SOQE 2021

The Second Workshop on
Second-Order Quantifier Elimination
and Related Topics

Online Event, November 4, 2021

Associated with KR 2021, the 18th International
Conference on Principles of Knowledge Representation
and Reasoning, November 3–12, 2021

Proceedings

Renate A. Schmidt · Christoph Wernhard · Yizheng Zhao (Eds.)
Preface

This volume contains the papers presented at the Second Workshop on Second-order Quantifier Elimination and Related Topics (SOQE 2021) held on November 4, 2021, due to the pandemic situation as an online event, associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021). It continues the SOQE Workshop Series, which was initiated with SOQE 2017 in Dresden.

Second-order quantifier elimination (SOQE) is the problem of equivalently reducing a formula with quantifiers upon second-order objects such as predicates to a formula in which these quantified second-order objects no longer occur. In slight variations, SOQE is known as forgetting, projection, predicate elimination, and uniform interpolation. It can be combined with various underlying logics, including propositional, modal, description and first-order logics. SOQE and its variations bear strong relationships to Craig interpolation, definability and computation of definientia, the notion of conservative theory extension, abduction, notions of weakest sufficient and strongest necessary condition, and generalizations of Boolean unification to predicate logic. It is attractive as a logic-based approach to various computational tasks, for example, the computation of circumscription, the computation of modal correspondence properties, forgetting in knowledge bases, knowledge-base modularization, abductive reasoning and generating explanations, the specification of non-monotonic logic programming semantics, view-based query processing, and the characterization of formula simplifications in reasoner preprocessing.

Given the relevance of SOQE and related topics for various particular fields, our call for papers asked not just for novel contributions, but also for abstracts of pre-published work that so far was presented only in other contexts. We received 14 papers out which 12 were accepted for this volume, 5 as regular papers, 3 as short papers and 4 as abstracts of pre-published work. In addition to the contributed papers, the program included two invited talks by leading experts:

– Frank Wolter on Living Without Beth and Craig: Explicit Definitions and Interpolants without Beth Definability and Craig Interpolation
– David Toman on Projective Beth Definability and Craig Interpolation for Relational Query Optimization

We would like to thank all those involved for their enthusiasm and high-quality contributions, in particular, the invited speakers, the authors of research papers, the members of the Program Committee, and the KR 2021 Workshop Chairs Markus Krötzsch and Yongmei Liu who provided excellent support.

November 2021

Renate A. Schmidt
Christoph Wernhard
Yizheng Zhao
Organization

Program Committee Chairs and Organizers

Renate A. Schmidt  The University of Manchester, UK
Christoph Wernhard  University of Potsdam, Germany
Yizheng Zhao  Nanjing University, China

Program Committee

Philippe Balbiani  IRIT, CNRS, University of Toulouse, France
Jieying Chen  University of Oslo, Norway
James Delgrande  Simon Fraser University, Canada
Silvio Ghilardi  Università degli Studi di Milano, Italy
Stefan Hetzl  Technische Universität Wien, Austria
Patrick Koopmann  Technische Universität Dresden, Germany
Andreas Nonnengart  DFKI, Germany
Vladislav Ryzhikov  Birkbeck, University of London, UK
Stefan Schlobach  Vrije Universiteit Amsterdam, Netherlands
Renate A. Schmidt  The University of Manchester, UK
Viorica Sofronie-Stokkermans  Universität Koblenz-Landau, Germany
Andrzej Szałas  Uniwersytet Warszawski, Poland and Linköpings Universitet, Sweden
Sophie Tourret  Inria, France and Max Planck Institute for Informatics, Germany
Kewen Wang  Griffith University, Australia
Christoph Wernhard  University of Potsdam, Germany
Yizheng Zhao  Nanjing University, China

Additional Reviewer

Alessandro Gianola
Contents

Invited Talks

Projective Beth Definability and Craig Interpolation for Relational Query Optimization (Material to Accompany Invited Talk) ........................................... 1
  David Toman and Grant Wedell

Living Without Beth and Craig: Explicit Definitions and Interpolants without Beth Definability and Craig Interpolation (Abstract of Invited Talk) ....................... 14
  Frank Wolter

Research Papers

Resolution-Based Uniform Interpolation for Multi-Agent Modal Logic $K_n$ ........................................... 15
  Ruba Alassaf, Renate A. Schmidt and Uli Sattler

Second-Order Specifications and Quantifier Elimination for Consistent Query Answering in Databases (Abstract) .................................................. 28
  Leopoldo Bertossi

On Testing Containedness Between Geometric Graph Classes using Second-order Quantifier Elimination and Hierarchical Reasoning (Short Paper) ............... 37
  Lucas Böltz, Hannes Frey, Dennis Peuter and Viorica Sofronie-Stokkermans

Abduction in $\mathcal{EL}$ via Translation to FOL ........................................... 46
  Fajar Haifani, Patrick Koopmann and Sophie Tourret

An Abstract Fixed-Point Theorem for Horn Formula Equations (Abstract) 59
  Stefan Hetzl and Johannes Klobhofer

Signature-Based ABox Abduction in $\mathcal{ALC}$ is Hard ........................................... 61
  Patrick Koopmann

SEH-PILoT: A System for Property-Directed Symbol Elimination – Work in Progress (Short Paper) ........................................... 75
  Philipp Marohn and Viorica Sofronie-Stokkermans

Symbol Elimination and Applications to Parametric Entailment Problems (Abstract) ........................................... 83
  Dennis Peuter and Viorica Sofronie-Stokkermans

The Yoneda Reduction of Polymorphic Types (Abstract) ........................................... 92
  Paolo Pistone and Luca Tranchini
Applying Second-Order Quantifier Elimination in Inspecting Gödel’s Ontological Proof .......................................................... 98
  Christoph Wernhard

Sahlqvist-type Correspondence Theory for Second-Order Propositional Modal Logic (Short Paper) .................................................. 112
  Zhiquang Zhao

Metadata-based Term Selection for Modularization and Uniform Interpolation of OWL Ontologies ............................................. 122
  Xinhao Zhu, Xuan Wu, Ruiqing Zhao, Yu Dong and Yizheng Zhao
Abstract. Accessing information using a high-level data model or ontology has been a long-standing objective of research communities in several areas. The underlying idea of separating a logical/conceptual view of how information is understood by users from a physical view of the layout of data in data structures, called physical data independence, has been a focus of research for more than fifty years. Here, we explore the issues connected with optimizing and executing relational queries and updates in this setting. In particular, we consider how to find appropriate reformulations of user queries over the physical design, and show how these ideas naturally relate to first-order definability and interpolation. The talk elaborates on how logic-based approaches can be used to capture both the high-level conceptual views of information and the low-level physical layout of data. Based on such a formalism, we present the design of a relational query optimizer based on Craig interpolation that allows users to compile both queries and updates to low-level code that operates directly over a physical encoding of the data. The ultimate objective of this design is to produce low-level code that performs comparably with hand-written code in low-level programming languages such as C.

1 The Problem

Abstraction and high-level approaches to software development have been among the most significant factors in increasing both the productivity of developers and the quality of the applications they develop. Efforts along these lines date back to the late 60s and early 70s when the concepts of data independence [1,2] and abstract data types (ADTs) [13] were introduced. The concepts share the goal of enabling application programmers to develop applications with respect to a purely abstract or conceptual understanding of the application’s data or information, an understanding that is entirely independent of the concrete data structures and related algorithms that encode the data on physical storage devices. Indeed, modern file systems are examples: programmers manipulate files...
via operations that are entirely devoid of any need to understand low-level disk layout issues.

The idea of data independence gained popularity in the 70s, both in the area of programming languages, e.g., with languages such as SETL [8,14,10], and in the area of information and database systems mainly due to the development of the relational model (RM) [5] with accompanying data manipulation language(s) [6] based on first-order logic. However, with the passing of time (50 years later), approaches that use lower levels of abstraction, such as C or the various recent NoSQL database systems, have often displaced approaches that promote data independence, often at the cost of increasing development time and/or lowering the quality of deployed systems. The most common reason for this phenomenon is the need for massive scaleability and flexibility, capabilities often missing in systems with high levels of abstraction such as RM.

The goal of this presentation is to outline a direction of research that, in the realm of database and information systems, enables simultaneous high level abstractions at the user level and extreme flexibility at the physical design level, that is, in the choice of concrete data structures and their access algorithms. Indeed, our ultimate goal of this direction of research is to compete with hand-written code in low-level languages such as C, while providing the high level of abstraction in the original RM [5] that are not yet fully realized in existing relational database management systems.

2 Data Independence (through an Example)

We begin outlining how data independence can be understood more formally in terms of first-order (relational) signatures and integrity constraints (i.e., first-order sentences over these signatures).

2.1 The Logical Schema

The logical schema is a first order signature $S_L$ and an accompanying set of integrity constraints $\Sigma_L$ that are specific to the domain of the application (that require user familiarity). The situation can be depicted as follows:

$$\Sigma_L \rightarrow S_L \varphi \rightarrow \text{Logical Schema and User Queries}$$

The users interacting with the data use queries, in our case open first order formulae over $S_L$, to formulate their requests (we will deal with modifying the data later in Section 4). There are two important observations that follow from this arrangement:

1. The user only requires familiarity with $S_L$ and $\Sigma_L$ to be able to develop applications; and
2. The user can assume that the actual data is a single interpretation on $S_L$ that is a model of $\Sigma_L$ over which her requests are evaluated (i.e., without the need to comprehend subtle issues related to logical entailment and/or belief revision). We call such interpretations instances of the schema.

Example 1 (Logical Schema)
We will use the following logical schema formulated in SQL as our running example.

```sql
CREATE TABLE employee (  num INTEGER NOT NULL,  name CHAR(20),  worksin INTEGER NOT NULL,  PRIMARY KEY (num),  FOREIGN KEY (worksin) REFERENCES department )
CREATE TABLE department (  num INTEGER NOT NULL,  name CHAR(50),  manager INTEGER NOT NULL,  PRIMARY KEY (num),  FOREIGN KEY (manager) REFERENCES employee )
```

The (instances of) `employee` and `department` relation declarations are intuitively meant to store information about employee numbers, names and departments they work in, and about departments, their names and managers. In our formalism, this is simply a syntactic sugar for a signature

$$S_L = \{\text{employee}/3, \text{department}/3\}$$

(where “/i” indicates predicate arity) and integrity constraints

$$\Sigma_L = \{\text{employee}(x,y_1,z_1) \land \text{employee}(x,y_2,z_2) \rightarrow y_1 = y_2 \land z_1 = z_2, \text{employee}(x,y,z) \rightarrow \exists u,v. \text{department}(z,u,v), \ldots\}$$

stating that employees are identified by their number, that they must work for exactly one department, and so on.

The ability of specifying integrity constraints in $\Sigma_L$ allows one to go beyond what is available in typical implementations of the relational model, for example:

- managers are employees that manage a department (a view)
  $$\text{manager}(x,y,z) \leftrightarrow \text{employee}(x,y,z) \land \exists u,v. \text{department}(u,v,x)$$
- managers work in their own departments (business rule)
  $$\text{employee}(x,y,z) \land \text{department}(u,v,x) \rightarrow z = u$$
- workers and managers partition employees (partition)
  $$\text{employee}(x,y,z) \leftrightarrow (\text{manager}(x,y,z) \lor \text{worker}(x,y,z))$$
  $$\text{manager}(x,y,z) \land \text{worker}(x,y,z) \rightarrow \bot$$

Observe that this extends the signature of the logical schema with additional predicate symbols `manager/3` and `worker/3` that a user can now reference in queries.
2.2 The Physical Schema

We use a similar strategy to define the physical schema where we again use relational signatures and constraints for this purpose. However, these symbols will correspond to actual data structures that are called access paths in database literature. These access paths correspond to various ways to access data, ranging from dereferencing a pointer in main memory or extracting a field from a main memory record (abstracted by binary predicate symbols whose interpretations are address-value pairs) to using main memory data structures such as linked lists (again abstracted by appropriate predicate symbols) to reading data from external storage, and to communicating with other agents. The situation can be again depicted as follows:

Note that there can be additional helper predicate symbols in $S_P$ in addition to the access paths $S_A$.

There are two issues with this strategy that must be addressed:

1. Will it suffice to associate access paths (data structures and their associated search algorithms) with predicate symbols?
2. Is it reasonable to also think about generated code using access paths as formulae ($\psi$ above)?

To address the 1st question, we annotate the symbols in $S_A$ with so called binding patterns [19] indicating which arguments of the particular access path must be bound to a value before the access path can be executed. We indicate this by an additional integer in the signature specification, for example “pointer-nav/2/1” indicates that the access path representing address-value pairs in main memory can be only used when we have a value for the first component (i.e., an address). The implementation then consists of a simple statement for dereferencing this address to produce the a value of the second argument. This observation also leads to restrictions on the form of $\psi$ [16].

Example 2 (Physical Schema)

We illustrate the first issue by defining the physical schema for our running example. Our physical design consists of a linked list of employee records that use pointers (references) to indicate department records an employee works in. In a similar fashion, the department records use a pointer to indicate which
employee is a manager. The records in a Pascal-like notation are as follows:

<table>
<thead>
<tr>
<th>record emp of</th>
<th>record dept of</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer num</td>
<td>integer num</td>
</tr>
<tr>
<td>string name</td>
<td>string name</td>
</tr>
<tr>
<td>reference dept</td>
<td>reference mgr</td>
</tr>
</tbody>
</table>

In our formalism this looks as follows: we define the following predicates to be associated with access paths (i.e., in $S_A$):

- empfile/1/0: set of addresses of emp records; this access path abstracts navigating a linked list (of emp records) in main memory.
- emp-num/2/1: a set of address of emp records paired with the emp numbers; this access path corresponds to extracting a field (num in this case) from an emp record (given an address of such a record). The access paths emp-name/2/1 and emp-dept/2/1 and dept-num/2/1, dept-name/2/1, and dept-mgr/2/1 similarly abstract the field extraction of the remaining fields from the emp and dept records.

We also use two auxiliary predicates emp/1 and dept/1 to stand for the sets of addresses of emp and dept records. Integrity constraints ($\Sigma_P \cup \Sigma_{LP}$) then capture the properties of instances of the physical schema and how they relate to the logical schema. For example the fact that records have appropriate fields can be specified as follows:

\[
\begin{align*}
\text{emp}(e) \rightarrow & \exists d. \text{emp-dept}(e, d) & \text{emp records have a dept field} \\
\text{emp-dept}(e, d_1) \land \text{emp-dept}(e, d_2) \rightarrow & d_1 = d_2 & \text{the dept field is functional} \\
\text{emp-dept}(e, d) \rightarrow & \text{dept}(d) & \text{the value of the dept field is a pointer to a dept record}
\end{align*}
\]

For full listing of the constraints see Appendix A. This completes our description of the physical schema for our example.

### 2.3 Queries and Plans

Now we are ready to give an answer to our 2nd question, how to interpret formulae as query plans. This is straightforward: atomic formulae are mapped to (the code associated with) access paths and logical connectives and quantifiers to “control flow code fragments” as follows:

- atomic formula $\rightarrow$ access path (a get-first / get-next iterator)
- conjunction $\rightarrow$ nested loops join
- existential quantifier $\rightarrow$ projection (with optional duplicate information)
- disjunction $\rightarrow$ concatenation
- negation $\rightarrow$ simple complement

For a formula to correspond to a plan (i.e., executable code), it is also necessary to obey binding patterns [16]. While such a procedural interpretation of atoms
and logical connectives might seem over simplistic, we discuss in Section 3.2 below how this simple fine-grained interpretation suffices for most of the hard-coded solutions in other database systems.

**Example 3**

We illustrate this framework by worked examples of several user queries together with possible query plans for these queries over our running physical design case.

**Q1: List employee numbers, names, and departments** \((\text{employee}(x,y,z))\). We can show that this user query is **logically equivalent** under the integrity constraints to the following formula over \(S_A\):

\[
\exists e, d. \text{empfile}(e) \land \text{emp-num}(e,x) \land \text{emp-name}(e,y) \\
\land \text{emp-dept}(e,d) \land \text{dept-num}(d,z)
\]

Assuming our formulas as plans mapping, this formula would correspond to the following C-like code (with trivial simplifications and inlining of the \(\text{ea-xxx}(x,y)\) access paths to \(y := x->xxx\)):

```c
for e in empfile do
  x := e->num;  y := e->name;
  d := e->dept; z := d->num; return (x, y, z);
```

Note also that the formula above satisfies the binding patterns associated with the access paths used as it retrieves the address of an emp record **before** attempting to extract the values of it’s fields.

**Q2: List worker numbers and names** \((\exists z. \text{worker}(x,y,z))\). Again, this query is equivalent to the following formula over \(S_A\):

\[
\exists e, d. \text{empfile}(e) \land \text{emp-num}(e,x) \land \text{emp-name}(e,y) \\
\land \text{emp-dept}(e,d) \land \neg \text{dept-mgr}(d,e)
\]

Note that a negation, \(\neg \text{dept-mgr}(d,e)\), is required, and that there is no negation in the query nor in the schema that provides any direct clue that it is needed. (We are not aware of any system that can synthesize this plan, that is, that compiles queries using this framework.)

**Q3: List all department numbers and their names** \((\exists z. \text{department}(x,y,z))\). Finding a plan for this query is more difficult since we do not have a direct way to “scan” dept records. However, it is an easy exercise to verify that the following two formulae over \(S_A\) are logically equivalent to the query:

\[
\exists d, e. \text{empfile}(e) \land \text{emp-dept}(e,d) \\
\land \text{dept-num}(d, x) \land \text{dept-name}(d, y)
\]

(relying on the constraint that “departments have at least one employee”)  

\[
\exists d, e. \text{empfile}(e) \land \text{emp-dept}(e,d) \\
\land \text{dept-num}(d, x) \land \text{dept-name}(d, y) \land \text{dept-mgr}(d,e)
\]

(relying on the constraint that “managers work in their own departments”)
Both correspond to plans. However, while the second might seem to be less efficient than the first, a query optimizer should prefer it on the grounds that, in this case, the quantified variables \( d \) and \( e \) are *functionally determined* by the answer variable \( x \). Hence, the final projection generated for the second has no need to eliminate duplicate answers. This is not the case for the first of these formulae since it would return a copy of the department information for every employee of the department should duplicate elimination in the final projection not be performed.

Many other problems and issues in physical design and query plans can be revolved in this framework, including standard RDBMS physical designs (and more), access to search structures (index access and selection), horizontal partitioning/sharding, column store/index-only plans, hash-based access to data (including hash-joins), multi-level storage (aka disk/remote/distributed files), materialized views, etc., all without any need for coding in C beyond the need for the generic specifications of *get-first / get-next* templates for concrete data structures [16].

### 3 Interpolation and Query Optimization

Now we turn our attention to the description of a query compiler/optimizer that, given the logical and physical schemata and a user query, generates a query plan that correctly implements the user request.

#### 3.1 What Queries Make Sense?? (to users)

However, before we begin, it is important to resolve what queries make sense to a user who presumes there is a *single interpretation* of symbols in \( S_L \) at any point in time, no matter how it is represented/stored physically. To satisfy to this expectation, the queries that make sense should have the same answer in every model of the overall physical design \( \Sigma \) in which the interpretation of \( S_A \) is fixed, that is, where the stored data is always the same. This arrangement also guarantees that artifacts facilitating efficient storage and retrieval of information won’t be leaked in the results of queries (since they do not exist in the logical view of the data). The consequence of this observation is that either

1. there are situations in which a seemingly reasonable user query cannot be answered (that would be the case for \( Q_3 \) in Section 2.3, were the constraint “departments have at least one employee” absent from the schema), or
2. queries must adhere to syntactic restrictions in which, e.g., symbols corresponding to built-in operations cannot be used completely freely, and physical designs must also adhere to syntactic restrictions such as so-called *standard designs* (i.e., where an access path exists for every logical table in \( \Sigma_L \), thus guaranteeing that every user query can be answered).

To make the definition of *sensible queries* more formal, we appeal to a well-known notion of *definability*:
Proposition 4 (Projective Beth Definability [4])

Let $\Sigma \cup \{\varphi\}$ be a FO theory over symbols in $L$ and $L' \subseteq L$. Then t.f.a.e.:

1. For all $M_1, M_2$ models of $\Sigma$ such that $M_1|_{L'} = M_2|_{L'}$, and all $a$ tuples of individuals, it holds that $M_1 \models \varphi[a]$ iff $M_2 \models \varphi[a]$, and
2. $\varphi$ is equivalent under $\Sigma$ to some formula $\psi$ in $L'$.

We say that $\varphi$ is explicitly definable w.r.t. $\Sigma$ and $L'$.

Definability (over $S_A$ w.r.t. $\Sigma$) formally captures the idea of (physical) data independence, the illusion of a single interpretation of the logical schema that satisfies integrity constraints that is presented to the users, and therefore provides the means of determining which queries can be answered over a particular physical design.

The first question is how to test for definability. The following observation reduces this test to determining whether a particular formula constructed from the user query is entailed by a theory constructed from the schema: $\varphi$ is explicitly definable (w.r.t. $\Sigma$ and over $S_A$) if and only if

$$\Sigma \cup \Sigma' \models \varphi \rightarrow \varphi'$$

(1)

where $\Sigma'$ ($\varphi'$) is $\Sigma$ ($\varphi$) in which symbols NOT in $S_A$ are primed, respectively.

The next question is how to find a plan for a given query. Our observations on how formulae can be interpreted as query plans in Section 2.3 then mostly reduces query compilation to a search for the formula $\psi$ in Proposition 4(2). To find $\psi$, we rely on a variant of the following result [7]:

If $\Sigma \cup \Sigma' \models \varphi \rightarrow \varphi'$ then there is $\psi$ s.t. $\Sigma \cup \Sigma' \models \varphi \rightarrow \psi \rightarrow \varphi'$

where $L(\psi) \subseteq L(S_A)$. Here, $\psi$ is called the Craig interpolant. Moreover, we can extract any such $\psi$ from a TABLEAU proof of (1) in linear time [9].

3.2 Architecture

The above discussion might seem to solve the query compilation problem. However there are additional issues that need to be addressed:

1. The search for interpolants and their implied query plans must consider that alternative but logically equivalent plans might have vastly different performance characteristics.\(^1\) Hudek et al. [11] introduce an approach that separates the tableau-based search for interpolants from the cost-based exploration of alternative query plans.

---

\(^1\) This holds even for conjunctive formulae: hence database literature often focuses on the so-called join-order problem [15].

\(^2\) Cost-based query optimization is the cornerstone of relational systems [15]; advancements in the area of query plan cost estimation are easily incorporated in this framework.
2. Binding patterns for access paths (see Section 2.2) further restrict the space of executable query plans (and in turn of sensible queries). Benedikt et al. [3] have shown how the binding patterns can be accommodated in the search for interpolants (i.e., in the search for proofs of definability).

3. In addition, during the search for optimal query plans, we consider the impact of duplicate elimination as illustrated by plans for Q3 in Section 2.3. A detailed account for this facet of query compilation can be found in [16,18].

Figure 1 sketches an architecture of a query compilation/optimization system that addresses the above concerns. The compiler preprocesses the given schema and the user query into a normal form and generates a bytecode that drives a virtual machine-based (VM) tableau theorem prover [17]. Unlike standard theorem provers, including those that can generate interpolants [12], the tableau VM generates an intermediate representation of a space of equivalent interpolants called closing sets [11]. Closing sets are then explored by an A*-based planner to find a query plan with the lowest estimated cost. The planner also explores ways to avoid duplicate elimination in the process. The planner is then followed by a code generator that produces the ultimate query plans in a form of C source.

4 Updates

In this section, we sketch how the problem of compiling updates on a logical design can be translated to the problem of compiling queries on a related logical design, thus enabling the same framework above to also be used to compile inserts, updates and deletes on logical tables.

As already mentioned in Section 2.1, user updates are formulated with respect to the logical schema (SL and ΣL). Moreover, physical data independence presents the user with an illusion that he is modifying an instance of SL by adding/removing ground tuples to/from the interpretations of symbols in SL. This process can be formalized in three parts as follows:
1. For every symbol \( R \in S_L \), introduce two additional symbols, \( R^+ \) and \( R^- \), whose (disjoint) interpretations correspond to the ground tuples the user wants to add or remove to/from the current instance;

2. The updated instance is then defined by executing an simultaneous assignments \( R := (R \cup R^+) \setminus R^- \) for all \( R \in S_L \); and

3. At the end of the assignment the new interpretation must be a model of \( \Sigma_L \).

The symbols \( R^+ \) and \( R^- \) are commonly called the *delta relations* and Part 3 of this process on user updates ensures so-called *consistency preserving transactions*.

To convert the update problem to the problem of synthesizing plans for queries, consider two copies of the schema \( \Sigma \), in which all symbols are super-scripted by \( o \) and \( n \), respectively. The intuition is that the \( o \) and \( n \) symbols correspond to the interpretations of \( S_L \) before and after the update. The actual assignment (Part 2 of the above process) can be then captured as additional formulae

\[
R^o(x) \lor R^-(x) \leftrightarrow R^n(x) \lor R^+(x)
\]

for each \( R \in S_L \) as depicted below.

In the same way, the changes to access paths in \( S_A \) can be captured by analogous constraints, as depicted in the lower half of the figure. Thus, user inserts, updates and deletes on logical tables (comprising a transaction) are mapped to a *definability* question of the following form:

Is \( A^n \) (or \( A^+, A^- \)) definable in terms of \( A^o \) and \( U^+_j, U^-_j \) (user updates) for every access path \( A \in S_A \), given the instance of all access paths in \( S_A \) and of all *delta relations* for \( S_L \)?

A positive answer to this question yields a *update plan* that applies the delta relations corresponding to the access paths to their current interpretations.

5 Summary

We have outlined how projective Beth definability can be used in database and information systems to facilitate physical data independence. Moreover, we have
shown how a variation on Craig interpolation can be used to compile and optimize user queries and user updates that are formulated over a logical schema to an executable plan over a fine-grained physical design. There are many avenues for further research and development, including: (1) admitting more powerful languages for user requests, such as languages with aggregation; (2) enhancements to the tableau provers, as well as alternatives such as superposition-based provers; and (3) improvements to the planning component of query compilation responsible for exploring the search space of alternative query plans.

References

A Constraints for the Running Example

The following listing is a complete specification of constraints needed for our running example. Note that some of the constraints in Section 2.1 are entailed by the constraints below (and are thus omitted).

```
% logical schema (entailed constraints omitted)
%
% a (virtual) view for managers
manager(x,y,z) <-> (employee(x,y,z) and ex(n,department(z,n,x))),
%
% disjoint partition of employees to managers and workers
employee(x,y,z) <-> (manager(x,y,z) or worker(x,y,z)),
manager(x,y,z) and worker(x,u,v) -> bot,
%
% business logic: managers work for their own departments
(department(x,y,z) and employee(z,u,w)) -> x=w,
%
% physical schema and mappings
%
% design of emp and dept structs; emp/dept addresses, fields functional
emp(e) -> ex(y,emp_num(e,y)), emp_num(e,y) and emp_num(e,z) -> y=z,
  emp_num(y,x) and emp_num(z,x) -> y=z,
emp(e) -> ex(y,emp_name(e,y)), emp_name(e,y) and emp_name(e,z) -> y=z,
emp(e) -> ex(y,emp_dept(e,y)), emp_dept(e,y) and emp_dept(e,z) -> y=z,
  emp_dept(e,d) -> dept(d),
%
depth(d) -> ex(y,dept_num(d,y)), dept_num(d,y) and dept_num(d,z) -> y=z,
  dept_num(y,x) and dept_num(z,x) -> y=z,
depth(d) -> ex(y,dept_name(d,y)), dept_name(d,y) and dept_name(d,z) -> y=z,
depth(d) -> ex(y,dept_mgr(d,y)), dept_mgr(d,y) and dept_mgr(d,z) -> y=z,
  dept_mgr(d,e) -> emp(e),
```

17. David Toman and Grant E. Weddell. An interpolation-based compiler and optimizer for relational queries (system design report). In IWIL@LPAR 2017 Workshop and LPAR-21 Short Presentations, Maun, Botswana, May 7-12, 2017, 2017.
% linked list for ea’s and record attributes

empfile(x) <-> emp(x),

% user predicates and mappings

employee(x,y,z) <-> ex(baseemployee(e,x,y,z)),

emp(e) <-> ex([x,y,z],baseemployee(e,x,y,z)),

emp_num(e,x) <-> ex([y,z],baseemployee(e,x,y,z)),

emp_name(e,y) <-> ex([z,x],baseemployee(e,x,y,z)),

ex(d,emp_dept(e,d) and dept_num(d,z)) <-> ex([x,y],baseemployee(e,x,y,z)),

department(x,y,z) <-> ex(basedepartment(d,x,y,z)),

dep(d) <-> ex([x,y,z],basedepartment(d,x,y,z)),

dep_num(d,x) <-> ex([y,z],basedepartment(d,x,y,z)),

dep_name(d,y) <-> ex([z,x],basedepartment(d,x,y,z)),

ex(e,dept_mgr(d,e) and emp_num(e,z)) <-> ex([x,y],basedepartment(d,x,y,z))
In logics with the Craig interpolation property (CIP) the existence of an interpolant for an implication follows from the validity of the implication. In logics with the projective Beth definability property (PBDP), the existence of an explicit definition of a relation follows from the validity of a formula expressing its implicit definability. From an algorithmic viewpoint, the CIP and PBDP are of interest because they reduce existence problems to validity checking: an interpolant exists if, and only if, an implication is valid and an explicit definition exists if, and only if, a straightforward formula stating implicit definability is valid. The interpolant and explicit definition existence problems are thus not harder than validity.

While many logics enjoy the CIP and the PBDP (for instance, first-order logic (FO), propositional logic, intuitionistic logic, and many modal and description logics), there are also many important logics that neither enjoy the CIP nor the PBDP. Examples include modal and description logics with nominals, the two-variable fragment of FO, the guarded fragment of FO, and most Horn-fragments of modal and description logics. In this talk, I will present recent results on the decidability and complexity of interpolant and explicit definition existence for logics that do not enjoy the CIP nor PBDP. For example, we show that the existence of explicit definitions of concept names (and individual names) relative to an ontology in the extension $\mathcal{ALCO}$ of $\mathcal{ALC}$ with nominals is $2\text{ExpTime}$-complete and that the existence of explicit definitions of relation in the guarded fragment is $3\text{ExpTime}$-complete, thus in both cases by one exponential harder than deduction. The presentation is based on [1,3,2].

References

Resolution-Based Uniform Interpolation for Multi-Agent Modal Logic $K_n$

Ruba Alassaf, Renate A. Schmidt, and Uli Sattler

University of Manchester, UK
{ruba.alassaf,renate.schmidt,uli.sattler}@manchester.ac.uk

Abstract. Research on uniform interpolation in modal logic has been largely focused on the theoretical investigation of the problem. This paper presents a system to compute uniform interpolants for a locally satisfiable formula in the multi-agent modal logic $K_n$. The system is based on a direct resolution approach. The idea of the system is that given a formula $\phi$ and a signature as input, it computes the strongest local consequence of $\phi$ over the input signature. We have shown that the system is guaranteed to terminate, soundness and completeness can be shown using model-theoretic proofs, and the worst-case space complexity bound is double exponential. We illustrate how the system is used via examples.

Keywords: Uniform Interpolation · Resolution · Modal logic · $K_n$.

1 Introduction

Uniform interpolation is the task of computing a formula that captures all logical consequences up to a given signature for a given formula. The problem of computing a uniform interpolant is generally not decidable. We are interested in logics that have the uniform interpolation property: that is, the property that for any formula $\phi$ and any signature $\Sigma$, a uniform interpolant for $\phi$ over $\Sigma$ exists in the logic.

Uniform interpolation amounts to the second-order quantifier elimination problem: given a formula with an existential second-order quantifier prefix, the process of eliminating the second-order quantifiers is essentially the process of producing a formula that does not contain the symbols that are being quantified.

A related notion is Craig interpolation. A logic has the Craig interpolation property if given two formulas $\phi$ and $\psi$ with a shared signature $\Sigma$ such that $\phi \rightarrow \psi$ holds, there exists a middle formula $\phi'$ such that $\phi \rightarrow \phi'$ and $\phi' \rightarrow \psi$ hold. A uniform interpolation method can be used to compute Craig interpolants by performing uniform interpolation to keep the shared signature $\Sigma$.

The importance of having uniform interpolation methods stems from the significant potential it has for applications. For example, in agent-based applications, it is often assumed that agents communicate using the same language.
Uniform interpolation becomes very useful when this assumption is relaxed; it can be used to allow an agent to express knowledge about a certain topic by computing a view that only uses some signature symbols. This gives agents the ability to share their knowledge with other agents who specialise in different domains.

The modal logic community has focused on uncovering theoretical results. It has been shown via constructive proofs that the modal logic $K$ has the uniform interpolation property [7,14]. An approach to constructing uniform interpolants was given in [2] for the modal logics $K$ and $T$. Wolter [15] proved that the modal logic $S5$ has the uniform interpolation property, and that uniform interpolation for any normal single-agent modal logic can be generalised to its multi-agent case. Recently, it was shown that $K45_n$ and $KD45_n$ have the uniform interpolation property in [4]. It is known that $S4$ and $K4$ do not have the uniform interpolation property [8].

