z, x) ∧ P(z, y);     %(Dd2_Overlap)%
           At(x : s) ⇔ ¬ ∃ y : s • PP(y, x)                %(Dd3_Atom)%
       ∀ x, y : s • ∃ z : s • At(z) ∧ P(z, x);                   %(Ad18)%

spec GenMereology_DL[sort s] =
       GenParthood_DL[sort s] with pred P
and    InvPP_P [sort s preds PP, P : s × s]
       with preds P_i, PP_i
then preds O(x, y : s) ⇔ O(y, x);                      %(Dd2_Overlap_sym)%
           At(x : s) ⇔ ¬ ∃ y : s • PP_i(x, y)              %(Dd3_Atom)%
       ∀ x, y, z : s
       • ∃ z' : s • P_i(x, z') ∧ At(z')                        %(Ad18_rw)%
       • PP(x, y) ⇒ P(x, y)                        %(Dd1_Proper_Part_impl)%
       • PP(x, y) ∧ PP(y, z) ⇒ PP(x, z)          %(Dd1_Proper_Part_trans)%
```

Figure 15: Approximation of mereological definitions from Dolce

```
spec 2-FOL_Specification_EXT =
       2-FOL_Specification
then %def
       <inverse_role_definitions>

view Validation_of_Approximation :
       Approximation_in_DL to 2-FOL_Specification_EXT
```

Figure 16: Relation of a 2-FOL theory and its approximation in DL

approximated axioms and the whole example is shown in Appendix B.2. Furthermore, the author expects that such an entailment is always provable with an ATP system like Spass (see Section 4.2 for ATP systems usable with Casl).

Evaluation of the approximation. The quality of the DL approximation is determined by how close the weaker DL theory is to the 2-FOL theory. The approximation method described here yields a weaker theory. This is sufficient if only the target language of the approximation is used for queries [13]. A minimal upper-bound is also described by the term *maximally strong weaker theory*, which is easier to comprehend. The quality of a 2-FOL approximation with DL described in this section yields a maximally strong DL theory with respect to the role axioms of the original theory since all possible role axioms are generated and the proven ones are incorporated in the approximation. With respect to the concept descriptions, the quality of the algorithm is still to be determined. The main focus of the author has been to find an approximation of FOL theories with DL which is human readable and not a maximally strong theory with respect to the original.

Furthermore, a good readability of the approximation result is achieved by using patterns which transform the original axioms instead of using a logical normal-form, which might introduce additional symbols. However, an implementation of the algorithm still needs to be carried out; so, this question will remain open until more examples beyond that presented in Appendix B.2 are available.

3.4 Summary

The combination of FOL and DL ontologies has been described in detail along with CASL-DL (an extended sublanguage of CASL), its translation from and to OWL DL, and the approximation of FOL ontologies with DL ontologies. This leads to a formal framework for studying the relation of foundational ontologies and applied ontologies, where the former are formalised as precisely as needed for a comprehensive understanding in rich logical systems such as FOL or modal logic. The latter are typically written in a logical formalism tailored towards efficient reasoning in the context of large data sets and do not support a great variety of logical constructs. The approximation algorithm offers a bridge between these logical systems. It can be either used to compile a FOL theory into a DL ontology, which is then extended by a weak formalisation of the application scenario, or the foundational ontology is extended within the context of a strong formalisation in CASL, and the result is compiled into DL for efficient usage in the application scenario.

4 Tool Support for Ontologies in Casl

Ontologies written in FOL (as e.g. a major part of Dolce) need tool support and connection to other ontology languages. In the preceding section the connection between OWL DL, Casl-DL and Casl has been discussed. All these methods will be integrated into the Heterogeneous Tool Set (Hets) [KL-11]. This section deals with the benefits of Hets for structured ontologies written in many-sorted FOL (Casl) and Casl extensions. A key point for the development of ontologies is the support of Automated Theorem Proving (ATP) systems for Casl which is described in Section 4.2 and [KL-9].

4.1 Features of Hets for Ontologies in Casl

In [KL-11] a brief general description of Hets has been published, which is tailored towards software engineering; the User Guide [KL-13] provides an overview of Hets. Here these results are expanded and applied for the analysis of ontologies written in Casl (see Section 2 for a discussion of ontologies developed in Casl). Casl-DL inherits most of the features described for Casl in this section because it is technically an extension of the Casl logic restricted to $\mathcal{SHOIN}$(D) (see Section 3.1 and [KL-10]). Hets provides analysis, transformation and connection to provers for Casl and its extensions. In general each logic that is formalised as an institution [25] can be easily integrated into Hets. Comorphisms translate between these logics by providing the necessary transformation of signatures and rewriting of sentences. Figure 17 provides an overview of the currently supported logics and comorphisms, where the planned comorphisms are shown as dashed lines. The different shades of the nodes reflect the current stability of the logic's implementation; a darker grey denotes a higher stability than a lighter grey. Only one arrow corresponding to the approximation algorithm presented in Section 3.3 is not a comorphism and is hence marked with '≈'. HetCasl (Heterogeneous Casl) [53] serves as the integration language for heterogeneous specifications by providing constructs for choosing a specific logic and a comorphism for the heterogeneous translation of specifications. The logics SoftFOL and Isabelle provide support for theorem provers, where the former logic is described in greater detail in Section 4.2 and the latter will be briefly explained in this section.

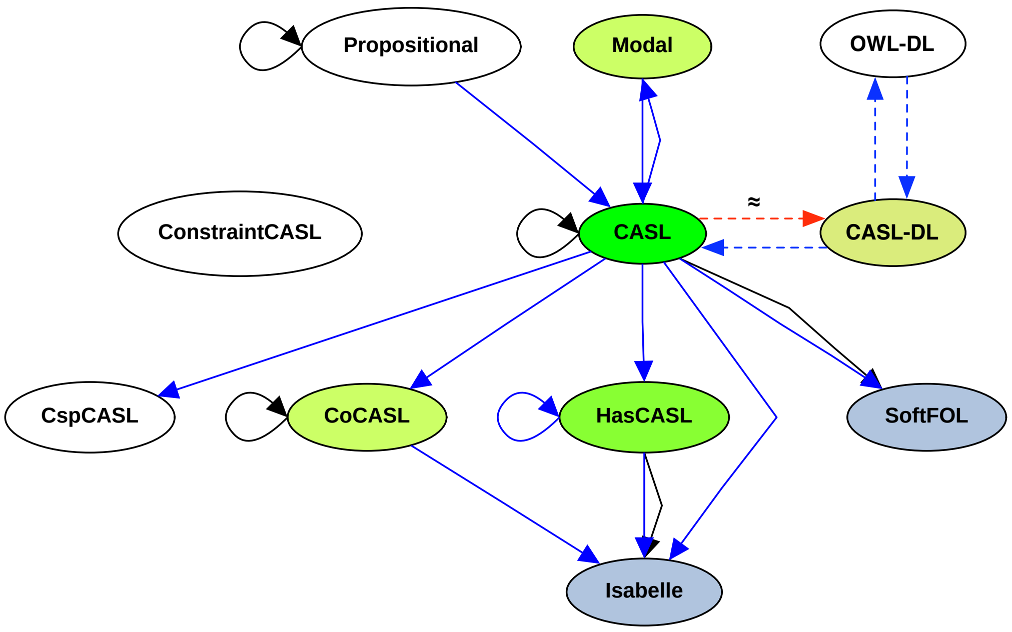


Figure 17: Currently supported logics of Hets

Apart from the logics Casl, Casl-DL and OWL DL, which have been presented in Section 2 and Section 3.1, only ModalCasl (labelled with "Modal" in Figure 17) is of further interest for the specification of ontologies. It is an extension of Casl with multi-modalities and term modalities. It allows the specification of modal systems with Kripke's possible worlds semantics, where the specific modal system(s) are specified by the user with propositional formulas. A translation from ModalCasl into Casl with possible worlds encoding has been implemented, such that ATP systems and Isabelle are available for theorem proving via further comorphisms.

The logic Isabelle provides proof support with the semi-automatic theorem prover Isabelle/HOL [58]. It uses proof scripts with a notation resembling mathematical pen-and-paper proofs; the logic is a typed higher-order logic with support for induction. However, Casl's partial functions and subsorting need to be coded via comorphisms [52], since they are not available in Isabelle/HOL. The benefits of a combination of automatic and semi-automatic theorem proving offered for Casl by Hets have been presented by Wölfl et.al. along the verification of the Region Connection Calculus [75].

Casl is input as simple textual specification in ISO latin 1 with a syntax resembling logical and mathematical notation, e.g. a '$\vee$' is input as '\/'. For the editor GNU Emacs [18] a Casl-mode [KL-12] is available, offering syntax highlighting and automatic indentation; it starts Hets for static analysis and inspection of the Casl library currently being edited. For the presentation of specifications, e.g. in papers or on slides, Hets offers the automatic generation of LaTeX code, which is still readable enough to edit some formatting for the display on slides. Display annotations may be used to provide special LaTeX notation for sorts, predicates, function symbols and variables. As examples consider Fig 14 and Appendix B.1, where display annotations are used for the display of sort '`s1`' as 's_1'.

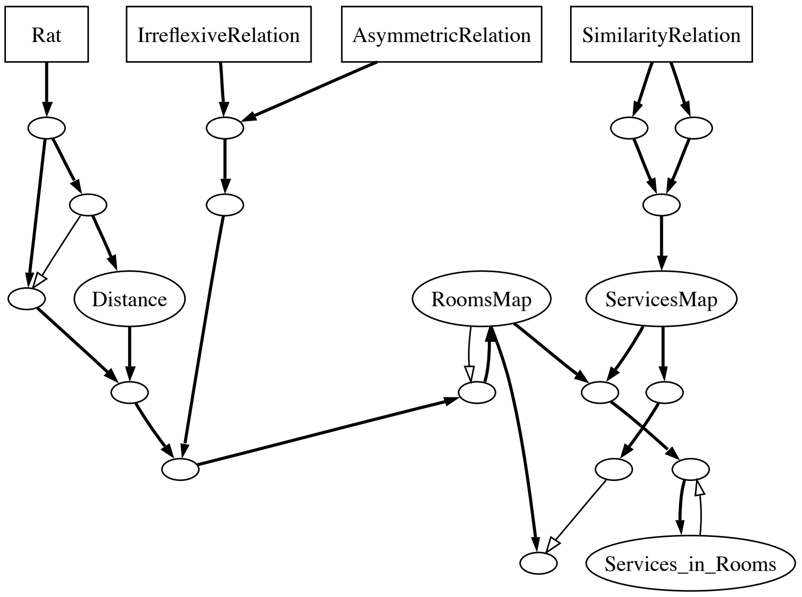


Figure 18: Structure graph of the library RoomsAndServices

A call of Hets starts the static analysis of a Casl library by parsing and analysing the structure and by calling logic-specific parsers and analysing functions for each basic specification. The graphical user interface (GUI) of Hets starts with the display of a structure graph[10] showing the structure of a Casl library with definitional (import) and

[10]Originally such a graph has been called development graph inherited from the development graph calculus that covers the whole process of software development with formal refinement steps presented as views. But in fact for ontologies only the structure and not the development of specifications in terms of refinement is reflected.