This paper presents the first complete resolution-based system for computing uniform interpolants in the multi-agent modal logic $K_n$. As far as the authors know, the only other paper which considers this logic is [4]. Different from our method, they construct a uniform interpolant by considering canonical formulas, which are conceptually simple but, as the authors explicitly state, inefficient to compute [4]. We show that our system has double exponential worst-case space complexity. We prove that the termination of our method is guaranteed, and that it is sound and complete. We are the first to use bisimulations to prove completeness for a resolution-based uniform interpolation system. We illustrate how the method is used via examples. Due to the lack of space, proof are provided in the full version of the paper which can be found here: https://personalpages.manchester.ac.uk/staff/ruba.alassaf/publications.html

2 Preliminaries

We assume the reader is familiar with the multi-modal logic $K_n$ [5,10]. We use $\mathcal{F} = (W, R)$ to denote a Kripke frame and $\mathcal{M} = (W, R, V)$ to denote a Kripke model. A formula $\phi$ is (locally) satisfiable in a model $\mathcal{M}$, denoted $\mathcal{M}, w \models \phi$, if there is a point $w$ in $W$ at which $\phi$ is true. A formula $\phi$ is (unconditionally) satisfiable if it is true at some point in some model. A formula $\phi$ is globally satisfied (or true) in a model $\mathcal{M}$, denoted $\mathcal{M} \models \phi$, if it is true at every $w$ in $W$. A formula $\phi$ is valid if it is satisfied in all models over any frame $\mathcal{F}$. A set of formulae $N$ is globally satisfied by a model $\mathcal{M}$, denoted $\mathcal{M} \models N$, if for each formula $\phi$ in $N$, $\mathcal{M}$ globally satisfies $\phi$.

We are interested in the problem of computing a uniform interpolant of a locally satisfiable formula and a signature.

**Definition 1 (Uniform Interpolation).** Given a formula $\phi$, a uniform interpolant of $\phi$ with respect to a signature $\Sigma$ of propositional symbols is a formula $\phi'$ such that:

1. $\phi'$ does not contain symbols outside of $\Sigma$, and
2. for any modal formula $\psi$ over $\Sigma$, we have that for all models $\mathcal{M}$, $\mathcal{M} \models \phi \rightarrow \psi$ iff for all models $\mathcal{M}$, $\mathcal{M} \models \phi' \rightarrow \psi$.

3 Related Work

In this section, we outline the methods we found related to our method, and explain how our method is different to these systems. A summary of the related methods is given in Table 1. In the table, we give the logic over which each method is defined, the expressivity of the input and output, and we state if the method is complete.

The first method is a uniform interpolation algorithm of Bilkova [2]. In her work, she describes an approach for constructing a uniform interpolant from a table. She uses a sequent calculus to prove that her algorithm is sound and complete.

The second is a resolution-based calculus introduced in Herzig and Ménigin [9]. There are two differences to our method, the first is that the method proposed by the present paper is for $K_n$ which is an extension of $K$, and the second is that we use a kind of labelling technique that allows us to flatten the input and apply resolution almost classically.

There are three more resolution-based systems for computing uniform interpolation: the SCAN approach [6] for first-order logic, and the LETHE system [11] and the system of Ludwig and Konev [12], both for description logics. These systems are designed for logics where a solution does not always exist. In the case of SCAN, the computation may not terminate [6]. In the case of LETHE, nominals/definer symbols may remain in the solution [11], or solutions may be approximated by a depth bound as in the method in [12]. We prove that a solution is always achievable via our method in a finite number of steps and without extending the logic or the signature. The completeness proofs provided for these methods are based on consequence finding, whereas our proof uses bisimulations. Moreover, compared to [11], the method we describe does not use unification-based reasoning.

Finally, second-order quantifier elimination methods which can be used to compute uniform interpolants often use Ackermann’s lemma [1]. Such methods include the DLS algorithm [3] for second-order quantifier elimination of first-order logic formulae, the MA system [13] for computing frame correspondence properties for modal axioms and the FAME tool [16] computing semantic forgetting in description logic.

4 Uniform Interpolation Method $UI_{K_n}$ for $K_n$

We start with a high-level description of our uniform interpolation system for multi-modal logic $K_n$. 
The calculus is based on resolution, with adaptations for modal logic. The idea behind our approach is the following: for each symbol $x$ outside the given signature $\Sigma$, we generate a sufficient set of conclusions for the given formula and subsequently eliminate any formulae that contain $x$. We repeat the process for all propositional symbols outside $\Sigma$.

The calculus uses special world symbols, or $W$-symbols for short, which are propositional symbols that help in two related ways:

1. They are used to flatten the input formula to surface some parts of it. E.g., $\square(\psi \lor \Diamond \phi)$ becomes $\square W_1$, $W_1 \Rightarrow \psi \lor \Diamond W_2$ and $W_2 \Rightarrow \phi$.
2. They allow our rules to detect legal inferences between the subformulae by labelling them with a $W$-symbol. E.g., $\square (x \land (\neg x \lor p))$ becomes $\square W$, $W \Rightarrow x$ and $W \Rightarrow \neg x \lor p$. Later on, we see that one of our rules allows us to apply a resolution step on $x$.

The idea behind using $W$-symbols is similar to using constants in a labelled tableau algorithm.

For a formula $\phi$, a signature $\Sigma$, and an ordering $\succ$ over the symbols outside the input signature $\Sigma$, the calculus is provided a clause set $N_0 = \{W_0 \Rightarrow \phi\}$ as input, and applies its rules exhaustively to the formulae in the set until no rule can be applied, resulting in a clause set of the form $N_n = \{W_0 \Rightarrow \phi_1, ..., W_0 \Rightarrow \phi_m\}$. The formula $\phi' = \phi_1 \land \ldots \land \phi_m$ is then a uniform $\Sigma$-interpolant of $\phi$, which is proved later.

The role of $W_0$ is to capture a specific world that satisfies $\phi$. Any model $M$ that satisfies $\phi$ at point $w$ can be extended to one that satisfies $W_0$ and $W_0 \Rightarrow \phi$ in a non-vacuously way by setting $w \in V(W_0)$. In this extended model, $W_0 \Rightarrow \phi$ is globally and witnessed as non-vacuously true.

The process of constructing a uniform interpolant is iterative with respect to the symbols outside $\Sigma$, and the ordering $\succ$ fixes the order in which these symbols are eliminated. For some uniform interpolation problems, a good ordering may allow the calculus to solve a problem in far fewer steps. For simplicity, and since the ordering does not improve any worst-case complexity results, we can assume

Table 1: A comparison between our method $UIK_n$ for $K_n$ and related methods.

<table>
<thead>
<tr>
<th>Logic</th>
<th>Method</th>
<th>Input Language</th>
<th>Output Language</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modal logic</td>
<td>Resolution</td>
<td>$K_n$ (locally satisfiable formula)</td>
<td>$K_n$ (locally satisfiable formula)</td>
<td>Yes</td>
</tr>
<tr>
<td>Modal logic</td>
<td>Sequent</td>
<td>$K$ (locally satisfiable formula)</td>
<td>$K$ (locally satisfiable formula)</td>
<td>Yes</td>
</tr>
<tr>
<td>Modal logic</td>
<td>Resolution</td>
<td>$K$ (locally satisfiable formula)</td>
<td>$K$ (locally satisfiable formula)</td>
<td>Yes</td>
</tr>
<tr>
<td>First-order logic</td>
<td>Resolution</td>
<td>Full first-order logic</td>
<td>Full first-order logic</td>
<td>No</td>
</tr>
<tr>
<td>Description logic</td>
<td>Resolution</td>
<td>$ALC(Tbox)$</td>
<td>$ALC(Tbox)$</td>
<td>No</td>
</tr>
<tr>
<td>Description logic</td>
<td>Resolution</td>
<td>$ALC(\mu(Tbox) + Abox)$</td>
<td>$ALC(\mu(Tbox) + Abox)$</td>
<td>No</td>
</tr>
<tr>
<td>Description logic</td>
<td>Ackermann</td>
<td>$Full first-order logic</td>
<td>$Full first-order logic</td>
<td>No</td>
</tr>
</tbody>
</table>

4.1 Overview

The calculus is based on resolution, with adaptations for modal logic. The idea behind our approach is the following: for each symbol $x$ outside the given signature $\Sigma$, we generate a sufficient set of conclusions for the given formula and subsequently eliminate any formulae that contain $x$. We repeat the process for all propositional symbols outside $\Sigma$.

The calculus uses special world symbols, or $W$-symbols for short, which are propositional symbols that help in two related ways:

1. They are used to flatten the input formula to surface some parts of it. E.g., $\square(\psi \lor \Diamond \phi)$ becomes $\square W_1$, $W_1 \Rightarrow \psi \lor \Diamond W_2$ and $W_2 \Rightarrow \phi$.
2. They allow our rules to detect legal inferences between the subformulae by labelling them with a $W$-symbol. E.g., $\square (x \land (\neg x \lor p))$ becomes $\square W$, $W \Rightarrow x$ and $W \Rightarrow \neg x \lor p$. Later on, we see that one of our rules allows us to apply a resolution step on $x$.

The idea behind using $W$-symbols is similar to using constants in a labelled tableau algorithm.

For a formula $\phi$, a signature $\Sigma$, and an ordering $\succ$ over the symbols outside the input signature $\Sigma$, the calculus is provided a clause set $N_0 = \{W_0 \Rightarrow \phi\}$ as input, and applies its rules exhaustively to the formulae in the set until no rule can be applied, resulting in a clause set of the form $N_n = \{W_0 \Rightarrow \phi_1, ..., W_0 \Rightarrow \phi_m\}$. The formula $\phi' = \phi_1 \land \ldots \land \phi_m$ is then a uniform $\Sigma$-interpolant of $\phi$, which is proved later.

The role of $W_0$ is to capture a specific world that satisfies $\phi$. Any model $M$ that satisfies $\phi$ at point $w$ can be extended to one that satisfies $W_0$ and $W_0 \Rightarrow \phi$ in a non-vacuously way by setting $w \in V(W_0)$. In this extended model, $W_0 \Rightarrow \phi$ is globally and witnessed as non-vacuously true.

The process of constructing a uniform interpolant is iterative with respect to the symbols outside $\Sigma$, and the ordering $\succ$ fixes the order in which these symbols are eliminated. For some uniform interpolation problems, a good ordering may allow the calculus to solve a problem in far fewer steps. For simplicity, and since the ordering does not improve any worst-case complexity results, we can assume
that this ordering is arbitrary. We use $x$ to denote the maximal propositional symbol occurring in the current clause set $N_i$.

### 4.2 The Calculus

The rules of our uniform interpolant calculus are given in Figures 1, 2 and 3. Each rule has a premise, some conditions and a conclusion. The rules are structured with the premise above a horizontal line and the conclusion below it. The premise (respectively conclusion) can be one or more clauses depending on which rule is being applied. There are three types of rules in the calculus: preprocessing rules, resolution rules, and elimination rules.

The preprocessing rules and the elimination rules are replacement rules; they replace the premise in the current working clause set with the conclusion. The resolution rules are saturation rules; they keep the premise and extend the clause set with the conclusion. The rules can be applied in any order as long as the conditions for each rule are met.

Generally, we can expect that for a formula in the clause set, it is preprocessed into another formula, or formulae, that is then involved in a few resolution rule applications and subsequently purified, if an elimination rule is applicable.

The clauses obtained and handled by our calculus are in a normal form. They are all labelled with a $W$-symbol in the condition of the implication. We can have a formula or another $W$-symbol in the consequence of the implication. Concretely, for some $W$-symbols $W_i$ and $W_j$, and some modal formula $\psi$, a clause can be in the form

$$W_i \Rightarrow \psi \quad \text{or} \quad W_i \Rightarrow W_j.$$  

If $\psi$ is a disjunction of modal formulas, we assume that it is a set, i.e., there is no repetition. This is essential for the correctness of the method. We use $\Rightarrow$, in contrast to $\rightarrow$, to distinguish an implication that is generated by our system, to maintain our normal form, from an implication provided as part of the input. Semantically, they are identical.

To describe the different types of $W$-symbols, we introduce some terminology and the function $\text{Corr}$ which will be used in the conditions of our system, and later on in the proofs.

**Definition 2.** Given a set $N$ of clauses, the set $S_w$ is the set of $W$-symbols introduced for subformulas appearing under a modal operator via the world introduction rule. We call these symbols base $W$-symbols.

The set $C_w$ is the set of $W$-symbols introduced by the $\Box\Box$ rule. We call these symbols combinatory $W$-symbols.

We define a function $\text{Corr}$ that maps $W$-symbols to subsets of $S_w$ as follows:

$$\text{Corr}(W_i) = \begin{cases} 
\{W_i\}, & \text{if } W_i \in S_w \\
\text{Corr}(W_n) \cup \text{Corr}(W_m), & \text{if } W_i \in C_w \text{ where } W_n \text{ and } W_m \text{ come from the premise of the } \text{Res } \Box\Box \\
& \text{rule that has introduced } W_i.
\end{cases}$$
Intuitively, a base $W$-symbol is introduced to represent a subformula, and a combinatory $W$-symbol can be seen as a unique representative of a subset of the base $W$-symbols.

We now describe the three groups of rules which together make up our calculus. We use $N$ to refer to the current working clause set. We assume that $x$ is the current symbol we would like to eliminate, i.e., it is the maximal symbol with respect to a given ordering $\succ$ for symbols outside $\Sigma$. The $W$-symbol $W_i$ is the $i$th $W$-symbol introduced during the inference process.

**Preprocessing.** The purpose of the preprocessing rules is to apply transformations to the members of the working clause set so that they can be handled by the other rules. Generally, the idea is to surface symbols appearing in $\phi$ that are not in $\Sigma$, i.e., to surface $x$ in $\phi$.

The normal form is based on pushing negation inwards, clausifying and applying structural transformation. The rules are applied in a lazy manner which means their application can be deferred to whenever they are necessary. The preprocessing rules are provided in Figure 1.

The first five rules are standard rules to transform modal formulae into negation normal form. The clausification rule distributes disjunction over conjunction. The world introduction rule performs structural transformation that flattens the modal formulae. Consider a clause $W_i \Rightarrow \neg \neg \psi$, the first negation normal form rule replaces this clause with $W_i \Rightarrow \psi$, so the original clause is no longer in the working set.

**Resolution.** The second type of rules are the resolution rules. The purpose of these rules is to deduce a sufficient number of clauses/formulas to generate a uniform interpolant. The rules are given in Figure 2.

The literal resolution rule is the heart of our calculus; it computes a formula by resolving on a maximal symbol $x$ if the premise is labelled with the same $W$-symbol. The world resolution rule is used to propagate formulas labelled by another $W$-symbol, which is essentially a resolution step between world symbols. The $\Box \Box$ resolution rule is used to capture combinations of successor relations. The second and third conditions are the blocking conditions; they aim to ensure that the rule application is not redundant which is important for complexity, and that the calculus does not infinitely introduce $W$-symbols which is essential for termination.

**Elimination.** The last type of rules are the elimination rules. These rules are responsible for eliminating symbols outside of $\Sigma \cup \{W_0\}$. They are applied once we have exhaustively applied the resolution rules to compute conclusions over $\Sigma$. The rules are given in Figure 3.

The positive and negative purification rules replace a maximal symbol $x$, occurring either positively or negatively, with $\top$. The world elimination rule collects modal formulas labelled with the same $W$-symbol, and replaces right
Negation Normal Form (1):
\[
N, W_i \Rightarrow \neg\neg\phi_1 \lor \phi_2
\]
\[
N, W_i \Rightarrow \phi_1 \lor \phi_2
\]
provided that \( \phi_1 \) contains \( x \).
\( \phi_2 \) may be empty.

Negation Normal Form (2):
\[
N, W_i \Rightarrow (\neg(\phi_1 \land \phi_2) \lor \phi_3)
\]
\[
N, W_i \Rightarrow \neg\phi_1 \lor \neg\phi_2 \lor \phi_3
\]
provided that either \( \phi_1 \) or \( \phi_2 \) contain \( x \).
\( \phi_3 \) may be empty.

Negation Normal Form (3):
\[
N, W_i \Rightarrow -\phi_1 \lor \phi_3
\]
provided that either \( \phi_1 \) or \( \phi_2 \) contain \( x \).
\( \phi_3 \) may be empty.

Negation Normal Form (4):
\[
N, W_i \Rightarrow \neg\Diamond a \phi_1 \lor \phi_2
\]
\[
N, W_i \Rightarrow \Box a \neg\phi_1 \lor \phi_2
\]
provided that \( \phi_1 \) contains \( x \).
\( \phi_2 \) may be empty.

Negation Normal Form (5):
\[
N, W_i \Rightarrow \neg\Box a \phi_1 \lor \phi_2
\]
\[
N, W_i \Rightarrow \Diamond a \neg\phi_1 \lor \phi_2
\]
provided that \( \phi_1 \) contains \( x \).
\( \phi_2 \) may be empty.

Implication Elimination:
\[
N, W_i \Rightarrow (\phi_1 \rightarrow \phi_2) \lor \phi_3
\]
\[
N, W_i \Rightarrow \neg\phi_1 \lor \phi_2 \lor \phi_3
\]
provided that either \( \phi_1 \) or \( \phi_2 \) contain \( x \).
\( \phi_3 \) may be empty.

Clausification:
\[
N, W_i \Rightarrow (\phi_1 \land \phi_2) \lor \phi_3
\]
\[
N, W_i \Rightarrow \phi_1 \lor \phi_2 \lor \phi_3
\]
provided that either \( \phi_1 \) or \( \phi_2 \) contain \( x \).
\( \phi_3 \) may be empty.

World Introduction (\text{INT } W):
\[
N, W_i \Rightarrow \Box a \phi_1 \lor \phi_2
\]
\[
N, W_1 \Rightarrow \Box a \phi_1 \lor \phi_2
\]
provided that
\begin{enumerate}
  \item \( \Box \in \{\Box, \Diamond\} \),
  \item \( \phi_1 \) must contain \( x \),
  \item if \( \phi_2 \) contains \( x \) then \( x \) must occur under a modal operator, and
  \item \( W_j \) is a fresh \( W \)-symbol, and \( \text{Corr}(W_j) = \{W_j\} \).
\end{enumerate}
\( \phi_2 \) may be empty.

Fig. 1: The preprocessing rules for UI_{K_n} calculus for the modal logic \( K_n \). The rules are replacement rules: each rule replaces the premise with an equisatisfiable formula. In each rule, \( x \) is assumed to be the maximal symbol specified by the given ordering \( \succ \) on the symbols outside \( \Sigma \) occurring in the premises.
Literal Resolution (Res):
\[ W_i \Rightarrow \psi_1 \land x \quad W_i \Rightarrow \psi_2 \land \neg x \]
\[ W_i \Rightarrow \psi_1 \land \psi_2 \]
\( \psi_1 \) and/or \( \psi_2 \) may be empty.

World Resolution (Res W):
\[ W_i \Rightarrow \psi \quad W_j \Rightarrow W_i \]
\[ W_j \Rightarrow \psi \]
provided that \( \psi \) contains \( x \).
\( \psi \) may be a \( W \)-symbol.

□□ Resolution (Res □□):
\[ W_i \Rightarrow \psi_1 \land □ a W_n \quad W_i \Rightarrow \psi_2 \land \Diamond a W_m \]
\[ W_i \Rightarrow \psi_1 \land \psi_2 \land □ a W_j, \quad W_j \Rightarrow W_n, \quad W_j \Rightarrow W_m \]
provided that:
(i) \( \circ \in \{□, \Diamond\} \),
(ii) \( \text{Corr}(W_n) \cap \text{Corr}(W_m) \) is empty,
(iii) if there is a \( W_k \) such that \( \text{Corr}(W_k) = \text{Corr}(W_n) \cup \text{Corr}(W_m) \) then
\( W_j = W_k \), otherwise \( W_j \) is a fresh \( W \)-symbol, and \( \text{Corr}(W_j) = \text{Corr}(W_n) \cup \text{Corr}(W_m) \).
\( \psi_1 \) and/or \( \psi_2 \) may be empty.

Fig. 2: The resolution rules of the \( UIK_n \) calculus for the modal logic \( K_n \). The rules given here are saturation rules: they add conclusions to the current working clause set. We use \( x \) to refer to the maximal symbol in the working set specified by a given ordering \( \succ \) on the symbols outside \( \Sigma \).

Positive Purification (+Pur):
\[ N, W_i \Rightarrow \psi \land x \]
\[ N, W_i \Rightarrow \psi \land \top \]
provided that no more non-purification inference rules can be applied. \( \psi \) may be empty.

Negative Purification (-Pur):
\[ N, W_i \Rightarrow \psi \land \neg x \]
\[ N, W_i \Rightarrow \psi \land \top \]
provided that no more non-purification inference rules can be applied. \( \psi \) may be empty.

World Elimination (Elm W):
\[ N, W_i \Rightarrow \psi_1, \ldots, W_i \Rightarrow \psi_n \]
\[ N^{W_i}_{(\psi_1 \land \ldots \land \psi_n)} \]
provided that \( i \neq 0, \psi_1, \ldots, \psi_n \) do not contain \( x \) or any \( W \)-symbol, and \( N \) only contains \( W_i \) on the right hand side of \( \Rightarrow \) clauses. The expression \( N^\phi \) denotes the set of clauses that is obtained by replacing each occurrence of \( \phi \) in \( N \) by \( \psi \).

Fig. 3: The purification and elimination rules of the \( UIK_n \) calculus for modal logic \( K_n \). The rules are replacement rules: each rule replaces the premise with an equisatisfiable formula. In each rule, \( x \) is assumed to be the maximal symbol specified by the given ordering \( \succ \) on the symbols outside \( \Sigma \) occurring in the premises.
hand side occurrences of the $W$-symbol with the conjunction of these formulas, effectively eliminating the $W$-symbol from the set of clauses.

### 4.3 Examples

In the following examples, we demonstrate how the $UI_{K_n}$ system is used to compute a uniform interpolant with respect to $\Sigma = \{p, q\}$. Starting from $i = 0$, we use $N_i$ to refer to the clause set that is obtained after applying the $i$th step in the derivation.

**Example 1.** Consider a formula $\phi = (\neg p \lor \Diamond x) \land (\neg x \lor \Box q)$.

The input to the system is the set $N_0 = \{W_0 \Rightarrow (\neg p \lor \Diamond x) \land (\neg x \lor \Box q)\}$. The only rule applicable to $N_0$ is the clausification rule which gives

$$N_1 = \{W_0 \Rightarrow \neg p \lor \Diamond x, W_0 \Rightarrow \neg x \lor \Box q\}.$$ 

Now we apply the world introduction rule to get

$$N_2 = \{W_0 \Rightarrow \neg p \lor \Diamond W_1, W_1 \Rightarrow x, W_0 \Rightarrow \neg x \lor \Box q\}.$$ 

The only applicable rules are the positive and negative purification rules. We achieve

$$N_3 = \{W_0 \Rightarrow \neg p \lor \Diamond W_1, W_1 \Rightarrow \top, W_0 \Rightarrow \top \lor \Box q\}.$$ 

Eliminating $W_1$, we obtain

$$N_4 = \{W_0 \Rightarrow \neg p \lor \top, W_0 \Rightarrow \top \lor \Box q\}.$$ 

The $\Sigma$-uniform interpolant is $\phi' = (\neg p \lor \top) \land (\top \lor \Box q)$.

Notice that this example illustrates the local flavour of the system. We see that the occurrences of $x$ at two different modal levels do not interact via any resolution rule.

**Example 2.** Consider a formula $\phi = (\neg p \lor \Diamond x) \land (\Box (\neg x \lor \Box q))$. We start with the set $N_0 = \{W_0 \Rightarrow (\neg p \lor \Diamond x) \land (\Box (\neg x \lor \Box q))\}$. Applying clasufication to $N_0$ we get

$$N_1 = \{W_0 \Rightarrow \neg p \lor \Diamond x, W_0 \Rightarrow \Box (\neg x \lor \Box q)\}.$$ 

By applying the world introduction rule twice, we have

$$N_3 = \{W_0 \Rightarrow \neg p \lor \Diamond W_1, W_1 \Rightarrow x, W_0 \Rightarrow \Box W_2, W_2 \Rightarrow \neg x \lor \Box q\}.$$ 

The only applicable rule is the $\Box \Diamond$ rule, and it yields

$$N_4 = N_3 \cup \{W_0 \Rightarrow \neg p \lor \Diamond W_3, W_3 \Rightarrow W_1, W_3 \Rightarrow W_2\}.$$ 

By applying the world resolution rule twice, we obtain

$$N_6 = N_4 \cup \{W_3 \Rightarrow x, W_3 \Rightarrow \neg x \lor \Box q\}.$$
Now, we can apply the literal resolution rule which yields

\[ N_7 = N_6 \cup \{ W_3 \Rightarrow \Box \neg q \} \]

We apply the positive and negative purification rules (4 applications) and achieve

\[ N_{11} = \{ W_0 \Rightarrow \neg p \lor \Diamond W_1, \quad W_1 \Rightarrow \top, \quad W_0 \Rightarrow \Box W_2, \]
\[ W_2 \Rightarrow \top \lor \Box q, \quad W_0 \Rightarrow \neg p \lor \Diamond W_3, \quad W_3 \Rightarrow W_1, \]
\[ W_3 \Rightarrow W_2, \quad W_3 \Rightarrow \top, \quad W_3 \Rightarrow \top \lor \Box q, \]
\[ W_3 \Rightarrow \Box q \}. \]

Now, \( x \) does not appear anywhere. We eliminate the world variables \( W_1, W_2, W_3 \) via the world elimination rule.

To eliminate \( W_1 \), we look for clauses labelled with \( W_1 \), in this case we only have \( W_1 \Rightarrow \top \). We remove \( W_1 \Rightarrow \top \) and replace each occurrence of \( W_1 \) on the right hand side of \( \Rightarrow \) with \( \top \) as follows:

\[ N_{12} = \{ W_0 \Rightarrow \neg p \lor \Diamond \top, \quad W_0 \Rightarrow \Box W_2, \quad W_2 \Rightarrow \top \lor \Box q, \]
\[ W_0 \Rightarrow \neg p \lor \Diamond W_3, \quad W_3 \Rightarrow \top, \quad W_3 \Rightarrow W_2, \]
\[ W_3 \Rightarrow \top \lor \Box q, \quad W_3 \Rightarrow \Box q \}. \]

Similarly for \( W_2 \), we remove \( W_2 \Rightarrow \top \lor \Box q \), and replace the other occurrences of \( W_2 \) with \( \top \lor \Box q \).

\[ N_{13} = \{ W_0 \Rightarrow \neg p \lor \Diamond \top, \quad W_0 \Rightarrow \Box (\top \lor \Box q), \quad W_0 \Rightarrow \neg p \lor \Diamond W_3, \]
\[ W_3 \Rightarrow \top, \quad W_3 \Rightarrow \top \lor \Box q, \quad W_3 \Rightarrow \Box q \}. \]

Finally, we eliminate \( W_3 \),

\[ N_{14} = \{ W_0 \Rightarrow \neg p \lor \Diamond \top, \quad W_0 \Rightarrow \Box (\top \lor \Box q), \]
\[ W_0 \Rightarrow \neg p \lor \Diamond (\top \land (\top \lor \Box q) \land \Box q) \}. \]

The uniform interpolant is

\[ \phi' = (\neg p \lor \Diamond \top) \land (\Box (\top \lor \Box q)) \land (\neg p \lor \Diamond (\top \land (\top \lor \Box q) \land \Box q)) \]

which is equivalent to \( \phi' = (\neg p \lor \Diamond \neg q) \) by standard simplifications.

### 4.4 Correctness

The output \( \phi' \) is correct if it is a uniform interpolant of a formula \( \phi \) and a signature \( \Sigma \), produced in a finite number of steps. There are three issues at hand: termination, soundness and completeness. We state the theorems and lemmas that are relevant to these topics. For the proofs, we refer the reader to the full version of the paper\(^1\).

First are lemmas which are relevant to termination. We prove termination by showing that any derivation uses a finite number of symbols, and we argue that because of this, the calculus will stop generating new clauses.

**Lemma 1.** For a given formula \( \phi \) and a signature \( \Sigma \), the UIK\(_n\) calculus introduces a finite number of \( W \)-symbols.

\(^1\) [https://personalpages.manchester.ac.uk/staff/ruba.alassaf/publications.html](https://personalpages.manchester.ac.uk/staff/ruba.alassaf/publications.html)
Lemma 2. For a given formula \( \phi \) and a signature \( \Sigma \), the \( UI_{K_n} \) calculus will stop generating new clauses.

Lemma 3. For a given formula \( \phi \) and a signature \( \Sigma \), the \( UI_{K_n} \) system will not reintroduce a \( W \)-symbol that was eliminated before.

From Lemma 1, 2 and 3, we conclude the following theorem.

Theorem 1 (Termination). Given a formula \( \phi \) and a signature \( \Sigma \), the uniform interpolation system \( UI_{K_n} \) computes a formula \( \phi' \) in a finite number of steps.

The following lemma addresses the space complexity of our system.

Lemma 4. The space complexity of the \( UI_{K_n} \) calculus is double exponentially bounded in the length of the input.

The idea of the proof is to show each clause is exponentially bounded in the length of the input \( n \), and that the number of clauses produced by the system is double exponentially bounded by \( n \).

The next lemmas argue that the signature of \( \phi' \) is \( \Sigma \).

Lemma 5. The \( UI_{K_n} \) system will always be able to eliminate every \( W \)-symbol that is not \( W_0 \), using the world elimination rule.

Lemma 6. The \( UI_{K_n} \) system will always be able to eliminate symbols in the signature of \( \phi \) that are not in \( \Sigma \).

Next, we state the soundness theorem.

Theorem 2 (Soundness). Given a formula \( \phi \) and a signature \( \Sigma \), the uniform interpolation system \( UI_{K_n} \) computes a formula \( \phi' \) such that for any formula \( \psi \) over \( \Sigma \), we have that

\[
\text{if } \models \phi' \rightarrow \psi \text{ then } \models \phi \rightarrow \psi.
\]

For our completeness proof, we are interested in understanding models that are invariant up to the satisfaction of \( \Sigma \)-modal formulas. \( \Sigma \)-modal formulas are modal formulas described using a signature of propositional symbols \( \Sigma \). For this purpose, we use the following notion.

Definition 3 (\( \Sigma \)-bisimulation). Let \( (M, w) \) and \( (M', w') \) be two Kripke models where \( M = (W, R, V) \) and \( M' = (W', R', V') \). A \( \Sigma \)-bisimulation between \( M \) and \( M' \) is a relation \( \rho \subseteq W \times W' \) such that \( w \rho w' \), and whenever \( u \rho u' \), the following holds:

- \( \text{atoms} \) \( u \) and \( u' \) satisfy the same propositional symbols from \( \Sigma \);
- \( \text{forth} \) For all \( a \), if \( u R_a t \), then there is a \( t' \) such that \( u' R'_a t' \) and \( t pt' \);
- \( \text{back} \) For all \( a \), if \( u' R'_a t' \), then there is a \( t \) such that \( u R_a t \) and \( t pt' \).
The following is our completeness theorem.

**Theorem 3 (Completeness).** Given a formula $\phi$ and a signature $\Sigma$, the uniform interpolation system $UI_{K_n}$ computes a formula $\phi'$ such that, for any formula $\psi$ over $\Sigma$, we have that

$$\text{if } \models \phi \rightarrow \psi \text{ then } \models \phi' \rightarrow \psi.$$ 

Using proof by contradiction, we assume that $\models \phi \rightarrow \psi$ but $\not\models \phi' \rightarrow \psi$. The assumption implies that there exists a counter-model $\mathcal{M}'$ and a world $w_0$ such that, $\mathcal{M}', w_0 \models \phi'$ and $\mathcal{M}', w_0 \not\models \psi$. We use $\Sigma$-bisimulation to prove by induction that this is not possible.

5 Conclusion

The paper presented a resolution-based method to compute uniform interpolants for the multi-agent modal logic $K_n$. It has been shown that our method terminates, and is sound and complete. The space complexity was proven to be at most double exponential in the length of the input. This work is intended to be the basis of our future work. We would like to study logics which are known to have the uniform interpolation property, and show that the presented system can be extended to solve the uniform interpolation problem for more modal logics. An implementation is being developed to demonstrate practicality.

References


Second-Order Specifications and Quantifier Elimination for Consistent Query Answering in Databases

(Abstract)

Leopoldo Bertossi

Universidad Adolfo Ibáñez
Faculty of Engineering and Sciences
and
Millennium Institute for Foundational Research on Data (IMFD)
Santiago, Chile
leopoldo.bertossi@uai.cl

Abstract. Consistent answers to a query from a possibly inconsistent database are answers that are simultaneously retrieved from every possible repair of the database. Repairs are consistent instances that minimally differ from the original inconsistent instance. It has been shown before that database repairs can be specified as the stable models of a disjunctive logic program. We show how to use the repair programs to transform the problem of consistent query answering into a problem of reasoning w.r.t. a theory written in second-order predicate logic. We show how a first-order theory can be obtained instead by applying second-order quantifier elimination techniques.

1 Introduction

Integrity constraints (ICs) on databases are expected to be satisfied by the instances of the given schema \( S \). If an instance does not satisfy the ICs, it is said to be inconsistent, and becomes only partially semantically correct. Consistent query answering (CQA) attempts to characterize and compute answers to a query that are consistent with respect to (w.r.t.) a given set of ICs \([5, 8, 15, 11]\). Informally, a tuple of constants \( \bar{t} \) is a consistent answer from an instance \( D \) to a query \( Q(\bar{x}) \) w.r.t. a set of ICs \( IC \) if \( \bar{t} \) can be obtained as a usual answer to \( Q \) from every repair of \( D \). Here, a repair is a consistent instance for the schema \( S \) that differs from \( D \) by a minimal set of database atoms under set inclusion [5].

It has been shown \([6, 13]\) that repairs can be specified as the stable models of a disjunctive logic program \([22]\) \( II \), by a so-called repair program. In this way, CQA becomes a problem of reasoning with program \( II \). Logic programs with stable model semantics are also called answer-set programs \([22]\), and their stable models are also called answer sets. Answer-set programming has become a powerful paradigm and tool for the specification and solution of hard combinatorial problems \([12]\).
Ideally, consistent answers to a query $Q$ from a database instance $D$ should be obtained by posing a new query $Q'$ to $D$, as an ordinary query that is, hopefully, easy to evaluate against $D$. This is the case, for example, when $Q'$ is a query expressed in the first-order (FO) language $L(S)$. Some classes of queries and ICs with this property have been already identified [5, 14, 21]; and many more by Wijsen in a series of papers on conjunctive queries and key constraints [1]. C.f. [36] and [37] for excellent surveys, and [23] for more recent results and references.

The main result for CQA for conjunctive queries (CQs) under key constraints (KCs), tells us that one can syntactically classify and decide CQs in terms of their data complexity for CQA.\footnote{As usual in databases, all the complexity results in this paper are about data complexity, i.e. in terms of the size of the database instance.} A trichotomy appears: a CQ can be FO-rewritable, or in $\text{PTIME}$ ($L$-complete), or $\text{coNP}$-complete. There are queries for these three classes. For the first class, the rewriting can be computed, in which case, it is possible to compute the consistent answers in polynomial time. It is worth emphasizing that there are CQs for which CQA can be done in polynomial time, but provably not via FO-rewriting [35, 34]. This opens the question about the right logical language for a rewriting, if any.

At the other extreme, repair programs provide a general mechanism for computing consistent answers. Actually, the data complexity of CQA can be as high as the data complexity of cautious query evaluation from disjunctive logic programs under the stable model semantics, namely $\Pi^P_2$-complete [16, 14]. Apart from providing the right expressive power and complexity for dealing with repairs and CQA, the semantics of answer-set programming is a non-monotonic, non-classical logical semantics, which is particularly suitable for applications in databases, through the implicit use of the closed-world assumption [32], and the minimality of models under set inclusion. This last feature is useful in relation to the minimality of database repairs.

In those cases where a FO rewriting for CQA is possible, one can transform the problem of CQA into one of reasoning in classical predicate logic, because the original database can be “logically reconstructed” as a FO theory [32]. In this work we investigate how repair programs can be used to generate a theory written in classical logic from which CQA can be captured as logical entailment. This theory can be written in second-order or first-order predicate logic. We start by trying to achieve the former, by providing specifications of database repairs in second-order (SO) predicate logic. They are obtained by applying recent results on the specification in SO logic of the stable models of a logic program [19, 20] -in our case, a repair program- and older results on their characterization as the models of a circumscripive theory [29] for the case of disjunctive stratified programs [30, 31]. This circumscription can be specified in SO predicate logic [25, 33].

In order to achieve a FO specification, for some cases related to queries and KCs, we apply techniques for SO quantifier elimination that have been introduced in [17]. In this way it is possible to obtain a FO specification of the database repairs. This transforms CQA into a problem of logical reasoning in FO logic. We illustrate by means of an example how to obtain a FO rewriting for CQA under a KC. We illustrate the SO quantifier elimination technique. Generalizing the methodology to more general cases is left for future investigation. C.f. Section 5), where we also discuss the possibility
of obtaining rewritings in fixed-point logic, when it is provably the case that no FO rewriting exists. This paper is an excerpt from [9]. In [10] one can find an extended and updated version of both the latter and this paper, containing all the details and much more.

2 Database Repairs and Repair Programs

Consider a database instance $D$ and a set of integrity constraints (ICs), that is a set $\Sigma$ of sentences in the first-order language of predicate logic associated to the database schema. The database may not satisfy $\Sigma$ in which case we say that $D$ is inconsistent. The database can be repaired by inserting or deleting full tuples into/from $D$, in such a way that the resulting instance becomes consistent. A (minimal) repair of $D$ is an instance $D'$ that satisfies $\Sigma$ and minimally differs from $D$ under set inclusion, i.e. $D \Delta D'$, the symmetric set difference, is minimal under set inclusion [5, 11]. For monotone ICs (they are never violated by tuple deletions), like the ones we will consider below, the repairs are always maximal-subsets (subinstances) of $D$. The repairs of an inconsistent database can be specified by means of answer-set programs (c.f. [11] for details and references). Those are the repair programs.

Repair programs use annotation constants in an extra argument for each of the database predicates. More precisely, for each $n$-ary $P \in S$, we make a copy $P'$, which is $(n+1)$-ary [13]. Here, we need only the following annotations, with its intended semantics: (a) $f$ in atoms $P(\bar{a}, f)$, meaning “made false (deleted)”, (b) $t^{**}$ in atoms $P(\bar{a}, t^{**})$, meaning “true in repair”.

Example 1. The relational schema $\mathcal{S}$ contains predicate $P(X,Y)$, and the functional dependency (FD) $X \to Y$, actually a KC, stating that the first attribute functionally determines the second. It can be expressed as the first-order (FO) sentence

$$FD: \forall x \forall y \forall z (P(x, y) \land P(x, z) \to y = z).$$

The database instance $D = \{P(a, b), P(a, c), P(d, e)\}$ is inconsistent since the first two tuples jointly violate the FD. We have two repairs: $D_1 = \{P(a, b), P(d, e)\}$ and $D_2 = \{P(a, c), P(d, e)\}$. The query $Q_1(y): \exists x P(x, y)$ has the consistent answer $(e)$, whereas the query $Q_2(x): \exists y P(x, y)$ has $(a), (d)$ as consistent answers. They are standard answers from both repairs. These repairs can be specified as the stable models of the following repair program $\Pi(D, FD)$:

1. Original database facts: $P(a, b), P(a, c), P(d, e)$.
2. The repair rule: $P_\bot(x, y, f) \lor P_\bot(x, z, f) \leftrightarrow P(x, y), P(x, z), y \neq z$.

It specifies that whenever the FD is violated, as captured by the rule body (the RHS), then one (and only one if possible) of the two tuples involved in the violation has to be made false (deleted), as captured by the disjunctive rule head (LHS).

3. Annotations constant $t^{**}$ is used to read off the atoms in a repair, saying that whichever atom was in the original instance and not deleted stays in the repair:

$$P(\bar{x}, t^{**}) \leftarrow P(\bar{x}), \text{ not } P(\bar{x}, f).$$

For simplicity, and from now on, we use new predicates $P_f(\cdot, \cdot)$ for $P_\bot(\cdot, \cdot, f)$, $P_{**}(\cdot, \cdot)$ for $P(\cdot, \cdot, t^{**})$. The repairs are in one-to-one correspondence with the restric-
tion of the stable models to the predicates of the form $P_{**}$ [13]. In this example, they are: $D_1 = \{P_{**}(a, b), P_{**}(d, e)\}$ and $D_2 = \{P_{**}(a, c), P_{**}(d, e)\}$.

In order to obtain the consistent answers to a FO query $Q$, a query program $\Pi^Q$, containing a query-answer predicate $Ans^Q$, is combined with the repair program $\Pi(D, FD)$. Next, as is common with ASPs, we can use the cautious entailment semantics from ASP, denoted $|=_{cs}$, which means that the right-hand side is true in all the stable models of the program on the left-hand side.

The extension of the answer predicate $Ans^Q$ in the intersection of all stable models of $\Pi := \Pi(D, FD) \cup \Pi^Q$ contains exactly the consistent answers. That is, $\bar{a}$ is a consistent answer to $Q$, denoted $D \models Q(\bar{a})$, iff $\Pi(D, FD) \cup \Pi^Q \models_{cs} Ans^Q(\bar{a})$. In general, $\Pi^Q$ will be a (stratified) non-recursive and normal Datalog query $\Pi^Q$ with answer predicate $Ans^Q(\bar{x})$ appearing only in rule heads [1, 27].

**Example 2.** (ex. 1 cont.) A possible query is $Q(x, y) : P(x, y)$, which can be represented by the simple query program $\Pi^Q : Ans(x, y) \leftarrow P_{**}(x, y)$. This program is combined with 1.-3. above, and the consistent answers to $Q$ are those tuples $\bar{a}$, such that $\Pi^Q \cup \Pi(D, FD) \models_{cs} Ans(\bar{a})$, obtaining the only consistent answer is $(d, e)$. ■

### 3 Second-Order Specification of Repairs

In [19, 20], the stable model semantics of logic programs introduced in [22] is reobtained via an explicit specification in classical SO predicate logic that is based on circumscription. First, the program $\Pi$ is transformed into (or seen as) a FO sentence $\psi(\Pi)$. Next, the latter is transformed into a SO sentence $\Phi(\Pi)$. Here, $\psi(\Pi)$ is obtained from $\Pi$ as follows: (a) Replace every comma by $\land$, and every not by $\neg$. (b) Turn every rule $Head \leftarrow Body$ into the formula $Body \rightarrow Head$. (c) Form the conjunction of the universal closures of those formulas.

Now, given a FO sentence $\psi$ (e.g. the $\psi(\Pi)$ above), a SO sentence $\Phi$ is defined as $\psi \land \neg\exists\bar{X}((\bar{X} < \bar{P}) \land \psi^o(\bar{X}))$, where $\bar{P}$ is the list of all predicates $P_1, ..., P_n$ in $\psi$ that are going to be circumscribed, and $\bar{X}$ is a list of distinct predicate variables $X^{P_1}, ..., X^{P_n}$, with $P_i$ and $X^{P_i}$ of the same arity. Here, $(\bar{X} < \bar{P})$ means $(\bar{X} \leq \bar{P}) \land (\bar{X} \neq \bar{P})$, i.e. $\bigwedge^n_i \forall\bar{x}(X^{P_i}(\bar{x}) \rightarrow P_i(\bar{x})) \land \bigvee^n_i (X^{P_i} \neq P_i)$. $X^{P_i} \neq P_i$ stands for $\exists\bar{x}_i(P_i(\bar{x}_i) \land \neg X^{P_i}(\bar{x}_i))$.

$\psi^o(\bar{X})$ is defined recursively as follows: (a) $P_1(t_1, ..., t_m)^o := X^{P_1}(t_1, ..., t_m)$. (b) $(t_1 = t_2)^o := (t_1 = t_2)$. (c) $\bot^o := \bot$. (d) $(F \land G)^o := (F^o \land G^o)$ for $\odot \in \{\land, \lor\}$. (e) $(F \lor G)^o := (F^o \lor G^o)$ for $\odot \in \{\land, \lor\}$. (f) $(Q \land F)^o := QxF^o$ for $Q \in \{\land, \lor, \forall, \exists\}$. Notice that we assume there is no explicit logical negation in formulas. Instead, a formula of the form $\neg \chi$ is assumed to be represented as $(\chi \rightarrow \bot)$, with $\bot$ standing for an always false propositional formula.

The Herbrand models of the SO sentence $\Phi(\Pi)$ associated to $\psi(\Pi)$ correspond to the stable models of the original program $\Pi$ [19]. We can see that $\Phi(\Pi)$ is similar to a parallel circumscription of the predicates in program $\Pi$ w.r.t. the FO sentence $\psi(\Pi)$

---

2 In circumscription, some predicate may be minimized, others may stay flexible (or variable) to accommodate to the minimization of others, and some may stay fixed [25].
associated to $\Pi \ [29, 26]$. In principle, the transformation rule (e) above could make formula $\Phi(\Pi)$ differ from a circumscription.

Now, let $D$ be a relational database, $\Pi^r$ the repair program without the database facts. From now on $\Pi = D \cup \Pi^r \cup \Pi^Q$. $\Pi^r$ depends only on the integrity constraints, and includes definitions for the annotation predicates. The only predicates shared by $\Pi^r$ and $\Pi^Q$ are of the form $P_{**}$, which appear only in the rule bodies of $\Pi^Q$. These predicates produce a splitting of the combined program [28], which allows us to analyze separately $\Pi^r$ and $\Pi^Q$. The latter can be translated into classical logic by predicate completion, or a prioritized circumscription [30]. If the query is FO, we can use query itself.

Example 3. (ex. 2 cont.) Leaving aside many details that can be found in [10], we obtain the following SO formula $\Phi(\Pi)$ that captures the stable models of the original program:

\[
\forall xy(P(x, y) \equiv (x = a \land y = b) \lor (x = a \land y = c) \lor (x = d \land y = e)) \land \tag{1}
\]

\[
\forall xy(P_{**}(x, y) \equiv \text{Ans}(x, y)) \land \tag{2}
\]

\[
\forall xy((P(x, y) \land \neg P_f(x, y)) \equiv P_{**}(x, y)) \land \tag{3}
\]

\[
\forall xyz((P(x, y) \land P(x, z) \land y \neq z \rightarrow (P_f(x, y) \lor P_f(x, z))) \land \tag{4}
\]

\[
\neg \exists U_f((U_f < P_f) \land \forall xyz(P(x, y) \land P(x, z) \land y \neq z \rightarrow (U_f(x, y) \lor U_f(x, z)))) \lor \forall U_f((U_f(x, y) \land \exists xy(P_f(x, y) \land \neg U_f(x, y))). \tag{5}
\]

Here, $U_f < P_f$ stands for the formula $\forall xy(U_f(x, y) \rightarrow P_f(x, y)) \land \exists xy(P_f(x, y) \land \neg U_f(x, y))$. In this sentence, the minimizations of the predicates $P, P_{**}$ and $\text{Ans}$ are expressed as their predicate completion. Predicate $P_f$ is minimized via (5).

We obtain the SO sentence for program $\Pi$ as a parallel circumscription of the predicates in the repair program seen as a FO sentence. The circumscription actually becomes a prioritized circumscription [25] given the stratified nature of the repair program: first the database predicate is minimized, next $P_f$, next $P_{**}$, and finally $\text{Ans}$. $\blacksquare$

Generalizations of the result in the previous example can be found in [10]. In the following we concentrate on the problem of possibly turning this SO reasoning problem into one at the FO level.

4 From Second-Order to First-Order CQA

In this section we discuss the possibility of obtaining a FO rewriting of the original query as posed to the repair program. We do this through the analysis of the SO sentence obtained in Example 3, concentrating on the SO sentence (5). In the rest of this section many details are missing. They can be all found in [10], the extended version of this work.

Sentence (5) can be expressed as

\[
\neg \exists U_f((U_f < P_f) \land \forall xyz(\kappa(x, y, z) \rightarrow (U_f(x, y) \lor U_f(x, z))), \tag{6}
\]

where $\kappa(x, y, z)$ is the formula $P(x, y) \land P(x, z) \land y \neq z$. For simplicity, we use $U$ instead of $U_f$. We will apply to (6) the SO quantifier elimination techniques in [17]. The negation of (6) turns out to be -after several steps [10]- logically equivalent to:

\[
\exists st \exists U (\forall xyz(\neg \kappa(x, y, z) \lor U(x, y) \lor U(x, z)) \land \tag{7}
\]

\[
\forall uv(\neg U(u, v) \lor P_f(u, v)) \land (P_f(s, t) \land \neg U(s, t))).
\]
The first conjunct in (7), with \( w = \vee(y, z) \) standing for \( (w = y \lor w = z) \), can be equivalently written as

\[
\exists f \forall r (\forall x_1 y_1 z_1 (\neg \kappa(x_1, y_1, z_1) \lor f(x_1, y_1, z_1) = \vee(y_1, z_1)) \land \\
\forall x_1 y_2 z_2 (\neg \kappa(x_1, y_2, z_2) \lor r \neq f(x_1, y_2) \lor U(x, r))),
\]

where \( \exists f \) is a quantification over functions. Formula (7) becomes:

\[
\exists st \exists f \exists U \forall x \forall y (\forall x_1 y_1 z_1 (\neg \kappa(x_1, y_1, z_1) \lor f(x_1, y_1, z_1) = \vee(y_1, z_1)) \land \\
\forall yz (\neg \kappa(x, y, z) \lor r \neq f(x, y, z) \lor U(x, r)) \land \\
\forall uv (\neg U(u, v) \lor P_f(u, v)) \land (P_f(s, t) \land \neg U(s, t))).
\]

We are ready to apply Ackermann’s Lemma [2, 3], with the last formula written as:

\[
\exists st \exists f \exists U \forall x \forall r ((A(x, r) \lor U(x, r)) \land B(U, \overline{U}(A(x, r))), \tag{8}
\]

where \( B(U, \overline{U}(A(x, r))) \) is formula \( B \) with predicate \( U \) replaced by \( \overline{U} \); and formulas \( A, B \) are:

\( A(x, r) : \forall yz (\forall yz (\neg \kappa(x, y, z) \lor r \neq f(x, y, z))) \); and \( B(U) : \forall x_1 y_1 z_1 (\neg \kappa(x_1, y_1, z_1) \lor f(x_1, y_1, z_1) = \vee(y_1, z_1)) \land \\
\forall uv (\neg U(u, v) \lor P_f(u, v)) \land (P_f(s, t) \land \neg U(s, t)) \).

Formula \( B \) is positive in \( U \), then the whole subformula in (8) starting with \( \exists U \) can be equivalently replaced by \( B(U, \overline{U}(A(x, r))) \) [17, lemma 1], getting rid of the SO variable \( U \), and obtaining:

\[
\exists st \exists f \forall x \forall y (\neg \kappa(x, y, z) \lor f(x, y, z) = \vee(y, z)) \land (\neg \kappa(x, y, z) \lor P_f(u, f(x, y, z)) \land \\
(\neg \kappa(x, y, z) \lor P_f(s, t) \land (x \neq s \lor \neg \kappa(x, y, z) \lor t \neq f(x, y, z))).
\]

Unskolemizing, getting rid of function variable \( f \), we obtain

\[
\exists st \forall x \forall y \forall w (\neg \kappa(x, y, z) \lor w = \vee(y, z)) \land (\neg \kappa(x, y, z) \lor P_f(u, w)) \land \\
(\neg \kappa(x, y, z) \lor P_f(s, t) \land (x \neq s \lor \neg \kappa(x, y, z) \lor t \neq w)),
\]

which is equivalent to the negation of (6). Negating again, we obtain a formula equivalent to (6):

\[
\forall st (P_f(s, t) \rightarrow \exists x \forall y \forall z ((w \neq y \lor w \neq z) \lor \\
\neg P_f(x, w) \lor (x = s \land t = w))).
\]

The formula in the square bracket inside can be equivalently replaced by

\[
((w = y \lor w = z) \lor P_f(x, w)) \rightarrow (s = x \land t = w).
\]

So, we obtain

\[
\forall st (P_f(s, t) \rightarrow \exists x \forall y (\kappa(x, y, z) \land (P_f(x, y) \rightarrow s = x \land t = y) \land \\
(P_f(x, z) \rightarrow s = x \land t = z))).
\]

Due to the definition of \( \kappa(x, y, z) \), it must hold \( y \neq z \). In consequence, we obtain:

\[
\forall st (P_f(s, t) \rightarrow \exists z (\kappa(s, t, z) \land \neg P_f(s, z))).
\]

Summing up, the SO sentence for the repair program \( \Pi(D, IC) \) is logically equivalent to a FO sentence, \( \psi \), that is the conjunction of (1), (3), (4), and

\[
\forall st (P_f(s, t) \rightarrow \exists z (\kappa(s, t, z) \land \neg P_f(s, z))), \tag{9}
\]

which says, in particular, that whenever there is a conflict between two tuples, one of them must be deleted, and for every deleted tuple due to a violation, there must be a tuple with the same key value that has not been deleted. Thus, not all mutually conflicting tuples can be deleted.

Coming back to CQA, for consistent answers \( \tilde{t} \), we now have classical FO entailment:

\[
\psi \land \forall \tilde{x} (\text{Ans}^\varnothing(\tilde{x})) \equiv \chi(\tilde{x}) \models \text{Ans}^\varnothing(\tilde{t}), \tag{10}
\]

where \( \chi \) is the FO definition of \( \text{Ans}^\varnothing \) in terms of the \( P_\ast \) predicate. This is not FO query rewriting in the sense of obtaining a FO query to be posed to the original database.
However, and for example, it is not difficult to show [10] that for the query $Q : P(x, y)$, and any consistent answer $\langle t_1, t_2 \rangle$, this is equivalent to having:

$$D \models P(t_1, t_2) \land \neg \exists z (P(t_1, z) \land z \neq t_2).$$

(11)

The query rewriting on the RHS of in (11) is one of those obtained in [5] using a completely different and more general resolution-based rewriting methodology.

5 Towards Fixed-Point Logic

As described in Section 1, there are syntactic classes of CQs for which consistent query answering can be done in polynomial time in data complexity. For one class, this can be done via FO query rewriting. For a different class, its queries provably do not admit a first-order rewriting. Even more, one can decide if a CQ falls in this case or not [36, 37].

For example, the Boolean conjunctive query $Q : \exists x \exists y (R(x, y) \land S(y, x))$, with the first attributes of $R$ and $S$ as keys for them, is a query in the second class in that it can be consistently answered in polynomial time, but no FO rewriting for it exists. Results of this kind are established in [35, 34] by means of the notions of Hanf-locality and Ehrenfeucht-Fraïssé games for FO-logic [24].

This opens the ground for investigating two problems:

1. Apply the second-order quantifier elimination technique in [17], that we applied in this work, with the purpose of recovering the FO rewritings for the whole class of queries that admit FO consistent rewritings (as determined by Wijsen [35]).
2. Identify and obtain logical languages that can be used for rewriting the queries in the second class, in such a way that query answering for the rewritten query can be done in polynomial time.

For the second problem, it would be interesting to see if second-order quantifier elimination could be applied to second-order specification of Section 3, in such a way that the resulting query is expressed, not in FO logic, but in fixed-point logic, which would lead to a polynomial-time answer [24]. Actually, in [18], the authors have been able to eliminate second-order quantifiers, obtaining fixed-point formulas. It is worth investigating if this is a way to obtain polynomial-time, logical, but non-FO, rewritings for CQA. This undertaking is not a priori impossible. The existence of non-FO rewritable but PTIME-complete queries (in data) already identified [35, 23] is in principle compatible with the PTIME-completeness of fixed-point logic (in data) [16].

Acknowledgements: Useful comments from anonymous reviewers for a previous and the submitted version of this paper are much appreciated. Leopoldo Bertossi has been partially funded by the ANID - Millennium Science Initiative Program - Code ICN17-002.

References


On Testing Containedness between Geometric Graph Classes using Second-order Quantifier Elimination and Hierarchical Reasoning
(Short Paper)

Lucas Böltz, Hannes Frey, Dennis Peuter, and Viorica Sofronie-Stokkermans

Computer Science Department, University of Koblenz-Landau, Germany
{boeltz,frey,dpeuter,sofronie}@uni-koblenz.de

Abstract. We consider geometric graphs in terms of graph classes which are axiomatically described by logical formulae. Our general method to prove graph properties is to show that one graph class is contained in another one, thus, the subclass inherits all properties already known for the superclass. The methods we use for automatically proving such subclass relations are based on second-order quantifier elimination and hierarchical reasoning. As a proof of concept we apply our method to show in how far given specific geometric graph classes with vertices on a plane are planar drawings. Specifically the graph classes we consider here have been used in the context of algorithmic wireless network research.

1 Introduction

Classes of geometric graphs occurring in algorithmic wireless network research can be described using axioms over suitable theories. Such axioms typically refer to points in $\mathbb{R}^2$ (and possibly also their coordinates) and often describe geometrical conditions for the existence resp. non-existence of edges.

In this paper we devise methods for checking containedness between graph classes (our focus here are graph classes that can be described with universally quantified axioms). If containedness cannot be proved, the methods we use allow us to generate counterexamples.

Checking containedness is in so far of interest, as it provides a general tool to check graph properties resulting from distributed graph algorithms. For example, showing containedness of the class of graphs resulting from such an algorithm with respect to the class of planar graphs implies that the algorithm produces planar graphs as well. Though in this paper we focus on containedness and planarity as a proof of concept, by bridging two very different disciplines, this work is intended to provide a general future pioneering direction for the design of local distributed algorithms in the context of wireless networks and in graphs in general (such algorithms are used, for example, to route messages [12] or
control the network topology [13]). The approach we propose (and the tools we use for this) can be a good instrument in theoretical graph and network theory, which would allow the user to test whether certain generalizations of concept are problematic and to locate possible problems with the general formalizations. A typical example we consider is the formalization of a distance between two points: We can assume that graphs have vertices which are points in $\mathbb{R}^2$ and the distance between points is the Euclidean distance, but we might also use abstract metric spaces or cost functions for the edges.

Testing containedness of graph classes has been studied by the authors of this paper in recent work. In [5], formulae representing necessary and sufficient conditions for containedness between certain types of graph classes were derived by hand. In [11,10] possibilities for combining symbol elimination methods were proposed and applied to (parametric) entailment problems, and used e.g. for checking inclusion between graph classes. In this paper, we propose automated methods for checking inclusion without the manual component in [5]; for this we rely on methods introduced in [11]. We are not aware of other similar approaches to the area of computational (geometric) graph theory. Existing approaches use a logical representation of graphs based on monadic second order logic [3] or higher order theorem provers like Isabelle/HOL [1]. Our approach is orthogonal: it relies on methods for second-order quantifier elimination which allow a reduction of many problems to satisfiability modulo a suitable theory for which state of the art SMT solvers can be used; the procedure is sound and our approach allows us to identify situations in which completeness can be guaranteed.

Structure of the paper. In Sect. 2, we introduce concepts from geometric graph theory. In Sect. 3 we introduce the theoretical tools for our work. In Sect. 4, using concrete examples, we demonstrate the feasibility and strength of our approach for showing containedness and planarity of geometrically described graphs. In Sect. 5 we present conclusions and plans for future work.

2 Graph Classes Related to Planarity Conditions

In the following we consider graphs with vertices in a set $X$, which can be the set of points in the Euclidean space $\mathbb{R}^2$ (or, in some cases, the set of points in an abstract metric space $(X, d)$). We present a type of graph classes which can be proved to be plane drawings (i.e. planar graphs which are drawn on the plane in such a way that edges do not intersect). We model such graph classes by using a unary predicate $V$ and a binary predicate $E$, such that for every element $x \in X$, $V(x)$ is true if $x$ is a vertex of the graph and $E(x, y)$ is true if $x$, $y$ are vertices and there is an edge between $x$ and $y$. We consider undirected graphs without self-loops. Such properties can be described by the axioms $K_0$:

$$K_0 = \{ \forall x (\neg E(x, x)), \forall x, y E(x, y) \rightarrow E(y, x), \forall x, y E(x, y) \rightarrow V(x) \wedge V(y) \}.$$

We consider the following classes of graphs.

The class $P$ of plane drawings can be described using the axiom:

$$(P) \quad \forall u, v, w, x : E(w, x) \wedge \pi_P(u, v, w, x) \rightarrow \neg E(u, v)$$
where $\pi_P(u, v, w, x)$ is a predicate which is true iff $w$ and $x$ are in different half planes defined by the line $uv$ passing through $u$ and $v$, and $u$ and $v$ are in different half planes defined by the line $wx$ passing through $w$ and $x$.\footnote{For the existence of these lines it is necessary that $u \neq v$ and $w \neq x$. If, for instance, the vertex $w$ is located on the line through $u$ and $v$, but not equal to $u$ or $v$, it is located in both half planes; hence, in this case the segments $uv$ and $wx$ intersect.} We can express this either using analytic geometry or (if $d$ is a metric) by the following formula:

$$
\pi_P(u, v, w, x) := V(u) \land V(v) \land V(x) \land V(w) \land u \neq v \land w \neq x \land \exists m (d(u, m) + d(m, v) = d(u, v) \land d(w, m) + d(m, x) = d(w, x))
$$

**The class $G$ of Gabriel graphs** [2] is axiomatized by $K_0$ together with:

\[(G) \quad \forall u, v \ E(u, v) \leftrightarrow \pi_G(u, v)\]

where $\pi_G(u, v)$ expresses the fact that $u$ and $v$ are vertices and every vertex different from $u$ and $v$ lies outside of the minimal circle passing through $u$ and $v$. If $m(u, v)$ is the middle of the segment $uv$, we can express this by:

$$
\pi_G(u, v) := V(u) \land V(v) \land u \neq v \land \forall w \ (w \neq u \land w \neq v \land V(w) \rightarrow d(m(u, v), w) > d(m(u, v), u)).
$$

We can define superclasses $G^\rightarrow$ and $G^\leftarrow$ of $G$: $G^\rightarrow$ is defined by only keeping the "$\rightarrow$" implication in $(G)$, and $G^\leftarrow$ by only keeping the "$\leftarrow$" implication.

**The class $R$ of relative neighborhood graphs** [9] is described by $K_0$ and:

\[(R) \quad \forall u, v \ E(u, v) \leftrightarrow \pi_R(u, v)\]

where $\pi_R$ is defined by:

$$
\pi_R(u, v) := V(u) \land V(v) \land u \neq v \land \forall w \ (V(w) \rightarrow d(u, w) \geq d(u, v) \land d(w, v) \geq d(u, v)).
$$

We can define superclasses $R^\rightarrow$ and $R^\leftarrow$ of $R$: $R^\rightarrow$ is defined by only keeping the "$\rightarrow$" implication in $(R)$, and $R^\leftarrow$ by only keeping the "$\leftarrow$" implication.

We model such graph classes using a logical language and consider extensions of the theory of real numbers and of theories modeling the points with additional functions symbols ($V$, $E$, distances, etc.). In [11] we proved that the properties of theory extensions occurring in this context allow us to define sound, complete and terminating proof procedures. We present a summary of the theoretical results and then several examples.

### 3 Theories and Theory Extensions

We assume that basic notions in (many-sorted) first-order logic such as signature $\Pi$, $\Pi$-structure, theory, satisfiability, unsatisfiability and entailment, possibly w.r.t. a theory are known.

Let $\Pi_0 = (\Sigma_0, \text{Pred})$ be a signature, and $T_0$ be a "base" theory with signature $\Pi_0$. We consider extensions $T := T_0 \cup K$ of $T_0$ with new function symbols $\Sigma$ (extension functions) whose properties are axiomatized using a set $K$ of (universally closed) clauses in the extended signature $\Pi = (\Sigma_0 \cup \Sigma, \text{Pred})$, such that each clause in $K$ contains function symbols in $\Sigma$. Especially well-behaved are the $\Psi$-local theory
extensions, i.e. theory extensions $T_0 \subseteq T_0 \cup K$ in which for every finite set $G$ of ground $\Pi^C$-clauses (containing an additional set $C$ of constants) it holds that $T_0 \cup K \cup G \models \bot$ if and only if $T_0 \cup K[\Psi_K(G)] \cup G$ is unsatisfiable, where, for every set $G$ of ground $\Pi^C$-clauses, $K[\Psi_K(G)]$ is the set of instances of $K$ in which the terms starting with a function symbol in $\Sigma$ are in $\Psi_K(G) = \Psi(\text{est}(K,G))$, (where $\text{est}(K,G)$ is the set of ground terms starting with a function in $\Sigma$ occurring in $G$ or $K$). $\Psi$-local extensions can be recognized by showing that certain partial models embed into total ones [14,8].

In local theory extensions hierarchical reasoning is possible: Assume that $T_0 \subseteq T_1 = T_0 \cup K$ is a $\Psi$-local theory extension. We can reduce the problem of checking the satisfiability of any finite set $G$ of ground clauses w.r.t. $T_0 \cup K$ to a satisfiability test w.r.t. $T_0$ as follows: For any finite set $G$ of $\Pi^C$-ground clauses, $T_0 \cup K \cup G$ is satisfiable iff $T_0 \cup K[\Psi_K(G)] \cup G$ is satisfiable. Let $K_0 \cup G_0 \cup \text{Def}$ be obtained from $K[\Psi_K(G)] \cup G$ by introducing, in a bottom-up manner, new constants $c_i \in C$ for subterms $t = f(c_1, \ldots, c_n)$ where $f \in \Sigma$ and $c_i$ are constants, together with definitions $c_i = f(c_1, \ldots, c_n)$ (included in $\text{Def}$) and replacing the corresponding terms $t$ with the constants $c_i$ in $K$ and $G$.

**Theorem 1** ([14]) Let $K$ be a set of clauses. Assume that $T_0 \subseteq T_1 = T_0 \cup K$ is a $\Psi$-local theory extension and let $G$ be a finite set of $\Pi^C$-ground clauses. Let $K_0 \cup G_0 \cup \text{Def}$ be obtained from $K[\Psi_K(G)] \cup G$ as explained before. Then $T_1 \cup G \models \bot$ iff $T_0 \cup K_0 \cup G_0 \cup \text{Con}_0 \models \bot$, where $\text{Con}_0 = \{ \bigwedge_{i=1}^{n} c_i \approx d_i \rightarrow c \approx d \mid f(c_1, \ldots, c_n) \approx c \in \text{Def}, f(d_1, \ldots, d_n) \approx d \in \text{Def} \}$.