```
spec SERVICESMAP =
     SIMILARITYRELATION
     with Elem ↦ Service, __~__ ↦ __near__
and  SIMILARITYRELATION
     with Elem ↦ Service, __~__ ↦ __connected__
then ∀ x, y : Service
     • x connected y ⇒ x near y          %(connected implies near)%
```

Figure 19: The specification SERVICESMAP

theorem links between the theories represented as nodes. Nodes of external specifications have a rectangular shape. Definitional links are drawn with bold lines and filled arrow heads; theorem links are drawn with thin lines and open arrow heads. Figure 18 shows the structure graph of the example library ROOMSANDSERVICES published in [KL-7]. In fact this graph is the black-and-white version generated from HETS while the GUI uses a coloured display. Unamed nodes show intermediate structuring of a specification, e.g. the specification SIMILARITYRELATION is imported and renamed for two different predicates and the union forms the starting point for SERVICESMAP. The corresponding structuring constructs of CASL are shown in Figure 19.

The main interaction with a CASL library of specifications is driven by the display of the structure graph. It offers reasoning and translation services at the structured level, where the former apply the proof calculus for development graphs [54, 53] which yields proof obligations at the level of theories, e.g. the node below ROOMSMAP in Figure 18 would provide the goals of the implied extension. Context menus offer functionality at edges and nodes. At edges the signature morphism and (for theorem links) the proof status can be inspected. Also, all links provide conservativeness checking in their context menus. At each node the signature, theory, sublogic and taxonomy can be inspected (Figure 1b on page 4 shows a taxonomy graph such as displayed by HETS).

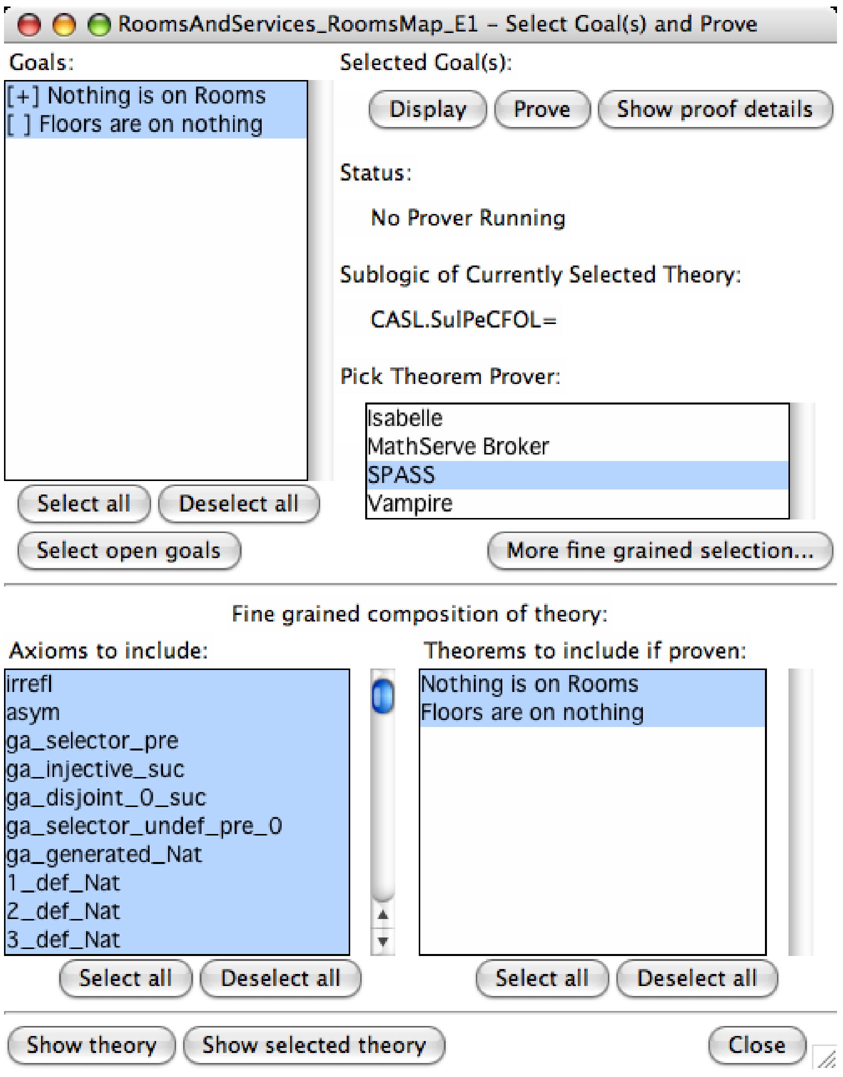
`[+]`	proved goal
`[-]`	disproved goal
`[×]`	proved goal revealing an inconsistent theory
`[ ]`	open goal

Table 3: Proof status indicators

Additionally, a proof management GUI is available at each node, which shows the names of the goals and sentences of a theory as in Figure 20. The proof status of the goals can be discovered at a glance by the status indicator in square brackets in front of the goal names, e.g. a '`[+]`' indicates a proven goal. Table 3 gives an overview of all indicators, where an inconsistent theory is detected by checking the used formulas indicated by an ATP system for the proof of the conjecture. If the conjecture is not among the used formulas, the theory is inconsistent[11], because a resolution based theorem prover needs to resolve with the CNF[12] clauses generated from the conjecture during a proof in a consisten theory. The list of known provers gives easy access to commonly used (composed) comorphisms translating from the theory's sublogic into the provers logic. However, the button "More fine grained selection..." allows one to choose out of a list of comorphisms translating into a prover supported logic. The list of known provers and the list of comorphisms is determined by the sublogic calculated from the currently selected theory. The bottom half of the proof management window allows the selection of axioms and proven theorems to be included in the theory for a proof attempt. Section 4.2 discusses the need for shrinking the theory sent

[11] See Sect 4.2 for details about consistency checking with ATP systems.

[12] Clause Normal Form



RoomsAndServices_RoomsMap_E1 – Select Goal(s) and Prove

Goals:

[+] Nothing is on Rooms
[ ] Floors are on nothing

Selected Goal(s):

Status:

No Prover Running

Sublogic of Currently Selected Theory:

CASL.SulPeCFOL=

Pick Theorem Prover:

Isabelle
MathServe Broker
SPASS
Vampire

Fine grained composition of theory:

Axioms to include:

irrefl
asym
ga_selector_pre
ga_injective_suc
ga_disjoint_0_suc
ga_selector_undef_pre_0
ga_generated_Nat
1_def_Nat
2_def_Nat
3_def_Nat

Theorems to include if proven:

Nothing is on Rooms
Floors are on nothing

Figure 20: Proof management GUI of HETS

to ATP systems. Note that the sublogic used for determining the known provers list (and the possible comorphisms) is recalculated when axioms or goals are selected or deselected.

Figure 1b shows a subsort graph which has been generated by HETS automatically. Although subsort relations have been a CASL basic specification construct from the beginning, the need for a display as subsort graph only has arisen from usage of CASL for ontologies. Furthermore, the display of unary and binary predicates integrated into the subsort graph forms the concept graph and gives graphical access to a large part of the signature of ontologies written in CASL and CASL-DL since the implementation of the CASL-DL logic reuses the functionality implemented for CASL.

HETS is implemented in Haskell [36] as an open source project. Each logic has to implement the type class `Logic` that provides a common function API for access to logics implementing the concept of institutions. This design allows for a clear separation of the logics' implementations from the logic independent parts (e.g. structure analysis), which enables a parametrisation of HETS over different logic graphs tailored towards different application scenarios such as ontology engineering or software engineering. Comorphisms are special heterogeneous functions which translate theories between sublogics of the same or of different logics, and form the edges of a logic graph which is displayed in Figure 17.

The following section deals with the logic SoftFOL, the comorphisms translating into it, and the connection of different ATP systems to ease proving of theorems in large TBoxes.

4.2 Proof Support for Casl with Automated Theorem Proving Systems

In [KL-9] the author has introduced a new logic SoftFOL (softly typed first-order logic) aiming at the usage of automated theorem proving (ATP) systems for Casl. In the beginning, only the semi-automatic theorem prover Isabelle [58] was connected to Hets for discharging Casl's proof obligations. Although Isabelle is very powerful, it comes with the disadvantage that detailed technical knowledge is required of how to write proof tactic scripts. This can be very tedious, especially in proofs that use only some automatic proof tactics. Also, Isabelle/Isar scripts use a higher-order logic with two logical levels; most tactics require that the quantification has a special form, e.g. no explicitly quantified variables. For example, the verification of Cohn's Region Connection Calculus (RCC) composition tables [75] in Casl involves 112 goals which are provable with ATP systems. If only Isabelle had been available, 112 individual proof scripts would have been needed; each script would have used different axiom subsets of the theory's axiom set, which are specific to each goal. Moreover, the connection of several advanced ATP systems with Hets allows on the one hand for higher confidence in the proofs if different ATP systems determine the conjecture as theorem, and on the other hand allow for the usage of the most suitable ATP system.

The ATP system Spass [72] is available under open source licensing (GPL) since 1991. It is developed at the Max Planck Institut für Informatik in Saarbrücken, Germany by Christoph Weidenbach, Thomas Hillenbrand, Dalibor Topić et.al. Spass has been considered as an ATP system suitable for Casl, because its input language DFG offers subsorting and a (freely) generatedness concept like Casl (see Section 2.2). However, in DFG sorts may be empty, whereas they are always non-empty in Casl; this is overcome by adding an axiom. The generatedness properties are only used for the ordering of symbols by Spass, but not for proofs by induction. Spass is a saturation based ATP system, which uses elaborated FOL theorem proving methods, such as superposition calculus, specific inference and reduction rules for sorts, and splitting rules for explicit case analysis. Moreover, it implements a sophisticated CNF translation. The DFG format provides syntax for the definition of theories in fully sorted first-order logic with equality, subsorts, and generatedness properties for sorts.

Furthermore, the MathServe system [76] developed by Jürgen Zimmer has been considered as proof support for Casl, because it provides a unified interface to a range of different ATP systems; the most important systems are listed in Table 4, along with their capabilities. These capabilities are derived from the *Specialist Problem Classes* (SPCs) defined upon the basis of logical, language and syntactical properties by Sutcliffe and Suttner [67]. The classes "effectively propositional" and "real first-order" apply to first-order problems that are distinguished by the finiteness of the Herbrand universe; an effectively propositional problem has only constants (generated by finitely many terms) whereas a real first-order problem contains true functions with an infinite Herbrand universe. Also, a brokering service is available as Web service from the MathServe system, which chooses the most appropriate ATP system upon a classification based on the SPCs, and on a training with the library Thousands of Problems for Theorem Provers (TPTP) [76]. Note that the abbreviation TPTP is used for both the library of problems and its logical format. The TPTP format has been introduced by Sutcliffe and Suttner for the annual competition CASC [68] and provides a unified syntax for untyped FOL with equality, but without any symbol declaration. The ATP systems are offered as Web Services using standardised protocols and formats such as SOAP, HTTP and XML. Currently, the ATP system Vampire may be accessed from Hets via MathServe; the other systems are only reached after brokering.

The logic SoftFOL has been designed along the DFG format which is used as input language by Spass. Later SoftFOL was also adapted to cover TPTP, the input language

ATP System	Version	Suitable Problem Classes[a]
DCTP	10.21p	effectively propositional
EP	0.91	effectively propositional; real first-order, no equality; real first-order, equality
Otter	3.3	real first-order, no equality
SPASS	2.2	effectively propositional; real first-order, no equality; real first-order, equality
Vampire	8.0	effectively propositional; pure equality, equality clauses contain non-unit equality clauses; real first-order, no equality, non-Horn
Waldmeister	704	pure equality, equality clauses are unit equality clauses

[a] The list of problem classes for each ATP system is not exhaustive, but only the most appropriate problem classes are named according to benchmark tests made with MathServe by Jürgen Zimmer.

Table 4: ATP systems provided as Web services by MathServe

of MathServe, since DFG and TPTP are very similar with respect to the allowed logical constructs. However, TPTP does not provide any declaration of symbols, whereas the DFG format allows for the declaration of symbols, and SPASS demands at least fixing the kind (sort, predicate or function) and the arity of predicates and functions for an untyped usage. The possibility to define a signature along with sorts, subsort relation and type profiles has been a further reason for investigating whether SPASS can be used as an ATP system for CASL. The three kinds of symbols form pairwise disjoint sets, and overloading is only allowed if the arity is the same. Another restriction inherited from DFG is a locally filtered subsort relation. However, SPASS uses sorts as special unary predicates during reasoning and translates type profile declarations and subsorting declarations into axioms; HETS does the same while generating TPTP. Note that SOFTFOL only provides an abstract syntax and the DFG format is used as default for displaying theories in the HETS GUI.

Furthermore, SPASS turns formulas quantified over sorted variables into implications where the antecedents are formed by applications of the unary sort predicates; again for TPTP the same transformation is achieved by HETS. So, SPASS does not provide any static type checking like HETS; but no erroneous reasoning due to ill-typed formulas arises, since only formulas statically checked by HETS are fed into the ATP systems. Compared to CASL, SOFTFOL offers the common FOL constructs, except for the CASL specific constructs such as conditional terms, unique existential quantifications and subsort projections. Additionally, SOFTFOL has the same generatedness axioms as CASL for declaring generated and freely generated sorts in DFG, which is a further motivation for investigating SPASS as an ATP system for CASL — the generation of TPTP ignores generatedness axioms.

To reach SOFTFOL from full CASL several comorphisms are needed, since SOFTFOL can only be translated from a sublogic of CASL with at least a locally filtered or discrete subsort relation and without partiality. Figure 21 presents the comorphisms involved and the syntax translations of SOFTFOL into its syntactical representations DFG and TPTP. The comorphism marked by a star is discussed in detail later in this section and has been published in [KL-9], but first the other comorphisms are presented briefly. Since SOFTFOL has no partiality, it needs to be coded. This is done completely among CASL's sublogics as shown in the upper part of Figure 21. $SulPCFOL^{=}$ and $SulCFOL^{=}$ are derived from the CASL logic $SubPCFOL^{=}$ and its sublogic $SubCFOL^{=}$ respectively by allowing only a locally filtered subsort pre-order. Recall that a pre-order is locally filtered if each connected pair has an upper bound. $PCFOL^{=}$ is many-sorted partial first-order logic with sort generation constraints and equality; $SubPCFOL^{=}$ adds subsorting to the former sublogic; both sublogics have been introduced by [10]. Sort generation constraints are used to allow the usual induction scheme for datatypes such as natural numbers and lists.

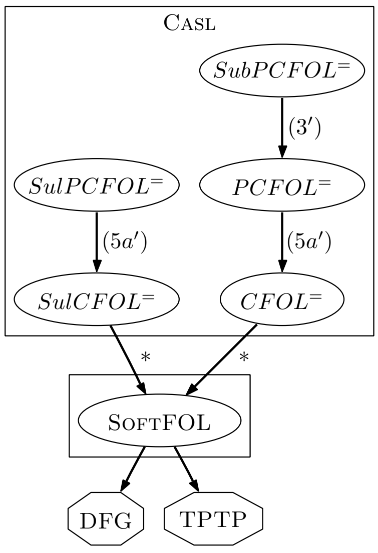


Figure 21: Relation of logics, sublogics and translations connected with SOFTFOL

If locally filtered subsorting and partiality are used in a specification, partiality is coded by a comorphism described as translation $(5a')$ in [52]. However, the distinction between a locally filtered subsort relation and an unrestricted subsort relation is not covered there, but it is not an essential difference. Even if the subsort relation is not locally filtered, a comorphism can be used for coding the subsort relation into explicit embedding and projection functions, which has been introduced in [52] as translation $(3')$. Afterwards the translation $(5a')$ is used to code partiality, since it does no harm if no subsort relation is present. Actually, Figure 21 shows all composed comorphisms leading to SOFTFOL, which are tried from the shortest "$*$" to the longest "$(3') \circ (5a') \circ *$", when the list of known provers (cf. Figure 20) is used for proving. The comorphism $(5a')$ translates the partial functions, including partial subsort projections, by introducing bottom elements, definedness predicates and total projection functions. The $*$ translation represents the comorphism translating $SulCFOL^=$ into SOFTFOL, which can also translate every sublogic of $SulCFOL^=$ such as $CFOL^=$. This comorphism and also the optimisations, introduced after meeting the developers of SPASS, is explained in greater detail below.