**Chains of extensions.** We can also consider chains of theory extensions:

$$T_0 \subseteq T_1 = T_0 \cup K_1 \subseteq T_2 = T_0 \cup K_1 \cup K_2 \subseteq \cdots \subseteq T_n = T_0 \cup K_1 \cup \cdots \cup K_n$$

in which each theory is a local extension of the preceding one. For a chain of $n$ local extensions a satisfiability check w.r.t. the last extension can be reduced (in $n$ steps) to a satisfiability check w.r.t. $T_0$. This iterated instantiation procedure has been implemented in H-PiLoT [7] – the only restriction needed in this case is that at each step the clauses reduced so far need to be ground.

## 4 Examples

In this section, we demonstrate the power of hierarchical reasoning and quantifier elimination for automatically proving graph properties. We show here as a proof-of-concept that the methods described in Section 3 can be used to prove the planarity of specific Euclidean graph classes. For determining the concrete proof tasks for testing containment of graph classes we used a form of abstraction that allowed us to use SCAN [6] for second-order quantifier elimination (Section 4.1). The proof tasks are then solved (Section 4.2) using the hierarchical reduction.
method described in Theorem 1. This method is implemented in H-PILoT [7] for the case when $\Psi$ is the identity. For a proof task of the form $\mathcal{T}_0 \cup \mathcal{K} \cup \mathcal{G}$, H-PILoT computes $\mathcal{T}_0 \cup \mathcal{K}[\mathcal{G}] \cup \mathcal{G}$, performs the hierarchical reduction described in Theorem 1 constructing the formula $\mathcal{T}_0 \cup \mathcal{K}_0 \cup \mathcal{G}_0 \cup \mathcal{Con}_0$, and calls a prover for the base theory (for instance Z3 or Redlog) and outputs the answer of the selected prover$^3$. (In general, chains of theory extensions are considered and the instantiation procedure is iterated.)

4.1 Second-order Quantifier Elimination for Class Inclusion

We present some simple examples in which second-order quantifier elimination allows us to determine constraints on abstract predicates used in the description of the classes under which inclusions hold. We analyze the type of axioms considered in Sect. 2 using a binary predicate $E$ for the edges and (abstract) predicates representing additional conditions.

Example 1. Consider the classes $C_1$ and $C_2$, described by axioms of the form

$$\text{Ax}_{C_i} : \forall u, v \ E(u, v) \leftrightarrow \pi_i(u, v) \quad i = 1, 2$$

We know that $C_1 \subseteq C_2$ iff $\text{Ax}_{C_1} \land \neg \text{Ax}_{C_2}$ is unsatisfiable. We have:

$$\text{Ax}_{C_1} \land \neg \text{Ax}_{C_2} \equiv \forall u, v \ (E(u, v) \leftrightarrow \pi_1(u, v)) \land \exists u, v (E(u, v) \land \neg \pi_2(u, v)) \lor$$

$$\forall u, v \ (E(u, v) \leftrightarrow \pi_1(u, v)) \land \exists u, v (\neg E(u, v) \land \pi_2(u, v))$$

We used SCAN to eliminate the predicate symbol $E$ and obtained:

$$\exists E(\forall u, v \ E(u, v) \leftrightarrow \pi_1(u, v) \land \exists u, v (E(u, v) \land \neg \pi_2(u, v))) \equiv \exists u, v (\pi_1(u, v) \land \neg \pi_2(u, v))$$

$$\exists E(\forall u, v \ E(u, v) \leftrightarrow \pi_1(u, v) \land \exists u, v (\neg E(u, v) \land \pi_2(u, v))) \equiv \exists u, v (\neg \pi_1(u, v) \land \pi_2(u, v)).$$

Hence, $C_1 \subseteq C_2$ iff $\exists u, v (\pi_1(u, v) \land \neg \pi_2(u, v)) \lor (\neg \pi_1(u, v) \land \pi_2(u, v))$ is false w.r.t. $\mathcal{T}$. This is the case iff $\forall u, v (\pi_1(u, v) \leftrightarrow \pi_2(u, v))$ is true w.r.t. $\mathcal{T}$. It is easy to see that these conditions are also equivalent to $C_2 \subseteq C_1$ and to $C_1 = C_2$. ■

Example 2. Let $\mathcal{C}$ be a class of graphs described by axioms $\text{Ax}_\mathcal{C}$ and $P$ the class of plane drawings, described by the axiom $\text{Ax}_P$, where:

$$\text{Ax}_\mathcal{C} : \forall u, v \ E(u, v) \leftrightarrow \pi_\mathcal{C}(u, v)$$

$$\text{Ax}_P : \forall u, v, w, x \ E(w, x) \land \pi_P(u, v, w, x) \rightarrow \neg E(u, v).$$

$\mathcal{C} \subseteq \mathcal{P}$ iff $\text{Ax}_\mathcal{C} \land \neg \text{Ax}_P \models \bot$. $\text{Ax}_\mathcal{C} \land \neg \text{Ax}_P$ is equivalent to:

$$\forall u, v \ (E(u, v) \leftrightarrow \pi_\mathcal{C}(u, v)) \land \exists u, v, w, x(E(w, x) \land \pi_P(u, v, w, x) \land E(u, v)).$$

We used SCAN to eliminate the predicate symbol $E$, and obtained:

$$\exists E(\text{Ax}_\mathcal{C} \land \neg \text{Ax}_P) \equiv \exists u, v, w, x (\pi_P(u, v, w, x) \land \pi_\mathcal{C}(w, x) \land \pi_\mathcal{C}(u, v)).$$

Thus, $\mathcal{C} \subseteq \mathcal{P}$ iff $\pi_P(a, b, c, d) \land \pi_\mathcal{C}(c, d) \land \pi_\mathcal{C}(a, b)$ is unsatisfiable w.r.t. $\mathcal{T}$. ■

$^3$ The answer can only be trusted if all theory extensions in the chain are local (or have complete finite instantiation and we ensure, when preparing the input for H-PILoT, that all necessary instances are considered).
Remark 1 If we consider the class $C \to$ (described by axiom $Ax_{C \to}$ obtained by only taking the direct implication in $Ax_C$) and use SCAN we obtain:

$$\exists E(Ax_{C \to} \land \neg Ax_p) \equiv \exists u,v,w,x(\pi_p(u,v,w,x) \land \pi_C(w,x) \land \pi_C(u,v)).$$

Therefore $C \subseteq P$ iff $C \subseteq P$.

Example 3. $C \to \subseteq D \to$ iff $\forall u,v(E(u,v) \to \pi_C(u,v)) \land \exists a,b(E(a,b) \land \neg \pi_D(a,b))$ is unsatisfiable w.r.t. $T$. Using e.g. SCAN we prove that this is the case iff $\pi_C(a,b) \land \neg \pi_D(a,b) \models T$ i.e. iff $\forall x,y(\pi_C(x,y) \to \pi_D(x,y))$ is valid w.r.t. $T$. □

For the type of axioms considered above, SCAN terminated and returned relatively simple first-order formulae. In [4] a method for computing weakest sufficient conditions for a formula $F$ based on second-order elimination techniques is presented; situations are identified under which conditions generated this way are first-order formulae. These results might be useful when analyzing more general graph classes.

4.2 Using Hierarchical Reasoning for Checking Class Inclusion

Let $T$ be a theory used for formalizing points, distance and vertices, with two sorts $p$ (points, uninterpreted) and $num$ (reals, interpreted). The concrete form of this theory depends on the model for the set of points and distances we choose. If we model the Euclidean plane with the Euclidean distance, $T$ can be defined starting from the combination of a theory of points $P$ and the theory $R$ of real numbers, using a chain of theory extensions:

$$(E) \quad P \cup R \subseteq P \cup R \cup \text{Free}_{x,y} \subseteq T = P \cup R \cup \text{Free}_{x,y} \cup T_d^c$$

where $x$ and $y$ are the coordinate functions and $T_d^c$ is the axiom stating that the Euclidean distance between points $u,v$ is $d(u,v) = \sqrt{(x(u) - x(v))^2 + (y(u) - y(v))^2}$.

If we model arbitrary metric spaces $(X,d)$, $T$ is the extension:

$$(M) \quad P \cup R \subseteq T = P \cup R \cup T_d^m$$

where $T_d^m$ are the axioms of a metric. All these extensions are local: in $(E)$ we have extensions with free function symbols and definitional extensions [14]; in [11] we proved that the axioms of a metric define a $\Psi$-local extension of $P \cup R$.

Example 4. We analyze the relation between the classes $G \to$ and $R \to$:

$$Ax_{G \to} : \forall u,v \,(E(u,v) \to \pi_G(u,v)) \quad Ax_{R \to} : \forall u,v \,(E(u,v) \to \pi_R(u,v)).$$

By Example 3, $G \to \subseteq R \to$ iff $\pi_G(u,v) \land \neg \pi_R(u,v)$ is unsatisfiable w.r.t. $T$, and $R \to \subseteq G \to$ iff $\pi_R(u,v) \land \neg \pi_G(u,v)$ is unsatisfiable w.r.t. $T$, where the predicates $\pi_G$ and $\pi_R$ are described (for an arbitrary distance function $d$) by:

$$\pi_G(u,v) := V(u) \land V(v) \land u \neq v \land \forall w \,(w \neq u \land w \neq v \land V(w) \to d(m(u,v),w) > d(m(u,v),u)),$$

$$\pi_R(u,v) := V(u) \land V(v) \land u \neq v \land \forall w \,(V(w) \to d(u,w) \geq d(u,v) \lor d(w,v) \geq d(u,v))$$

where $m(u,v)$ is the middle point of the segment $uv$. If $u$ and $v$ are considered to be constants then both the extension of $T$ with a function $V$ satisfying condition
\( \pi_G(u,v) \), and the extension of \( T \) with a function \( V \) satisfying condition \( \pi_R(u,v) \) can be proved to be local. We therefore can use H-PILoT for the proof tasks.

We first consider the case where \( d \) is the Euclidean distance. Then we analyze this problem for the more general case where \( d \) is an arbitrary metric.

**Case 1: \( d \) is the Euclidean distance.** We check both inclusions.

(i) To prove that \( R \to \subseteq G \to \) holds, it is sufficient to show that \( \pi_R(u,v) \land \neg \pi_G(u,v) \) is unsatisfiable. The formula obtained after Skolemization and simplification from \( \neg \pi_G(u,v) \) is a ground formula; the coordinates of the middle \( m \) of \( uv \) can be computed as usual; we add conditions: \( x(p) \neq x(q) \lor y(p) \neq y(q) \) for \( p, q \in \{u,v,w,m\} \). H-PILoT derives unsatisfiability, i.e. proves that \( R \to \subseteq G \to \).

(ii) For proving that \( G \to \not\subseteq R \to \) we show that \( \pi_G(u,v) \land \neg \pi_R(u,v) \) is satisfiable. H-PILoT gives the answer satisfiable and produces the following model:

\[
\begin{align*}
    u &= (1,0), & v &= (7,-8), & w &= (-1,-5), & m &= (4,-4).
\end{align*}
\]

**Case 2: \( d \) is an arbitrary metric.** We again check for both directions of containment whether the corresponding formula holds (assuming that we have distinct vertices). For \( d \) we have the metric axioms, proved to be \( \Psi \)-local in [11]. For the direction \( \pi_R(u,v) \to \pi_G(u,v) \) H-PILoT answers satisfiable and returns the following model (impossible in the Euclidean space):

\[
\begin{align*}
    d(u,v) &= 6, & d(u,m) &= 3, & d(v,m) &= 3, & d(u,w) &= 6, & d(v,w) &= 5, & d(w,m) &= 3.
\end{align*}
\]

Thus, the containment \( R \to \subseteq G \to \) is true in the Euclidean space, but not for arbitrary metric spaces. For the direction \( \pi_G(u,v) \to \pi_R(u,v) \) we also get a model describing a situation which cannot occur in the Euclidean space. ■

**Example 5.** We show that \( G \to \subseteq P \). By Example 2, this holds iff \( \pi_P(u,v,w,x) \land \pi_G(w,x) \land \pi_G(u,v) \) is unsatisfiable. With the encoding with the Euclidean distance H-PILoT did not terminate after 3 minutes. If \( d \) is an arbitrary metric then H-PILoT returns "unsatisfiable" (after 102.62 s.), which shows that \( G \to \subseteq P \). By Example 4, \( R \to \subseteq G \to \subseteq P \), hence, by Remark 1, \( G \subseteq P \) and \( R \subseteq P \). ■

We thus proved that the class \( G \) of Gabriel graphs and the class \( R \) of relative neighborhood graphs are subclasses of the class \( P \) of plane drawings.

## 5 Conclusions and Future Work

We demonstrated by example that hierarchical reasoning and quantifier elimination is a powerful tool to analyze properties of graph classes defined by general and Euclidean metrics. In subsequent work, we intend to investigate many more graph properties, e.g., spanner properties (Euclidean, topological, energy), and degree limitation. These concepts are of interest for algorithm design in wireless graph models but also on graphs in general. Furthermore, we plan to significantly expand the set of graph classes that can be analyzed with our tool set.

\footnote{The full test where all instances are considered is available under \( \text{https://userpages.uni-koblenz.de/~sofronie/tests-graphs-2021.} \).}
References


Abduction in $\mathcal{EL}$ via Translation to FOL

Fajar Haifani$^1$, Patrick Koopmann$^2$, and Sophie Tourret$^{1,3}$

1 Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany (\{fhaifani, stourret\}@mpi-inf.mpg.de)

2 Institute for Theoretical Computer Science, Technische Universität Dresden, Germany (patrick.koopmann@tu-dresden.de)

3 Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

Abstract. We present a technique for performing TBox abduction in the description logic $\mathcal{EL}$. The input problem is converted into first-order formulas on which a prime implicate generation technique is applied, then $\mathcal{EL}$ hypotheses are reconstructed by combining the generated positive and negative implicates.

Keywords: Abduction · Description Logic $\mathcal{EL}$ · Prime Implicates · First-order Logic.

1 Introduction

Description logic (DL) ontologies are used to formalize terminological knowledge in diverse areas such as medicine, biology, or the semantic web. Common reasoning tasks for ontologies are subsumption checking, which is to decide whether one description of a concept generalizes another, and classification, which is to compute the entire subsumption hierarchy of an ontology, that is, the set of all subsumption relationships that hold between atomic concepts in the ontology. Both problems can nowadays be computed rather efficiently using highly optimized description logic reasoners [24]. Modern ontologies can become very large and complex: for instance, the medical ontology SNOMED CT$^4$ contains over 350,000 statements, while the Gene Ontology GO defines over 50,000 concepts. But even for less complex ontologies, it is easy to introduce bugs, and possible inferences may not always be transparent to ontology users and engineers. Correspondingly, services for explaining those inferences can be very helpful. Typical techniques for explaining why subsumption can be inferred include justifications [4, 16, 27] and proofs [1, 2], both of which have been included in the popular ontology development tool Protégé [17, 18].

In order to explain why a subsumption relationship cannot be inferred, one can use abduction [21]. The general setup in abduction is that we have an

* Funded by the DFG grant 389792660 as part of TRR 248 – CPEC, see https://perspicuous-computing.science

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

4 https://www.snomed.org
input consisting of a knowledge base $K$ and some logical statement $\alpha$—the observation—that is not logically entailed by $K$. The aim is then to compute a hypothesis, a set of axioms whose addition to $K$ would result in the entailment $\alpha$. This way, the hypothesis not only provides for a possible explanation, it also offers a fix to repair the entailment in case it is supposed to hold. One way to avoid trivial hypotheses (such as the axiom $\alpha$ itself) is to restrict solutions syntactically, for instance by specifying a signature of predicates that can be used in the solution [20,22].

In the context of DLs, depending on the shape of the observation to be explained, abduction can be classified into different categories:

- In concept abduction observation and hypothesis are DL concepts. That means we are looking for a concept that is subsumed by a user-given concept (with respect to an ontology) [5].
- In ABox abduction, observations and hypotheses have the form of ABoxes, that is, they describe facts, which is the more typical scenario for diagnosis [6–9,11,15,19,20,25,26].
- In TBox abduction, observation and hypothesis consist of TBox axioms, which can be used to find fixes for missing subsumption relationships [10,28].
- Finally, in knowledge base abduction, the most general form, both hypotheses and observations can be arbitrarily composed of TBox and ABox axioms [13, 22].

We are interested in explaining missing subsumption relationships, which is what is done by TBox abduction, in which background knowledge, observation, and hypothesis all consist of TBox axioms, that is, DL axioms expressing subsumption relationships between concepts. Optionally, a set of abducible concepts can be specified as part of the input, of which every hypothesis needs to be composed.

We work in the lightweight DL $\mathcal{EL}$, used in many large scale ontologies [3].

TBox abduction has been considered before, albeit in different settings. In [10], instead of a set of abducibles, a set of justification patterns is given, which is a set of syntactical patterns that hypotheses have to fit in. In contrast, [28] generalises both this work and the one in [10], in that it uses an arbitrary oracle function to decide whether a solution is admissible or not (which may use abducibles, justification patterns, or something else). The work in [28] also considers $\mathcal{EL}$, and abduction under various minimality notions such as subset minimality and size minimality. It is there shown that deciding the existence of hypotheses is tractable, while for certain minimality criteria, it can become coNP-complete. In addition to the theoretical results, the authors of [28] present practical algorithms, and an evaluation of an implementation. Different to our approach, they use an external DL reasoner to decide entailment relationships. In contrast, we present an approach that directly exploits first-order reasoning, and thus has the potential to be generalisable to more expressive DLs.

The TBox abduction technique we present proceeds in three steps. The first step is the translation of the abduction problem into a first-order formula $\Phi$. Then, we compute the prime implicates of $\Phi$, that is, a minimal set of logical consequences of $\Phi$ that subsume all other consequences of $\Phi$. In the final step, we
Fig. 1. $\mathcal{EL}$ abduction using prime implicate generation in FOL.

construct TBoxes that are hypotheses of the original abduction problem, from the prime implicates of $\Phi$.

Figure 1 illustrates this process with some more details. The translation to first-order relies on the assumption that the TBox $\mathcal{T}$ is normalized and this results in a set of definite Horn clauses $\Phi$. Prime implicate (PI) generation in first-order logic can be done with an off-the-shelf tool [12, 23] or, in our case, by slightly altering a resolution-based theorem prover, such as included in the SPASS theorem prover [29]. Since the input is Horn, $\mathcal{P}I_{\Sigma}^+(\Phi)$, the set of ground positive prime implicates of $\Phi$, contains only unit clauses. The recombination step looks at the negative ground prime implicates in $\mathcal{P}I_{\Sigma}^-(\Phi)$ one after the other and attempts to match each literal in one such clause with clauses in $\mathcal{P}I_{\Sigma}^+(\Phi)$. If this succeeds, the result is a solution to the original problem. We denote $\mathcal{S}$ the set of all solutions obtained in this way.

In this paper, we describe in detail the translation and recombination steps and prove that the $\mathcal{S}$ obtained in that way contains only solutions to the original problem.

2 Preliminaries

We begin with an introduction of the relevant notions, first in $\mathcal{EL}$, then in first-order logic.

2.1 The Description Logic $\mathcal{EL}$

Let $\mathbb{N}_C$ and $\mathbb{N}_R$ be pair-wise disjoint, countably infinite sets of respectively atomic concepts and roles. Generally, we use letters $A, B, E, F, \ldots$ for atomic concepts, and $r$ for roles, possibly annotated, letters $C, D$, possibly annotated, denote $\mathcal{EL}$ concepts, built according to the syntax rule

$$C ::= \top \mid A \mid C \sqcap C \mid \exists r.C.$$ 

We use $\bigcap\{C_1, \ldots, C_m\}$ as abbreviation for $C_1 \cap \ldots \cap C_m$, and identify the empty conjunction (when $m = 0$) with $\top$. An $\mathcal{EL}$ TBox $\mathcal{T}$ is a finite set of concept inclusions (CIs) of the form $C \sqsubseteq D$. We use $C \equiv D$ as abbreviation for $\{C \sqsubseteq D, D \sqsubseteq C\}$. The semantics of $\mathcal{EL}$ is defined as usual using DL interpretations, which are tuples $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ made of a domain $\Delta^\mathcal{I}$ and an interpretation function...
\( \mathcal{I} \) that maps atomic concepts \( A \in \mathcal{N}_C \) to sets \( A^\mathcal{I} \subseteq \Delta^\mathcal{I} \) and roles \( r \in \mathcal{N}_R \) to relations \( r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I} \). The interpretation function \( \cdot \mathcal{I} \) is extended to complex concepts as follows:

\[
\top^\mathcal{I} = \Delta^\mathcal{I} \quad (C \cap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I} \\
(\exists r.C)^\mathcal{I} = \{ d \in \Delta^\mathcal{I} \mid \exists (d, e) \in r^\mathcal{I} \text{ s.t. } e \in C^\mathcal{I} \}
\]

We say that \( \mathcal{I} \) satisfies a CI \( C \subseteq D \), in symbols \( \mathcal{I} \models C \subseteq D \), if and only if \( C^\mathcal{I} \subseteq D^\mathcal{I} \), and if \( \mathcal{I} \) satisfies all axioms in a TBox \( \mathcal{T} \), we write \( \mathcal{I} \models \mathcal{T} \) and call \( \mathcal{I} \) a model of \( \mathcal{T} \). If a CI \( C \subseteq D \) is satisfied in every model of \( \mathcal{T} \), we write \( \mathcal{T} \models C \subseteq D \) and say that \( C \subseteq D \) is entailed by \( \mathcal{T} \). In this case, we say that \( C \) is subsumed by \( D \) and call \( C \) a subsumer of \( D \) and \( D \) a subsume of \( C \).

Note that, if we identify \( \mathcal{N}_C \) with the set of unary predicates and \( \mathcal{N}_R \) with the set of binary predicates, \( \mathcal{I} \) is a classical first-order structure for those predicates. We can thus define satisfaction of first-order formulas in DL interpretations as usual, and also relate first-order formulas with CIs and TBoxes via the usual entailment relation.

## 2.2 Abduction in \( \mathcal{EL} \)

The abduction problem we are concerned with in this paper is the following.

**Definition 1.** Given a TBox \( \mathcal{T} \), a set of atomic concepts \( \Sigma \subseteq \mathcal{N}_C \) and a concept inclusion \( C_1 \subseteq C_2 \), called the observation, where \( C_1 \) and \( C_2 \) are atomic concepts and \( \mathcal{T} \not\models C_1 \subseteq C_2 \), the corresponding TBox abduction problem, denoted by the tuple \( \langle \mathcal{T}, \Sigma, C_1 \subseteq C_2 \rangle \), is to find a TBox

\[
\mathcal{H} \subseteq \{ A_{i_1} \cap \cdots \cap A_{i_n} \subseteq B_{i_1} \cap \cdots \cap B_{i_m} \mid \{ A_{i_1}, \ldots, A_{i_n}, B_{i_1}, \ldots, B_{i_m} \} \subseteq \Sigma \}
\]

where \( m > 0 \), \( n \geq 0 \) and such that \( \mathcal{T} \cup \mathcal{H} \models C_1 \subseteq C_2 \) and \( \mathcal{T} \cap \mathcal{H} = \emptyset \). Such a solution \( \mathcal{H} \) to the abduction problem is called a hypothesis.

As an example, suppose we have a TBox \( \mathcal{T} = \{ C_1 \subseteq \exists r.A, \exists r.B \subseteq C_2 \} \). For the abduction problem \( \langle \mathcal{T}, \{ A, B \}, C_1 \subseteq C_2 \rangle \), the set \( \mathcal{H} = \{ A \subseteq B \} \) is a solution since \( \mathcal{T} \cup \mathcal{H} \models C_1 \subseteq C_2 \). Note that since \( \mathcal{EL} \) TBoxes are always consistent, the consistency condition usually required on \( \mathcal{T} \cup \mathcal{H} \) is not needed in our setting.

## 2.3 Prime Implicate Generation in First Order Logic

We consider first-order logic with unary and binary predicates, which we identify with the corresponding elements in \( \mathcal{N}_C \) and \( \mathcal{N}_R \). Let \( V \) be a set of variables, and \( \Sigma_{\text{FOL}} \) be a signature such that \( \Sigma_{\text{FOL}} = \mathcal{N}_C \cup \mathcal{N}_R \), where \( \mathcal{N}_C \) and \( \mathcal{N}_R \) are the sets of \( \mathcal{EL} \) atomic concepts and roles, used as unary and binary predicate symbols respectively. Furthermore, let \( F \) be a set of function symbols, which includes a set \( \mathcal{C} \) of constants as nullary function symbols. A term \( t \), possibly annotated, is either a variable \( x \in V \), a constant \( c \in \mathcal{C} \), or of the form \( f(t_1, \ldots, t_m) \), where \( f \) is an \( m \)-ary function symbol and \( t_1, \ldots, t_m \) are terms. Atoms are either of
the form $A(t_1)$ or $r(t_1, t_2)$, where $t_1$ and $t_2$ are terms. Literals are either atoms $\kappa$ (positive literals) or negated atoms $\neg \kappa$ (negative literals). Clauses, denoted $\varphi$ with possible annotations, are disjunctions of literals implicitly universally quantified. Formulas, denoted $\Phi$, are described using universal and existential quantification, conjunction, disjunction, and negation. For convenience, by abuse of notation, we may sometimes identify sets of formulas or clauses with the conjunction over those.

Skolemization removes existentially quantified variables from formulas in the usual way: for each existentially quantified variable, it adds a fresh function with variable arguments taken from the universally quantified variables outside the scope of the variable under consideration (if no such variables exist, we use a nullary function, i.e., a constant). For example, $\forall x. \neg A(x) \lor \exists y. r(x, y)$ is Skolemized into $\forall x. \neg A(x) \lor r(x, f(x))$. Here, we assume that the formula is in negation normal form, that is, every negation symbol occurs only in front of an atom. Furthermore, we assume that such Skolem functions are taken from a set $N_S$ not in the signature. In our context, we only require a single Skolem constant, namely $sk_0$, since Skolemized atoms, literals, and clauses range over Skolem terms either built over a given variable or $sk_0$ instead of ranging over variables. We let $T_x(N_S)$ be the set of Skolem terms built on $x$, where $x$ is either $sk_0$ or a variable in $\mathcal{V}$.

Since we only consider formulas over unary and binary predicates, they can be satisfied by DL interpretations as defined above, understood naturally as first-order structures. We call interpretations $I$ with $\Delta^I = T_{sk_0}(N_S)$ Herbrand interpretations, which for convenience, by abuse of notation, we may treat as sets of ground atoms s.t. $A(t) \in I$ if and only if $t \in A^I$ and $r(t, t') \in I$ if and only if $(t, t') \in r^I$.

**Definition 2 (Prime Implicate).** Given a set of clauses $\Phi$, a clause $\varphi$ is an implicate of $\Phi$ if $\Phi \models \varphi$. Additionally, $\varphi$ is a prime implicate of $\Phi$ if for any other implicate $\varphi'$ of $\Phi$ s.t. $\varphi' \models \varphi$, it also holds that $\varphi \models \varphi'$. Given a Skolemized $\Phi$, the set $\mathcal{PI}_\Sigma^+(\Phi)$ is the set of all positive ground prime implicants of $\Phi$ and the set $\mathcal{PI}_\Sigma^-(\Phi)$ is the set of all negative ground prime implicants where all of the predicate symbols belong to $\Sigma$.

**Example 3.** Given $\Phi$ as follows:

$$\Phi = \{ A(sk_0), \neg B(sk_0), \neg A(x) \lor r(x, sk(x)), \neg A(x) \lor E(sk(x)),$$
$$\neg G(x) \lor \neg r(x, y) \lor \neg E(y) \lor B(sk_0) \},$$

where $sk_0$ is a constant and $sk$ is a function symbol, the ground prime implicants of $\Phi$ are $\mathcal{PI}_\Sigma^+(\Phi) = \{ A(sk_0), E(sk(sk_0)) \}$ and $\mathcal{PI}_\Sigma^-(\Phi) = \{ \neg B(sk_0), \neg G(sk_0) \lor \neg E(sk(sk_0)) \}$. They are implicants because all of them are consequences of $\Phi$ (they are entailed by $\Phi$). Moreover, these implicants are also prime implicants because it is not possible to obtain an implicate of $\Phi$ by removing a literal from one of them.

In some cases, the set of prime implicants can even be infinite.
Example 4. Consider $\Phi'$, a slight variation on $\Phi$ from the previous example, where $E(sk(x))$ is replaced by $A(sk(x))$, so that:

$$
\Phi' = \{ A(sk_0), \neg B(sk_0), \neg A(x) \lor r(x, sk(x)), \neg A(x) \lor A(sk(x)), \\
\neg G(x) \lor \neg r(x, y) \lor \neg E(y) \lor B(sk_0) \},
$$

Whereas the negative ground prime implicates are unchanged ($\Pi_{g-}^\Sigma(\Phi') = \Pi_{g-}^\Sigma(\Phi)$), there are now infinitely many positive ground prime implicates, because $\Pi_{g+}^\Sigma(\Phi')$ contains $A(sk_0)$, $A(sk(sk_0))$, $A(sk(sk(sk_0)))) \ldots$

3 From $\mathcal{EL}$ to FOL

We assume the $\mathcal{EL}$ TBox in the input to be in normal form as defined in [3], which means that every CI is of the form $A \sqsubseteq B$, $A_1 \sqcap A_2 \sqsubseteq B$, $\exists r. A \sqsubseteq B$ or $A \sqsubseteq \exists r. B$, where $r \in \mathbb{N}_R$ and $A, A_1, A_2, B \in \mathbb{N}_C \cup \{\top\}$. Every $\mathcal{EL}$ TBox can be transformed in polynomial time into this normal form through the introduction of fresh concept names. This transformation preserves all entailments in the signature of the original ontology, even after the addition of CIs (assuming the same CIs are added also to the input ontology, and they don’t use any of the introduced concept names). Note that this transformation extends the signature, and thus potentially the solution space of the abduction problem: to avoid this, we may simply intersect the set of abducibles $\Sigma$ with the signature of the original input ontology.

At the level of TBoxes, we use the standard, semantics-preserving translation from $\mathcal{EL}$ to FOL [3].

**Definition 5 (TBox translation).** The function $\pi$ translates concepts into first-order formulas:

$$
\pi(\top, x) = \text{true} \\
\pi(A, x) = A(x) \\
\pi(\exists r.C, x) = \exists y. (r(x, y) \land \pi(C, y)) \\
\pi(C \sqcap D, x) = \pi(C, x) \land \pi(D, x)
$$

We lift $\pi$ to concept inclusions with: $\pi(C \sqsubseteq D) = \forall x. \neg \pi(C, x) \lor \pi(D, x)$. Finally, we lift $\pi$ to TBoxes with:

$$
\pi(\mathcal{T}) = \{ \pi(C \sqsubseteq D) \mid C \sqsubseteq D \in \mathcal{T} \}.
$$
If \( T \) is in normal form, \( \pi(T) \) contains only axioms of the following shapes:

shapes from CIs with \( \top \):
\[
\forall x.B(x) \\
\forall x.A(x) \vee \exists y.r(x, y), \\
\forall x.\neg(\exists y.r(x, y)) \vee B(x)
\]

shapes from CIs without \( \top \):
\[
\forall x.\neg A(x) \vee B(x), \\
\forall x.\neg A_1(x) \vee \neg A_2(x) \vee B(x), \\
\forall x.\neg(\exists y.r(x, y) \wedge A(y)) \vee B(x), \\
\forall x.\neg A(x) \vee \exists y.(r(x, y) \wedge B(y)).
\]

In practice, since the symbol \( \top \) will be preprocessed, only the four last forms are relevant in our work.

The function \( \pi \) is semantics-preserving in the following sense:

**Theorem 6 (First-Order Semantics Preservation [3])**. For any \( \mathcal{EL} \) TBox \( T \) and possibly complex concepts \( C \) and \( D \), \( T \models C \sqsubseteq D \) if and only if \( \pi(T) \models \pi(C \sqsubseteq D) \).

The translation of a TBox abduction problem, built on top of the standard TBox translation, is less straightforward.

The translation \( \Pi(T, C_1 \sqsubseteq C_2) \) of \( \langle T, \Sigma, C_1 \sqsubseteq C_2 \rangle \) to first-order logic is the Skolemization of
\[
\pi(T \uplus T') \wedge \neg \pi(C_1 \sqsubseteq C_2')
\]

where \( \text{sk}_0 \) is used as the unique fresh Skolem constant such that the Skolemization of \( \neg \pi(C_1 \sqsubseteq C_2') \) results in \( \{C_1(\text{sk}_0), \neg C_2'(\text{sk}_0)\} \), and where several other things happen that are explained in the following paragraphs.

Before Skolemization, \( T \) is first preprocessed by replacing every occurrence of the concept \( \top \) in \( T \) with that of the fresh atomic concept \( A_{\top} \) and we add \( \exists r.A_{\top} \sqsubseteq A_{\top} \) and \( B \sqsubseteq A_{\top} \) to \( T \) for every role \( r \) and atomic concept \( B \). This simulates for \( A_{\top} \) the implicit property that \( C \sqsubseteq \top \) holds for any \( C \) no matter what the TBox is. In particular, this ensures that whenever there is a positive prime implicate \( B(t) \) or \( r(t, t_1) \), \( A_{\top}(t) \) also becomes a prime implicate.

Finally, we define \( T' \) by renaming all atomic concepts \( A \) from \( T \) using fresh symbols \( A' \). Notice that this is also done for \( C_2 \), renamed to \( C_2' \) in \( \Pi(T, C_1 \sqsubseteq C_2) \).

Thanks to this encoding trick, the inferences can be traced back from \( C_1(\text{sk}_0) \) or from \( \neg C_2'(\text{sk}_0) \) and do not interfere with each other, making it easier to relate ground prime implicates and concept inclusions and capturing interesting solutions that would otherwise not be found.

We now look at a complete encoding example in details.

**Example 7.** Suppose given \( T \) over the signature where \( \mathcal{N}_C = \{A, B, C_1, C_2, E, F, G, H\} \) and \( \mathcal{N}_R = \{r_1, r_2\} \) as follows.
\[
T = \{C_1 \sqsubseteq \exists r_1.A, C_1 \sqsubseteq B, C_1 \sqsubseteq \exists r_2.G, \\
\exists r_2.E \sqsubseteq B, \exists r_1.F \sqsubseteq H, B \sqcap H \sqsubseteq C_2\}
\]
Consider the abduction problem \( \langle T, N_C, C_1 \sqsubseteq C_2 \rangle \). Preprocessing the \( \top \) symbol turns \( T \) into:

\[
T = \{ C_1 \sqsubseteq \exists r_1.A, \ C_1 \sqsubseteq B, \ C_1 \sqsubseteq \exists r_2.G, \ \\
\exists r_2.E \sqsubseteq B, \ \exists r_1.F \sqsubseteq H, \ B \cap H \sqsubseteq C_2 \}
\]

and the renaming of atomic concepts produces:

\[
T' = \{ C'_1 \sqsubseteq \exists r_1.A', \ C'_1 \sqsubseteq B', \ C'_1 \sqsubseteq \exists r_2.G', \ \\
\exists r_2.E' \sqsubseteq B', \ \exists r_1.F' \sqsubseteq H', \ B' \cap H' \sqsubseteq C'_2 \}
\]

The translation of \( T \) results in \( \Phi_T \), the Skolemization of \( \pi(T) \):

\[
\Phi_T = \{ \neg C_1(x) \lor r_1(x, sk_1(x)), \ \neg C_1(x) \lor A(sk_1(x)), \ \\
\neg C_1(x) \lor B(x), \ \\
\neg C_1(x) \lor r_2(x, sk_2(x)), \ \neg C_1(x) \lor G(sk_2(x)), \ \\
\neg r_2(x, y) \lor \neg E(y) \lor B(x), \ \\
\neg r_1(x, y) \lor \neg F(y) \lor H(x), \ \\
\neg B(x) \lor \neg H(x) \lor C_2(x), \ \\
\forall \{ \neg r_1(x, y) \lor A_\top(y) \lor A_\top(x), \ \neg r_2(x, y) \lor A_\top(y) \lor A_\top(x) \} \}
\]

\( \Phi_{T'} \), the Skolemization of \( T' \), is similar except that the atomic concepts are renamed. Finally, \( \Pi(T, C_1 \sqsubseteq C_2) = \Phi_T \cup \Phi_{T'} \cup \{ C_1(sk_0), \neg C'_2(sk_0) \} \).

Let us now look in more detail at the reason for replacing \( \top \) with \( A_\top \). First, remember that \( \top \) is not considered as a part of \( N_C \) and represents an empty conjunction. Since it is translated to \texttt{true} in first-order logic, it never appears in a translated formula, as can be seen in the text following Definition 5. However, our technique is only looking for the ground prime implicates. There are many other non-ground prime implicates that are generated and that we only use in the hope of finally grounding them. It would not be practical to have to sort out the non-ground PIs due to the \( \top \) symbol from the rest of them. Consider the following example.

\textbf{Example 8.} Given \( \Sigma = \{ A, E, F, G \} \) and the TBox

\[
T = \{ \top \sqsubseteq \exists r_1.F, \ C_1 \sqsubseteq \exists r_2.A, \ A \sqsubseteq \exists r_3.G, \ \exists r_2.E \sqsubseteq J_1, \ \exists r_1.J_1 \sqsubseteq J_0, \ \\
\exists r_2.\top \sqsubseteq J_3, \ J_0 \cap J_3 \sqsubseteq C_2 \}
\]

Some of the ground PIs of \( \Pi(T, C_1 \sqsubseteq C_2) \) are as follows:
– $A(sk_2(sk_0))$, $F(sk_1(sk_2(sk_0)))$, $G(sk_3(sk_2(sk_0)))$ are positive PIs,
– $\neg E'(sk_1(sk_2(sk_0))) \lor \neg A'(sk_3(sk_2(sk_0)))$ is a negative PI,

where the Skolem functions $sk_1$, $sk_2$, and $sk_3$, are from the Skolemization of $r_1$, $r_2$, and $r_3$ respectively in the first three axioms.

Let us take a close look at the reason why $F(sk_1(sk_2(sk_0)))$ is an implicate. We observe that $A(sk_2(sk_0))$ is obtained as a consequence of the presence of $C_1 \sqsubseteq \exists r_2.A$ in $T$. Thanks to the addition of $A \sqsubseteq A_\top$ to $T$, $A_\top(sk_2(sk_0))$ is also an implicate. Finally due to the presence of the renaming of $\top \sqsubseteq \exists r_1.F$ in $T$, namely $A_\top \sqsubseteq \exists r_1.F$, the clause $F(sk_1(sk_2(sk_0)))$ is also an implicate. With a direct translation of $\top$, we would only be able to derive $F(sk_1(x))$ and there would be no way to obtain a ground implicate from this clause.

Similarly, it is only thanks to the presence of $\exists r_3.A_\top \sqsubseteq A_\top$ in $T$ that the prime implicate $\neg E'(sk_1(sk_2(sk_0))) \lor \neg A'(sk_3(sk_2(sk_0)))$ can be deduced.

### 4 Building $\mathcal{EL}$ hypotheses from prime implicates

We use the ground prime implicates of $\Phi = \Pi(T, C_1 \sqsubseteq C_2)$ to produce solutions to the abduction problem $\langle T, \Sigma, C_1 \sqsubseteq C_2 \rangle$ in $\mathcal{EL}$. To do this, we identify subsumers of $C_1$ and subsumees of $C_2$ that can be generated from the ground prime implicates in $\Phi$, and that can be matched with each other based on the Skolem terms they contain.

**Definition 9 (Solution reconstruction).** Let $\langle T, \Sigma, C_1 \sqsubseteq C_2 \rangle$ be an abduction problem and let $\Phi = \Pi(T, C_1 \sqsubseteq C_2)$, and assume there are prime implicates

$$\{A_1(t_1), \ldots, A_n(t_n)\} \subseteq \mathcal{PT}_+^g(\Phi) \text{ and } \neg B'_1(t'_1) \lor \ldots \lor \neg B'_m(t'_m) \in \mathcal{PT}_-^g(\Phi)$$

such that $\{t_i\}_{i \in [1,n]} = \{t'_i\}_{i \in [1,m]}$. We can then obtain a hypothesis $\mathcal{H}$ for the abduction problem

$$\mathcal{H} = \{l_1(t) \sqsubseteq l_2(t) \mid t \in L \text{ and } T \not\models l_1(t) \sqsubseteq l_2(t)\},$$

such that

– $L = \{t_i\}_{i \in [1,n]} = \{t'_i\}_{i \in [1,m]}$,
– $l_1(t) = \bigcap\{A \mid A(t) \in \{A_1(t_1), \ldots, A_n(t_n)\}\}$, and
– $l_2(t) = \bigcap\{B \mid B'(t) \in \{B'_1(t'_1), \ldots, B'_m(t'_m)\}\}$.

We denote by $S(\Phi, \Sigma)$ the set containing all such $\mathcal{H}$.

**Example 10.** From Example 7,

$$T = \{ C_1 \sqsubseteq \exists r_1.A, \ C_1 \sqsubseteq B, \ C_1 \sqsubseteq \exists r_2.F, \ B \sqcap \exists r_1.E \sqsubseteq C_2, \ \exists r_1.A_\top \sqsubseteq A_\top, \ \exists r_2.A_\top \sqsubseteq A_\top \}$$

$$\cup \{J \sqsubseteq A_\top \mid J \in \{B, C_1, C_2, A, B, E, F, G\}\}.$$ 

Consider the prime implicates from $\Pi(T, C_1 \sqsubseteq C_2)$
\[- \{B(\sk_0), A(\sk_1(\sk_0))\} \subseteq \Pi_\Sigma^{g,+}(\Phi), \text{ and} \]
\[- \neg B'(\sk_0) \lor C'(\sk_1(\sk_0)) \in \Pi_\Sigma^{g-}(\Phi). \]

From the Skolem term \(\sk_0\) and \(\sk_1(\sk_0)\), we have \(l_1(\sk_0) \subseteq l_2(\sk_0) = B \subseteq B\) and \(l_1(\sk_1(\sk_0)) \subseteq l_2(\sk_1(\sk_0)) = A \subseteq C\) respectively. However, \(B \subseteq B\) is a tautology. Consequently, \(\mathcal{H} = \{A \subseteq C\}\) is one possible hypothesis.

**Theorem 11.** Given a TBox abduction problem \(\langle T, \Sigma, C_1 \sqsubseteq C_2\rangle\), and denoting \(\Phi\) the set \(\Pi(T, C_1 \sqsubseteq C_2)\), the set \(\mathcal{S}(\Phi, \Sigma)\) contains only hypotheses for the abduction problem.

**Proof.** Let \(\Phi_T\) denote the Skolemization of \(\pi(T)\). If \(-B'_1(t'_1) \lor \ldots \lor -B'_m(t'_m) \in \Pi_\Sigma^{g-}(\Phi)\), then \(\Phi_T \cup \{C_1(\sk_0), -C_2(\sk_0)\} \models -B_1(t_1) \lor \ldots \lor -B_m(t_m)\) because using, e.g., resolution, any inference to derive the primed disjunction can be imitated without primes. Moreover, \(A(t) \in \Pi_\Sigma^{g+}(\Phi)\) implies \(A(t) \in \Pi_\Sigma^{g+}(\Phi_T \cup \{C_1(\sk_0)\})\) since only non-primed clauses are relevant to entail positive literals. Thus \(\Phi_T \cup \{C_1(\sk_0)\} \models A_1(t_1) \land \ldots \land A_n(t_n)\).

By combining the two results for compatible positive and negative clauses, we obtain \(\Phi_T \cup \{C_1(\sk_0), -C_2(\sk_0)\} \models \pi(l_1(t), t) \land \neg \pi(l_2(t), t)\) for all \(t \in L\) where \(L\), \(l_1(t)\) and \(l_2(t)\) are as defined in Definition 9. In turn, this implies \(\Phi_T \cup \{C_1(\sk_0), -C_2(\sk_0)\} \models \exists x.(\pi(l_1(t), x) \land \neg \pi(l_2(t), x))\) for all \(t \in L\). Thus \(\Phi_T \cup \{C_1(\sk_0), -C_2(\sk_0)\} \cup \{\forall x.(-\pi(l_1(t), x) \lor \pi(l_2(t), x))\} \models \Pi_\Sigma(\mathcal{H})\) for all \(t \in L\). Hence, for any hypothesis \(\mathcal{H} \in \mathcal{S}(\Phi, \Sigma)\), we know that \(\Phi_T \cup \{C_1(\sk_0), -C_2(\sk_0)\} \cup \Pi_\Sigma(\mathcal{H})\) is unsatisfiable. Consequently, because Skolemization is satisfiability-preserving, \(\pi(T) \cup \neg \pi(C_1 \sqsubseteq C_2) \cup \pi(\mathcal{H})\) is also unsatisfiable, hence \(\pi(T) \cup \pi(\mathcal{H}) \models \pi(C_1 \sqsubseteq C_2)\). By Theorem 6, this in turn means that \(T \cup H \models C_1 \sqsubseteq C_2\). \(\square\)

## 5 Conclusion

We have introduced a technique to compute hypotheses to solve TBox abduction problems in \(\mathcal{EL}\). This technique turns the input TBox into a first-order formula, of which the ground prime implicates can be used to build hypotheses answering the problem.

We have not shown the prime implicate computation step itself for two reasons. First, existing tools such as SOLAR [23] and GPiD [12] can compute prime implicates automatically in first-order logic. Second, there is an issue with the termination of such tools for problems with cyclic dependencies in the CIs, e.g., such that \(T \models A \sqsubseteq \exists r.A\), because in that case, the set of prime implicates may be unbounded. However, there exists only a finite number of hypotheses to be found and we conjecture that prime implicates with polynomial nesting depth of Skolem terms are sufficient for finding all relevant hypotheses. We plan to verify this conjecture and implement a prime implicate generation tool on top of SPASS [29] as the next step of this work.

An orthogonal concern that we are currently addressing is to relate the obtained hypotheses with connection-minimality [14], a notion we recently introduced to characterize solutions to the abduction problem in \(\mathcal{EL}\) that we find...
particularly interesting, because they connect the subsumers of $C_1$ to the subsumees of $C_2$ in a less contrived way than other solutions. We believe that the present technique is sound and complete with respect to finding connection-minimal hypotheses, but that remains to be formally proved.

Finally, as our approach works by generating ground prime implicates starting from a set of ground literals, it should also be possible to use it for a mix of ABox and TBox abduction, where our background knowledge consists of a DL knowledge base containing both a TBox and ground facts, the observation is another set of ground facts, and we are looking for hypotheses in the form of a TBox. This would have applications for TBox learning from data.

References


An Abstract Fixed-point Theorem
for Horn Formula Equations
(Abstract)

Stefan Hetzl and Johannes Kloibhofer

Institute of Discrete Mathematics and Geometry, TU Wien, Austria
stefan.hetzl@tuwien.ac.at, j.kloibhofer@gmx.net

We consider the problem of solving a formula equation, i.e., given a formula $\exists X \varphi$ where $\varphi$ is first-order and $X$ is a tuple of predicate variables, to find a substitution $[X \backslash \psi]$ s.t. $\varphi[X \backslash \psi]$ is a valid first-order formula. This problem is also known as Boolean solution problem in the literature [9] and is closely related to second-order quantifier elimination.

More specifically, we focus on the class of Horn formula equations, which is defined by restricting $\varphi$ to be a Horn clause set w.r.t. the predicate variables. We state and prove a fixed-point theorem for Horn formula equations based on expressing the fixed-point computation of a minimal model (in the sense of logic programming) of a set of Horn clauses on the object level as a formula in first-order logic with a least fixed-point operator. This result is shown by an extension of the fixed-point approach of Nonnengart and Sza/

suppress las to second-order quantifier elimination [7]. Our fixed-point theorem applies not only to the usual semantics of second-order logic and first-order logic with a least fixed-point operator but also to model abstractions, a semantics for logical formulas that corresponds to abstract interpretation of programs using Galois connections [2].

Our fixed-point theorem allows both new results and simpler proofs of existing results as applications and corollaries.

1. It entails expressibility of the weakest precondition and the strongest postcondition, and thus the partial correctness of an imperative program, in first-order logic with a least fixed-point operator.
2. It allows a generalisation of a result by Ackermann [1] on approximating a second-order formula by first-order formulas in a direction different from the recent generalisation [8].
3. It allows to obtain a result from a recently introduced approach to automated inductive theorem proving with tree grammars [3] as another straightforward corollary.
4. Since it incorporates abstract interpretation, it permits to considerably simplify the proof of the decidability of affine formula equations originally presented in [5].

This work is rooted in the second author’s master’s thesis [6]. Some of these results have been presented at the 8th Workshop on Horn Clauses for Verification and Synthesis [4].

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
References


Signature-Based ABox Abduction in $\mathcal{ALC}$ is Hard

Patrick Koopmann

Institute for Theoretical Computer Science, Technische Universität Dresden, Germany

Abstract. In ABox abduction, we are given a knowledge base together with an ABox—the observation—that is not logically entailed by the knowledge base, and are looking for another ABox—the hypothesis—that, when added to the knowledge base, would make the observation entailed. This has applications in explaining negative entailments and missing query answers, as well as in diagnosis. To get useful hypotheses, in signature-based abduction, we additionally provide a signature of abducible names, and require the hypothesis to use only names from that signature. In the variant we are considering, the hypothesis may otherwise use fresh individual names, as well as complex concepts constructed in arbitrary way using the names in the signature. It was recently shown that this variant of abduction is in N2EXPTIME$^\mathsf{NP}$, and that hypotheses may require concepts that are of triple exponential size. We complement those results by showing a matching N2EXPTIME$^\mathsf{NP}$ lower bound, and show that in the worst case, hypotheses may also use a double exponential number of fresh individual names.

1 Introduction

Since inferences performed by description logic reasoners can be complex, and real ontologies often contain 10,000s of axioms, explanations of description logic reasoning has since longer been in the focus of research [26]. In particular for explaining positive entailments, that is, entailments that hold for a given knowledge base, there is a plethora of research, mostly focused on using justifications [33,4,30,17], but recently also on using proofs [1,2]. Explaining negative entailments, that is, entailments that do not hold for a given knowledge base, has been less in the focus of attention. Here, common approaches are showing the user a counter example in form of a description logic interpretation [5,3], or using abduction [29,23]. In abduction, one is given a knowledge base and an observation, a formula that is not logically entailed by the knowledge base, and is interested in finding a missing piece in the knowledge base that would make the observation logically entailed, called a hypothesis for the observation. Abduction can be used in different ways to explain negative entailments to ontology users [23,7,8] and to repair missing entailments [34]. Another application of abduction is diagnosis: here, the observation describes symptoms, for instance of a medical condition or a faulty machine [27,8], and the hypothesis is supposed to give a possible explanation for how the symptoms came into place.

* This work was supported by the DFG in grant 389792660 as part of TRR 248.

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
To illustrate, we reuse a simplified example from [21] from the geology domain. The observation is that the ground under a street has become unstable: \{Unstable(street)\}, and the background knowledge involves the following axioms:

1. EvaporiteFormation \(\sqcap\exists\text{borders.}(\text{WaterWay} \sqsubset \neg\exists\text{lining.}\text{WaterProof})\)
2. EvaporiteFormation \(\sqcap\exists\text{affectedBy.}\text{Dissolution}\)
3. \((\text{Waterway} \sqcup \text{Street}) \sqcap \text{EvaporiteFormation} \sqsubset \bot\)
4. Waterway(canal) 5. Street(street),

These state: 1. an evaporite formation bordering to a waterway without waterproof lining is affected by dissolution; 2. if an evaporite formation is affected by dissolution, then everything above it is unstable; 3. waterways and streets are different from evaporite formations; 4. canal is a waterway and 5. street is a street. A meaningful hypothesis could then be the following, which uses \(e\) as a fresh individual referring to a possible unknown formation under the street, as well as the complex concept \(\forall\text{lining.}\bot\):

\[
\mathcal{H} = \{ \text{EvaporiteFormation}(e), \text{above}(e, \text{street}), \\
\text{borders}(e, \text{canal}), \forall\text{lining.}\bot(\text{canal}) \}
\]

Depending on whether observation and hypothesis consist of concepts, facts, terminological axioms or knowledge bases, one distinguishes between concept abduction [6], ABox abduction [9,8,32,31,7,12,10,16,19], TBox abduction [11,34,15,14], and knowledge base abduction [24,13]. In this paper, we focus on ABox abduction, where, as in the above example, both the observation and the hypothesis consist of a set of facts about some set of individuals, a variant especially suited for diagnosis and for generating counter examples as in [23].

To avoid trivial answers (such as the observation itself), usually, the abduction problem is specified with additional constraints on the hypothesis. Those may be purely of logical nature [13], or based on syntactic properties. For instance, in [11], the user specifies syntactical patterns of axioms that the hypothesis must conform to. Another approach is to fix a finite set of abducible axioms—either directly as part of the input [31,8], or indirectly by posing strict conditions on the shape of axioms, such as being flat ABox assertions using only names and individuals from the input [32,10]. However, such a restriction might not be feasible in applications where we do not know the exact shape of the axioms we are looking for, for instance because they involve unknown individuals or complex concepts as in the example above. Another approach is to instead specify a set of abducible names or predicates, from which hypotheses can then be constructed in arbitrary fashion using the constructors of the respective description logic. In the context of diagnosis, this set would contain names that are connected to causes of symptoms rather than the symptoms themselves, specific terms rather than generic categories, and possibly describing aspects that can be verified to check the hypothesis. In the example above, examples for names that would not be included would be Unstable (because it is about the observation, not about the cause), Waterway and Street (because we know all waterways and streets in the area), affectedBy and Dissolution (because they are too unspecific, and we are not looking for the processes, but the objects that caused the street to become unstable.).
This form of abduction, which we call signature-based abduction, has been theoretically investigated for DL-Lite in [7], and for more expressive DLs in [21]. There is also a practical implementation for signature-based ABox abduction [9] and one for signature-based KB abduction [24]. These two practical implementation reduce abduction to uniform interpolation via (extensions of) the tool LETHE [20], which can lead to triple exponentially large solutions [25]. As shown in [21], this is indeed in theory unavoidable. However, some crucial questions were left open in [21]: for the corresponding decision problem whether there exists a hypothesis for a given abduction problem, using \( \mathcal{ALC} \) as description logic and allowing for both fresh individuals and complex concepts in the hypothesis, only an upper bound of \( \text{N2ExpTime}^{\text{NP}} \) is provided, while the lower bound is left open. Furthermore, for the number of fresh individuals one may need to introduce in the worst case, also only an upper bound was developed. In this paper, we show that those bounds are indeed tight, by proving that the general version of the signature-based ABox abduction problem for \( \mathcal{ALC} \) is \( \text{N2ExpTime}^{\text{NP}} \)-complete, and that hypotheses for those problems may require a number of fresh individuals that is double exponential in the size of the abduction problem.

2 Preliminaries

We recall the description logic \( \mathcal{ALC} \) and define the signature-based ABox abduction problem formally.

Let \( N_C, N_R \) and \( N_I \) be pair-wise disjoint, countably infinite sets of respectively concept, role and individual names. A signature is a finite set \( \Sigma \subseteq N_C \cup N_R \). Concepts are then built according to the following syntax rule, where \( A \in N_C \) and \( r \in N_R \):

\[
C := \top | \bot | A | \neg C | C \sqcup C | C \sqcap C | \exists r.C | \forall r.C
\]

A TBox is a finite set of general concept inclusion axioms (GCIs) of the form \( C \sqsubseteq D \) and concept equivalence axioms of the form \( C \equiv D \), where \( C \) and \( D \) are concepts. An ABox is a finite set of concept assertions of the form \( C(a) \) and role assertions of the form \( r(a,b) \), where \( C \) is a concept, \( r \in N_R \) and \( a,b \in N_I \). A knowledge base (KB) \( K \) is a union of a TBox and an ABox, and thus generalises both notions. The contents of a KB are called axioms. For a concept/axiom/KB \( E \), we denote its signature, that is, the concept and role names that occur in \( E \), by \( \text{Sig}(E) \).

The semantics of KBs is defined in terms of interpretations. An interpretation \( \mathcal{I} \) is a tuple \( \langle \Delta^\mathcal{I}, \mathcal{I} \rangle \) of a countable but possibly infinite domain \( \Delta^\mathcal{I} \) and an interpretation function \( \mathcal{I} \) that assigns to each concept name \( A \in N_C \) a set \( A^\mathcal{I} \subseteq \Delta^\mathcal{I} \), and to each role \( r \in N_R \) a relation \( r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I} \). The interpretation function is extended as follows to concepts:

\[
\begin{align*}
\top^\mathcal{I} &= \Delta^\mathcal{I} \\
\bot^\mathcal{I} &= \emptyset \\
(\neg C)^\mathcal{I} &= \Delta^\mathcal{I} \setminus C^\mathcal{I} \\
(C \sqcap D)^\mathcal{I} &= C^\mathcal{I} \cap D^\mathcal{I} \\
(C \sqcup D)^\mathcal{I} &= C^\mathcal{I} \cup D^\mathcal{I} \\
(\exists r.C)^\mathcal{I} &= \{d \in \Delta^\mathcal{I} \mid \text{there is } \langle d, e \rangle \in r^\mathcal{I} \text{ s.t. } e \in C^\mathcal{I} \} \\
(\forall r.C)^\mathcal{I} &= \{d \in \Delta^\mathcal{I} \mid \text{for every } \langle d, e \rangle \in r^\mathcal{I} \text{ we have } e \in C^\mathcal{I} \}.
\end{align*}
\]
Such an interpretation $I$ satisfies an axiom $\alpha$, in symbols $I \models \alpha$ if $\alpha = C \sqsubseteq D$ and $C^I \subseteq D^I$, $\alpha = C \sqsubseteq D$ and $C^I = D^I$, $\alpha = C(a)$ and $a^I \in C^I$, or $\alpha = r(a, b)$ and $\langle a^I, b^I \rangle \in r^I$. If $I$ satisfies all axioms in a KB $K$, we write $I \models K$ and call $I$ a model of $K$. If a KB $K$ has no model, we say it is inconsistent and write $K \models \bot$. If an axiom $\alpha$ is satisfied in every model of a KB $K$, we write $K \models \alpha$ and say $\alpha$ is entailed by $K$.

We are now ready to define the signature-based ABox abduction problem for $\mathcal{ALC}$.

Definition 1. A signature based ABox abduction problem is given by a tuple $\mathfrak{A} = \langle K, \Phi, \Sigma \rangle$ of a KB $K$ called the background knowledge, an ABox $\Phi$ called the observation, and a signature $\Sigma$ of abducible names, for which we want to determine whether there exists a hypothesis $\mathcal{H}$, which is an ABox that satisfies the following conditions:

\begin{align*}
A1 & \quad K \cup \mathcal{H} \not\models \bot, & A2 & \quad K \cup \mathcal{H} \models \Phi, \text{ and} & A3 & \quad \text{sig}(\mathcal{H}) \subseteq \Sigma.
\end{align*}

Intuitively, the hypothesis should not contradict what we know (Condition A1), it should be sufficient to explain the observation (Condition A2), and it should only be constructed using the abducible names (Condition A3). Apart from that, we put no further restriction on $\mathcal{H}$: it may use individual names that occur neither in $K$ nor in $\Phi$ (fresh individuals), and it may contain assertions $C(a)$ where $C$ is a complex concept, arbitrarily composed using the constructors of $\mathcal{ALC}$ and the names in $\Sigma$.

Regarding the corresponding decision problem, the following was shown in [21]:

Theorem 1. Existence of hypotheses for signature-based ABox abduction problems is in $\text{N2EXP\text{-TIME}}^{\text{NP}}$.

However, a corresponding lower bound was not yet provided, which is the contribution of this paper. In addition, [21] provided the following bounds on the size of hypotheses.

Theorem 2. There exists a family of signature-based ABox abduction problems for which every hypothesis is of size triple exponential in the size of the problem. Furthermore, for every signature-based ABox abduction problem, if there exists a hypothesis, then there exist one that is of size at most triple exponential in the size of the problem.

The family of abduction problems used for the lower bound is such that only a single assertion is needed for the hypothesis. Therefore, an open problem remained how many fresh individuals may be required in a hypothesis. Inspection of the proof used for the upper bound of Theorem 2 gives at least an upper bound for this.

Theorem 3. For every signature-based $\mathcal{ALC}$-ABox abduction problem, if there exists a hypothesis, then there exist one that uses a number of fresh individual names that is at most double exponential in the size of the abduction problem.

Proof. This is a consequence of the following lemmas from [21]: 1. every hypothesis can be converted into a so-called hypothesis abstraction (Lemma 1), 2. every hypothesis abstraction can be restricted to refer to at most double exponentially many individual names (Lemma 2), and 3. those bounds are preserved when translating the hypothesis abstraction back into a hypothesis (Lemma 3). □

We will also show that this bound is tight: in our reduction, we create a class of signature-based abduction problems whose hypotheses always need a double exponential number of fresh individuals.
3 Overview of the Reduction

To prove hardness for N2EXP\text{TIME}^{NP}, we provide a reduction from the word problem for non-deterministic, double exponential Turing machines with NP oracle. Specifically, given such a Turing machine $M$ and a word $w$, we will construct an abduction problem $\mathcal{A}_{M,w}$ in polynomial time s.t. $M$ accepts $w$ iff there exists a hypothesis for $\mathcal{A}_{M,w}$. The individual names in this hypothesis will be organised into a $N \times N \times N$-cube, where $N = \{0, \ldots, 2^n \cdot 2^{2^n}\}$, and $n$ is polynomial in the size of $w$. That is, for every $\langle x, y, z \rangle \in N \times N \times N$, there will be a corresponding individual $a_{x,y,z}$. The individuals on one side of the cube, that is, those of the form $a_{x,y,0}$, will encode an accepting configuration history for the Turing machine. The remaining individuals supply the “guessing space” for the oracle via the models of the hypothesis. Specifically, for a fixed $y$, the individuals $a_{x,y,z}$ will be such that they cannot possibly encode a successful computation history for the oracle in any model of the hypothesis and the background knowledge.

Our abduction problem is composed of different components, that we define as separate abduction problems:

1. we reuse a construction from [21] to obtain three double exponential counters;
2. those are used to create the different individuals for all coordinates in $N \times N \times N$;
3. we then enforce that every hypothesis forms a cube in which every coordinate is represented by exactly one individual;
4. finally, we make sure that hypothesis corresponds to a computation history of the Turing machine as described above.

4 The Double Exponential Counters

We encode numbers with $2^n$ bits using chains of $2^n$ individuals, where each individual is an instance of the concept name $\text{Bit}$ if it represents a bit with value 1, and otherwise an instance of $\neg \text{Bit}$. An abduction problem that exactly provides this is constructed in the proof for Theorem 5 in [21,22] to show the triple exponential lower bound on the size of hypothesis with complex concepts. This abduction problem is of the form $\mathcal{A}_c = \langle K_c, \text{Goal}(a), \Sigma_c \rangle$, where $\{r, \text{Bit}, B\} \subseteq \Sigma_c$. $K_c$ is such that, for any individual name $b$, the only way to entail $\text{Goal}(b)$ using only names from $\Sigma_c$ is by enforcing that every path of $r$-successors starting from $b$ encodes a $2^n$-bit counter as required, starting from a value of $2^n \cdot 2^{2^n}$ and counting down to 0, ending at an individual satisfying $B$. (The construction also increments this counter along another role $s$, which is however not relevant for our purposes here.) This also means that no such path can contain any cycles or shortcuts before reaching the instance of $B$ after $2^n \cdot 2^{2^n}$ $r$-successors.

We will need three such double exponential counters, for which purpose we rename all names in $\mathcal{A}_c$ to obtain the three signature-disjoint variants $\mathcal{A}_x = \langle K_x, \text{Goal}_x(a), \Sigma_x \rangle$, $\mathcal{A}_y = \langle K_y, \text{Goal}_y(a), \Sigma_y \rangle$ and $\mathcal{A}_z = \langle K_z, \text{Goal}_z(a), \Sigma_z \rangle$. Specifically, these use now roles $r_x$, $r_y$ and $r_z$ to connect the individuals in the counting sequence, the concept names $\text{Bit}_x$, $\text{Bit}_y$ and $\text{Bit}_z$ for the respective bit values, and $B_x$, $B_y$ and $B_z$ to mark the end points of the counters. All counters count backwards: that is, they start at the maximal value, and then step-wise decrease along the role-successors.
5 Creating the Coordinates

To generate all coordinates in our cube, we now need to connect those abduction problems appropriately. Specifically, we need the counter for a coordinate \( d \in \{x, y, z\} \) to start not only at the individual name \( a \) (the root of the cube), but also at any other individual name along the corresponding side of the cube.

For \( d \in \{x, y, z\} \), we use a concept name \( \text{Max}_d \) to mark individuals that are at the beginning of the sequence for the \( d \)-counter (specifically: at the last bit of the maximal counter value). This is established using the following axioms:

\[
\text{Max}_d(a) \quad \text{Max}_d \sqsubseteq \bigcap_{e \in \{x, y, z\} \setminus d} \forall r_e. \text{Max}_e \sqcap \forall r_d. \neg \text{Max}_d
\]

\[
\neg \text{Max}_d \sqsubseteq \bigcap_{e \in \{x, y, z\}} \forall r_e. \neg \text{Max}_e
\]

Intuitively, the counter for the coordinate \( d \) should be initialised at all points where \( \text{Max}_d \) is satisfied. However, we cannot initialise the counters directly, but need to use the respective abduction problem. Here, we use that \( \text{Goal}_d(b) \) is only entailed for individual names \( b \) from which every path along \( r_d \) corresponds to a double exponential counter for \( d \). To make sure all counters are initialised at the right positions, we use another concept name \( \text{AllStarted} \) that requires that all reachable individuals that satisfy \( \text{Max}_d \) also satisfy \( \text{Goal}_d \). Recall that every counter for \( d \) reaches its end at a domain element satisfying \( B \).

\[
B_x \sqcup B_y \sqcup B_z \sqsubseteq \text{AllStarted}
\]

\[
\bigcap_{d \in \{x, y, z\}} \bigcap_{e \in \{x, y, z\}} \exists r_d. (\text{AllStarted} \sqcap (\neg \text{Max}_e \sqcup \text{Goal}_e)) \sqsubseteq \text{AllStarted}
\]

The concept name \( \text{Coords} \) now requires all counters to have started:

\[
\text{AllStarted} \sqcap \text{Goal}_x \sqcap \text{Goal}_y \sqcap \text{Goal}_z \sqsubseteq \text{Coords}.
\]

Set \( K_{\text{coord}} \) to be the union of \( K_x, K_y, K_z \) and these new axioms. The abduction problem

\[
\mathcal{A}_{\text{coord}} = \langle K_{\text{coord}}, \text{Coords}(a), \Sigma_x \cup \Sigma_y \cup \Sigma_z \rangle
\]

now produces the required counters in its hypotheses, reachable by the corresponding paths of \( r_x, r_y \) and \( r_z \)-successors.