The comorphism translating $SulCFOL^=$ into SOFTFOL takes care of renaming all CASL symbols into unique symbols since overloading is only allowed for symbols of the same kind (sort, pred, op) and arity in SOFTFOL. All CASL symbols are qualified with their kind and arity, e.g. sort symbol s is translated into `sort_s`. For every sort an axiom is added stating non-emptiness. As an optimisation the only sort of a single sorted theory is omitted, since it adds no additional information, but it would add a superfluous sort symbol, used as a special predicate, to the reasoning process. Subsorting constructs of CASL like injections are deleted, except those in sort generation constraints. All binary predicates, which are equivalent to the built-in equality of CASL, are removed from the signature. In the sentences the predications of the removed equality-predicates are substituted with applications of the built-in equality. This is also an optimisation found in the discussion with the developers of SPASS. Furthermore, the coding of unique existential quantification has been revised after the discussions with the developers of SPASS: A unique existential quantification is turned into a conjunction where first the existence of such an individual is stated, and next it is stated that there exists only the individual of the first conjunct fulfilling the formula.

After some experiments with theories consisting of many sentences such as the verification of the RCC composition tables, it turned out that a high number of sentences and certain kinds of sentences lead to state explosion in the ATP systems, which significantly slows down

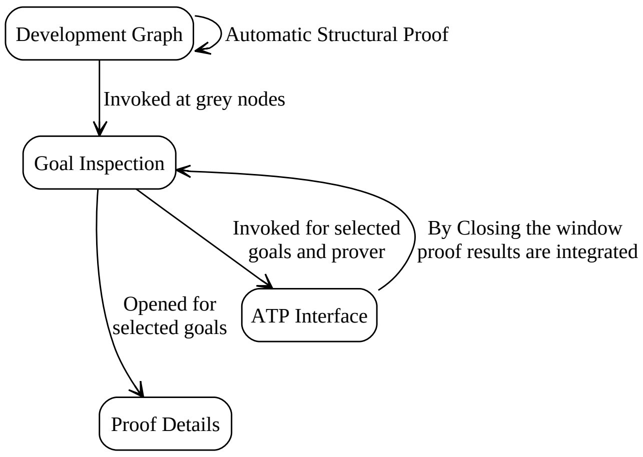
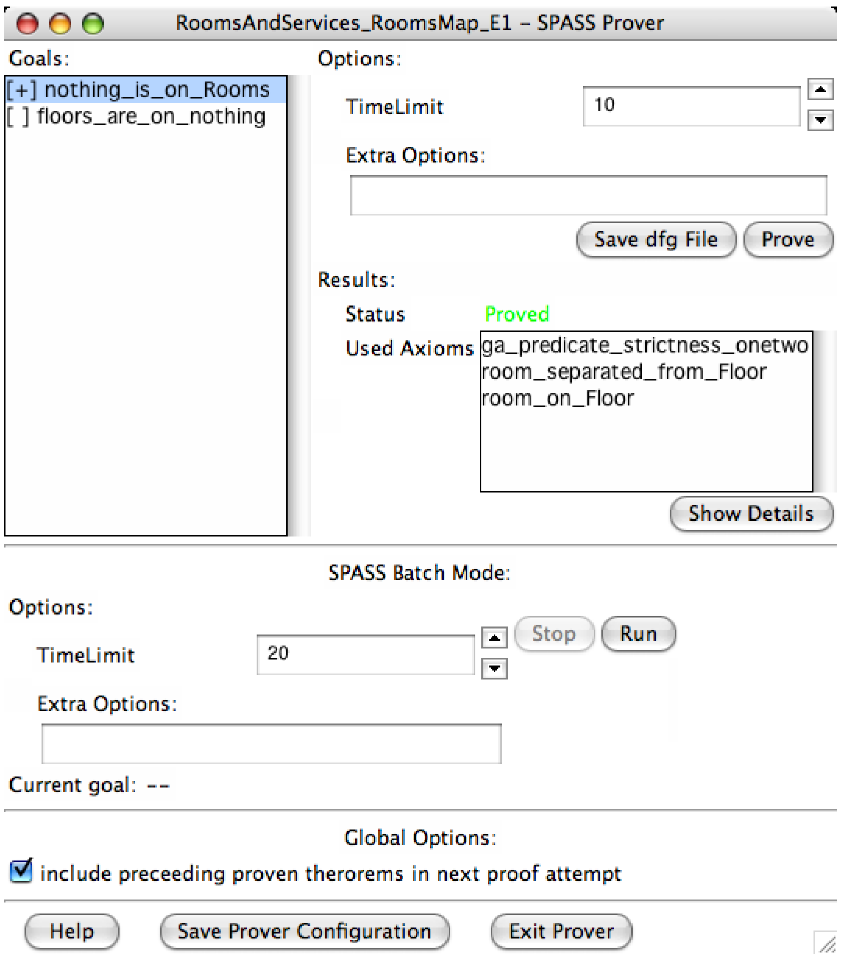

Development Graph
Automatic Structural Proof
Invoked at grey nodes
Goal Inspection
Invoked for selected goals and prover
By Closing the window proof results are integrated
Opened for selected goals
ATP Interface
Proof Details

Figure 22: Proof work-flow used for ATP proofs in HETS

the process of proving. Problematic sentences are e.g. commutativity and symmetry axioms. For this reason it is possible to select the sentences that form the theory sent to the prover using the two lists of sentence labels in the bottom half of the proof management window (cf. Figure 20). Although the semantics of such a theory shrunken by selection of axioms may be weaker, it is not possible to prove the entailment of a conjecture which is not a theorem of the complete theory. If a conjecture is disproved by a shrunken theory, the proof is rejected, since the goal might still be an entailment of the complete theory. Another optimisation of the theory sent to the prover removes symbols not used in the selected theory from the signature, which again reduces the number of sentences used during reasoning, since typing is turned into axioms.

A typical work-flow with ATP systems from HETS is shown in Figure 22. The three ATP systems (SPASS, Vampire and MathServe Broker) instantiate the same generic ATP interface which is shown in Figure 23. However, the Broker interface does not provide a field for extra options, since these are specific to each reasoner. Also, the generic reasoner interface allows for easier integration of further ATP systems into HETS, since only the implementation of calling the additional reasoner needs to be provided according to a unified set of functions which are used by the generic graphical interface — the interface implementation itself does not need to be adapted.

Consistency checking with SPASS is provided by generating theory files for each specification. These files contain the sentence "*false*" as conjecture, and if SPASS finds a proof, the theory is inconsistent and the list of used formulae provides a hint to the cause. If SPASS disproves the conjecture "*false*", the theory is consistent. If it runs into a timeout with a limit of t seconds it is called to be t-consistent. Most inconsistent theories have been discovered with a value for t of 500 or 1000 seconds before the time limit was reached by testing with theories from the basic datatypes published in [10]. So, by choosing a reasonably high time limit, a high confidence in the consistency of the theory is reached.


RoomsAndServices_RoomsMap_E1 – SPASS Prover

Goals:

[+] nothing_is_on_Rooms
[ ] floors_are_on_nothing

Options:

TimeLimit 10

Extra Options:

Results:

Status Proved

Used Axioms ga_predicate_strictness_onetwo
room_separated_from_Floor
room_on_Floor

SPASS Batch Mode:

Options:

TimeLimit 20

Extra Options:

Current goal: --

Global Options:

☑ include preceeding proven therorems in next proof attempt

Figure 23: ATP interface instantiated for SPASS

4.3 Summary

The tool HETS supports the development of structured and strongly typed ontologies with CASL by providing static analysis at the levels of structured and basic specification. Furthermore, it allows to inspect the import structure of the ontologies and also the inspection of taxonomy graphs. But most important is the proof support for CASL offered by HETS through interfacing to ATP systems and Isabelle. ATP systems allow us to discharge the simpler theorems automatically, such that the ontology engineer can concentrate on proofs of the complicated theorems with Isabelle. Additionally, HETS enables consistency checks of the ontologies. So, combined with the Emacs mode for editing the ontology specification, HETS is a reasonable tool for developing foundational ontologies such as DOLCE in the expressive FOL language CASL. The full integration of CASL-DL, OWL DL and approximation of FOL theories (see Section 3) into HETS will offer access to domain ontologies written in DL.

5 Examples of Ontologies in CASL

This section introduces some examples where CASL has been applied as an ontological specification language. Also, HETS has been applied for the analysis of the CASL specifications, for the generation of output formats, and for discharging theorems to ATP systems. All presented examples have been developed within the context of the collaborative research center SFB/TR 8 "Spatial Cognition". The example presented in Section 5.3 is also the result of an international cooperation with the Laboratory for Applied Ontology (LOA) in Trento, Italy. The example in Section 5.1 explains wayfinding as metaphor for navigating the Web; Section 5.2 introduces the rather abstract and unrestricted concept Route Graph and describes different instantiations of the concept. Furthermore, Section 5.4 shows the approximation relation between two alternative Route Graph specifications.

5.1 Ontology for Wayfinding

Hochmair investigated the applicability of metaphors [29] such as navigation in the context of the World Wide Web (WWW) by writing specification-like Haskell programs, which simulate navigation in the real world and in the WWW. In [KL-2] Hochmair and the author abstracted these Haskell programs into CASL specifications and derived a formal specification of the concept[13] "navigate" from various definitions. Figure 24 gives an overview of the conducted work to achieve a formalisation of the "Navigation" metaphor for the WWW.

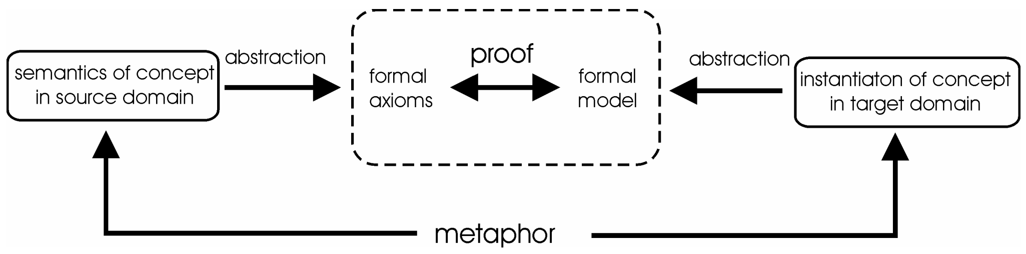

semantics of concept in source domain
abstraction
formal axioms
proof
formal model
abstraction
instantiaton of concept in target domain
metaphor

Figure 24: Research method for explaining the navigation metaphor formally[14]

The application of spatial concepts as metaphors for human-computer interaction (HCI) has been studied extensively e.g. in [66]. Features of the physical world are mapped to a computational domain, which enables the user of the computer to apply her experience of the real world to the computational domain. Maglio and Matlock [46] conducted a study whether Web users comprehend the Web as physical space. Their result is that experienced Web users refer to the Web as it were a two dimensional physical space or a physical container. They applied extensively spatial phrases such as "I couldn't get back to where I was" or "Yahoo had what I wanted" to the non-physical Web space. Using metaphors for HCI helps to master the abstract nature of information systems [41]. Montello [50] claims that navigation is a versatile concept, which can be applied to various domains, including many which are not intrinsically spatial. The approach of [KL-2] is to define an axiomatic formalisation of a concept in some source domain, e.g. navigation in a spatial environment, and to establish evidence for the metaphor by proving these axioms within the target domain, WWW in this case.

For the investigation of the applicability of the navigation metaphor, two scenarios have been chosen based on previous work of Hochmair. In [30] a Web agent has been simulated, searching for some Web shop providing running shoes with certain properties. Marchionini

[13]The term "concept" is used here in a broader sense than in DL; it denotes a whole specification in the context of Section 5.1.

[14]Figure 24 has been published as Figure 1 in [KL-2, p. 237].

[48] describes navigating as one of four browsing strategies. The navigation strategy is applied by following some of the possible links on a Web page in order to reach the goal. The real world scenario is based on the vector based *least-angle* strategy for wayfinding discussed also in [31]. The agent finds its goal by deciding at each intersection of a street network which street has less deviation from the target vector than the others. This can be achieved either by direct sensing or dead reckoning [44], where the latter is needed in city environments where the target is not always visible. Furthermore, the agent lacks any other knowledge about its environment.

The formalisation of metaphors by algebraic specifications follows the approaches taken in e.g. [42, 24]. Structure preserving morphisms are used to demonstrate the metaphorical mapping of the concept in the source domain onto the target domain in [42] by using Haskell type classes. But no axioms of the source domain are specified, since Haskell type classes only allow for signature definitions. Thus the morphisms are only signature mappings. The usage of Casl in [KL-2] develops this approach towards formal proofs of the morphisms by defining Casl views. Figure 25 shows the relation and origin of the Casl specifications used to formalise the concept "navigate". At first, definitions of the concept navigate are reviewed and turned into a Casl specification of the key properties of navigation. In a second step, two Haskell programs are used to simulate the key properties in a real world scenario and a Web scenario. In a third step, the Haskell programs are abstracted into Casl specifications of the key functions used for the simulation. The definition of the Casl views relating the key properties in Casl with the specifications derived from the simulations finalises the approach. Due to limitations in Hets regarding the hiding of symbols, the views only provide signature morphisms and no proofs have been carried out.

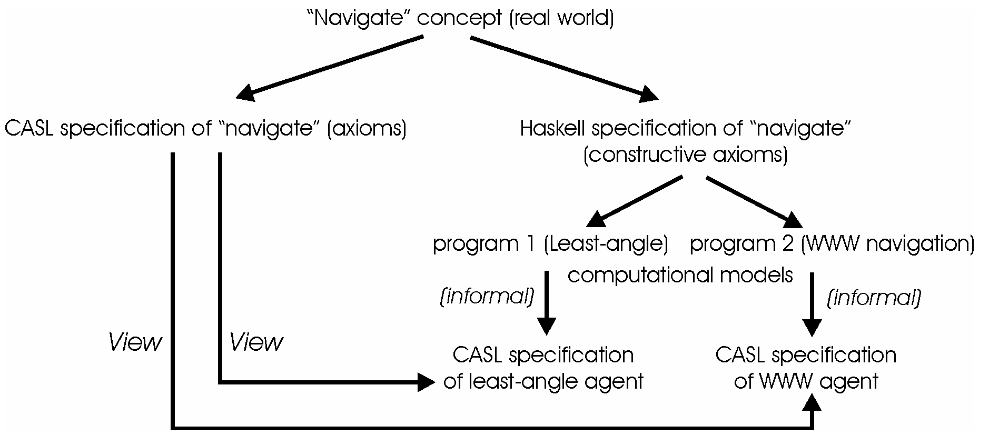

"Navigate" concept (real world)
CASL specification of "navigate" (axioms)
Haskell specification of "navigate" (constructive axioms)
program 1 (Least-angle)
program 2 (WWW navigation)
computational models
(informal)
(informal)
View
View
CASL specification of least-angle agent
CASL specification of WWW agent

Figure 25: Relation of Casl specifications and views, and Haskell programs[15]

The semantics of navigation and wayfinding used to define the key axioms of the concept "navigate" are taken from different definitions found in the Geographical Information Science (GIS) and ontology communities. The lexical definitions found in the GIS literature provide two distinct views on the relation of navigation and wayfinding, where both treat them as processes. On the one hand, Golledge [26] and other authors assign different properties to wayfinding and navigation; their conclusion is that both are independent processes. On the other hand, navigation is treated as a process composed by locomotion and wayfinding [11, 15]. Furthermore, WordNet with its relation to SUMO and Dolce Lite+ (as provided by the OntoWordNet project [20]) are considered for formal definitions of wayfinding and navigation, although these definitions are merely taxonomic.

Figure 26 gives an overview of the axioms derived from OntoWordNet descriptions leading to the definition of the concept "navigate". The ground axioms are derived from the term

[15]Figure 25 has been published as Figure 4 in [KL-2, p. 245].

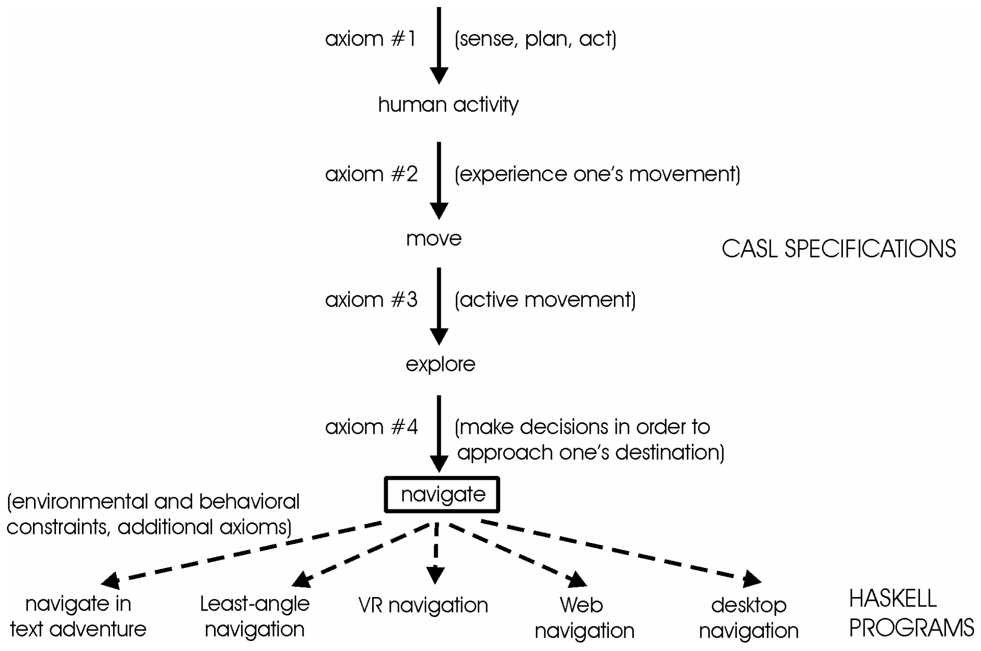


Figure 26: Development of concept "Navigate"[16]

agent as conceptualised by Russel and Norvig [61]. Furthermore, beliefs and reality are distinguished, where beliefs may not be reflecting the reality of the environment. The agent acts in a Sense-Plan-Act (SPA) cycle obtaining the decomposition of an agent's control functions as defined by Nilson [57].

In [KL-2] HumanActivity is defined in Casl in terms of an SPA cycle where the beliefs of the agent are stored in a stategraph. Based on perception, the stategraph is adapted and a decision is planned, which leads to an effect on the environment. Additionally, at all discrete time steps of the *World* (consisting of environment and agent) an *environmentalAction* may change the environment, too. Figure 27 shows the Casl specification and the key axiom defining the function *worldStep*. The remaining axioms are defined by instantiation of a graph specification for the sort *StateGraph* and by providing abstract definitions of predicates in terms of world steps and changes on the *StateGraph* for moving, exploring and navigating. Movement includes exploration, but may be controlled by the environment. Exploration is a movement triggered by the agent and includes navigation. Navigation only takes place if the agent decides for an option which takes the agent closer to the goal. The specifications in the lower half of Figure 27 illustrate the axioms involved (further details may be found in [KL-2]).

The specifications derived from the Haskell programs use almost the same approach regarding the agent moving around an environment, but the definitions of the functions are constructive, acting on concrete datatypes. For example, each world in the simulation is stored as a trace list. The structuring facilities of Casl are used extensively to achieve the reuse of shared semantics between the least-angle agent and the WWW agent. For example, the definition of iterating world steps is provided as a generic specification which is instantiated for both agents by appropriate parameter specifications (see Figure 28). The generic specifications in Casl are used to model the type classes which are used in the Haskell simulations for shared signatures. Furthermore, the specification AbsEnvAgent provides the signature of functions and predicates that need to be provided for implementing an agent acting in an environment, called WorldImpl. The specification derived from

[16]Figure 26 has been published as Figure 5 in [KL-2, p. 248].