6 Enforcing the Grid Shape

So far, our abduction problem would still allow for a solution with just a single individual, which would produce the counters using a complex concept. This changes with the next extension, which makes sure that hypotheses need to use a number of fresh individuals that is double exponential in the size of the input, and that those individuals are organised into a three-dimensional grid via the three roles \( r_x, r_y \) and \( r_z \). Rather than
enforcing this grid shape directly, we define a concept to detect whether a hypothesis is not of the desired grid shape, and then require for the observation to be entailed that the root individual is not an instance of this concept.

For every two \(d, e \in \{x, y, z\}\) st. \(d \neq e\), we add the following axiom

\[
\text{NonSquare} \sqsubseteq \exists r_d.\exists r_e. D \land \exists r_e.\exists r_d. \neg D,
\]

NonSquare is only satisfiable by domain elements for which there is a path along \(r_d-r_e\) that ends in a different domain element than a path along \(r_e-r_d\). As in an interpretation, there may be more \(r_d\) and \(r_e\) successors than specified in the ABox, we use the following axioms to make sure NonSquare cannot be satisfied by an individual in an ABox in which the next two \(r_d\) and \(r_e\)-successors form a square.

\[
\exists r_d. D \equiv \forall r_d. D \\
\exists r_d.\exists r_e. D \equiv \forall r_d.\exists r_e. D \\
\exists r_e.\exists r_d. D \equiv \forall r_e.\exists r_d. D
\]

The only way to entail \(\neg\text{NonSquare}(b)\) for an individual \(b\), and without using \(D\) or NonSquare, is now by providing an \(r_d-r_e\)-path and an \(r_e-r_d\)-path that both end on the same individual, and thus by creating one cell of a grid. Next, we define the concept NonGrid to detect appearances of NonSquare anywhere on a path between the root element and the end of our paths marked by the concept names \(B_x\), \(B_y\) and \(B_z\). Specifically, for all \(d \in \{x, y, z\}\), we add:

\[
\begin{align*}
\text{NonGrid} & \sqsubseteq \neg (B_x \sqcup B_y \sqcup B_z) \land (\text{NonSquare} \sqcup \exists r_d.\text{NonGrid}) \\
\exists r_d.\neg\text{NonGrid} & \sqsubseteq \forall r_d.\neg\text{NonGrid}
\end{align*}
\]

The first axiom ensures that, if a domain element is in NonGrid, then there must be a path to some element in NonSquare that does not go through \(B_x \sqcup B_y \sqcup B_z\). Together with the first axiom, the second axiom makes sure that, once one path fails to satisfy NonGrid at some point, all predecessors will fail to do so as well, unless NonSquare is already satisfied. Consequently, to satisfy \(\neg\text{NonGrid}\), we have to ensure that on all paths, \(\neg\text{NonSquare}\) remains satisfied before the paths reach \(B_x \sqcup B_y \sqcup B_z\). The only way to do so is by organising the role successors into a grid as required.

The final axiom to be added is the following:

\[
\text{Coords} \sqsubseteq \text{NonGrid} \sqcup \text{Grid}
\]

Every instance of Coords is either in NonGrid or Grid. To entail Grid\((a)\), we need to entail Coords\((a)\) and \(\neg\text{NonGrid}(a)\), which is only possible by having all coordinates organised into a cube.

Denote the resulting KB with all axioms mentioned until now by \(K_{\text{grid}}\). Our abduction problem for this section is defined as \(\mathcal{A}_{\text{grid}} \sqsubseteq \{K_{\text{grid}}, \text{Grid} \sqcap \Sigma_x \sqcup \Sigma_y \sqcup \Sigma_z\}\). As now every hypothesis needs an individual for every coordinate in the cube, we obtain the following theorem.

**Theorem 4.** There exists an unbounded family of \(\text{ALC}\) ABox abduction problems such that every hypothesis uses a number of fresh individuals that is at least double exponential in the size of the abduction problem.
7 Encoding the Turing machine

It remains to encode the Turing machine to finish the reduction. Let

\[ M_o = \langle Q_o, \Sigma_o, \Gamma_o, \Delta_o, q_{o0}, \hat{b}, F_o \rangle \]

be a non-deterministic Turing machine with states \( Q_o \), input alphabet \( \Sigma_o \), tape alphabet \( \Gamma_o \), transition relation \( \Delta_o \subseteq (Q_o \times \Gamma_o) \times (Q_o \times \Gamma_o \times \{-1, +1\}) \), initial state \( q_{o0} \), blank symbol \( \hat{b} \) and accepting states \( F_o \). We assume that there exists a polynomial \( p \) such that for input words \( w \), \( M_o \) stops after at most \( p(|w|) \) steps. Furthermore, let

\[ M = \langle Q, \Sigma, \Gamma, \delta, q_0, q?, \hat{b}, F, M_o \rangle \]

be a non-deterministic Turing machine that uses \( M_o \) as oracle, where \( Q \) is the set of states, \( \Sigma \) the input alphabet, \( \Gamma \) the tape alphabet, \( \Delta \subseteq (Q \times \Gamma \times \Gamma_o) \to (Q \times \Gamma \times \Gamma_o \times \{-1, +1\}) \) the transition relation that now operates on two tapes (the work tape and the oracle tape), \( q_0 \) the starting state, \( q? \) the oracle query state, \( q- \) the oracle answer state, and \( F \) the set of accepting states. The definition of runs and accepting runs is as usual, however, for convenience, we define it with two differences: 1) our tape head is always on the same position for the work tape and the oracle tape, and 2) when in the oracle state, the Turing machine \( M \) goes into state \( q- \) if the oracle rejects the current content of the oracle tape, and otherwise \( M \) rejects the input (instead of going into a special state reserved for positive query answers). That 1) is without loss of generality can be seen using a standard trick for reducing multi tape machines to single tape machines: specifically, to simulate a different tape head for the oracle tape, we introduce a “marked” tape symbol \( \gamma_T \) for every tape symbol \( \gamma \) in the original TM, and use those marked symbols to mark the virtual position of the head on the oracle tape, and similarly for the work tape. In order to move the two tapes in different positions, we just move back and forth between the cells on the tapes that are marked in this way. This is possible with only quadratic time overhead.

For Difference 2), we use the following argument: assume we have a Turing machine that, based on the result of the oracle, would go either into a state \( q_+ \) (if the oracle accepts) or into a state \( q- \) (if the oracle rejects). We can simulate this behaviour with our type of Turing machine by first guessing the result of the oracle. If we guess for a positive answer, we verify it without the oracle (since the Turing machine is non-deterministic), if we guess for a negative answer, we use the oracle to verify this. We assume that there is a polynomial \( q \) s.t. for words \( w \) on the input, \( M \) stops after at most \( 2^{q(|w|)} \) steps, that is, it accepts a language in \( \text{N2EXP TIME}^{\text{NP}} \). Fix one such input word \( w \). We choose our \( n \) used in the previous sections s.t. \( n \geq p(q(|w|)) \), this way ensuring that \( n \) bounds the length of accepting runs, as well as the maximal length of the tape content, both for \( M \) and the oracle machine \( M_o \). Our reduction will now proceed in such a way that for the resulting abduction problem, every hypothesis will encode a successful run on the side of the cube for which the \( z \)-coordinate is 0: specifically, we will represent the content of the two tapes of \( M \) along the \( x \)-axis, and succeeding configurations along the \( y \)-axis.
For convenience, we assume $Q, \Gamma, Q_o$ and $\Gamma_o$ to be pairwise disjoint, and $Q \cup \Gamma \cup Q_o \cup \Gamma_o \subseteq N_C$, so that we can identify states and tape symbols directly with concept names. We however need to make sure that every domain element can represent at most one state and tape symbol at a time:

$$q \cap q' \subseteq \bot \quad \text{for every } q, q' \in Q, q \neq q'$$ (1)
$$\gamma \cap \gamma' \subseteq \bot \quad \text{for every } \gamma, \gamma' \in \Gamma, \gamma \neq \gamma'$$ (2)
$$\gamma_o \cap \gamma'_o \subseteq \bot \quad \text{for every } \gamma_o, \gamma'_o \in \Gamma_o, \gamma_o \neq \gamma'_o$$ (3)

The input word $w = w_0 \ldots w_m$ is encoded as follows:

$$w_0(a), r_x(a, a_1), w_1(a_1), r_x(a_1, a_2), \ldots, w_m(a_m), r_x(a_m, a_{m+1}), \check{b}(a_{m+1})$$ (4)

To make sure the blank symbol is propagated along the available space on the tape, we use the following TBox axiom. Recall that $\text{Max}_y$ and $\text{Max}_z$ are satisfied on all individuals where the $y$ and $z$-counters are initialised, and $B_x$ at the end of the counting sequence along $r_x$.

$$\check{b} \cap \text{Max}_y \cap \text{Max}_z \cap \neg B_x \subseteq \forall r_x.\check{b}$$ (5)

We represent the current state on every node in the grid, and mark the position of the tape head with a special concept name $\text{Head}$:

$$\text{Head}(a)$$ (6)
$$\text{Max}_y \cap \text{Max}_z \subseteq q_0$$ (7)

This completes the encoding of the initial configuration.

We further make sure that also on the other configurations, every individual along the $x$-axis gets assigned the same state $q \in Q$:

$$q \equiv \forall r_x.q$$ (8)

and that at most one individual satisfies $\text{Head}$. For the latter, we use the concept $\text{LeftOfHead}$ to mark individuals left of the head. $q \in Q$:

$$q \equiv \forall r_x.q$$ (9)

$$(\text{Max}_x \cap \neg \text{Head}) \subseteq \text{LeftOfHead}$$ (10)
$$\text{LeftOfHead} \subseteq \forall r_x(\text{Head} \cup \text{LeftOfHead})$$ (11)
$$\text{Head} \subseteq \neg \text{LeftOfHead} \cap \forall r_x(\neg \text{Head} \cap \neg \text{LeftOfHead})$$ (12)
$$\neg \text{LeftOfHead} \cap \neg \text{Head} \subseteq \forall r_x(\neg \text{Head} \cap \neg \text{LeftOfHead})$$ (13)

The transitions that do not use the oracle are encoded using the following TBox axioms. For every $q \in Q \setminus (F \cup \{q_0\}), \gamma \in \Gamma, \gamma_o \in \Gamma_o$:

$$\exists r_x.(q \cap \text{Head} \cap q \cap q_{-1})$$ (14)
$$\subseteq \bigcup \langle (q, \gamma, q_o), (q', \gamma', q'_o, -1) \rangle \in \Delta \forall r_y.(\text{Head} \cap q' \cap \forall r_x.(\gamma' \cap q'_o))$$ (15)

$$\cup \bigcup \langle (q, \gamma, q_o), (q', \gamma', q'_o, +1) \rangle \in \Delta \forall r_y.\forall r_x.(\gamma' \cap q'_o \cap q' \cap \forall r_x.\text{Head})$$ (16)

$$\gamma \cap \neg \text{Head} \subseteq \forall r_y.\gamma$$ (17)
$$\gamma_o \cap \neg \text{Head} \subseteq \forall r_y.\gamma_o$$ (18)
The following axioms now propagate the acceptance back to the root, so that we can require it for the observation to be entailed. Here, we use a fresh concept name $\text{Accept}$.

\[
\bigcup_{q \in F} q \sqsubseteq \text{Accept} \quad (19)
\]

\[\exists r_y. \text{Accept} \sqsubseteq \text{Accept} \quad (20)\]

\[\exists r_x. \text{Accept} \sqsubseteq \text{Accept} \quad (21)\]

Our signature $\Sigma_{T,w}$ for the final abduction problem is defined as

\[
\Sigma_{T,w} = \Sigma_x \cup \Sigma_y \cup \Sigma_z \cup Q \cup \Gamma \cup \Gamma_o \cup \text{Head} \quad (22)
\]

It remains to encode the oracle. While we want the run of $M$ to be represented in the hypothesis, we cannot do the same for the oracle, as we need to quantify over all possible runs (and ensure that none of them is accepting). We thus use a fresh concept name $\gamma_o^*$ for every $\gamma_o \in \Gamma_o$, since $\Gamma_o$ is already used in $\Sigma_{T,w}$. The following axioms, for every $\gamma_o \in \Gamma_o$, copy the information from the oracle tape to the input tape of the oracle machine, when the oracle is queried:

\[
q \sqcap \gamma_o \sqsubseteq \gamma_o^*. 
\]

Similarly, we use a concept name $\text{Tape}^*$ outside of the signature $\Sigma_{T,w}$ for the tape positions of the oracle machine. The following axiom ensures that the head of the oracle tape is always at the left-most position at the beginning:

\[
\text{Max}_x \sqcap \text{Max}_z \sqsubseteq \text{Tape}^* 
\]

Except for axioms 14–16, the rest is encoded using the same axioms as in 7 – 18, however with the role $r_y$ replaced by $r_z$, $\text{Tape}$ replaced by $\text{Tape}^*$, the tape alphabet now using the names $\gamma_o^*$ instead of $\gamma_o$, and of course all states and transitions now referring to the Turing machine $M_o$.

To detect whether the oracle is rejecting, instead of the axioms 14–16, we represent the transitions as follows for every $q \in Q_o \setminus F_o$ and $\gamma \in \Gamma_o$.

\[
\exists r_x. (q \sqcap \text{Head}^* \sqcap \gamma^*) \sqsubseteq \bigcup_{(q, \gamma), (q', \gamma', -1)} \forall r_x. (\text{Head}^* \sqcap q' \sqcap \forall r_x. (\gamma')^*) \\
\sqcup \bigcup_{(q, \gamma), (q', \gamma', +1)} \forall r_x. \forall r_z. ((\gamma')^* \sqcap q' \sqcap \forall r_x. \text{Head}^*) \\
\sqcup \text{Reject} \quad \\
\exists x. \text{Reject} \sqsubseteq \text{Reject} \\
\exists z. \text{Reject} \sqsubseteq \text{Reject} 
\]

Those state that, if every computation path in $M_o$ ends in a dead end that is not an accepting state, then in every model, $\text{Reject}$ will be satisfied by the domain elements that represent the beginning of the computation.

Finally, we connect both components, the encoding of the Turing machine $M$ and the encoding of the Turing machine $M_o$, by introducing a concept name $\text{OracleChecked}$ that is satisfied if on every configuration between the accepting configuration and the
initial configuration, either there is no oracle call (¬q?), or there is an oracle call (q?), the oracle rejects, and the next state is q−.

\[
\bigcup_{q \in F} q \sqsubseteq \text{OracleChecked}
\]

\[
\neg q? \sqcap \exists r_y.\text{OracleChecked} \sqsubseteq \text{OracleChecked}
\]

\[
q? \sqcap \text{Reject} \sqcap \exists r_y. (\text{OracleChecked} \sqcap q−) \sqsubseteq \text{OracleChecked}
\]

To complete the construction, we add the following GCI:

\[
\text{Accept} \sqcap \text{OracleChecked} \sqcap \text{Grid} \sqsubseteq \text{Observation}
\]

Denote the knowledge base consisting of all axioms shown so far by \( \mathcal{K}_{T,w} \). The final abduction problem is now the tuple \( \mathfrak{A}_{T,w} = (\mathcal{K}_{T,w}, \text{Observation}(a), \Sigma_{T,w}) \).

**Lemma 1.** There is a hypothesis for \( \mathfrak{A}_{T,w} \) iff \( T \) accepts \( w \).

Together with the upper bound from [21], we thus obtain the following theorem.

**Theorem 5.** Signature-based ABox abduction in \( \mathcal{ALC} \) is \( \text{N2ExpTime}^{\text{NP}} \)-complete.

## 8 Conclusion

One conclusion one could draw from the high complexity is that it may make sense to look into more restricted variants of ABox abduction, such as flat ABox abduction, where complex concepts are not allowed in the hypothesis: here hypotheses can get at most exponentially large, and the decision problem (for \( \mathcal{ALC} \)) is only \( \text{coNExpTime} \)-complete [21]. On the other hand, especially in the context of description logics, it is often observed that high theoretical complexity results do not mean that there cannot be implementations that work well in practice: for instance, reasoning in \( \mathcal{SROIQ} \), the DL underlying the web ontology language standard OWL, is \( \text{N2ExpTime} \)-complete [18], while modern optimised DL reasoners process even large OWL ontologies in practice [28]. An example more related to signature-based abduction is uniform interpolation, which for \( \mathcal{ALC} \) may also lead to triple exponentially large ontologies [25], while systems like LETHE [20] and FAME [35] can often successfully compute uniform interpolants of realistic ontologies in practice.

The current prototypes for signature-based abduction for \( \mathcal{ALC} \) [9,24] performed reasonably well in an evaluation on real ontologies, however, they do not introduce fresh individual names. Instead, the more general approach in [24] uses additional DL constructors like inverse roles and nominals to express connections between named individuals. Arguably, the results of abduction would become more user friendly if they would use fresh individual names instead. An approach could be to first compute hypotheses with those systems, and afterwards simplify concepts by introduction of fresh individuals. The results in this paper indicate however that this could turn out challenging in practice, as the number of those individuals may become large in theory.
References


SEH-PILoT: A System for Property-Directed Symbol Elimination – Work in Progress
(Short Paper)

Philipp Marohn and Viorica Sofronie-Stokkermans
University Koblenz-Landau, Koblenz, Germany
{pmarohn,sofronie}@uni-koblenz.de

Abstract. We describe the implementation of a system for property-directed symbol elimination in extensions of a theory $\mathcal{T}_0$ with additional function symbols whose properties are axiomatised using a set of clauses. The system performs the symbol elimination in a hierarchical way, relying on existing mechanisms for symbol elimination in $\mathcal{T}_0$.

1 Introduction

Many reasoning problems in mathematics or program verification can be reduced to checking satisfiability of ground formulae w.r.t. a theory (this can be a standard theory, e.g. linear arithmetic, or a complex theory – e.g. the extension of a base theory with additional function symbols axiomatized by a set of formulae, or a combination of theories). More interesting is to go beyond yes/no answers, i.e. to consider problems – in mathematics or verification – in which the properties of certain function symbols are underspecified (these symbols are considered to be parametric) and (weakest) additional conditions need to be derived under which given properties hold. In [13] a method for property-directed symbol elimination in local theory extensions was proposed which can be used for obtaining such constraints on parameters. The goal of this paper is to present the current state of an implementation of this method in the system SEH-PILoT.

Structure of the paper: In Section 2.2 we first present the theoretical background and then the implementation details. In Section 3 we discuss in detail an example, then present an overview of a (small) subset of the tests we considered so far.

2 Description of the SEH-PILoT Implementation

SEH-PILoT (Symbol Elimination based on Hierarchical Proving In Local Theory extensions) is an implementation of the method for symbol elimination presented in [12,13]. SEH-PILoT is implemented in Python 3.9. Its general structure is presented in Figure 1. Examples which show how SEH-PILoT can be used in various application areas (mathematics, verification, wireless network theory) can be found at https://userpages.uni-koblenz.de/~sofronie/sehpilot/.

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
2.1 Theoretical Background

Let $\Pi_0 = (\Sigma_0, \text{Pred})$ be a signature, and $T_0$ be a “base” theory with signature $\Pi_0$. We consider extensions $T := T_0 \cup K$ of $T_0$ with new function symbols $\Sigma$ (extension functions) whose properties are axiomatized using a set $K$ of (universally closed) clauses in the extended signature $\Pi = (\Sigma_0 \cup \Sigma, \text{Pred})$, such that each clause in $K$ contains function symbols in $\Sigma$. Let $\Sigma_{\text{par}} \subseteq \Sigma$ be a set of parameters. Let $\Sigma_c$ be an additional set of constants.

**Task:** Let $G$ be a set of ground $\Pi \cup \Sigma_c$ clauses. We want to check whether $G$ is satisfiable w.r.t. $T_0 \cup K$ or not and – if it is satisfiable – to generate a weakest universal $\Pi_0 \cup \Sigma_{\text{par}}$-formula $\Gamma$ such that $T_0 \cup K \cup \Gamma \cup G$ is unsatisfiable.

**Method.** For satisfiability checking we use a method for hierarchical reduction to checking satisfiability in the base theory. For symbol elimination (i.e. for determining $\Gamma$) we use a method for hierarchically reducing the problem to a quantifier elimination problem w.r.t. $T_0$. If $T_0$ allows quantifier elimination (i.e. for every formula $\phi$ over $\Pi_0$ there exists a quantifier-free formula $\phi^*$ over $\Pi_0$ which is equivalent to $\phi$ modulo $T_0$) a method for quantifier elimination w.r.t. $T_0$ can be used for this.

In what follows we present situations in which hierarchical reasoning is complete and weakest constraints on parameters can be generated.

**Local Theory Extensions.** Let $\Psi$ be a closure operator on sets of ground terms. A theory extension $T_0 \subseteq T_0 \cup K$ is $\Psi$-local if it satisfies the condition:

$$(\text{Loc}_f^\Psi) \quad \text{For every finite set } G \text{ of ground } \Pi_c \text{-clauses (for an additional set } C \text{ of constants) it holds that } T_0 \cup K \cup G \models \bot \text{ if and only if } T_0 \cup K \cup [\Psi_K(G)] \cup G \text{ is unsatisfiable.}$$

where, for every set $G$ of ground $\Pi_c$-clauses, $K[\Psi_K(G)]$ is the set of instances of $K$ in which the terms starting with a function symbol in $\Sigma$ are in $\Psi_K(G) = \Psi(\text{est}(K, G))$, where $\text{est}(K, G)$ is the set of ground terms starting with a function in $\Sigma$ occurring in $G$ or $K$.

$\Psi$-local extensions can be recognized by showing that certain partial models embed into total ones [11,7]. Especially well-behaved are theory extensions with the property $(\text{Comp}_f^\Psi)$ which requires that every partial model of $T$ whose reduct to $\Pi_0$ is total and the “set of defined terms” is finite and closed under $\Psi$, embeds into a total model of $T$ with the same support (cf. e.g. [5]). If $\Psi$ is the identity, we denote $\text{Loc}_f^\Psi$ by $\text{Loc}_f$ and $\text{Comp}_f^\Psi$ by $\text{Comp}_f$. Examples of local theory extensions can be found in [14].

**Hierarchical Reasoning.** Consider a $\Psi$-local theory extension $T_0 \subseteq T_0 \cup K$. Condition $(\text{Loc}_f^\Psi)$ requires that for every finite set $G$ of ground $\Pi_c$ clauses: $T_0 \cup K \cup G \models \bot$ if and only if $T_0 \cup K \cup \Psi_K(G) \cup G \models \bot$. In all clauses in $K[\Psi_K(G)] \cup G$ the function symbols in $\Sigma$ only have ground terms as arguments, so $K[\Psi_K(G)] \cup G$ can be flattened and purified\(^1\) by introducing, in a bottom-up manner, new

\(^1\) i.e. the function symbols in $\Sigma$ are separated from the other symbols.
constants $c_t \in C$ for subterms $t = f(c_1, \ldots, c_n)$ where $f \in \Sigma$ and $c_i$ are constants, together with definitions $c_t \approx f(c_1, \ldots, c_n)$ which are all included in a set $\text{Def}$. We thus obtain a set of clauses $K_0 \cup G_0 \cup \text{Def}$, where $K_0$ and $G_0$ do not contain $\Sigma$-function symbols and $\text{Def}$ contains clauses of the form $c_t \approx f(c_1, \ldots, c_n)$, where $f \in \Sigma$, $c, c_1, \ldots, c_n$ are constants.

**Theorem 1 ([11,5])** Let $\mathcal{K}$ be a set of clauses. Assume that $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is a $\Psi$-local theory extension. For any finite set $G$ of ground clauses, let $\mathcal{K}_0 \cup G_0 \cup \text{Def}$ be obtained from $\mathcal{K}[\Psi_{\mathcal{K}}(G)] \cup G$ by flattening and purification, as explained above. Then the following are equivalent to $\mathcal{T}_1 \cup G \models \bot$:

1. $\mathcal{T}_0 \cup \mathcal{K}[\Psi_{\mathcal{K}}(G)] \cup G \models \bot$.
2. $\mathcal{T}_0 \cup \mathcal{K}_0 \cup G_0 \cup \text{Con}_0 \models \bot$, where $\text{Con}_0 = \{ \bigwedge_{i=1}^n c_i \approx d_i \rightarrow c \approx d \mid f(c_1, \ldots, c_n) \approx c \in \text{Def} \}$.

We can also consider chains of theory extensions:

$$\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}_1 \subseteq \mathcal{T}_2 = \mathcal{T}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2 \subseteq \cdots \subseteq \mathcal{T}_n = \mathcal{T}_0 \cup \mathcal{K}_1 \cup \ldots \cup \mathcal{K}_n$$

in which each theory is a local extension of the preceding one. For a chain of $n$ local extensions a satisfiability check w.r.t. the last extension can be reduced (in $n$ steps) to a satisfiability check w.r.t. $\mathcal{T}_0$. The only restriction we need to impose in order to ensure that such a reduction is possible is that at each step the clauses reduced so far need to be ground. Groundness is assured if each variable in a clause appears at least once under an extension function. This iterated instantiation procedure for chains of local theory extensions has been implemented in H-PILoT [6].

**Hierarchical Symbol Elimination.** In [13] we proposed a method for property-directed symbol elimination described in Algorithm 1.

**Theorem 2 ([12,13])** Let $\mathcal{T}_0$ be a $\Pi_0$-theory allowing quantifier elimination, $\Sigma_{\text{par}}$ be a set of parameters (function and constant symbols) and $\Sigma$ a set of function symbols such that $\Sigma \cap (\Sigma_0 \cup \Sigma_{\text{par}}) = \emptyset$. Let $\mathcal{K}$ be a set of clauses in the signature $\Pi_0 \cup \Sigma_{\text{par}} \cup \Sigma$ in which all variables occur also below functions in $\Sigma_1 = \Sigma_{\text{par}} \cup \Sigma$. Assume $\mathcal{T} \subseteq \mathcal{T}_0 \cup \mathcal{K}$ satisfies condition ($\text{Comp}_f^\Psi$) for a suitable closure operator $\Psi$. Let $T = \Psi_{\mathcal{K}}(G)$. Then Algorithm 1 yields a universal $\Pi_0 \cup \Sigma_{\text{par}}$-formula $\forall \bar{x}. \Gamma_T(\bar{x})$ such that $\mathcal{T}_0 \cup \forall \bar{x}. \Gamma_T(\bar{x}) \cup \mathcal{K} \cup G \models \bot$ which is entailed by every universal formula $\Gamma$ with $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K} \cup G \models \bot$.

Algorithm 1 yields a formula $\forall \bar{x}. \Gamma_T(\bar{x})$ with $\mathcal{T}_0 \cup \forall \bar{x}. \Gamma_T(\bar{x}) \cup \mathcal{K} \cup G \models \bot$ also if the extension $\mathcal{T} \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is not $\Psi$-local or $T \neq \Psi_{\mathcal{K}}(G)$, but in this case there is no guarantee that $\forall \bar{x}. \Gamma_T(\bar{x})$ is the weakest universal formula with this property. A similar result holds for chains of local theory extensions.

H-PILoT allows the user to specify a chain of extensions by tagging the extension functions with their place in the chain (e.g., if $f$ occurs in $\mathcal{K}_3$ but not in $\mathcal{K}_1 \cup \mathcal{K}_2$ it is declared as level 3).
Algorithm 1 Symbol elimination in theory extensions [12,13]

**Input:** Theory extension $T_0 \cup K$ with signature $\Pi = \Pi_0 \cup (\Sigma \cup \Sigma_{\text{par}})$ where $\Sigma_{\text{par}}$ is a set of parameters
Set $T$ of ground $\Pi^G$-terms

**Output:** $\forall \overline{y}. \Gamma_T(\overline{y})$ (universal $\Pi_0 \cup \Sigma_{\text{par}}$-formula)

**Step 1** Purify $K[T] \cup G$ as described in Theorem 1 (with set of extension symbols $\Sigma_1$). Let $K_0 \cup G_0 \cup \text{Con}_0$ be the set of $\Pi^G_0$-clauses obtained this way.

**Step 2** Let $G_1 = K_0 \cup G_0 \cup \text{Con}_0$. Among the constants in $G_1$, we identify
(i) the constants $c_f, f \in \Sigma_{\text{par}}$, where $c_f$ is a constant parameter or $c_f$ is introduced by a definition $c_f \approx f(c_1, \ldots, c_k)$ in the hierarchical reasoning method,
(ii) all constants $\overline{c}_p$ occurring as arguments of functions in $\Sigma_{\text{par}}$ in such definitions. Replace all the other constants $\overline{c}$ with existentially quantified variables $\overline{x}$ (i.e. replace $G_1(\overline{c}_p, c_f, \overline{c})$ with $\exists \overline{x}. G_1(\overline{c}_p, c_f, \overline{x})$).

**Step 3** Construct a formula $\Gamma_1(\overline{c}_p, \overline{c}_f)$ equivalent to $\exists \overline{x}. G_1(\overline{c}_p, c_f, \overline{x})$ w.r.t. $T_0$ using a method for quantifier elimination in $T_0$ and let $\Gamma_2(\overline{c}_p, \overline{c}_f)$ be $\neg \Gamma_1(\overline{c}_p, \overline{c}_f)$.

**Step 4** Replace (i) each constant $c_f$ introduced by definition $c_f \approx f(c_1, \ldots, c_k)$ with the term $f(c_1, \ldots, c_k)$ and (ii) $\overline{c}_p$ with universally quantified variables $\overline{y}$ in $\Gamma_2(\overline{c}_p, \overline{c}_f)$. The formula obtained this way is $\forall \overline{y}. \Gamma_T(\overline{y})$.

---

**Theorem 3 ([13])** Consider the following chain of theory extensions:

$T_0 \subseteq T_0 \cup K_1 \subseteq T_0 \cup K_1 \cup K_2 \subseteq \ldots \subseteq T_0 \cup K_1 \cup K_2 \cup \cdots \cup K_n$

where every extension in the chain satisfies condition $(\text{Comp}_f)$, $K_i$ are all flat and linear, and in all $K_i$ all variables occur below the extension terms on level $i$.

Let $G$ be a set of ground clauses, and let $G_1$ be the result of the hierarchical reduction of satisfiability of $G$ to a satisfiability test w.r.t. $T_0$. Let $T = T(G)$ be the set of all instances used in the chain of hierarchical reductions and let $\forall y. \Gamma_T(G)(y)$ be the formula obtained by applying Steps 2–4 of Alg. 1 to $G_1$. Then $\forall y. \Gamma_T(G)(y)$ is entailed by every conjunction $\Gamma$ of clauses with the property that $T_0 \cup \Gamma \cup K_1 \cup \cdots \cup K_n \cup G$ is unsatisfiable (i.e. it is the weakest such constraint).

---

**2.2 Implementation**

**Hierarchical reasoning: H-PILoT.** The method for hierarchical reasoning in local theory extensions described before was implemented in the system H-PILoT [6]. H-PILoT carries out a hierarchical reduction to the base theory. Standard SMT provers or specialized provers can be used for testing the satisfiability of the formulae obtained after the reduction. H-PILoT uses eager instantiation and the hierarchical reduction, so provers like CVC4 [1] or Z3 [2] are in general faster in proving unsatisfiability. The advantage of using H-PILoT is that knowing the instances needed for a complete instantiation allows us to correctly detect satisfiability (and generate models) in situations in which e.g. CVC4 returns “unknown”, and use property-directed symbol elimination to obtain additional constraints on parameters which ensure unsatisfiability.
Symbol Elimination: SEH-PILoT (Symbol Elimination with H-PILoT):
For obtaining constraints on parameters we used the method described in Algorithm 1 [13] which was implemented in SEH-PILoT for the case in which the theory can be structured as a local theory extension or a chain of theory extensions and the base theory $T_0$ is the theory of real closed fields.

**Input.** SEH-PILoT receives a list of symbols to be eliminated (and possibly a list of already existing constraints on the parameters) and an input file for H-PILoT$^3$. This file contains (i) the specification of the signature and of the hierarchy of local theory extensions to be considered; (ii) an axiomatization $K$ of the theory extension(s); (iii) a set $G$ of ground clauses possibly containing additional constants. Currently the only supported base theory for SEH-PILoT is the theory of real numbers (the theory of real closed fields).

**Main Algorithm.** SEH-PILoT follows the steps of Algorithm 1.

*Step 1:* SEH-PILoT uses H-PILoT (with option \texttt{-redlog}) for the hierarchical reduction to a problem in the base theory. H-PILoT computes the necessary instances $K[T_G]$, where $T_G$ is the set of ground terms necessary for instantiation (cf. Thm.3), generates the formula $K_0 \cup G_0 \cup \text{Con}_0$ and writes it in a file which can be used as input for Redlog [4].

*Step 2:* Taking into account the function symbols to be eliminated, the constants are classified as required in Step 2 of Alg. 1 and the Redlog file is changed accordingly such that only those symbols that do not correspond to a parameter or argument of a parameter are considered to be existentially quantified.