```
spec HumanActivity =
     sorts Environment, Perception, Decision, Agent, StateGraph
     type  World ::= World(Environment; Agent)
     ops   updateState : Decision × Agent → Agent;
           perception : Environment × StateGraph → Perception;
           plan : Perception × StateGraph → Decision;
           physicalAction : Decision × Environment → Environment;
           environmentalAction : Environment → Environment;
           graph : Agent → StateGraph;
           worldStep : World → World
     ∀ a1, a2 : Agent; w1, w2 : World; e1, e2 : Environment
     • worldStep(World(e1, a1)) = World(e2, a2)
       ⇔ a2 = updateState(plan(perception(e1, graph(a1)), graph(a1)), a1)
          ∧ (e2 = physicalAction(plan(perception(e1, graph(a1)), graph(a1)), e1)
             ∨ e2 = environmentalAction(
                        physicalAction(plan(perception(e1, graph(a1)), graph(a1)),
                                       e1))
             ∨ e2 = physicalAction(plan(perception(e1, graph(a1)), graph(a1)),
                                   environmentalAction(e1)))
                                                                  %(Axiom #1)%

spec Move = HumanActivity hide environmentalAction
then RichGraph2 [sort MentalPos fit sort Node ↦ MentalPos]
                [sort Transition fit sort Edge ↦ Transition]
     with Graph ↦ StateGraph
then op     mpos : Agent → MentalPos
     . . .
     • hasMoved(World(e1, a1), World(e2, a2))
       ⇔ worldStep(World(e1, a1)) = World(e2, a2)
          ∧ isMoveTransition(getEdge(graph(a2), mpos(a1), mpos(a2)))
                                                                  %(Axiom #2)%

spec Explore = Move
then . . .
     • hasExplored(World(e1, a1), World(e2, a2))
       ⇔ worldStep(World(e1, a1)) = World(e2, a2)
          ∧ hasMoved(World(e1, a1), World(e2, a2))
          ∧ isSelfMoveTransition(getEdge(graph(a2), mpos(a1), mpos(a2)))
                                                                  %(Axiom #3)%

spec Navigate = Explore
then . . .
     • navigate(World(e1, a1), World(e2, a2))
       ⇔ worldStep(World(e1, a1)) = World(e2, a2)
          ∧ (hasExplored(World(e1, a1), World(e2, a2)) if getEDegree(e1) > 0)
          ∧ isdecisionPoint(mpos(a1), graph(a1))
          ∧ (approach(World(e1, a1), World(e2, a2)) if getEDegree(e1) > 0)
                                                                  %(Axiom #4)%
```

Figure 27: Navigate specification based on HumanActivity in Casl

the Haskell simulation is used as parameter for WayfindSig, which provides the needed observer predicates for mapping the metaphor specification Navigate to the simulations. The specification WayfindSig is a further example of a shared specification which is used for both simulations. Of course, the definitions are specific to each simulation.

```
spec AbsEnvAgent =
     sorts AbsEnv, AbsAgent, AbsMPos
then ops   plan, updateHistStateGraph : AbsAgent → AbsAgent;
     ...

spec WorldImpl[AbsEnvAgent] given Nat =
     free type World ::= World(AbsEnv; AbsAgent)
     op    worldStep : World → World
     ...

spec WorldLA =
     WorldImpl [LeastAngleAgent fit...]

spec WorldLAObserver =
     WayfindSig [WorldLA fit ...]
then ...
```

Figure 28: Shared implementation of world iteration and instantiation for least-angle agent

The verification of the metaphorical mapping is done in two steps:

1. the signature mapping between the definition of the metaphor and the simulations is shown formally to be a view in Casl;
2. the behaviour of the two simulations is compared by checking similarities in the traces.

Since the second verification step has already been published by Hochmair, the first step is explained in greater detail here. Hets has been used to statically type-check the specification library and the signature morphisms provided by the views which formalise the mapping of the metaphor specification to the source and target domains (see Figure 29). Due to extensive usage of hiding in the structuring of the specifications, the development-graph calculus implemented by Hets cannot determine the proof obligations generated by the views. However, since the modelling of the simulations and the abstract metaphor are very similar, the proofs for the source and target domain of the metaphor should be easily obtained once Hets can provide the obligations correctly.

```
view Wayfind_in_WorldLA : Navigate to WorldLAObserver =
     sorts World ↦ World, Agent ↦ AgentLA, ...
     preds hasExplored ↦ hasTraveled, navigate ↦ wayfind

view Wayfind_in_WorldWWW : Navigate to WorldWWWObserver =
     sorts World ↦ World, Agent ↦ AgentWeb, ...
     preds hasExplored ↦ hasTraveled, navigate ↦ wayfind
```

Figure 29: Views mapping the metaphor specification Navigate to the simulated source and target domains

Summarising the results achieved in [KL-2], the authors provided a formal specification of a behaviour of cognitive agents instantiated for two different domains. The applicability of CASL for the definition of ontological concepts is demonstrated by giving an abstract definition of the spatial concept "navigate" based on ontological theories, and by mapping this definition to a simulation of navigation in the real world and to a simulation of navigating the Web. A further step would be to use heterogeneous specification for such a verification. On the one hand this could lead to a formal view between the Haskell simulations and their CASL counterparts, and on the other hand MODALCASL could be used for the specification of the agents behaviour without direct references to the world steps.

5.2 Route Graph Ontologies

Route Graphs have been introduced by Werner et.al. [73] for the description of knowledge about structured spatial environments such as cities, buildings or harbours. The general idea is to cover the knowledge as graph of directed route segments which connect decision points, called places, along a route. Figure 30 shows some spatial scenarios where Route Graphs are used for navigation. The original description of Route Graphs in [73] was rather informal, in prose only, allowing different interpretations. The main focus was to introduce a common terminology for the description of navigational knowledge for different agents such as humans, animals or robots which act in spatial environments.

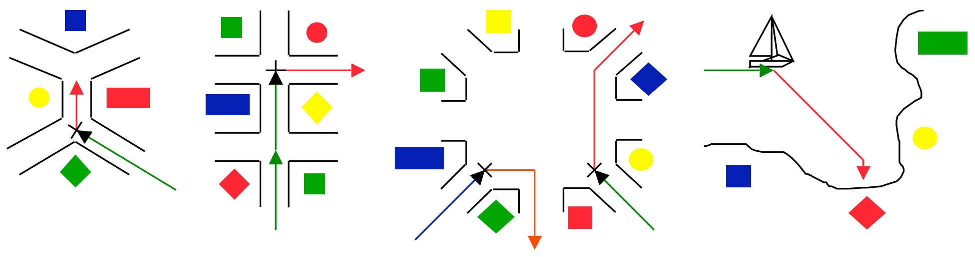


Figure 30: Route Graph example scenarios[17]

In [KL-3] Route Graphs are studied in different ontological formats such as CASL and MMiSSLaTeX [38, 39], and in the functional programming language Haskell [36]. The author focuses on the CASL specifications and covers briefly the derived Haskell implementations, which are not intended as running programs but merely as design patterns for the derivation of programs from ontological CASL specifications. The ontology specification of Route Graphs extensively uses generic specifications and has been restricted to binary predicates allowing easier connection to CASL-DL, although it has not been checked whether it obeys the restrictions of CASL-DL (cf. Section 3.1). Furthermore, the design specification of Route Graphs reuses datatypes such as lists and graphs defined in the basic CASL libraries (cf. [10, Part V]).

The concept Route Graph specialises the notions of *node* and *directed edge* taken from graph theory into the notions of *place* and *route segment* respectively. A place in a Route Graph holds enough information for the decision about where the agent currently is, and which route segments originate from here. This is usually information about the surrounding route-marks or landmarks forming a reference system local to each place. The possible information attached to route segments is further divided into *entry*, *exit* and *course*. The deviation of a route segment's direction from the direction given by the reference system of a place is stored in the entry and exit information, whereas the course information might characterise the trajectory of a route segment in terms of e.g. width, curves or route-marks.

Furthermore, two specific sequences of route segments are defined: (1) *paths* collect route segments which are connected to each other and describe a path through a Route Graph where some route segments and places may be traversed more than once such as in car races; (2) *routes* are special paths where each place is visited at most once while travelling along a

[17]Figure 30 has been published as Figure 2 in [KL-3, p. 392].

route. Different Route Graphs are distinguished by their *kind*; this allows for e.g. pedestrian, robot or tramway Route Graphs, which may be related by special route segments called *transfer segments.* Heterogeneous routes are formed by using transfer segments. Another relation of two Route Graphs is described by abstraction, where Route Graphs of different granularities are related by an abstraction function.

The most difficult task in the context of Route Graphs is the integration of routes into one Route Graph which is called *route integration* or *place integration.* For each place, it must be decided whether it is a new one or whether it corresponds to some existing place. This is achieved by integration of the reference systems of two places on (different) routes. If the integration is successful, only one unified reference system is attached to the place, and for each incoming and outgoing route segment, exit and entry information is recalculated according to the new united reference system. This notion of route integration is a very abstract description of a part of the SLAM (Simultaneous Localisation And Mapping) problem [19, 51] in the field of autonomous robot exploration.

Ontological Route Graph specifications. The specification of ontologies for Route Graphs in [KL-3] starts with light-weight ontologies in MMiSSLaTeX and the derived CASL specifications which provide merely the taxonomy and very few axioms in CASL. Figure 31 presents the generic specification of a relational graph, which also provides the sequence types path and route. The sort *Kind* serves as marker distinguishing different Route Graphs when specification GENGRAPH is instantiated for distinct agents or environments.

```
spec GENGRAPH [sort Kind] =
       sorts  Graph[Kind], Node[Kind], Edge[Kind]
       preds  hasNode : Graph[Kind] × Node[Kind];
              hasEdge : Graph[Kind] × Edge[Kind]
       ops    hasSource, hasTarget : Edge[Kind] → Node[Kind]
       ∀ e: Edge[Kind]; s, t: Node[Kind]; g: Graph[Kind]
       • hasEdge(g, e) ∧ hasSource(e) = s ∧ hasTarget(e) = t ⇒
         hasNode(g, s) ∧ hasNode(g, t)
then
       GENSEQUENCE [sort Edge[Kind]] with hasHead, hasTail, EmptySeq, freq
then . . .
       op     sources : Sequence[Edge[Kind]] → Sequence[Node[Kind]]
       . . .
       pred   connected(l: Sequence[Edge[Kind]]) ⇔
              (∀ e1, e2: Edge[Kind]
               • hasHead(l) = e1 ∧ hasHead(hasTail(l)) = e2 ⇔
                 hasTarget(e1) = hasSource(e2)) ∧ connected(hasTail(l))
       sorts  Path[Kind] = {l: Sequence[Edge[Kind]] • connected(l)};
              Route[Kind] = {p:Path[Kind]
                                   • (∀ n: Node[Kind] • freq(sources(p), n) ≤ 1) ∧
                                     (∀ e: Edge[Kind] • freq(p, e) ≤ 1)}
```

Figure 31: GENGRAPH provides the starting point for Route Graph specifications

The design specification for the implementation of GENGRAPH is presented in Figure 32. The specification GRAPH2 is also parameterised with a sort *Kind*, but it is based on already published datatypes (cf. [10, Part V]) with constructive axioms which allow for the derivation of functional programs in Haskell. Both Figure 31 and Figure 32 are taken from [KL-3]. After checking again the involved CASL libraries, the author revised the definition of the predicate *connected* as presented in Figure 31, because it does not reflect the whole coverage of the predicate *connected* as defined in the design specification; i.e. the cases of empty list and single element list have not been included. Moreover, the original definition has been

```
from Basic/StructuredDatatypes get List
from Basic/Numbers get Nat
from Basic/Graphs get NonUniqueEdgesGraph

spec Graph2 [sort Kind] given Nat =
      sorts  Node, Edge
      ops    source, target : Edge → Node
then
      NonUniqueEdgesGraph [sort Node] [sort Edge] and
      List2 [sort Edge] with [], __::__, atMostOnce
then
      pred   connected : List[Edge]
      ∀ rs1, rs2: Edge; rsl: List[Edge]
      • connected([])                                    %(connected_empty)%
      • connected([rs1])                                 %(connected_singleton)%
      • connected(rs1 :: rs2 :: rsl) ⇔
        target(rs1) = source(rs2) ∧ connected(rs2 :: rsl)  %(connected_rec)%
then ...
      op     sources : List[Edge] → List[Node]
      ...
      sorts  Path = {l: List[Edge] • connected(l)};          %(def_path)%
             Route = {l: Path • atMostOnce(sources(l)) ∧ atMostOnce(l)}
                                                             %(def_route)%
```

Figure 32: Design specification of graphs tagged with a *Kind*

too strong, i.e. every two segments e_1, e_2 with $hasTarget(e_1) = hasSource(e_2)$ would have implied that they are the first and second element of every connected list (see Figure 33 for the revised definition of pred *connected*).

Furthermore, Appendix C.1 presents the complete graph specifications including the revised specifications GenSequence and List2. In addition to the revised specifications of [KL-3], the relation of the specifications GenGraph and Graph2 is formalised as a view (see Figure 34, and Appendix C.1 for the complete signature mapping). Most conjectures arising from the view are provable automatically with Spass. Only the definition of *connected* (as show in Figure 33) and the definition of *atMostOnce* (labelled with %(atMostOnce_def)% in Appendix C.1), which arise from the view as goals, need to be proved with Isabelle.

```
pred connected(l : Sequence[Edge[Kind]]) ⇔
          l = EmptySeq
          ∨ (∃! e1 : Edge[Kind] • hasElem(l, e1))
          ∨ ((∀ e1, e2 : Edge[Kind]
                • head(l) = e1 ∧ head(tail(l)) = e2 ⇒ target(e1) = source(e2))
             ∧ connected(tail(l)))
```

Figure 33: Revised definition of connected sequences of edges

```
view GRAPH2_IMPLEMENTS_GRAPHONT :
        GENGRAPH[sort k fit Kind ↦ k] to
        {GRAPH2[sort k fit Kind ↦ k]
         then %def
            preds hasEdge(g : Graph; e : Edge) ⇔ e ϵ g;              %(hasEdge_def)%
                  hasNode(g : Graph; n : Node) ⇔ n ϵ g;              %(hasNode_def)%
                  hasElem(l : List[Edge]; e : Edge) ⇔ e ϵ l;  %(hasElem_Edge_def)%
                  hasElem(l : List[Node]; n : Node) ⇔ n ϵ l  %(hasElem_Node_def)%
        } = ...
```

Figure 34: View GRAPH2_IMPLEMENTS_GRAPHONT

Route Graph for an Indoor Scenario. The instantiation of the Route Graph specification in an indoor environment for the Bremen autonomous wheelchair Rolland [47] has been informally introduced in [KL-3]. The general idea is to abstract a Voronoi-based graph representation [71] into a Route Graph. The Route Graph is used for the navigation and linguistic communication with the Rolland wheelchair. However, the generic approach and the taxonomy of indoor kinds have complicated the situation rather than clarified. The implementation of the generic specifications as generic programs has led to complex datastructures and complex derived XML serialisations that were not usable as easy exchange format, because almost each individual segment ended up with its own kind, such as doorway, corridor, or lift kind. Recent developments will lead to a Route Graph specification that is homogeneous; qualitative and quantitative data is attached via option types to places and route segments. Furthermore, a specification of an abstract reference system for places [40] has been introduced after publication of [KL-3].

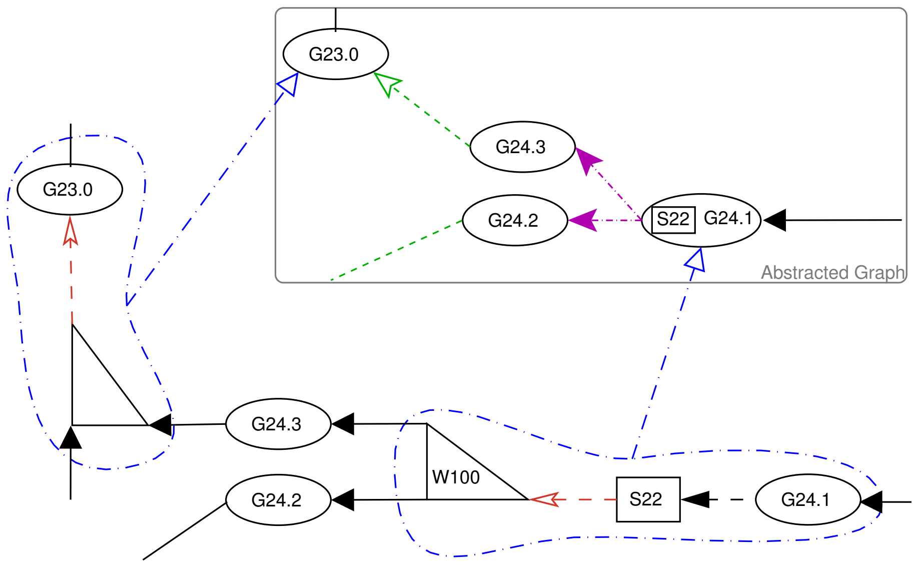


Figure 35: Abstraction from the tramway devices Route Graph

Abstraction of Fine Grained Route Graphs. The formal specification of place abstraction has been published in [KL-6] whereas in [KL-3] only informal figures illustrate place and route abstractions. Figure 35 displays an example of place abstraction, where some devices located together are composed to an abstract node (Figure 35 has been published in [KL-6]). The abstraction serves the goal of information reduction, i.e. the detailed planning of the tramway network needs information about every device, whereas the safety

```
spec PlaceAbstraction [GenGraph[sort Kind1]] [GenGraph[sort Kind2]]
      given Nat =
      ops   abstractPlaces : Graph[Kind1] → Graph[Kind2];
            mapNode : Node[Kind1] → Node[Kind2];
            mapEdge : Edge[Kind1] →? Edge[Kind2]
      ∀ e_K1 : Edge[Kind1]; e_K2 : Edge[Kind2];
        n_K1 : Node[Kind1]; n_K2 : Node[Kind2];
        g_K1 : Graph[Kind1]; g_K2 : Graph[Kind2]
      • def mapEdge(e_K1) ⇔
        ¬ mapNode(source(e_K1)) = mapNode(target(e_K1))
      • g_K2 = abstractPlaces(g_K1)
        ⇒ (hasEdge(g_K1, e_K1) ∧ def mapEdge(e_K1)
            ⇒ source(mapEdge(e_K1)) = mapNode(source(e_K1))
               ∧ target(mapEdge(e_K1)) = mapNode(target(e_K1))
               ∧ hasEdge(g_K2, mapEdge(e_K1)))
          ∧ ∃ n1, n2 : Node[Kind1]; e1, e2 : Edge[Kind1]
            • mapNode(n1) = mapNode(n2) ∧ hasNode(g_K1, n1)
               ∧ hasNode(g_K1, n2) ∧ hasEdge(g_K1, e1)
               ∧ hasEdge(g_K1, e2) ∧ target(e1) = n1 ∧ source(e2) = n2
               ⇒ ∃ r : Route[Kind1]
                  • pathIn(r, g_K1) ∧ startOf(r) = n1 ∧ endOf(r) = n2
      • g_K2 = abstractPlaces(g_K1) ⇔
        (hasNode(g_K2, n_K2) ⇔
         ∃ n1 : Node[Kind1] • hasNode(g_K1, n1) ∧ mapNode(n1) = n_K2)
        ∧ (hasEdge(g_K2, e_K2) ⇔
           ∃ e1 : Edge[Kind1] • hasEdge(g_K1, e1) ∧ mapEdge(e1) = e_K2)
```

Figure 36: Specification of place abstraction

predicates on the routes only need a simplified Route Graph for the detection of critical situations.

In general the abstraction leads to fewer places, which leads to better computational behaviour of graph algorithms such as shortest path. On the one hand, a route where each intermediate place has only one incoming and outgoing segment, may be abstracted into a single route segment; on the other hand, a connected subgraph can be abstracted to a single place if its nodes share a common feature such as being in the same room or in close distance to each other. For example, the Route Graph of a marketplace might be abstracted into one place on a more abstract layer. Note that the abstraction function still associates the fine grained graph with the abstracted one. The information at the fine grained layer is not discarded, but hidden on the more abstract layer. If a robot needs the detailed information stored at a fine grained layer after calculation of a route at the abstract layer, a subgraph (a connected set of places) is provided for further route planning at a lower level where some outgoing edge of the subgraph is already marked for usage.

Tramway Route Graphs. In [KL-6] an instantiation of the Route Graph specifications for tramways with refinement of the abstraction of detailed layers has been published. Furthermore, an application of a Route Graph representation for the investigation of safety requirements is demonstrated. Figure 36 shows the formal specification of place abstraction, where only abstract requirements regarding the interplay of the functions *abstractPlaces*, *mapEdges* and *mapNodes* are defined, but the concrete implementation depends on the actual graphs given as parameters. For completeness, Appendix C.2 shows the specifications of the two tramway Route Graphs and their relation in terms of an instantiation of PlaceAbstraction. The specifications in Figure 36 and Appendix C.2 have been published in [KL-6].

The application of an ontological representation of tramway Route Graphs serves as a starting point for the definition of safety predicates at the abstract level which is derived via abstraction from the underlying device layer.

5.3 Dolce in Casl

In 2003 a collaboration with the LOA in Trento has been started for sharing knowledge about the Descriptive Ontology for Linguistic and Cognitive Engineering (Dolce) and for using Casl as its ontology specification language. At first, a simplified version of Mereology (as it is part of Dolce) has been defined by instantiation of generic specifications of parthood [KL-8]. The specifications GenMereology and GenParthood as presented in Figure 15 on page 20 and Appendix B.2 have been published in [KL-8].

Furthermore, the development of Dolce is still an ongoing project carried out collaboratively between the LOA and the SFB/TR 8 projects I4-[SPIN] and I1-[OntoSpace] in Bremen. In [KL-1] the quality predicates as defined in [49] are revised and applied for a case study showing the combination of two quality spaces representing space in the framework of Dolce: Mereology and Route Graphs. This case study includes a short spatial scenario, where the wayfinding task has been proved with Spass after fine-tuning of some intermediate theorems. By using the notion of strongly typed ontology and Casl's structuring facilities this case study is a proof of concept for considering Casl as an ontology specification language.

Additionally, the integration of different information sources is pointed out. The notions of strongly typed and structured ontologies are presented at work by applying Casl and Hets for the formalisation and analysis of the highly modularised specifications. This is augmented by presenting a spatial scenario along with the necessary concrete physical qualities, also formalised in Casl. The actual wayfinding task has been formulated by the author in Casl as theorems suitable for applying Spass. The author applied also the concepts of knowledge approximation to the specification of the Route Graph. However, only a weaker FOL theory is used instead of a DL theory. The required view has been proved with Spass and Isabelle, and the weaker version of the Route Graph sped up the reasoning about the actual wayfinding problem. Figure 37 presents the weak Route Graph which enables the modelling of spatial knowledge, which is sufficient for the scenario studied in [KL-1]. A sort of graphs or edges is omitted completely and even the nodes do not provide any reference system, since only the linkage between nodes is necessary for carrying out the wayfinding in the example.

```
spec WeakRouteGraph =
  sort   node
  preds link, path : node × node
  ∀ x, y, z : node
  • path(x, x)                                         %(path refl)%
  • link(x, y) ⇒ path(x, y)                           %(links are paths)%
  • link(x, y) ∧ path(y, z) ⇒ path(x, z)    %(link prefix of path yields path)%
```

Figure 37: Weak Route Graph specification[18]

Figure 38 shows the definition of physical positions (pred *pos*) in quality spaces (sort *s*) which are inhered (pred *inh*) in some arbitrary elements (sort *Elem*). The actual instantiation for physical endurants (sort *PED*) and different quality spaces is presented in Figure 4 on page 7. Partial knowledge about the environment is modelled by demanding only one physical position in spatial quality spaces.

[18]The specification presented in Figure 37 have been published in [KL-1, Section 4.2].

```
spec QualityType[sorts q, Elem] =
       pred  inh : q × Elem                                        %(A1)%
       ∀ x, x′ : q; y, y′ : Elem
       • inh(x, y) ∧ inh(x, y′) ⇒ y = y′                           %(A2)%
       • inh(x, y) ∧ inh(x′, y) ⇒ x = x′                           %(A3)%
       • ∃ y : Elem • inh(x, y)                                    %(A4)%

spec StrongQualityType[sorts q, Elem] =
       QualityType[sorts q, Elem]
then ∀ y : Elem • ∃ x : q • inh(x, y)                            %(A4 a)%

spec QualitySpace[sorts s, q] =
       pred  pos : s × q                                           %(A6)%
       ∀ x, x′ : s; y : q • pos(x, y) ∧ pos(x′, y) ⇒ x = x′      %(A7-A8)%
```

Figure 38: Qualities and physical positions in Dolce [19]

5.4 Route Graph Approximation

Moreover, [KL-1] includes an example for the approximation of theories, which is detailed in this section (cf. Section 3.3 for the notion of theory approximation). The case study started with the necessary and sufficient specification WeakRouteGraph (Figure 37). Later a more datatype-like specification was investigated instead, based on a simple list specification for the representation of paths (see Figure 39 for the list based Route Graph and the approximation view). However, the list-based Route Graph specification prevents that any ATP system connected to Hets finds proofs of the actual wayfinding problem due to the complexity of the list datatype. The less complex specification WeakRouteGraph allows that these proofs are achieved automatically with ATP systems (cf. Section 4.2). The relation of the weak Route Graph specification is an approximation of the list based Route Graph specification and has been made explicit by defining a Casl view, which has been formally verified. Although this approximation is shown completely in Casl, the advantages of more efficient reasoning with approximated theories has been demonstrated. Most theorems can be proved automatically; only one theorem needs to be proved interactively using Isabelle. Since this is an approximation within Casl, this occurrence does not invalidate the claim that each DL approximation can be proved automatically. Furthermore, the proof with Isabelle only needs to be done once before the online reasoning for wayfinding is carried out. The imported specification of linked lists is presented in Appendix C.3.

```
spec RouteGraph =
       LinkedList [sort node
                   pred link : node × node]
then %def
   pred  path : node × node
   ∀ x, y : node
   • path(x, x)                                                %(refl path)%
   • path(x, y) ⇔ ∃ L : List[node] • LinkedList(x :: L ++ [ y ])  %(path def)%