*Step 3:* SEH-PILoT uses Redlog to eliminate the existentially quantified symbols and afterwards to negate the resulting formula.

*Step 4:* The constants contained in the formula obtained by Step 3 are replaced back with the terms they represent and the constants occurring as arguments are replaced by universally quantified variables. Finally SEH-PILoT translates the generated constraints from the syntax of Redlog back to the syntax of H-PILoT such that they can easily be used for verification or an iterative approach of constraint generation. Since Redlog is not very efficient in simplifying formulae, SLFQ can be used, which allows Redlog to use the possibilities of simplification offered by QEPCAD B [3].

$^3$ A detailed description of the form of such input files can be found in the system description of H-PILoT [6].
3 Examples

We illustrate the way SEH-PILoT works on the following example\(^4\). Consider a discrete water level controller in which the inflow varies during the evolution of the system, and can be modeled by a function \(\text{inflow} : \mathbb{R} \to \mathbb{R}\), where \(\text{inflow}(t)\) is the inflow at time step \(t\). If the water level becomes greater than an alarm level \(L_{\text{alarm}}\) (positioned below the overflow level \(L_{\text{overflow}}\)) a valve is opened and a fixed quantity of water (\(\text{outflow}\)) is left out. Otherwise, the valve remains closed.

Assume that the formula \(\text{Init}(L) := L < L_{\text{overflow}}\) describes the initial states. Then \(L \leq L_{\text{overflow}}\) is an inductive invariant iff the following formulae are unsatisfiable w.r.t. the extension of the theory of real numbers with a function \(\text{inflow}\):

1. \(\exists L. (L < L_{\text{overflow}} \land L > L_{\text{overflow}})\);
2. \(\exists L, L', t, t'. (L \leq L_{\text{overflow}} \land L > L_{\text{alarm}} \land L' \approx L + \text{inflow}(t) - \text{outflow} \land t' \approx t + 1 \land L' > L_{\text{overflow}})\);
3. \(\exists L, L', t, t'. (L \leq L_{\text{overflow}} \land L \leq L_{\text{alarm}} \land L' \approx L + \text{inflow}(t) \land t' \approx t + 1 \land L' > L_{\text{overflow}})\).

We want to obtain conditions on the parameters \((\text{inflow}, \text{outflow}, L_{\text{alarm}}, L_{\text{overflow}})\) under which \(L < L_{\text{overflow}}\) is an inductive invariant. (1) is clearly unsatisfiable. SEH-PILoT (with assumptions \(L_{\text{alarm}} < L_{\text{overflow}} \land \forall x. \text{inflow}(x) > 0\)) generated weakest universal conditions under which (2) resp. (3) are unsatisfiable. Consider e.g. (2). The problem is described in the H-PILoT syntax as follows:

<table>
<thead>
<tr>
<th>Base functions:</th>
<th>{(+,2), (-,2), (*,2)}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension functions:</td>
<td>{(\text{inflow}, 1, 1)}</td>
</tr>
<tr>
<td>Relations:</td>
<td>{(&lt;=, 2), (&lt;, 2), (&gt;=, 2), (&gt;, 2)}</td>
</tr>
<tr>
<td>Clauses:</td>
<td>1 &lt;= \text{overflow} ; 1 &gt; \text{alam}; lp = (1 + \text{inflow}(t)) - \text{outflow} ; tp = t+1 ;</td>
</tr>
<tr>
<td>Query:</td>
<td>lp &gt; \text{overflow} ;</td>
</tr>
</tbody>
</table>

It can be checked that the extension \(\mathbb{R} \subseteq \mathbb{R} \cup \mathcal{K}\) of \(\mathbb{R}\) with a function symbol \(\text{inflow}\) satisfying the axiom \(\mathcal{K} = \forall x. (\text{inflow}(x) > 0)\) defines a local theory extension.

When using SEH-PILoT the user has to specify the name of the H-PILoT file (in this case \text{inv2.loc}) as well as several additional examples can be found under \texttt{https://userpages.uni-koblenz.de/~sofronie/sehpilot/}.

\(^4\) This example as well as several additional examples can be found under \texttt{https://userpages.uni-koblenz.de/~sofronie/sehpilot/}.
SEH-PILoT classifies the constants and updates the Redlog file by simplifying the input and changing `vars` to the set of symbols to be eliminated \(1, l, p, t, p\), eliminates these variables, negates the result and simplifies the resulting formula and obtains \(e_1 - \text{outflow} \leq 0\). It then replaces \(e_1\) with \(\text{inflow}(t)\), quantifies \(t\) universally and obtains the weakest constraint:

\((I_1)\ \ \forall t. (\text{inflow}(t) \leq \text{outflow})\)

for which (2) becomes unsatisfiable. A similar procedure yields the constraint

\((I_2) \ \ \forall t. (\text{lalarm} + \text{inflow}(t) \leq \text{loverflow})\)

for (3).

**Test runs for SEH-PILoT.** We analyzed examples from mathematics, verification and wireless network theory. We used H-PILoT for testing satisfiability of the formulae; if the formulae were satisfiable SEH-PILoT was used for symbol elimination and generating constraints on the parameters. The table below provides some data on the size of the problems we analyzed and the time H-PILoT needed for hierarchical reduction and SEH-PILoT for symbol elimination.

<table>
<thead>
<tr>
<th>Name</th>
<th># clauses</th>
<th>time H-PILoT (s)</th>
<th># atoms (1)</th>
<th># atoms (2)</th>
<th>time QE (ms)</th>
<th># atoms (3)</th>
<th># atoms (4)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mathematics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case distinction</td>
<td>4</td>
<td>0.23</td>
<td>16</td>
<td>12</td>
<td>0.81</td>
<td>34</td>
<td>8</td>
<td>2.5</td>
</tr>
<tr>
<td>Example 5.6 in [13]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lipschitz</td>
<td>12</td>
<td>0.25</td>
<td>64</td>
<td>22</td>
<td>5.26</td>
<td>2925</td>
<td>3</td>
<td>9.5</td>
</tr>
<tr>
<td><strong>Verification</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Water tank, -a</td>
<td>7</td>
<td>0.22</td>
<td>5</td>
<td>5</td>
<td>0.19</td>
<td>2</td>
<td>2</td>
<td>1.7</td>
</tr>
<tr>
<td>without SLFQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Water tank, no -a</td>
<td>6</td>
<td>0.10</td>
<td>6</td>
<td>6</td>
<td>0.08</td>
<td>2</td>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>without SLFQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array sorted Example 4 in [9]</td>
<td>6</td>
<td>0.23</td>
<td>11</td>
<td>8</td>
<td>0.82</td>
<td>2</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>Maximum in array Example 4.13 [14]</td>
<td>7</td>
<td>0.22</td>
<td>11</td>
<td>6</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
<td>3.0</td>
</tr>
<tr>
<td><strong>Networks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph class consist.</td>
<td>3</td>
<td>0.22</td>
<td>3</td>
<td>3</td>
<td>0.79</td>
<td>1</td>
<td>1</td>
<td>2.4</td>
</tr>
<tr>
<td>Example 4 in [10]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class inclusion ((g_4))</td>
<td>19</td>
<td>0.24</td>
<td>111</td>
<td>22</td>
<td>0.90</td>
<td>20</td>
<td>4</td>
<td>2.5</td>
</tr>
<tr>
<td>Class inclusion ((g_5))</td>
<td>19</td>
<td>0.24</td>
<td>114</td>
<td>22</td>
<td>1.01</td>
<td>20</td>
<td>4</td>
<td>2.7</td>
</tr>
<tr>
<td>Example 8 in [10,8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 1.** Run on an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 8192 K-byte cache.

\# clauses is the number of clauses in the input to H-PILoT. \# atoms refer to the number of atoms in the Redlog file generated by H-PILoT before (1) resp. after (2) simplification; resp. in the formula obtained after quantifier elimination before (3) resp. after (4) simplification.

For all examples (with exception of the water tank) simplification with SLFQ was used, which is responsible for a significant amount of the runtime.

---

If SEH-PILoT is called with assumptions (-a) in the command line, redlog-simplification with assumptions is performed and the resulting formula is simpler.
References


Symbol Elimination and Applications to Parametric Entailment Problems

(Abstract)

Dennis Peuter and Viorica Sofronie-Stokkermans

University Koblenz-Landau, Koblenz, Germany
{dpeuter,sofronie}@uni-koblenz.de

Abstract. We analyze possibilities of second-order quantifier elimination for formulae containing parameters – constants or functions. For this, we use a constraint resolution calculus obtained from specializing the hierarchical superposition calculus. If the saturated set of formulae has a finite representation, we analyze possibilities of obtaining weakest constraints on parameters which guarantee satisfiability. We identify situations in which entailment between formulae expressed using second-order quantification can be effectively checked. We illustrate the ideas on an example from wireless network research.

1 Introduction

The motivation for this work was a study of models for graph classes naturally occurring in wireless network research – in which nodes that are close are always connected, nodes that are far apart from each other are never connected and any other node pairs can, but do not need to be connected. Transformations can be applied to such graphs to make them symmetric; this leads to further graph classes. When checking inclusion between graph classes described using transformations we need to check entailment of second-order formulae. If inclusion cannot be proved and the graph class descriptions are parametric we want to obtain (weakest) conditions on these parameters that guarantee that inclusions hold. This can be achieved by eliminating “non-parametric” constants or function symbols used in the description of such classes.

We show that it is possible to combine methods for general symbol elimination (for eliminating existentially quantified predicates) with methods for property-directed symbol elimination (for obtaining conditions on “parameters” under which formulae are satisfiable or second-order entailment holds). For general second-order quantifier elimination we use a form of ordered resolution similar to that proposed in [11]. For property-directed symbol elimination we use a

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
method we proposed in [24]. The advantage of using such a two-layered approach is that it avoids non-termination that might occur if using only general symbol elimination methods. The main application area we consider in this paper is the analysis of inclusions between graph classes arising in wireless network research. The study of second-order quantifier elimination goes back to [7,1,2]. Most of its known applications are in the study of modal logics or knowledge representation [12,13]. In [11], Gabbay and Ohlbach proposed a resolution-based algorithm for second-order quantifier elimination which is implemented in the system SCAN. In [4], Bachmair et al. mention that hierarchical superposition (cf. [5,6] for further refinements) can be used for second-order quantifier elimination modulo a theory. In [19,14], Hoder et al. study possibilities of symbol elimination in inference systems (e.g. the superposition calculus and its extension with ground linear rational arithmetic and uninterpreted functions). Since the saturated sets might be infinite, possibilities of finding finite representations were investigated in the context of superposition [15] or in verification, in relationship to acceleration [9,10,3,8]. Orthogonal to this direction of study is the “property-directed” symbol elimination: Given a theory \( T \) and a ground formula \( G \) satisfiable w.r.t. \( T \), the goal is to derive a (weakest) universal formula \( \Gamma \) over a subset of the signature, such that \( \Gamma \land G \) is unsatisfiable w.r.t. \( T \). In [24] we proved that this is possible for local extensions of theories allowing quantifier elimination.

Structure of the paper. In Sect. 2 we introduce the theoretical results needed in the paper. In Sect. 3 we present an ordered hierarchical resolution calculus, \( \text{HRes}_P^{\succ} \), which can be used for eliminating predicate \( P \), and mention possibilities of giving finite representations for infinite saturated sets and of investigating the satisfiability of the saturated sets. In Sect. 4 we use these ideas for checking entailment and illustrate the method with an example.

This is an abstract containing results published in [21]. Details of the proofs and additional examples can be found in [20] (which is the extended version of [21]).

2 Local Extensions; Hierarchical Symbol Elimination

We assume known the basic notions in (many-sorted) first-order logic. We consider signatures of the form \( \Pi = (S, \Sigma, \text{Pred}) \), where \( S \) is a set of sorts, \( \Sigma \) is a family of function symbols and \( \text{Pred} \) a family of predicate symbols, such that for every function symbol \( f \) (resp. predicate symbol \( p \)) their arity \( a(f) = s_1 \ldots s_n \to s \) (resp. \( a(p) = s_1 \ldots s_m \)), where \( s_1, \ldots, s_n, s \in S \), is specified. If \( C \) is a fixed countable set of fresh constants, we denote by \( \Pi^C \) the extension of \( \Pi \) with constants in \( C \). A \( \Pi \)-structure \( A \) is a tuple \( (\{A_s\}_{s \in S}, \{f_A\}_{f \in \Sigma}, \{p_A\}_{p \in \text{Pred}}) \) where for every function symbol \( f \) with arity \( a(f) = s_1 \ldots s_n \to s \), \( f_A : A_{s_1} \times \ldots \times A_{s_n} \to A_s \), and for every predicate symbol \( p \) with \( a(p) = s_1 \ldots s_m \), \( p_A \subseteq A_{s_1} \times \ldots \times A_{s_m} \). If \( \Pi \subseteq \Pi' \) and \( A \) is a \( \Pi' \)-structure, we denote its reduct to \( \Pi \) by \( A\mid_{\Pi} \).

A theory \( T \) can be defined by specifying a set of axioms, or by specifying a class of structures (the models of the theory). If \( F \) and \( G \) are formulae we write \( F \models G \) (resp. \( F \models_T G \) – also written as \( \mathcal{T} \cup F \models G \)) to express the fact that
every model of $F$ (resp. every model of $F$ which is also a model of $T$) is a model of $G$. We write $F \models \bot$ (resp. $F \models T \bot$) to express the fact that $F$ has no model (resp. that there is no model of $T$ which is a model of $F$). A theory $T$ over a signature $\Pi$ allows quantifier elimination (QE) if for every formula $\phi$ over $\Pi$ there exists a quantifier-free formula $\phi^*$ over $\Pi$ which is equivalent to $\phi$ modulo $T$. Examples of theories which allow quantifier elimination are rational and real linear arithmetic ($\text{LI}(\mathbb{Q})$, $\text{LI}(\mathbb{R})$) and the theory of real closed fields.

Let $\Pi_0 = (\Sigma_0, \text{Pred})$ be a signature, and $T_0$ be a “base” theory with signature $\Pi_0$. We consider extensions $T := T_0 \cup \mathcal{K}$ of $T_0$ with new function symbols $\Sigma$ (extension functions) whose properties are axiomatized using a set $\mathcal{K}$ of (universally closed) clauses in the extended signature $\Pi = (\Sigma_0 \cup \Sigma, \text{Pred})$, such that each clause in $\mathcal{K}$ contains function symbols in $\Sigma$. Especially well-behaved are the $\Psi$-local theory extensions, where $\Psi$ is a suitable closure operator (for details on the properties of the closure operators we consider we refer to [18]). $\Psi$-local theory extensions are extensions $T_0 \subseteq T_0 \cup \mathcal{K}$ satisfying the following condition:

$$(\text{Loc}_f^\Psi) \quad \text{For every finite set } G \text{ of ground } \Pi^C-\text{clauses (for an additional set } C \text{ of constants) it holds that } T_0 \cup \mathcal{K} \cup G \models \bot \text{ if and only if } T_0 \cup \mathcal{K}[\Psi_{\mathcal{K}}(G)] \cup G \text{ is unsatisfiable.}$$

where, for every set $G$ of ground $\Pi^C$-clauses, $\mathcal{K}[\Psi_{\mathcal{K}}(G)]$ is the set of instances of $\mathcal{K}$ in which the terms starting with a function symbol in $\Sigma$ are in $\Psi_{\mathcal{K}}(G) = \Psi(\text{est}(\mathcal{K}, G))$, where $\text{est}(\mathcal{K}, G)$ is the set of ground terms starting with a function in $G$ or $\mathcal{K}$. $\Psi$-local extensions can be recognized by showing that certain partial models embed into total ones [22,18]. Especially well-behaved are theory extensions with the property (Comp$_f^\Psi$) which requires that every partial model of $T$ whose reduct to $\Pi_0$ is total and the “set of defined terms” is finite and closed under $\Psi$ embeds into a total model of $T$ with the same support (cf. [16,18]). If $\Psi$ is the identity, we denote $\text{Loc}_f^\Psi$ by $\text{Loc}_f$ and $\text{Comp}_f^\Psi$ by $\text{Comp}_f$. In ($\Psi$)-local theory extensions hierarchical reasoning is possible. If the base theory allows quantifier elimination, a form of property-directed symbol elimination is also possible: the symbol elimination problem is hierarchically reduced to a quantifier elimination problem w.r.t. the base theory.

Theorem 1 ([23,24]) Let $T_0$ be a $\Pi_0$-theory allowing quantifier elimination, $\Sigma_{\text{par}}$ be a set of parameters (function and constant symbols) and $\Pi = (S, \Sigma, \text{Pred})$ be such that $\Sigma \cap (\Sigma_0 \cup \Sigma_{\text{par}}) = \emptyset$. Let $\mathcal{K}$ be a set of clauses in the signature $\Pi_0 \cup \Sigma_{\text{par}} \cup \Sigma$ in which all variables occur also below functions in $\Sigma_1 = \Sigma_{\text{par}} \cup \Sigma$. Assume $T \subseteq T_0 \cup \mathcal{K}$ satisfies condition (Comp$_f^\Psi$) for a suitable closure operator $\Psi$ with $\text{est}(G) \subseteq \Psi_{\mathcal{K}}(G)$ for every set $G$ of ground $\Pi^C$-clauses. Then Algorithm 1 can be used to construct a universal $\Pi_0 \cup \Sigma_{\text{par}}$-formula $\forall \vec{x} \Gamma_T(\vec{x})$ such that $T_0 \cup \forall \vec{x} \Gamma_T(\vec{x}) \cup \mathcal{K} \cup G \models \bot$ which is entailed by every universal formula $\Gamma$ with $T_0 \cup \Gamma \cup \mathcal{K} \cup G \models \bot$. 

Algorithm 1 Symbol elimination in theory extensions [23, 24]

Input: Theory extension \( T_0 \cup K \) with signature \( \Pi = \Pi_0 \cup \Sigma \), where \( \Sigma_1 = \Sigma \cup \Sigma_{\text{par}} \) and \( \Sigma_{\text{par}} \) is a set of parameters

\( G \): set of ground \( \Pi^C \)-clauses; \( T \): set of ground \( \Pi^C \)-terms with \( \Psi(K)(G) \subset T \)

Output: \( \forall \overline{y} \Gamma_T(\overline{y}) \) (universal \( \Pi_0 \cup \Sigma_{\text{par}} \)-formula)

Step 1 Purify \( K[T] \cup G \) (with set of extension symbols \( \Sigma_1 \)). Let \( K_0 \cup G_0 \cup \text{Con}_0 \) be the set of \( \Pi^C \)-clauses obtained this way.

Step 2 Let \( G_1 = K_0 \cup G_0 \cup \text{Con}_0 \). Among the constants in \( G_1 \), we identify

(i) the constants \( c_f, f \in \Sigma_{\text{par}} \), where \( c_f \) is a constant parameter or \( c_f \) is introduced by a definition \( c_f \approx f(c_1, \ldots, c_k) \) in the hierarchical reasoning method,

(ii) all constants \( \tau_p \) occurring as arguments of functions in \( \Sigma_{\text{par}} \) in such definitions.

Replace all the other constants \( \tau \) with existentially quantified variables \( \overline{\tau} \) (i.e. replace \( G_1(\tau_p, \tau_f, \tau) \) with \( \exists \overline{\tau} G_1(\tau_p, \tau_f, \overline{\tau}) \)).

Step 3 Construct a formula \( \Gamma_1(\tau_p, \tau_f) \) equivalent to \( \exists \overline{\tau} G_1(\tau_p, \tau_f, \overline{\tau}) \) w.r.t. \( \Gamma_0 \), using a method for quantifier elimination in \( \Gamma_0 \).

Step 4 Replace each constant \( c_f \) introduced by definition \( c_f = f(c_1, \ldots, c_k) \) with the term \( f(c_1, \ldots, c_k) \) in \( \Gamma_1(\tau_p, \tau_f) \). Let \( \Gamma_2(\tau_p) \) be the formula obtained this way. Replace \( \tau_p \) with existentially quantified variables \( \overline{\tau} \).

Step 5 Let \( \forall \overline{y} \Gamma_T(\overline{y}) \) be \( \forall \overline{y} \neg \Gamma_2(\overline{y}) \).

3 Second-Order Quantifier Elimination

We consider only the elimination of one predicate; for formulae of the form \( \exists P_1 \ldots P_n F \) the process can be iterated.

Let \( \mathcal{T} \) be a theory over a many-sorted signature \( \Pi = (S, \Sigma, \text{Pred}) \) where the set of sorts \( S = S_i \cup S_u \) consists of a set \( S_i \) of interpreted sorts and a set \( S_u \) of uninterpreted sorts. The models of the theories are \( \Pi \)-structures \( \mathcal{A} = (\{A_s\}_{s \in S}, \{f_A\}_{f \in \Sigma}, \{p_A\}_{p \in \text{Pred}}) \), where each support of interpreted sort is considered to be fixed. Following the terminology used in [5, 6], we will refer to elements in the fixed domain of sort \( s \in S_i \) as domain elements of sort \( s \).

Let \( \Pi' = (S, \Sigma, \text{Pred} \cup \{P\}) \), where \( P \not\in \text{Pred} \). Let \( F \) be a universal first-order \( \Pi' \)-formula. We can assume, without loss of generality, that \( F \) is a set of clauses of the form \( \forall \overline{x} D(\overline{x}) \lor C(\overline{x}) \), where \( D(\overline{x}) \) is a clause over the signature \( \Pi \) and \( C(\overline{x}) \) is a clause containing literals of the form \( \neg P(x_1, \ldots, x_n) \), where \( x_1, \ldots, x_n \) are variables\(^1\). Such clauses can also be represented as constrained clauses in the form \( \forall \overline{x} \phi(\overline{x}) \| C(\overline{x}) \), where \( \phi(\overline{x}) := \neg D(\overline{x}) \). We will refer to clauses of this form as constrained \( \Pi \)-clauses.

Our goal is to compute, if possible, a first-order \( \Pi \)-formula \( G \) such that \( G \equiv_{\mathcal{T}} \exists P F \).

Let \( \succ \) be a strict, well-founded ordering on terms that is compatible with contexts and stable under substitutions, total on ground terms and with the property that

\(^1\) We can bring the clauses to this form using variable abstraction.
Let $t \succ d$ for every domain element $d$ of interpreted sort $s$ and every ground term $t$ that is not a domain element. Let $HRes^P_\prec$ be the calculus containing the following ordered resolution and factorization rules for constrained $P$-clauses:

$\phi_1 \parallel P(\bar{x}) \lor C \quad \phi_2 \parallel \neg P(\bar{y}) \lor D$

$(\phi_1 \land \phi_2)\sigma \parallel (C \lor D)\sigma$

$\phi \parallel P(\bar{x}) \lor P(\bar{y}) \lor C$

$(\phi \sigma) \parallel (P(\bar{x}) \lor C)\sigma$

where (i) $\sigma = \text{mgu}(P(\bar{x}), P(\bar{y}))$
(ii) $P(\bar{x})\sigma$ is strictly maximal in $(P(\bar{x}) \lor C)\sigma$
(iii) $\neg P(\bar{y})\sigma$ is maximal in $(\neg P(\bar{y}) \lor D)\sigma$.

The inference rules are supplemented by a redundancy criterion $\mathcal{R} = (\mathcal{R}_c, \mathcal{R}_i)$ meant to specify a set $\mathcal{R}_c$ of redundant clauses (which can be removed) and a set $\mathcal{R}_i$ of redundant inferences (which do not need to be computed). The following notion of redundancy $\mathcal{R}_c^0$ for clauses is often used: A (constrained) clause is redundant w.r.t. a set $N$ of clauses if all its ground instances are entailed w.r.t. $\mathcal{T}$ by ground instances of clauses in $N$ which are strictly smaller w.r.t. $\succ$. We will use the following notion of redundancy for inferences: If $\mathcal{R}_c$ is a redundancy criterion for clauses, we say that an inference $\iota$ on ground clauses is redundant w.r.t. $N$ if either one of its premises is redundant w.r.t. $N$ and $\mathcal{R}_c$ or, if $C_0$ is the conclusion of $\iota$ then there exist clauses $C_1, \ldots, C_n \in N$ that are smaller w.r.t. $\succ$ than the maximal premise of $\iota$ and $C_1, \ldots, C_n \models C_0$. A non-ground inference is redundant if all its ground instances are redundant.

We say that a set of clauses $N^*$ is saturated up to $\mathcal{R}$-redundancy w.r.t. $HRes^P_\prec$ if every $HRes^P_\prec$ inference with premises in $N^*$ is redundant.

**Theorem 2** ([4, 21]) Let $N$ be a set of constrained $P$-clauses over background theory $\mathcal{T}$, $N^*$ its saturation (up to $\mathcal{R}$-redundancy) under $HRes^P_\prec$, and $N^*_0$ the set of clauses in $N^*$ not containing $P$. For every model $\mathcal{A}$ of $\mathcal{T}$, $\mathcal{A}$ is a model of $N^*_0$ iff there exists a $\Pi^1$-structure $\mathcal{B}$ with $\mathcal{B} \models N$ and $\mathcal{B}_{\Pi^1} = \mathcal{A}$.

If the saturation $N^*$ of $N$ under $HRes^P_\prec$ (up to $\mathcal{R}$-redundancy) is finite, the universal closure of the conjunction of the clauses in $N^*_0$ is equivalent to $\exists P \, N$.

**Theorem 3** ([21]) Let $\mathcal{T}$ be a theory with signature $\Pi = (S, \Sigma, \text{Pred})$, $N$ a set of constrained $P$-clauses. Assume that the saturation $N^*$ of $N$ (up to $\mathcal{R}$-redundancy) w.r.t. $HRes^P_\prec$ is finite; let $N^*_0$ be the set of clauses in $N^*$ not containing $P$. Let $\Sigma_{\text{par}} \subseteq \Sigma$ be a set of parameters. If (i) $\mathcal{T}$ allows quantifier elimination or (ii) $\mathcal{T}_0 \subseteq \mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$ is a local theory extension satisfying condition $(\text{Comp}^F_\psi)$ and $\mathcal{T}_0$ allows quantifier elimination, then we can use Algorithm 1 to obtain a (weakest) universal constraint $\Gamma$ on the parameters such that every model $\mathcal{A}$ of $\mathcal{T} \cup \Gamma$ is a model of (the universal closure of) $N^*_0$, hence $\mathcal{A} \models \exists P \, N$.

Since the implementations of the hierarchical superposition calculus we are aware of have as background theory linear arithmetic and in our examples we had more complex theories, in [21] and [20] we used a form of abstraction first: We renamed the constraints over more complex theories with new predicate symbols, and used SCAN [11] for second-order quantifier elimination.
The saturation of a set $N$ of constrained $P$-clauses up to redundancy under \textit{HResP} $\succ$ might be infinite. In [21] we discuss two possibilities of obtaining finite representations for it: using an encoding of the constraints as minimal models of suitable sets of constrained Horn clauses [8] or using acceleration [9,10].

4 Checking Entailment

Let $\mathcal{T}$ be a theory with signature $\Pi = (S, \Sigma, \text{Pred})$, and let $\mathcal{P}_1 = P^1_1, \ldots, P^n_1$ and $\mathcal{P}_2 = P^2_1, \ldots, P^2_{n_2}$ be finite sequences of different predicate symbols with $P^j_1 \notin \text{Pred}$, and $\Pi_i = \left( \Sigma, \text{Pred} \cup \{P^j_i \mid 1 \leq j \leq n_i \} \right)$ for $i = 1, 2$.

Let $F_1$ be a universal $\Pi_1$-formula and $F_2$ be a universal $\Pi_2$-formula. If there exist $\Pi$-formulae $G_1$ and $G_2$ such that $G_1 \equiv \exists \mathcal{P}_1 F_1$ and $G_2 \equiv \exists \mathcal{P}_2 F_2$ (which can be found either by saturation or by using acceleration techniques or other methods) then $\exists \mathcal{P}_1 F_1 \models \exists \mathcal{P}_2 F_2$ iff $G_1 \models \exists \mathcal{P}_2 F_2$ (which is the case iff $G_1 \land \neg G_2 \models \bot$). Below are some situations in which this can be effectively decided.

**Theorem 4** ([21]) Assume that there exist $\Pi$-formulae $G_1$ and $G_2$ such that $G_1 \equiv \exists \mathcal{P}_1 F_1$ and $G_2 \equiv \exists \mathcal{P}_2 F_2$. If $\mathcal{T}$ is a decidable theory then we can effectively check whether $\exists \mathcal{P}_1 F_1 \models \exists \mathcal{P}_2 F_2$. If $\mathcal{T}$ has quantifier elimination and the formulae $F_1, F_2$ contain parametric constants, we can use quantifier elimination in $\mathcal{T}$ to derive conditions on these parameters under which $\exists \mathcal{P}_1 F_1 \models \exists \mathcal{P}_2 F_2$.

**Theorem 5** ([21]) Assume that there exist universal $\Pi$-formulae $G_1$ and $G_2$ such that $G_1 \equiv \exists \mathcal{P}_1 F_1$ and $G_2 \equiv \exists \mathcal{P}_2 F_2$, and that $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$, where $\mathcal{T}_0$ is a decidable theory with signature $\Pi_0 = (S_0, \Sigma_0, \text{Pred}_0)$ where $S_0$ is a set of interpreted sorts and $\mathcal{K}$ is a set of (universally quantified) clauses over $\Pi = (S_0 \cup S_1, \Sigma_0 \cup \Sigma_1, \text{Pred}_0 \cup \text{Pred}_1)$, where (i) $S_1$ is a new set of uninterpreted sorts, (ii) $\Sigma_1, \text{Pred}_1$ are sets of new function, resp. predicate symbols which have only arguments of uninterpreted sort $\in S_1$, and all function symbols in $\Sigma_1$ have interpreted output sort $\in S_0$. Assume, in addition, that all variables and constants of sort $\in S_1$ in $\mathcal{K}, G_1$ and $\neg G_2$ occur below function symbols in $\Sigma_1$.

We can use the decision procedure for $\mathcal{T}_0$ to effectively check whether $G_1 \land \neg G_2 \models \bot$ (hence whether $\exists \mathcal{P}_1 F_1 \models \exists \mathcal{P}_2 F_2$). If $\mathcal{T}_0$ allows quantifier elimination and the formulae $F_1, F_2$ (hence also $G_1, G_2$) contain parametric constants and functions, we can use the property-directed symbol elimination in Algorithm 1 (cf. Theorem 1) for obtaining a universal formula $\Gamma$ representing weakest universal constraints on the parameters under which $\exists \mathcal{P}_1 F_1 \models \exists \mathcal{P}_2 F_2$.

We illustrate how the previous results can be used for checking an inclusion between two classes of graphs of interest in wireless network theory.

**Example 1.** Let $\text{QUDG}(r) = (\text{MinDG}(r) \cap \text{MaxDG}(1))^{-} \text{ be axiomatized by } \text{MinDG}(r) \land \text{MaxDG}(1) \land \text{Tr}^{-}(E, F)$, where $r$ is a function symbol (where $r(v)$ models the maximum communication distance of node $v$), and:
MinDG(r) : \( \forall x, y \pi^i_r(x, y) \rightarrow E(x, y) \) where \( \pi^i_r(x, y) = x \neq y \land d(x, y) \leq r(x) \)
MaxDG(1) : \( \forall x, y \pi^e(x, y) \rightarrow \neg E(x, y) \) where \( \pi^e(x, y) = d(x, y) > 1 \)
Tr\(^{-}(E, F)\) : \( \forall x, y \ (F(x, y) \leftrightarrow E(x, y) \land E(y, x)) \)
Tr\(^{+}(E, F)\) : \( \forall x, y \ (F(x, y) \leftrightarrow E(x, y) \lor E(y, x)) \).

We want to check whether \( A(r) \subseteq B(r) \), where \( A(r) = \text{QUDG}(r) \) and \( B(r) = (\text{MinDG}(r) \cap \text{MaxDG}(1))^+ \) is described by \( \text{MinDG}(r) \land \text{MaxDG}(1) \land \text{Tr}^{-}(E, F) \).

We obtain axiomatizations \( G_1 \equiv \exists E(\text{MinDG}(r) \land \text{MaxDG}(1) \land \text{Tr}^{-}(E, F)) \) for \( A(r) \) and \( G_2 \equiv \exists E(\text{MinDG}(r) \land \text{MaxDG}(1) \land \text{Tr}^{+}(E, F)) \) for \( B(r) \) by eliminating \( E \). As mentioned before, the implementations of the hierarchical superposition calculus we are aware of allow only linear arithmetic as a background theory, whereas in our examples we had more complex theories. This is why we renamed the constraints over more complex theories with new predicate symbols \( \pi^i, \pi^e, \pi^t \) and used SCAN [11] for second-order quantifier elimination in first-order logic.