view PathApproximation : WeakRouteGraph to RouteGraph
```

Figure 39: List based Route Graph and approximation view[20]

[19]The specifications presented in Figure 38 have been published in [KL-1, Section 4.1].

[20]The specification and view presented in Figure 39 have been published in [KL-1, Section 4.2].

5.5 Summary

The examples of ontology specifications in CASL presented in this section support the notions of strongly typed and structured ontologies (cf. Section 2.2 and Section 2.3). Although the application has been mostly shown within the context of spatial information, CASL is applicable as an ontology specification language to any kind of knowledge. A further advantage of CASL is its tool HETS, which has been applied for the analysis of the presented specifications, including the usage of theorem provers interfaced by HETS. The language CASL also supports the specifications of approximation relations in terms of views following the definition of knowledge compilation as introduced in Section 3.3.

In particular, the navigation metaphor has been studied as algebraic specification of relational axioms derived from ontological definitions and of simulations in a real world and a WWW scenario [KL-2]. Views relate the signatures and the similarity of the formalisations is demonstrated informally. Furthermore, an ontological foundation of Route Graphs and a constructive design specification of Route Graphs are presented [KL-3]. Between the two base specifications of graphs on the ontological and design sides a view is shown. Furthermore, the abstraction function of Route Graphs is illustrated by instantiation of the ontological Route Graph to a tramway example [KL-6]. In addition, the relation of DOLCE and the Route Graph as one possible representation of space is explained through a wayfinding scenario combining different information sources [KL-1].

6 Conclusion

This thesis introduces the notions of structured and strongly typed ontologies in terms of the algebraic specification language Casl. Since foundational ontologies written in a modular way and in FOL lack a well established, tool supported language, Casl is a well suited candidate by providing a logic independent, well studied structuring language with tool support. Furthermore, Casl offers many-sorted FOL, which enables statically type checked theories, where theorem proving systems only need to reason about the entailment of the explicit theorem, whereas the implicit type information has already been checked.

When an expressive FOL language such as Casl is considered for knowledge representation, the most prominent disadvantage is its lack of tractable reasoning. This thesis provides two possible ways to overcome this disadvantage: on the one hand, ATP systems may be used to carry out the theorem proving, which enables computationally efficient reasoning for many purposes, but no tractable reasoning; while on the other hand, knowledge compilation into DL offers the application of approximated foundational ontologies within the context of computational efficient reasoners, which are intended to deal with large knowledge bases.

The presented examples support the claim that Casl can serve as a multi-purpose ontology specification language allowing the specification of foundational and domain ontologies. Also, the refinement of foundational ontologies into domain ontologies is supported by Casl, since ontology refinement is similar to the refinement of software specifications.

In the future, the structured formalisation of the SUMO ontology might be a good candidate for validating Casl as an ontology specification language. Furthermore, the relation between SUMO[21] and Dolce can be studied formally, and a composed, well-founded "sweetened SUMO" might emerge, which combines the rigid and precise definitions of Dolce with the broad and deep coverage of SUMO.

6.1 Bibliographical Remarks

Below, contributions of this thesis's author to collaboratively written papers are detailed; "the author" always refers to the author of this thesis.

An overview of the advantages of Casl and Hets for the development of formal ontologies has been published in the internationally reviewed proceedings of the conference on "Formal Ontology in Information Systems" [KL-8]. The author explored the usage of Casl's structuring and strong typing facilities in the context of the foundational ontology Dolce and proved some of the theorems with Isabelle. However, this rewritten version of Dolce was far from complete; it only covered Mereology. In [KL-1] a revised version of the quality related predicates in Dolce has been published in the internationally reviewed journal Spatial Cognition and Computation. These quality predicates have been used in a case study by the author showing the combination of different information sources formalised as quality spaces for some wayfinding task. One colour and two spatial quality spaces (Mereology and Route Graphs) have been used. Also, the author checked the consistency of the specification and provided proofs of fundamental theorems with Isabelle. However, the actual reasoning needed for the wayfinding task has been delegated to ATP systems. Furthermore, [KL-7] has been internationally reviewed and disseminates Casl's structuring concepts into the Semantic Web community; here the author explored systematically the benefits of modularisation for ontology engineering.

Within the context of Hets and Casl the author revised the proof management at the structured level, laid out the theoretical concepts of using ATP systems for proving Casl theorems, established a Casl extension towards DL, and explored the approximation of FOL theories with DL theories. This resulted in the following internationally reviewed publications:

- [KL-11] gives a short overview of Hets presented as a tool paper at TACAS'07;

[21] Suggested Upper Merged Ontology

- [KL-9] introduces the theoretical foundations of reasonable proof management for Casl in the context of ATP systems and detailed the connection of ATP systems to Hets;
- in [KL-10] the author has constructed institutions for OWL DL and Casl-DL, and their translation into Casl as comorphism;
- [KL-4, KL-5] provides a theory approximation algorithm for compiling FOL theories into DL theories for the development of domain ontologies based on approximated foundational ontologies in FOL.

Implementations of the comorphism and connection of ATP systems to Hets as described in [KL-9] enabled the verification of the Region Connection Calculus based on Cohn's connection theory in Casl with the ATP system Spass [75]. Additionally, the author made available an Emacs mode [KL-12] for editing HetCasl specifications and the Hets User Guide [KL-13] in collaboration with Till Mossakowski and Christian Maeder.

Moreover, [KL-2] shows the application of Casl to the formalisation of cognitive aspects by analysing the navigation metaphor. In [KL-2] the author has shown the advantages of formalising the cognitive sense-plan-act model with Casl, instead of writing Haskell programs, which simulate only one given example. Furthermore, the specification has been derived from Hochmair's simulations [29] in Haskell and contrasted with it, showing the benefits of algebraic specification with Casl.

In addition, the author has formulated different specifications of the Route Graph [73] tailored towards Rolland, the autonomous wheelchair used as demonstrator in the SFB/TR 8, and towards tramway networks. The collaborative efforts in the context of Rolland included ontological definitions of graph datatypes in Casl, and derived Haskell programs providing computational access to the ontology based Route Graphs. The result has been an internationally reviewed publication in the proceedings of "Spatial Cognition 2004" [KL-3]. Here, the author demonstrated again the importance of structured and generic Casl specifications for the development of ontologies and how to derive Haskell implementations from the ontological Casl specifications. Again, the application of the Route Graph concept to tramway networks uses the ontological and generic specifications of graphs and Route Graphs introduced in [KL-3], and it combines two layers of different granularities via abstraction. It distinguishes a device and a network layer, where the latter is derived from the former by abstraction. The instantiation of the Route Graph for tramway networks has been elaborated on by the author in great detail with a qualitative approach to the representation of knowledge; it has been published in the internationally reviewed proceedings of the 2004 symposium for "Formal Methods for Automation and Safety in Railway and Automotive Systems" [KL-6].

The Section "Publications by the Author" on the following page lists all the publications accumulated for this thesis. However, [KL-4] has not been submitted with this summarising thesis, since it has been superseded by [KL-5]. Also, the technical documents [KL-12, KL-13] have not been included in the submission of this thesis.

Acknowledgements

The author thanks the following people (listed in no particular order) for fruitful discussions and helpful comments: Stefano Borgo, Claudio Masolo, Markus Roggenbach, Bernd Krieg-Brückner, Till Mossakowski, John Bateman, Robert Ross, Christian Maeder, and Christian Mandel. Matthias Intemann deserves special thanks for many encouraging chats at the time of writing this cumulative thesis. Furthermore, the author thanks the students Heng Jiang, Rene Wagner, and Rainer Grabbe for their help with various implementations in the context of Hets. This thesis has been supported by the Collaborative Research Center "Spatial Cognition" SFB/TR 8 funded by the Deutsche Forschungsgemeinschaft (DFG) and by the DAAD VIGONI cooperation program (Projekt D/04/42045: "ALT-INFO"). Very special thanks go to Wencke Lüttich for her outstanding support.

Publications by the Author

Refereed Publications

[KL-1] John Bateman, Stefano Borgo, Klaus Lüttich, Claudio Masolo, and Till Mossakowski. Ontological Modularity and Spatial Diversity. *Spatial Cognition and Computation*, 7(1). In press.

[KL-2] Hartwig H. Hochmair and Klaus Lüttich. An Analysis of the Navigation Metaphor – And Why It Works for the World Wide Web. *Spatial Cognition and Computation*, 6(3):235–278, 2006.

[KL-3] Bernd Krieg-Brückner, Udo Frese, Klaus Lüttich, Christian Mandel, Till Mossakowski, and Robert Ross. Specification of an Ontology for Route Graphs. In C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel, and T. Barkowsky, editors, *Spatial Cognition IV*, volume 3343 of *Lecture Notes in Artificial Intelligence*, pages 390–412. Springer; Heidelberg, 2005.

[KL-4] Klaus Lüttich. Approximation of Ontologies in CASL. In P. Mosses, J. Power, and M. Seisenberger, editors, *CALCO-jnr 2005 CALCO Young Researchers Workshop Selected Papers*, number CSR 18-2005 in Report Series, pages 41–53. University of Wales Swansea, 2005.

[KL-5] Klaus Lüttich. Approximation of Ontologies in CASL. In Brandon Bennett and Christiane Fellbaum, editors, *Formal Ontology in Information Systems – Proceedings of the Fourth International Conference (FOIS-2006)*, volume 150 of *Frontiers in Artificial Intelligence and Applications*, pages 335–346. IOS Press; Amsterdam, 2006.

[KL-6] Klaus Lüttich, Bernd Krieg-Brückner, and Till Mossakowski. Tramway Networks as Route Graphs. In E. Schnieder and G. Tarnai, editors, *FORMS/FORMAT 2004 – Formal Methods for Automation and Safety in Railway and Automotive Systems*, pages 109–119. Universität Braunschweig / Beyrich DigitalService, 2004.

[KL-7] Klaus Lüttich, Claudio Masolo, and Stefano Borgo. Development of Modular Ontologies in CASL. In P. Haase, V. Honavar, O. Kutz, Y. Sure, and A. Tamilin, editors, *Proceedings of the 1st International Workshop on Modular Ontologies, WoMO 2006*, volume 232 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[KL-8] Klaus Lüttich and Till Mossakowski. Specification of Ontologies in CASL. In Achille C. Varzi and Laure Vieu, editors, *Formal Ontology in Information Systems – Proceedings of the Third International Conference (FOIS-2004)*, volume 114 of *Frontiers in Artificial Intelligence and Applications*, pages 140–150. IOS Press; Amsterdam, 2004.

[KL-9] Klaus Lüttich and Till Mossakowski. Reasoning Support for CASL with Automated Theorem Proving Systems. In J. Fiadeiro, editor, *Recent Trends in Algebraic Development Techniques*, volume 4409 of *Lecture Notes in Computer Science*, pages 74–91. Springer; Heidelberg, 2007.

[KL-10] Klaus Lüttich, Till Mossakowski, and Bernd Krieg-Brückner. Ontologies for the Semantic Web in CASL. In J. L. Fiadeiro, P. Mosses, and F. Orejas, editors, *Recent Trends in Algebraic Development Techniques*, volume 3423 of *Lecture Notes in Computer Science*, pages 106–125. Springer; Heidelberg, 2005.

[KL-11] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Orna Grumberg and Michael Huth, editors, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer; Heidelberg, 2007.

Other Publications

[KL-12] Klaus Lüttich. Emacs mode for editing HetCasl specifications. `http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/hets/emacs_mode/`, March 2007.

[KL-13] Till Mossakowski, Christian Maeder, and Klaus Lüttich. Hets User Guide – Version 0.7 –. `http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/hets/UserGuide.pdf`, March 25, 2007.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, 2003.

[2] B. Bennett. *Logical Representations for Automated Reasoning about Spatial Relationships.* PhD thesis, School of Computer Studies, The University of Leeds, 1997.

[3] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic syntax. IETF Draft Standard (RFC 2396) `http://www.ietf.org/rfc/rfc2396.txt`, August 1998.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[5] M. Bidoit and P. D. Mosses. CASL *User Manual*, volume 2900 of *LNCS (IFIP Series).* Springer Verlag, 2004. With chapters by Till Mossakowski, Donald Sannella, and Andrzej Tarlecki.

[6] P. V. Biron and A. Malhotra, editors. XML schema part 2: Datatypes – W3C recommendation. `http://www.w3.org/TR/xmlschema-2/`, 02 May 2001.

[7] D. Brickley and R. Guha, editors. RDF vocabulary description language 1.0: RDF Schema. W3C Working Draft `http://www.w3.org/TR/rdf-schema/`, 10 October 2003.

[8] R. Casati and A. C. Varzi. *Parts and places: the structures of spatial representation.* MIT Press (Bradford Books), Cambridge, MA and London, 1999.

[9] P. Clark and B. Porter. Building concept representations from reusable components. In *Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 369–376. AAAI Press / MIT Press, 1997.

[10] CoFI (The Common Framework Initiative). CASL *Reference Manual*, volume 2960 of *LNCS (IFIP Series).* Springer Verlag, 2004.

[11] R. P. Darken, T. Allard, and L. B. Achille. Spatial orientation and wayfinding in large-scale virtual spaces: Guest editors' introduction. *Presence: Teleoperators and Virtual Environments*, 8(6):3–6, 1999.

[12] M. Dean and G. Schreiber, editors. OWL Web Ontology Language – reference. W3C Recommendation `http://www.w3.org/TR/owl-ref/`, 10 February 2004.

[13] A. del Val. An analysis of approximate knowledge compilation. In *IJCAI'95, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 830–836, 1995.

[14] A. del Val. First order LUB approximations: characterization and algorithms. *Artificial Intelligence*, 162(1-2):7–48, 2005.

[15] T. T. Elvins. Wayfinding 2: The lost worlds. *SIGGRAPH Computer Graphics*, 31(4):9–12, 1997.

[16] B. Falkenhainer and K. Forbus. Compositional modelling: Finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

[17] D. Fensel, J. Hendler, H. Liebermann, and W. Wahlster, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.* The MIT Press, Cambridge, MA, 2003.

[18] F. S. Foundation. GNU emacs. `http://www.gnu.org/software/emacs/`.

[19] U. Frese. A discussion of simultaneous localization and mapping. *Autonomous Robots*, 20(1):25–42, 2006.

[20] A. Gangemi, R. Navigli, and P. Velardi. The OntoWordNet project: Extension and axiomatization of conceptual relations in WordNet. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *LNCS*, pages 820–838. Springer Verlag, 2003.

[21] M. R. Genesereth. Knowledge Interchange Format. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning*, pages 238–249. Morgan Kaufman Publishers, 1991.

[22] M. R. Genesereth and R. E. Fikes. Knowlegde Interchange Format; version 3.0; Reference Manual. Stanford Logic Group, Report Logic-92-1 `http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps`, June 1992.

[23] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of Protégé: An environment for knowledge-based systems development. *Int. J. Hum.-Comput. Stud.*, 58(1):89–123, 2003.

[24] J. Goguen. An introduction to algebraic semiotics, with applications to user interface design. In C. Nehaniv, editor, *Computation for Metaphor, Analogy and Agents*, volume 1562 of *LNCS*, pages 242–291. Springer Verlag, 1999.

[25] J. Goguen and G. Rosu. Institution morphisms. *Formal Aspects of Computing*, 13(3–5):274–307, 2002.

[26] R. G. Golledge. Human wayfinding and cognitive maps. In R. G. Golledge, editor, *Wayfinding behavior: Cognitive mapping and other spatial processes*, pages 5–45. Johns Hopkins Press, 1999.

[27] B. C. Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularity and web ontologies. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proceedings of the 10th International Conference on Principles of Knowledge Representation*, pages 198–209. AAAI Press, 2006.

[28] V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR*, volume 2083 of *LNCS*, pages 701–706. Springer Verlag, 2001.

[29] H. H. Hochmair. The wayfinding metaphor – comparing the semantics of wayfinding in the physical world and the WWW. Unpublished doctoral dissertation, Institute for Geoinformation, Technical University of Vienna, Vienna, 2002.

[30] H. H. Hochmair and A. U. Frank. A semantic map as basis for the decision process in the WWW navigation. In D. R. Montello, editor, *Conference on Spatial Information Theory (COSIT)*, volume 2205 of *LNCS*, pages 173–188. Springer Verlag, 2001.

[31] H. H. Hochmair and A. U. Frank. Influence of estimation errors on wayfinding-decisions in unknown street networks—analyzing the least-angle strategy. *Spatial Cognition and Computation*, 2(4):283–313, 2002.

[32] I. Horrocks. FaCT++. Available at `http://owl.man.ac.uk/factplusplus/`.

[33] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, number 2870 in LNCS, pages 17–29. Springer Verlag, 2003.

[34] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.

[35] M. Jarrar. *Towards Methodological Principles for Ontology Engineering.* Phd thesis, Vrije Universiteit, 2005.

[36] S. P. Jones, editor. *Haskell 98 Language and Libraries — The Revised Report.* Cambridge University Press, 2003.

[37] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *LNCS*, pages 229–243. Springer Verlag, 2004.

[38] B. Krieg-Brückner, D. Hutter, A. Lindow, C. Lüth, A. Mahnke, E. Melis, P. Meier, A. Poetzsch-Heffter, M. Roggenbach, G. Russell, J.-G. Smaus, and M. Wirsing. Multimedia instruction in safe and secure systems. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques*, volume 2755 of *LNCS*, pages 82–117. Springer Verlag, 2003.

[39] B. Krieg-Brückner, A. Lindow, C. Lüth, A. Mahnke, and G. Russell. Semantic interrelation of documents via an ontology. In G. Engels and S. Seehusen, editors, *DeLFI 2004, Tagungsband der 2. e-Learning Fachtagung Informatik, 6.-8. September 2004, Paderborn, Germany*, volume P-52 of *Lecture Notes in Informatics*, pages 271–282. Gesellschaft für Informatik, 2004.

[40] B. Krieg-Brückner and H. Shi. Orientation calculi and RouteGraphs: Towards semantic representations for route descriptions. In M. Raubal, H. Miller, A. Frank, and M. Goodchild, editors, *Proc. International Conference GIScience 2006, Münster, Germany*, volume 4197 of *LNCS*, pages 234–250. Springer Verlag, 2006.

[41] W. Kuhn. 7 ± 2 questions and answers about metaphors for GIS user interfaces. In T. L. Nyerges, editor, *Cognitive Aspects of Human-Computer interaction for geographic information systems*, pages 113–122. Kluwer Academic Press, 1995.

[42] W. Kuhn and A. U. Frank. A formalization of metaphors and image-schemas in user interfaces. In D. M. Mark and A. U. Frank, editors, *Cognitive and linguistic aspects of geographic space*, pages 419–434. Kluwer Academic Press, 1991.

[43] J. Liem and B. Bredeweg. OWL and qualitative reasoning models. In *KI 2006 – 29th German Conference on Artificial Intelligence*, pages 29–43, 2006.

[44] J. M. Loomis and R. L. Klatzky. Human navigation by path integration. In R. G. Golledge, editor, *Wayfinding Behavior: Cognitive Mapping and Other Spatial processes*, pages 125–151. Johns Hopkins Press, 1999.

[45] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence IJCAI-07.* AAAI Press, 2007.

[46] P. P. Maglio and T. Matlock. Constructing social spaces in virtual environments: Metaphors we surf the web by. In K. Höök, A. Munro, and D. Benyon, editors, *Workshop on personalized and social navigation in information space*, number T98-02, pages 138–149. SICS Technical Report, 1998.

[47] C. Mandel, K. Huebner, and T. Vierhuff. Towards an autonomous wheelchair: Cognitive aspects in service robotics. In *Proceedings of Towards Autonomous Robotic Systems (TAROS 2005)*, pages 165–172, 2005.

[48] G. M. Marchionini. *Information Seeking in Electronic Environments.* Cambridge University Press, 1997.

[49] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. WonderWeb Deliverable D18: Ontology Library. Technical report, ISTC-CNR, 2003.

[50] D. R. Montello. Navigation. In P. Shah and A. Miyake, editors, *The Cambridge handbook of visuospatial thinking*, pages 257–294. Cambridge University Press, 2005.

[51] R. Moratz and J. O. Wallgrün. Spatial reasoning about relative orientation and distance for robot exploration. In W. Kuhn, M. Worboys, and S. Timpf, editors, *Spatial Information Theory: Foundations of Geographic Information Science. Conference on Spatial Information Theory (COSIT)*, Lecture Notes in Computer Science, pages 61–74. Springer-Verlag; D-69121 Heidelberg, Germany; http://www.springer.de, 2003.

[52] T. Mossakowski. Relating CASL with other specification languages: The institution level. *Theoretical Comput. Sci.*, 286:367–475, 2002.

[53] T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen, 2005.

[54] T. Mossakowski, S. Autexier, and D. Hutter. Development graphs – proof management for structured specifications. *Journal of Logic and Algebraic Programming*, 67(1-2):114–145, 2006.

[55] P. D. Mosses. CoFI: the common framework initiative for algebraic specification and development. In M. Bidoit and M. Dauchet, editors, *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Proceedings*, number 1214 in LNCS, pages 115–137. Springer Verlag, 1997.

[56] I. Niles and A. Pease. Towards a standard upper ontology. In C. Welty and B. Smith, editors, *Proceedings of the international conference on Formal Ontology in Information Systems (FOIS 2001)*, pages 2–9, New York, NY, USA, 2001. ACM Press.

[57] N. J. Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufmann Publishers, 1980.

[58] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic.* Springer Verlag, 2002.

[59] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, editors. OWL Web Ontology Language – semantics and abstract syntax. W3C Recommendation `http://www.w3.org/TR/owl-semantics/`, 10 February 2004.

[60] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In B. Nebel, W. Swartout, and C. Rich, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*, pages 165–176. Morgan Kaufmann, 1992.

[61] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall International, 2003.

[62] B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *J. ACM*, 43(2):193–224, 1996.

[63] B. Smith. Logic and formal ontology. In J. N. Mohanty and W. McKenna, editors, *Husserl's Phenomenology: A Textbook*, pages 29–67. University Press of America, 1989.

[64] B. Smith. Against Idiosyncrasy in Ontology Development. In B. Bennett and C. Fellbaum, editors, *Formal Ontology in Information Systems – Proceedings of the Fourth International Conference (FOIS-2006)*, volume 150 of *Frontiers in Artificial Intelligence and Applications*, pages 15–26. IOS Press; Amsterdam, 2006.

[65] M. Smith, C. Welty, and D. McGuinness, editors. OWL Web Ontology Language – Guide. W3C Recommendation `http://www.w3.org/TR/owl-guide/`, 10 February 2004.

[66] M. E. Sorrows and S. C. Hirtle. The nature of landmarks for real and electronic spaces. In C. Freksa and D. Mark, editors, *Spatial Information Theory: Cognitive and Computational Foundations of Geographic Information Science, International Conference COSIT '99*, volume 1661 of *LNCS*, pages 37–50. Springer Verlag, 1999.

[67] G. Sutcliffe and C. B. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[68] G. Sutcliffe and C. B. Suttner. The state of CASC. *AI Communications*, 19(1):35–48, 2006.

[69] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.

[70] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNAI*, pages 292–297. Springer Verlag, 2006.

[71] J. O. Wallgrün. Autonomous construction of hierarchical Voronoi-based route graph representations. In C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel, and T. Barkowsky, editors, *Spatial Cognition IV. Reasoning, Action, Interaction: International Conference Spatial Cognition 2004*, volume 3343 of *LNAI*, pages 413–433. Springer Verlag, 2005.

[72] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topic. SPASS version 2.0. In A. Voronkov, editor, *Automated Deduction – CADE-18*, volume 2392 of *LNCS*, pages 275–279. Springer Verlag, July27-30 2002.

[73] S. Werner, B. Krieg-Brückner, and T. Herrmann. Modelling navigational knowledge by route graphs. In C. Freksa, C. Habel, and K. Wender, editors, *Spatial Cognition II*, volume 1849 of *LNAI*, pages 295–317. Springer Verlag, 2000.

[74] S. Wölfl and T. Mossakowski. CASL specifications of qualitative calculi. In A. G. Cohn and D. M. Mark, editors, *Conference on Spatial Information Theory*, volume 3693 of *LNCS*, pages 200–217. Springer Verlag, 2005.

[75] S. Wölfl, T. Mossakowski, and L. Schröder. Qualitative constraint calculi: Heterogeneous verification of composition tables. In *20th International FLAIRS Conference, Florida*, May 2007. To appear.

[76] J. Zimmer and S. Autexier. The MathServe System for Semantic Web Reasoning Services. In U. Furbach and N. Shankar, editors, *Proceedings of the third International Joint Conference on Automated Reasoning*, volume 4130 of *LNCS*. Springer Verlag, 2006.

Appendix

A Predefined CASL-DL Datatypes

The following CASL library is used for bootstrapping the predefined CASL-DL datatypes in HETS. The proof obligations of the view INT_IMPLEMENTS_INTEGERLITERAL have been proved with the ATP system Vampire via MathServe.

```
library CASL_DL_DATATYPES version 0.1
%date 09.02.2006
%authors Klaus Lüttich <luettich@tzi.de>
%number __@@__
%string emptyString, __: @ :__

spec BOOLEANLITERAL =
     sort  boolean < DATA
     ops   True, False : boolean

spec INTEGERLITERAL =
     sorts nonNegativeInteger < integer;
           integer < DATA
     ops   1, 2, 3, 4, 5, 6, 7, 8, 9, 0 : nonNegativeInteger;
           __@@__ : nonNegativeInteger × nonNegativeInteger → nonNegativeInteger;
           −__ : integer → integer
     sorts positiveInteger = {y : nonNegativeInteger • ¬ y = 0};          %(posInt_def)%
           nonPositiveInteger < integer;
           negativeInteger < nonPositiveInteger
     op    0 : nonPositiveInteger
     • 00 = 0                                                             %(zero_eq)%
     ∀ x : nonNegativeInteger
     • 0 @@ x = x                                               %(no_preceeding_zeros)%

spec STRINGLITERAL =
     sorts Char;
           string < DATA
     free type string ::= emptyString | __: @ :__(Char; string)
     ops   ' ' : Char;                                          %(DL_printable_32)%
           '!' : Char;                                          %(DL_printable_33)%
           '\"' : Char;                                         %(DL_printable_34)%
           '#' : Char;                                          %(DL_printable_35)%
           ...
           'þ' : Char;                                          %(DL_printable_254)%
           'ÿ' : Char                                           %(DL_printable_255)%

spec DL_LITERAL = STRINGLITERAL and INTEGERLITERAL and BOOLEANLITERAL
then free type DATA ::= sorts string, integer, boolean

from BASIC/NUMBERS get INT

view INT_IMPLEMENTS_INTEGERLITERAL :
     INTEGERLITERAL to
     {INT
      then %def
           sorts nonPositiveInteger = {i : Int • i ≤ 0};
                 negativeInteger = {i : nonPositiveInteger • i < 0}
     } = integer ↦ Int, positiveInteger ↦ Pos, nonNegativeInteger ↦ Nat
```

B Knowledge Compilation Examples

B.1 Complete Reification Example

This library shows the complete reification example of Section 3.3. Both views have been automatically proven with SPASS.

library REIFICATIONEXAMPLE
%display(*s1* %LATEX s_1)%
%display(*s2* %LATEX s_2)%
%display(*x1* %LATEX x_1)%
%display(*x2* %LATEX x_2)%
%display(*t1* %LATEX t_1)%
%display(*t2* %LATEX t_2)%
%display(*r1* %LATEX r_1)%
%display(*r2* %LATEX r_2)%

spec TIME_MEREOLOGY =
  sort T
  pred $P : T \times T$

spec BINARY_TEMPORAL_DISSECTIVE =
  TIME_MEREOLOGY **with sort** T, **pred** $P : T \times T$
then sorts s_1, s_2
  pred $Rel : s_1 \times s_2 \times T$
  $\forall x_1 : s_1; x_2 : s_2; t_1, t_2 : T$
  $\bullet\ Rel(x_1, x_2, t_1) \wedge P(t_2, t_1) \Rightarrow Rel(x_1, x_2, t_2)$

spec REIFIED_BINARY_TEMPORAL_DISSECTIVE =
  TIME_MEREOLOGY **with sort** T, **pred** $P : T \times T$
then sorts s_1, s_2
  sort Rel
  ops $Rel\_1 : Rel \rightarrow s_1;$
    $Rel\_2 : Rel \rightarrow s_2;$
    $Rel\_3 : Rel \rightarrow T$
  $\forall r_1, r_2 : Rel; x_1 : s_1; x_2 : s_2; t_1, t_2 : T$
  $\bullet\ Rel\_1(r_1) = x_1 \wedge Rel\_2(r_1) = x_2 \wedge Rel\_3(r_1) = t_1 \wedge P(t_2, t_1)$
    $\Rightarrow \exists r_2 : Rel \bullet Rel\_1(r_2) = x_1 \wedge Rel\_2(r_2) = x_2 \wedge Rel\_3(r_2) = t_2$

view REIFICATIONENTAILSORIGINAL :
 BINARY_TEMPORAL_DISSECTIVE **to**
 {REIFIED_BINARY_TEMPORAL_DISSECTIVE
 then %def
  pred $Rel : s_1 \times s_2 \times T$
  $\forall x_1 : s_1; x_2 : s_2; t : T$
  $\bullet\ Rel(x_1, x_2, t) \Leftrightarrow$
    $\exists r : Rel \bullet Rel\_1(r) = x_1 \wedge Rel\_2(r) = x_2 \wedge Rel\_3(r) = t$ %(Rel_def)%
 }

view ORIGINALENTAILSREIFICATION :
 REIFIED_BINARY_TEMPORAL_DISSECTIVE **to**
 {BINARY_TEMPORAL_DISSECTIVE
 then sort Rel
  ops $Rel\_1 : Rel \rightarrow s_1;$
    $Rel\_2 : Rel \rightarrow s_2;$
    $Rel\_3 : Rel \rightarrow T$
  $\forall x_1 : s_1; x_2 : s_2; t : T$
  $\bullet\ Rel(x_1, x_2, t) \Leftrightarrow$

```
    ∃ r : Rel • Rel_1(r) = x₁ ∧ Rel_2(r) = x₂ ∧ Rel_3(r) = t          %(Rel_def)%
}
```

B.2 Complete Approximation Example

This library summarises the example used in [KL-4, KL-5] and shows the surrounding signature and structuring for the example in Figure 15.

```
library DOLCE_MEREOLOGY_APPROX version 0.4
%{This library is based on "A fragment of DOLCE for CASL"
  by Stefano Borgo, Claudio Masolo
  LOA−CNR
  March 5, 2004%

spec GENPARTHOOD[sort s] =
     pred P : s × s
     ∀ x, y, z : s
     • P(x, x)                                                  %(Ad11)%
     • P(x, y) ∧ P(y, x) ⇒ x = y                                %(Ad12)%
     • P(x, y) ∧ P(y, z) ⇒ P(x, z)                              %(Ad13)%

spec GENMEREOLOGY[sort s] =
     GENPARTHOOD[sort s]
then preds PP(x, y : s) ⇔ P(x, y) ∧ ¬ P(y, x);          %(Dd1_Proper_Part)%
           O(x, y : s) ⇔ ∃ z : s • P(z, x) ∧ P(z, y);        %(Dd2_Overlap)%
           At(x : s) ⇔ ¬ ∃ y : s • PP(y, x)                  %(Dd3_Atom)%
           ∀ x, y : s • ∃ z : s • At(z) ∧ P(z, x)               %(Ad18)%

%% added inverse predicates
spec INVPP_P [sort s
              preds PP, P : s × s] =
     preds PP_i(x, y : s) ⇔ PP(y, x);
           P_i(x, y : s) ⇔ P(y, x)

%% DL version
spec GENPARTHOOD_DL[sort s] =
     pred P : s × s
     ∀ x, y, z : s • P(x, y) ∧ P(y, z) ⇒ P(x, z);               %(Ad13)%

spec GENMEREOLOGY_DL[sort s] =
     GENPARTHOOD_DL[sort s]
and  INVPP_P [sort s
              preds PP, P : s × s]
then preds PP : s × s;
           O(x, y : s) ⇔ O(y, x);                         %(Dd2_Overlap_sym)%
           At(x : s) ⇔ ¬ ∃ y : s • PP_i(x, y);                  %(Dd3_Atom)%
     ∀ x, y, z : s
     • ∃ z' : s • P_i(x, z') ∧ At(z')                          %(Ad18_rw)%
     • PP(x, y) ⇒ P(x, y)                           %(Dd1_Proper_Part_impl)%
     • PP(x, y) ∧ PP(y, z) ⇒ PP(x, z)              %(Dd1_Proper_Part_trans)%

view MEREOLOGY_DL_TO_FOL :
     GENMEREOLOGY_DL[sort s] to
     {GENMEREOLOGY[sort s]
     then %def
          INVPP_P [sort s preds PP, P : s × s]
     }
```

C Specifications Related to Route Graphs

C.1 Revised Graph Specifications

The following libraries illustrate the Route Graph ontology and design specifications as used in [KL-3]. However, they are revised compared to the already published version. Additionally, a view is defined showing the relation between the ontology and its design specification.

```
library RouteGraphOnt version 0.1

from Basic/Numbers get Nat

spec GenSequence[sort Elem] given Nat =
     sort   Sequence[Elem]
     op     EmptySeq : Sequence[Elem]
     pred   hasElem : Sequence[Elem] × Elem
     ops    head : Sequence[Elem] →? Elem;
            tail : Sequence[Elem] →? Sequence[Elem];
            freq : Sequence[Elem] × Elem → Nat;
            last : Sequence[Elem] →? Elem;
            __++__ : Sequence[Elem] × Sequence[Elem] → Sequence[Elem]
     ∀ e : Elem • ¬ hasElem(EmptySeq, e)
     pred   atMostOnce(s : Sequence[Elem]) ⇔ ∀ e : Elem • freq(s, e) ≤ 1
                                                            %(atMostOnce_def)%

spec GenGraph[sort Kind] =
     sorts  Graph[Kind], Node[Kind], Edge[Kind]
     preds  hasNode : Graph[Kind] × Node[Kind];
            hasEdge : Graph[Kind] × Edge[Kind]
     ops    source, target : Edge[Kind] → Node[Kind]
     ∀ e : Edge[Kind]; s, t : Node[Kind]; g : Graph[Kind]
     • hasEdge(g, e) ∧ source(e) = s ∧ target(e) = t ⇒ hasNode(g, s) ∧ hasNode(g, t)
then GenSequence [sort Edge[Kind] fit Elem ↦ Edge[Kind]]
     with head, tail, EmptySeq, freq, __++__
then GenSequence [sort Node[Kind] fit Elem ↦ Node[Kind]]
     with head, tail, freq, last
then ops    sources, targets : Sequence[Edge[Kind]] → Sequence[Node[Kind]]
     ∀ l : Sequence[Edge[Kind]]
     • sources(EmptySeq) = EmptySeq
     • ¬ l = EmptySeq ⇒ head(sources(l)) = source(head(l))
     • ¬ l = EmptySeq ⇒ tail(sources(l)) = sources(tail(l))
     • targets(EmptySeq) = EmptySeq
     • ¬ l = EmptySeq ⇒ head(targets(l)) = target(head(l))
     • ¬ l = EmptySeq ⇒ tail(targets(l)) = targets(tail(l));
     pred   connected(l : Sequence[Edge[Kind]]) ⇔
            l = EmptySeq
            ∨ (∃! e1 : Edge[Kind] • hasElem(l, e1))
            ∨ ((∀ e1, e2 : Edge[Kind]
                  • head(l) = e1 ∧ head(tail(l)) = e2 ⇒ target(e1) = source(e2))
               ∧ connected(tail(l)))
     sorts  Path[Kind] = {l : Sequence[Edge[Kind]] • connected(l)};
            Route[Kind] = {p : Path[Kind] • atMostOnce(sources(p)) ∧ atMostOnce(p)}
     ops    startOf(l : Sequence[Edge[Kind]]) :? Node[Kind] = source(head(l));
            endOf(l : Sequence[Edge[Kind]]) :? Node[Kind] = target(last(l))
     pred   pathIn : Path[Kind] × Graph[Kind]
```

```
library RouteGraphDesign version 0.6

from RouteGraphOnt get RouteGraphIndoors, GenGraph
from Basic/StructuredDatatypes get List
from Basic/Numbers get Nat
from Basic/Graphs get NonUniqueEdgesGraph

spec List2[sort Elem] given Nat =
      List[sort Elem fit Elem ↦ Elem]
then pred atMostOnce : List[Elem]
      ∀ l : List[Elem]; x : Elem
      • atMostOnce([ ])
      • atMostOnce(x :: l) ⇔ freq(l, x) = 0 ∧ atMostOnce(l)

spec Graph2[sort Kind] given Nat =
      sorts Node, Edge
      ops   source, target : Edge → Node
then NonUniqueEdgesGraph[sort Node][sort Edge]
and  List2[sort Edge fit Elem ↦ Edge]
      with [], __::__, atMostOnce
then pred connected : List[Edge]
      ∀ rs1, rs2 : Edge; rsl : List[Edge]
      • connected([ ])                                          %(connected_empty)%
      • connected([ rs1 ])                                      %(connected_singleton)%
      • connected(rs1 :: rs2 :: rsl) ⇔
        target(rs1) = source(rs2) ∧ connected(rs2 :: rsl)      %(connected_rec)%
then List2[sort Node fit Elem ↦ Node]
then ops  sources, targets : List[Edge] → List[Node]
      ∀ l : List[Edge]; rs : Edge
      • sources([ ]) = [ ]
      • sources(rs :: l) = source(rs) :: sources(l);
      ∀ l : List[Edge]; rs : Edge
      • targets([ ]) = [ ]
      • targets(rs :: l) = target(rs) :: targets(l);
      sorts Path = {l : List[Edge] • connected(l)};                 %(def_path)%
            Route = {l: Path • atMostOnce(sources(l)) ∧ atMostOnce(l)} %(def_route)%
then %def
      pred  __pathIn__ : Path × Graph
      ops   startOf, endOf : List[Edge] →? Node
      ∀ p : Path; g : Graph; l : List[Edge]
      • p pathIn g ⇔ ∀ e : Edge • e ϵ p ⇒ e ϵ g
                                                                %(path_in_def)%
      • startOf(l) = source(first(l))                            %(startOf_def)%
      • endOf(l) = target(last(l))                               %(endOf_def)%

view Graph2_implements_GraphOnt :
      GenGraph[sort k fit Kind ↦ k] to
      {Graph2[sort k fit Kind ↦ k]
       then %def
            preds hasEdge(g : Graph; e : Edge) ⇔ e ϵ g;          %(hasEdge_def)%
                  hasNode(g : Graph; n : Node) ⇔ n ϵ g;          %(hasNode_def)%
                  hasElem(l : List[Edge]; e : Edge) ⇔ e ϵ l;     %(hasElem_Edge_def)%
                  hasElem(l : List[Node]; n : Node) ⇔ n ϵ l      %(hasElem_Node_def)%
      } =
      Edge[k] ↦ Edge, Graph[k] ↦ Graph, k ↦ k, Nat ↦ Nat,
      Node[k] ↦ Node, Path[k] ↦ Path, Pos ↦ Pos,
      Route[k] ↦ Route, Sequence[Edge[k]] ↦ List[Edge],
      Sequence[Node[k]] ↦ List[Node],
      EmptySeq : Sequence[Edge[k]] ↦ [] : List[Edge],
      EmptySeq : Sequence[Node[k]] ↦ [] : List[Node],
      head : Sequence[Edge[k]] →? Edge[k] ↦ first : List[Edge] →? Edge,
      last : Sequence[Edge[k]] →? Edge[k] ↦ last : List[Edge] →? Edge,
      tail : Sequence[Edge[k]] →? Sequence[Edge[k]] ↦ rest : List[Edge] →? List[Edge],
      head : Sequence[Node[k]] →? Node[k] ↦ first : List[Node] →? Node,
      last : Sequence[Node[k]] →? Node[k] ↦ last : List[Node] →? Node,
      tail : Sequence[Node[k]] →? Sequence[Node[k]] ↦ rest : List[Node] →? List[Node],
      connected ↦ connected
```

C.2 Tramway Route Graph Specifications

The specification GenGraph, instantiated here, is presented in Appendix C.1 on page 56.

```
spec TramDevices_RouteGraph =
      GenGraph[sort TDevices]
then sorts  ActivePoint, JointPoint, Sensor, Signal, AuxSegment, Segment,
            SignalSegment, PointSegment
      generated type Point ::= sorts ActivePoint, JointPoint
      generated type Node[TDevices] ::= sorts Sensor, Signal, Point
      free type Edge[TDevices] ::= sorts Segment, AuxSegment
      free type AuxSegment ::= sorts SignalSegment, PointSegment
      sorts SignalSegment = {e : Edge[TDevices]
                                 • source(e) ∈ Sensor ∧ target(e) ∈ Signal };
            PointSegment = {e : Edge[TDevices]
                                 • (source(e) ∈ JointPoint ∧ target(e) ∈ Sensor)
                                   ∨ (source(e) ∈ Signal ∧ target(e) ∈ ActivePoint) };
            Segment = {e : Edge[TDevices]
                            • (source(e) ∈ Sensor ∧ (target(e) ∈ Sensor ∨ target(e) ∈ JointPoint))
                              ∨ (source(e) ∈ ActivePoint ∧ target(e) ∈ Sensor) }
      pred  __crosses__ : Edge[TDevices] × Edge[TDevices]
      ∀ e1, e2 : Edge[TDevices] • e1 crosses e2 ⇒ e2 crosses e1
      free type ActivePointState ::= straight | left | right
      sorts Entry[TDevices], Course[TDevices]
      ops   entry : Edge[TDevices] → Entry[TDevices];
            pointDirection : Entry[TDevices] →? ActivePointState;
            course : Edge[TDevices] →? Course[TDevices];
            maxSpeed, duration : Course[TDevices] → Nat
      ∀ e : Edge[TDevices]
      • e ∈ Segment ∧ source(e) ∈ ActivePoint ⇔ def pointDirection(entry(e))
      • def course(e) ⇔ e ∈ Segment

spec TramNetwork_RouteGraph =
      GenGraph[sort TNetwork] with Edge[TNetwork] ↦ TrackSegment
then sorts SignalSensor, SimpleSensor
      free type Node[TNetwork] ::= sorts SignalSensor, SimpleSensor
      sorts TramRoute = {r : Route[TNetwork]
                              • startOf(r) ∈ SimpleSensor
                                ∧ target(head(r)) ∈ SignalSensor
                                ∧ ∀ e : TrackSegment
                                   • hasElem(tail(r), e)
                                     ⇒ source(e) ∈ SimpleSensor ∧ target(e) ∈ SimpleSensor };
            SignalTrackSegment, JoinTrackSegment < TrackSegment
```

```
        preds __exclusionConflictWith__(r1, r2 : TramRoute) ⇔
                (target(head(r1)) = target(head(r2)) ∨ last(r1) = last(r2)) ∧ ¬ r1 = r2
              __crosses__ : TrackSegment × TrackSegment;
              __crossConflictWith__ : TramRoute × TramRoute
        free type SignalState ::= straight | left | right | stop
        free type JoinPriority ::= main_track | incoming_track
        sorts Entry[TNetwork], Exit[TNetwork]
        ops   entry : SignalTrackSegment → Entry[TNetwork];
              exit : JoinTrackSegment → Exit[TNetwork];
              signalSetting : Entry[TNetwork] → SignalState;
              priority : Exit[TNetwork] → JoinPriority
        ∀ e, e' : TrackSegment • e crosses e' ⇒ e' crosses e
        ∀ r1, r2 : TramRoute
        • r1 crossConflictWith r2 ⇔
             ∃ n_e1, n_e2 : TrackSegment
             • hasElem(r1, n_e1) ∧ hasElem(r2, n_e2) ∧ n_e1 crosses n_e2

spec DEVICES_TO_TRACKSEGMENTS =
        PLACEABSTRACTION [TRAMDEVICES_ROUTEGRAPH fit
                           Kind1 ↦ TDevices, Route[Kind1] ↦ Route[TDevices],
                           Path[Kind1] ↦ Path[TDevices]]
                          [TRAMNETWORK_ROUTEGRAPH fit
                           Kind2 ↦ TNetwork, Edge[Kind2] ↦ TrackSegment,
                           Route[Kind2] ↦ Route[TNetwork],
                           Path[Kind2] ↦ Path[TNetwork]]
then ∀ g1 : Graph[TDevices]; g2 : Graph[TNetwork];
          sigSeg : SignalSegment; poiSeg : PointSegment;
          sig_n : Signal; seg : Segment
        • mapNode(source(sigSeg)) = mapNode(target(sigSeg))    %(places_con_SignalSeg)%
        • mapNode(source(poiSeg)) = mapNode(target(poiSeg))     %(places_con_PointSeg)%
        • mapNode(sig_n) ∈ SignalSensor                          %(Signal_SignalSensor)%
        • ¬ def mapEdge(sigSeg)
        • ¬ def mapEdge(poiSeg)
        • def mapEdge(seg) ∧ mapEdge(seg) ∈ TrackSegment
        • source(seg) ∈ ActivePoint ⇒ mapEdge(seg) ∈ SignalTrackSegment
        • target(seg) ∈ JointPoint ⇒ mapEdge(seg) ∈ JoinTrackSegment
        pred  __crosses__(d_e1, d_e2 : TrackSegment) ⇔
              ∃ e1, e2 : Edge[TDevices]
              • d_e1 = mapEdge(e1) ∧ d_e2 = mapEdge(e2) ∧ e1 crosses e2
```

C.3 Specification of Linked List

```
spec List[sort Elem] =
        free type List[Elem] ::= [] | __::__(Elem; List[Elem])
        op      __++__ : List[Elem] × List[Elem] → List[Elem]
        ∀ K, L, L1, L2 : List[Elem]; x, y, z : Elem
        • [ ] ++ K = K                                           %(concat_nil_List)%
        • x :: L ++ K = x :: (L ++ K)                            %(concat_NeList_List)%
        • x :: (L1 ++ y :: L2 ++ [ z ]) = (x :: L1 ++ y :: L2) ++ [ z ]
                                                        %(shift brackets)% %implied
        • [ x ] ++ [ y ] = [ x, y ]                  %(concat singletons)% %implied

spec LinkedList [sort Elem
                 pred link : Elem × Elem] =
        List[sort Elem]
then pred  LinkedList : List[Elem]
        ∀ K : List[Elem]; x, y : Elem
        • LinkedList([ ])                                        %(empty list is linked)%
        • LinkedList([ x ])                              %(single element list is linked)%
        • LinkedList(x :: y :: K) ⇔ link(x, y) ∧ LinkedList(y :: K)
                                   %(head elements are linked and the rest is linked
                                                                => the whole is linked)%
then %implies
        ∀ x, y, z : Elem; K : List[Elem]
        • link(x, y) ⇔ LinkedList([ x, y ])              %(a link implies a linked list)%
        • ∀ L1, L2 : List[Elem]
          • ∀ x : Elem
            • LinkedList(x :: (L1 ++ [ y ])) ∧ LinkedList(y :: (L2 ++ [ z ]))
              ⇒ LinkedList(x :: (L1 ++ y :: L2 ++ [ z ]))
                       %(if a linked list ends in an element which is the head of another linked list
                                                        the concatanation of these is also linked)%
```

Accumulated Papers