We obtained the following formulae:

\[
\begin{array}{ll}
G_1 & G_2 \\
\forall x, y \pi^i_r(x, y) \land \pi^e(x, y) \rightarrow \bot & \forall x, y \pi^i_r(x, y) \land \pi^e(x, y) \rightarrow \bot \\
\forall x, y \pi^i_r(x, y) \land \pi^i_r(y, x) \rightarrow F(y, x) & \forall x, y \pi^i_r(x, y) \land \pi^e(y, x) \rightarrow \neg F(y, x) \\
\forall x, y \pi^e(x, y) \rightarrow \neg F(x, y) & \forall x, y \pi^i_r(x, y) \rightarrow F(x, y) \\
\forall x, y \pi^e(x, y) \rightarrow \neg F(y, x) & \forall x, y \pi^i_r(x, y) \rightarrow F(y, x) \\
\forall x, y \ (F(x, y) \rightarrow F(y, x)) & \forall x, y \pi^e(x, y) \rightarrow \neg F(y, x) \\
\end{array}
\]

The task is now to check whether \( G_1 \models G_2 \), i.e. whether \( G_1 \land \neg G_2 \) is unsatisfiable w.r.t. \( \mathcal{T} \), where \( \neg G_2 \) is the disjunction of the following ground formulae (we ignore the negation of the first clause obviously implied by \( G_1 \)).

\[
\begin{align*}
(g_1) \ & \pi^e(a, b) \land \pi^e(b, a) \land F(b, a) \\
(g_2) \ & \pi^e(a, a) \land F(a, a) \\
(g_3) \ & F(a, b) \land \neg F(b, a) \\
(g_4) \ & \pi^i_r(a, b) \land \neg F(a, b) \\
(g_5) \ & \pi^i_r(a, b) \land \neg F(b, a)
\end{align*}
\]

Here \( \mathcal{T} = \mathcal{T}_d \cup \mathcal{UIF}_r \), where \( \mathcal{T}_d \) is a theory describing the properties of \( d \) and \( r \) is considered to be an uninterpreted function symbol. In [21] we analyzed the situations in which \( \mathcal{T}_d \) is one of the theories \( \mathcal{T}_d^m = \mathcal{T}_0 \cup \mathcal{K}_m \), where \( \mathcal{K}_m \) are axioms of a metric, \( \mathcal{T}_d^s \), the extension of \( \mathcal{T}_0 \) with a function \( d \) satisfying symmetry, \( \mathcal{T}_d^p = \mathcal{T}_0 \cup \mathcal{K}_p \), where \( \mathcal{K}_p = \forall x, y \ d(x, y) \geq 0 \), and \( \mathcal{T}_d^n \), the extension of \( \mathcal{T}_0 \) with an uninterpreted function \( d \) – where \( \mathcal{T}_0 \) is the disjoint combination of the theory \( \mathcal{E} \) of pure equality (sort \( p \)) and linear real arithmetic (sort \( \text{num} \)).

In [21] we proved that all these theories satisfy suitable locality properties. For testing entailment, by Theorem 5, we can consider the set of all instances of \( G_1 \) in which the variables of sort \( p \) are replaced with the constants \( a, b \), then use a method for checking ground satisfiability of \( G_1[T] \land g_i \) w.r.t. \( \mathcal{T}_d \cup \mathcal{UIF}_r \), where \( \mathcal{T}_d \in \{ \mathcal{T}_d^m, \mathcal{T}_d^s, \mathcal{T}_d^p, \mathcal{T}_d^n \} \). For this, we use H-PLoT [17]. This allows us to check that \( G_1[T] \land g_i \) is unsatisfiable for \( i \in \{ 1, 2, 3 \} \), but satisfiable for \( i \in \{ 4, 5 \} \) (this is so for all four theories). For cases 4 and 5 we can use Algorithm 1 to derive conditions on parameters under which \( G_1[T] \land g_i \) is unsatisfiable. If e.g. we consider \( d \) and \( r \) to be parameters, for \( \mathcal{T}_d^m \) we obtain condition

---

\(^2\) To check that the inclusion holds in one given model \( A \) we can choose \( \mathcal{T} = \text{Th}(A) \).
\[ C^{d,r} = \forall x, y(x \neq y \land d(x, y) \leq 1 \land d(x, y) \leq r(x) \rightarrow d(y, x) \leq r(y) \] (which is true in any model of \( T \) in which \( r \) is interpreted as a constant function). For further details cf. [20].

**Acknowledgments:** We thank Hannes Frey and Lucas Bölz for the numerous discussions we had on the problems in wireless network research, Renate Schmidt for maintaining a website where one can run SCAN online and for sending us the executables and instructions for running them.

**References**


The Yoneda Reduction of Polymorphic Types

(Abstract)*  **

Paolo Pistone¹ and Luca Tranchini²

¹ DISI, Università di Bologna, Mura Anteo Zamboni,7, I-40126, Bologna
² Wilhelm-Schickard-Institut, Universität Tübingen, Sand 13, D-72076, Tübingen
paolo.pistone2@unibo.it
luca.tranchini@gmail.com

Abstract. We explore a family of type isomorphisms in System F whose validity corresponds, semantically, to some form of the Yoneda isomorphism from category theory. These isomorphisms hold under theories of equivalence stronger than βη-equivalence, like those induced by parametricity and dinaturality. We show that the Yoneda type isomorphisms yield a rewriting over types, that we call Yoneda reduction, which can be used to eliminate quantifiers from a polymorphic type, replacing them with a combination of monomorphic type constructors. We establish some sufficient conditions under which quantifiers can be fully eliminated from a polymorphic type, and we show some application of these conditions to count the inhabitants of a type and to compute program equivalence in some fragments of System F.

Keywords: Type isomorphism · Yoneda Lemma · Propositional quantification.

1 Introduction

The study of type isomorphisms is a fundamental one both in the theory of programming languages and in logic, through the well-known proofs-as-programs correspondence: type isomorphisms supply programmers with transformations allowing them to obtain simpler and more optimized code, and offer new insights to understand and refine the syntax of type- and proof-systems.

* This extended abstract is an excerpt from the long version of a paper which the authors have recently published in the proceedings of CSL 21 [21]. The long version of the paper (available at https://arxiv.org/abs/1907.03481) contains more detailed proofs and some additional results compared to [21].

** We thank the anonymous reviewers for interesting insights, and in particular for pointing us to a similarity between the type isomorphisms studied in this paper and a result known as Ackermann’s Lemma in the field of second-order quantifier elimination (see [11], section 6).

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
Roughly speaking, two types $A, B$ are isomorphic when one can transform any call by a program to an object of type $A$ into a call to an object of type $B$, without altering the behavior of the program. Thus, type isomorphisms are tightly related to theories of program equivalence, which describe what counts as the observable behavior of a program, so that programs with the same behavior can be considered equivalent.

The connection between type isomorphisms and program equivalence is of special importance for polymorphic type systems like System F (hereafter $\Lambda_2$). In fact, while standard $\beta\eta$-equivalence for $\Lambda_2$ and the related isomorphisms are well-understood [7,8], stronger notions of equivalence (as those based on parametricity or free theorems [29,16,1]) are often more useful in practice but are generally intractable or difficult to compute, and little is known about the type isomorphisms holding under such theories.

A cornerstone result of category theory, the Yoneda lemma, is sometimes invoked [3,13,5,27,14] to justify some type isomorphisms in $\Lambda_2$ like e.g.

$$\forall X. (A \Rightarrow X) \Rightarrow (B \Rightarrow X) \equiv B \Rightarrow A \quad \forall X. (X \Rightarrow A) \Rightarrow (X \Rightarrow B) \equiv A \Rightarrow B \quad \text{(\star)}$$

which do not hold under $\beta\eta$-equivalence, but only under stronger equivalences. Such isomorphisms are usually justified by reference to the interpretation of polymorphic programs as (di)natural transformations [3], a well-known semantics of $\Lambda_2$ related to both parametricity [23] and free-theorems [28], and yielding a not yet well-understood equational theory over the programs of $\Lambda_2$ [12,17,20], that we call here the $\varepsilon$-theory. Other isomorphisms, like those in Fig. 1, can be justified in a similar way as soon as the language of $\Lambda_2$ is enriched with other type constructors like $1, 0, +, \times, \Rightarrow$ and least/greatest fixpoints $\mu X.A, \nu X.A$.

All such type isomorphisms have the effect of eliminating a quantifier, replacing it with a combination of monomorphic type constructors, and can be used to test if a polymorphic type has a finite number of inhabitants (as illustrated in Fig. 2) or, as suggested in [5], to devise decidable tests for program equivalence.

In this paper we develop a formal study of the elimination of quantifiers from polymorphic types using a class of type isomorphisms, that we will call Yoneda type isomorphisms, which generalize the examples above. Then, we explore the application of such type isomorphisms to establish properties of program equivalence for polymorphic programs.

2 A Type-Rewriting Theory of Polymorphic Types

In the first part of the paper we investigate the type-rewriting arising from Yoneda type isomorphisms and its connection with proof-theoretic techniques to count type inhabitants.

2.1 Counting type inhabitants with type isomorphisms.

Examples like the one in Fig. 2 suggest that, while arising from a categorical reading of polymorphic programs, Yoneda Type Isomorphisms have a proof-theoretic
Fig. 1: Other examples of Yoneda type isomorphisms, where $X$ only occurs positively in $A, B, C$ and only occurs negatively in $D$.

2.2 Eliminating Quantifiers with Yoneda Reduction

We then turn to investigate in a more systematic way the quantifier-eliminating rewriting over types arising from the left-to-right orientation of Yoneda type isomorphisms. A major obstacle here is that the rewriting must take into account possible applications of $\beta\eta$-isomorphisms, whose axiomatization is well-known to be challenging in presence of the constructors $+, 0$ [10,15] (not to say of $\mu, \nu$). For this reason we introduce a family of rewrite rules, that we call Yoneda reduction, defined not directly over types but over a class of finite trees which represent the types of $\Lambda_2$ (but crucially not those made with $0, +, \ldots$) up to $\beta\eta$-isomorphism.

Using this rewriting we establish some sufficient conditions for eliminating quantifiers, based on elementary graph-theoretic properties of such trees, which in turn provide some new sufficient conditions for the finiteness of type inhabitants of polymorphic types. First, we prove quantifier-elimination for the types satisfying a certain coherence condition which can be seen as an instance of the 2-SAT problem. We then introduce a more refined condition by associating each polymorphic type $A$ with a value $\kappa(A) \in \{0, 1, \infty\}$, that we call the characteristic of $A$, so that whenever $\kappa(A) \neq \infty$, $A$ rewrites into a monomorphic type, and when furthermore $\kappa(A) = 0$, $A$ converges to a finite type. In the last case our method provides an effective way to count the inhabitants of $A$.

The computation of $\kappa(A)$ is somehow reminiscent of linear logic proof-nets, as it is obtained by inspecting the existence of cyclic paths in a graph obtained by adding some “axiom-links” to the tree-representation of $A$. 

\[
\forall X. X \Rightarrow X \Rightarrow A \equiv A[X \mapsto 1 + 1] \quad (*)
\]
\[
\forall X. (A \Rightarrow X) \Rightarrow (B \Rightarrow X) \Rightarrow C \equiv C[X \mapsto \mu X.A + B] \quad (**) 
\]
\[
\forall X. (X \Rightarrow A) \Rightarrow (X \Rightarrow B) \Rightarrow D \equiv D[X \mapsto \nu X.A \times B] \quad (***) 
\]

① ∀Y.(∀Z.(Z ⇒ 1 + 1) ⇒ (∀W.(W ⇒ Z) ⇒ W ⇒ 1 + 1) ⇒ Z ⇒ Y) ⇒ (1 + 1 ⇒ Y) ⇒ Y

② ∀Y.(∀Z.(Z ⇒ 1 + 1) ⇒ (Z ⇒ 1 + 1) ⇒ Z ⇒ Y) ⇒ (1 + 1 ⇒ Y) ⇒ Y

③ Y.((μZ.(1 + 1) × (1 + 1)) ⇒ Y) ⇒ (1 + 1 ⇒ Y) ⇒ Y

④ μY.(μZ.(1 + 1) × (1 + 1)) + (1 + 1) = 1 + 1 + 1 + 1 + 1 + 1

Fig. 2: Short proof that a Λ2-type has 6 inhabitants, using type isomorphisms.

3 Program Equivalence in System F with Finite Characteristic

In the second part of the paper we direct our attention to programs rather than types, and we exploit our results on type isomorphisms to establish some non-trivial properties of program equivalence for polymorphic programs in some suitable fragments of Λ2.

3.1 Computing equivalence with type isomorphisms

Computing program equivalence under the ε-theory can be a challenging task, as this theory involves global permutations of rules which are difficult to detect and apply [17,20,26,22,28]. Things are even worse at the semantic level, since computing with dinatural transformations can be rather cumbersome, due to the well-known fact that such transformations need not compose [3,18].

Nevertheless, our approach to quantifier-elimination based on the notion of characteristic provides a way to compute program equivalence without the appeal to ε-rules, free theorems and parametricity, since all polymorphic programs having types of finite characteristic can be embedded inside well-known monomorphic systems. To demonstrate this fact, we introduce two fragments Λ2κ≤0 and Λ2κ≤1 of Λ2 in which types have a fixed finite characteristic, and we prove that these are equivalent, under the ε-theory, respectively, to the simply typed λ-calculus with finite products and co-products (or, equivalently, to the free bicartesian closed category B), and to its extension with μ, ν-types (that is, to the free cartesian closed μ-bicomplete category μB [24,4]). Using well-known facts about B and μB [25,4,19], we finally establish that the ε-theory is decidable in Λ2κ≤0 and undecidable in Λ2κ≤1.

3.2 Program equivalence and predicativity

We provide an example of how the correspondence between polymorphic types of finite characteristic and μ, ν-types can be used to prove non-trivial properties of program equivalence. A main source of difficulty with Λ2 is that polymorphic programs are impredicative, that is, a program of universal type ∀X.A can be instantiated at any type B, yielding a program of type A[B/X]. It is thus useful to be able to predict when a complex type instantiation can be replaced by a one of smaller complexity, without altering the program behavior.
Using the fact that a universal type $\forall X.A$ of finite characteristic in which $A$ is of the form $A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow X$ is isomorphic to the initial algebra $\mu X.T$ of some appropriate functor $T$, we establish a sufficient condition under which a program containing an instantiation of $\forall X.A$ as $A[B/X]$ can be transformed into one with instantiations of types strictly less complex than $B$.

We finally use this condition to provide a simpler proof of a result from [22], showing that all programs in a certain fragment of $\Lambda^{\omega<\omega}$ (the fragment freely generated by the embedding of finite sums and products) can be transformed into predicative programs only containing atomic type instantiations, a result related to some recent investigations on atomic polymorphism [9].

References

16. Johann, P.: On proving the correctness of program transformation based on free
theorems for higher-order polymorphic calculi. Mathematical Structures in Com-
puter Science 15, 201–229 (2005)

17. de Lataillade, J.: Dinatural terms in System F. In: Proceedings of the Twenty-

fuer Informatik, Dagstuhl, Germany (2018)

and Applications of Categories 6, 47–64 (1999)

20. Pistone, P.: On dinaturality, typability and $\beta\eta$-stable models. In: fuer Informatik,
S.D.L.Z. (ed.) 2nd International Conference on Formal Structures for Compu-
tation and Deduction (FSCD 2017). Leibniz International Proceedings in Informatics

puter Science Logic 2021 (CSL 2021). LIPIcs–Leibniz International Proceedings in

On atomic polymorphism and generalized propositional connectives. Studia Logica

ternational Conference on Typed Lambda Calculi and Applications. Lecture Notes

(2002)

of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages.


27. Uustalu, T., Vene, V.: The Recursion Scheme from the Cofree Recursive Comonad.

28. Voigtländer, J.: Free theorems simply, via dinaturality. In: Declarative Program-
ming and Knowledge Management. pp. 247–267. Springer International Publishing,
Cham (2020)

29. Wadler, P.: Theorems for free! In: Proceedings of the fourth international confer-
ence on functional programming languages and computer architecture - FPCA ’89
(1989)

Applying Second-Order Quantifier Elimination in Inspecting Gödel’s Ontological Proof

Christoph Wernhard

University of Potsdam, Germany

Abstract. In recent years, Gödel’s ontological proof and variations of it were formalized and analyzed with automated tools in various ways. We supplement these analyses with a modeling in an automated environment based on first-order logic extended by predicate quantification. Formula macros are used to structure complex formulas and tasks. The analysis is presented as a generated type-set document where informal explanations are interspersed with pretty-printed formulas and outputs of reasoners for first-order theorem proving and second-order quantifier elimination. Previously unnoticed or obscured aspects and details of Gödel’s proof become apparent. Practical application possibilities of second-order quantifier elimination are shown and the encountered elimination tasks may serve as benchmarks.

1 Introduction

Kurt Gödel’s ontological proof is bequeathed in hand-written notes by himself [14] and by Dana Scott [25]. Since transcriptions of these notes were published in 1987 [26], the proof was analyzed, formalized and varied in many different logical settings.\(^1\) Books by John Howard Sobel [27] and Melvin Fitting [13] include comprehensive discussions. Branden Fitelson, Paul E. Oppenheimer and Edward N. Zalta [12,24] investigated various metaphysical arguments in an automated first-order setting based on Prover9 [20]. Christoph Benzmüller and Bruno Woltzenlogel Paleo [7] initiated in 2014 the investigation of Gödel’s argument with automated systems, which led to a large number of follow-up studies concerning its verification with automated tools and the human-readable yet formal representation, e.g., [8,16,6,17,5]. Here we add to these lines of work an inspection of Gödel’s proof in a logical setting that so far has not been considered for this purpose. The expectation is that further, previously unnoticed or obscured aspects and details of the proof become apparent. The used framework, PIE (“Proving, Interpolating, Eliminating”) [29,30] embeds automated reasoners, in particular for first-order theorem proving and second-order quantifier elimination, in a system for defining formula macros and rendering \(\LaTeX\)-formatted

\(^1\) Recently discovered further sources by Gödel are presented in [15].

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
presentations of formula macro definitions and reasoner outputs. In fact, the present paper is the generated output of such a PIE document.  

The target logic of the macros is second-order logic, or, more precisely classical first-order logic extended by predicate quantifiers. The macro layer and the formulas obtained as expansions can be strictly separated. In our modeling of Gödel’s proof we proceed by expressing large-scale steps (axioms, theorems) with macros whose relationships are verified by invocations of embedded reasoners. In this sense our formalization of may be considered as semi-automated.

Aside of providing further material for the study of Gödel’s proof, the work shows possibilities of applying second-order quantifier elimination in a practical system. It appears that the functionality of the macro mechanism is necessary to express nontrivial applications on the basis of first- and second-order logic. The elimination problems that suggested themselves in the course of the investigation, some of which could not be solved by the current version of PIE, may be useful as benchmarks for implemented elimination systems.  

The rest of the paper is structured as follows: After introducing preliminaries in Sect. 2, Gödel’s proof in the version of Scott is developed in Sect. 3. An approach to obtain the weakest sufficient precondition on the accessibility relation for Gödel’s proof with second-order quantifier elimination is then discussed in Sect. 4. Section 5 concludes the paper. Supplementary material is provided in the report version [31] of the paper.

## 2 Preliminaries

A PIE document is a Prolog source file that contains declarative formula macro definitions and specifications of reasoner invocations, interspersed with \LaTeX comments in the manner of literate programming [18]. The PIE processor expands the formula macros, invokes the reasoners, and compiles a \LaTeX document where the formula macro definitions and the results of reasoner invocations are pretty-printed. Alternatively, the processor’s functionality is accessible from Prolog, via the interpreter and in programs. The overall processing time for the present paper, including reasoner invocations and \LaTeX processing to produce a PDF, is about 2.5 seconds.

Formula macros without parameters can play the role of formula names. Expressions with macros expand into formulas of first-order logic extended with predicate quantifiers. Hence, some means of expression that would naturally be used in a higher-order logic formalization of Gödel’s proof are not available in the expansion results. Specifically, predicates in argument position are not permitted and there is no abstraction mechanism to construct predicates from formulas.

---

2 The PIE source of this paper is available at http://cs.christophwernhard.com/pie.

3 Another recent system for second-order quantifier elimination on the basis of first-order logic is DLS-Forgetter [2], which, like the implementation in PIE, is based on the DLS algorithm [10]. An older resolution-based system, SCAN, [22,23] can currently be invoked via a Web interface.
However, these higher-order features are in Gödel’s proof actually only required with respect to specific instances that can be expressed in first-order logic.

As embedded reasoners we used the first-order theorem provers Prover9 \[20\] and CMPProver \[9,29\], the first-order model generator Mace4 \[20\], and an implementation \[29,30\] of the DLS algorithm \[10\] for second-order quantifier elimination, which is based on Ackermann’s Lemma \[1\]. Reasoner outputs computed during processing of the PIE document are presented with the introductory phrases This formula is valid; This formula is not valid; and Result of elimination: In addition, various methods for formula simplification, clausification and un-Skolemization are applied in preprocessing, inprocessing and for output presentation.

The generated \LaTeX{} presentation of formulas and macro definitions bears some footprint inherited from the Prolog syntax that is used to write formulas in PIE documents. As in Prolog, predicate and constant symbols are written in lower case. Macro parameters and bound logical variables that are to be instantiated with fresh symbols at macro expansion are printed like Prolog variables with a capitalized initial. Where-clauses in macro definitions are used to display in abstracted form auxiliary Prolog code executed at macro expansion.

We write formulas of modal predicate logic as formulas of classical first-order logic with one additional free world variable \(v\) by applying the standard translation from \[4, \text{Sec. 11.4}\] (see also \[3, \text{Chap. XII}\]), which can be defined as

\[
\begin{align*}
ST(P(t_1, \ldots, t_n)) & \overset{\text{def}}{=} P(v, t_1, \ldots, t_n) \\
ST(\neg F) & \overset{\text{def}}{=} \neg ST(F) \\
ST(F \lor G) & \overset{\text{def}}{=} ST(F) \lor ST(G) \\
ST(\exists x F) & \overset{\text{def}}{=} \exists x (e(v, x) \land ST(F)) \\
ST(\Box F) & \overset{\text{def}}{=} \exists w (r(v, w) \land \bigwedge_{i \text{ s.th. } x_i \text{ free in } F} e(w, x_i) \land ST(F)\{v \mapsto w\})
\end{align*}
\]

An \(n\)-ary predicate \(P\) in the modal logic is translated into an \(n+1\)-ary predicate, where the first argument represents a world. The binary predicates \(r\) and \(e\) are used for world accessibility and membership in the domain of a world. The logic operators \(\land, \rightarrow, \leftrightarrow, \forall, \Box\) can be understood as shorthands defined in terms of the shown operators. As target logic we neither use a two-sorted logic nor encode two-sortedness explicitly with relativizer predicates. However, the translation of modal formulas yields formulas in which all quantifications are relativized by \(r\) or by \(e\), which seems to subsume the effect of such relativizer predicates. To express that free individual symbols are of sort world we use the unary predicate world. Macro 4, defined below, can be used as an axiom that relates world and \(r\) as far as needed for our purposes. The standard translation realizes with respect to the represented modal logic varying domain semantics (actualist notion of quantification), expressed with the existence predicate \(e\). Constant domain semantics (possibilist notion of quantification) can be achieved with axioms that state domain increase and decrease.

As technical basis for Gödel’s proof we use the presentation of Scott’s version \[25\] in \[7, \text{Fig. 1}\], shown here as Fig. 1. The identifiers \(A1–A5, T1–T3, D1–D3\) and \(C\) of the involved axioms, theorems, definitions and corollary follow \[7\]. In addition, Lemma \(L\) is taken from \[6, \text{Fig. 1}\], where it is appears as \(L2\).
A1 Either a property or its negation is positive, but not both
\[ \forall P (\text{Pos}(\neg P) \leftrightarrow \neg\text{Pos}(P)) \]

A2 A property necessarily implied by a positive property is positive
\[ \forall P \forall Q ((\text{Pos}(P) \land \Box \forall x (P(x) \rightarrow Q(x))) \rightarrow \text{Pos}(Q)) \]

T1 Positive properties are possibly exemplified
\[ \forall P (\text{Pos}(P) \rightarrow \Diamond \exists x P(x)) \]

D1 A God-like being possesses all positive properties
\[ G(x) \leftrightarrow \forall P (\text{Pos}(P) \rightarrow P(x)) \]

A3 The property of being God-like is positive
\[ \text{Pos}(G) \]

C Possibly, God exists
\[ \Diamond \exists x G(x) \]

A4 Positive properties are necessarily positive
\[ \forall P (\text{Pos}(P) \rightarrow \Box \text{Pos}(P)) \]

D2 An essence of an individual is a property possessed by it and necessarily implying any of its properties
\[ \text{Ess}(P, x) \leftrightarrow P(x) \land \forall Q (Q(x) \rightarrow \Box \forall y (P(y) \rightarrow Q(y))) \]

T2 Being God-like is an essence of any God-like being
\[ \forall x (G(x) \rightarrow \text{Ess}(G, x)) \]

D3 Necessary existence of an individual is the necessary exemplification of all its essences
\[ \text{NE}(x) \leftrightarrow \forall P (\text{Ess}(P, x) \rightarrow \Box \exists y P(y)) \]

A5 Necessary existence is a positive property
\[ \text{Pos}(\text{NE}) \]

L If a God-like being exists, then necessarily a God-like being exists
\[ \exists x G(x) \rightarrow \Box \exists x G(x) \]

T3 Necessarily, God exists
\[ \Box \exists x G(x) \]

Fig. 1. Scott’s version of Gödel’s ontological argument [25], adapted from [7,6].

3 Rendering Gödel’s Ontological Proof

3.1 Positiveness – Proving Theorem T1

The first two axioms in Gödel’s proof, A1 and A2, are about positiveness of properties. Theorem T1 follows from them. The following macros render the left-to-right direction of A1 and A2, respectively.

Macro 1 $ax_\rightarrow(V, P)$ is defined as
\[ \text{world}(V) \rightarrow (\text{pos}(V, N') \rightarrow \neg\text{pos}(V, P')), \]
where
\[ N' := \neg\bar{P}, \]
\[ P' := \bar{P}. \]

Is positive is represented here by the binary predicate pos, which has a world and an individual representing a predicate as argument. $P'$ and $N'$ are individual constants that represent a supplied predicate $P$ and its complement $\lambda v x. \neg P(v, x)$, respectively. The where clause specifies that at macro expansion they are replaced by individual constants $\bar{P}$ and $\neg\bar{P}$, available for each predicate symbol $P$. 
Throughout this analysis, we expose the current world as macro parameter $V$, which facilitates identifying proofs steps where an axiom is not just applied with respect to the initially given current world but to some other reachable world.

**Macro 2** $ax_2(V, P, Q)$ is defined as

$$\text{world}(V) \quad \rightarrow \quad (\text{pos}(V, P') \quad \land \quad \forall W (r(V, W) \quad \rightarrow \quad \forall X (e(W, X) \rightarrow (P(W, X) \rightarrow Q(W, X)))) \quad \rightarrow \quad \text{pos}(V, Q')).$$

where

$$P' := \dot{P}, \quad Q' := \dot{Q}.$$

As an insight-providing intermediate step for proving $T1$ we can now derive the following lemma using just a single instance of each of $ax_1^T$ and $ax_2$, where verum ($\lambda x.x = x$) and falsum ($\lambda x.x \neq x$) are represented as binary predicates $\top$ and $\bot$, whose first argument is a world.

**Macro 3** $\text{lemma}_1(V)$ is defined as

$$\text{world}(V) \rightarrow \neg \text{pos}(V, \dot{\bot}).$$

**Macro 4** $r\_world_1$ is defined as $\forall v \forall w (r(v, w) \rightarrow \text{world}(w)).$

**Macro 5** $\text{topbot\_def}$ is defined as

$$\forall v \forall x (\text{world}(v) \rightarrow (\top(v, x) \leftrightarrow e(v, x))) \quad \land \quad \forall v \forall x (\text{world}(v) \rightarrow (\bot(v, x) \leftrightarrow \neg e(v, x))).$$

To express the precondition for $\text{lemma}_1$ we need some auxiliary macros concerning $\top$ and $\bot$. The following expresses equivalence of $\top$ and $\bot$, which first argument is a world.

**Macro 6** $\text{topbot\_equiv}$ is defined as $\forall v \forall x (\text{world}(v) \rightarrow (\top(v, x) \leftrightarrow \neg \bot(v, x))).$

This formula is valid: $\text{topbot\_def} \rightarrow \text{topbot\_equiv}$.

The constants $\dot{\top}$, $\dot{\bot}$ and $\neg \dot{\top}$ designate the individuals associated with $\top$, $\bot$ and $\lambda x.\neg \top(v, x)$, respectively. The following axiom leads from the equivalence expressed by Macro 6 to equality of the associated individuals.

**Macro 7** $\text{topbot\_equiv\_equal}$ is defined as $\text{topbot\_equiv} \rightarrow \dot{\bot} = \neg \dot{\top}$.

Equality is understood there with respect to first-order logic, not qualified by a world parameter. In [31] alternatives are shown, where equality is replaced by a weaker substitutivity property. We can now give the precondition for $\text{lemma}_1$.

**Macro 8** $\text{pre\_lemma}_1(V)$ is defined as

$$\text{r\_world}_1 \quad \land \quad \text{topbot\_def} \quad \land \quad \text{topbot\_equiv\_equal} \quad \land \quad ax_1^T(V, \top) \quad \land \quad ax_2(V, \bot, \top).$$

This formula is valid: $\text{pre\_lemma}_1(v) \rightarrow \text{lemma}_1(v)$.

$T1$ can be rendered by the following macro with a predicate parameter.
Macro 9 \( \text{thm}_1(V, P) \) is defined as
\[
\text{world}(V) \to (\text{pos}(V, P') \to \exists W (r(V, W) \land \exists X (e(W, X) \land P(W, X)))),
\]
where \( P' := \dot{P} \).

Macro 10 \( \text{pre}_{\text{thm}}_1(V, P) \) is defined as \( \text{lemma}_1(V) \land \text{ax}_2(V, P, \bot) \).

This formula is valid: \( \text{pre}_{\text{thm}}_1(v, p) \to \text{thm}_1(v, p) \).

Instances of \( \text{thm}_1(V, P) \) can be proven for arbitrary worlds \( V \) and predicates \( P \), from the respective instance of the precondition \( \text{pre}_{\text{thm}}_1(V, P) \). A further instance of \( \text{ax}_2 \) – beyond that used to prove \( \text{lemma}_1 \) – is required there, with respect to \( \bot \) and the given predicate \( P \).

3.2 Possibly, God Exists – Proving Corollary C

Axiom \( \text{A3} \) and \( \text{T1} \) instantiated by \( \text{God-like} \) together imply corollary \( \text{C} \). This is rendered as follows, where \( \text{God-like} \) is represented by \( g \).

Macro 11 \( \text{ax}_3(V) \) is defined as
\[
\text{world}(V) \to \text{pos}(V, g).
\]

Macro 12 \( \text{coro}(V) \) is defined as
\[
\text{world}(V) \to \exists W (r(V, W) \land \exists X (e(W, X) \land g(W, X)));
\]

Macro 13 \( \text{pre}_{\text{coro}}(V) \) is defined as \( \text{thm}_1(V, g) \land \text{ax}_3(V) \).

This formula is valid: \( \text{pre}_{\text{coro}}(v) \to \text{coro}(v) \).

Notice that, differently from the proofs reported in [7, Fig. 2], \( \text{C} \), represented here by \( \text{coro} \), can be proven independently from the definition of \( \text{God-like} \), \( \text{D1} \), which is represented here by the Macros \( \text{def}^{\rightarrow}_1 \) and \( \text{def}^{\rightarrow\neg}_1 \) defined below.

3.3 Essence – Proving Theorem T2

With macros \( \text{def}^{\rightarrow}_1 \) and \( \text{def}^{\rightarrow\neg}_1 \), defined now, we represent the left-to-right direction of \( \text{D1} \). Actually, only this direction of \( \text{D1} \) is required for the proving the further theorems.

Macro 14 \( \text{def}^{\rightarrow}_1(V, X, P) \) is defined as
\[
g(V, X) \to (\text{pos}(V, P') \to P(V, X)),
\]
where \( P' := \dot{P} \).

Macro 15 \( \text{def}^{\rightarrow\neg}_1(V, X, P) \) is defined as
\[
g(V, X) \to (\text{pos}(V, P') \to \neg P(V, X)),
\]
where \( P' := \neg \dot{P} \).

The following macro \( \text{val} \_\text{ess} \) renders the definiens of the \textit{essence} of relationship between a predicate and an individual in \( \text{D2} \). It is originally a formula with predicate quantification, but without application of a predicate to a predicate. The macro \( \text{val} \_\text{ess} \) exposes the universally quantified predicate as parameter \( Q \), permitting to use it instantiated with some specific predicate.
Macro 16 \( \text{val}_\text{ess}(V,P,X,Q) \) is defined as
\[
P(V,X) \land (Q(V,X) \rightarrow \forall W (r(V,W) \rightarrow \forall Y (e(W,Y) \rightarrow (P(W,Y) \rightarrow Q(W,Y))))).
\]
The universally quantified version of \( \text{val}_\text{ess} \) can be be expressed by prefixing a predicate quantifier upon \( Q \). Eliminating this second-order quantifier shows another view on \( \text{essence} \).

Input: \( \forall q \text{val}_\text{ess}(v,p,x,q) \).

Result of elimination:
\[
p(v,x) \land \forall y \forall z (e(y,z) \land p(y,z) \land r(v,y) \rightarrow y = v) \land \\
\forall y \forall z (e(y,z) \land p(y,z) \land r(v,y) \rightarrow z = x).
\]
We convert the elimination result “manually” to a more clear form and prove equivalence by referencing to the “last result” via a macro.

Macro 17 \( \text{last}_\text{result} \) is defined as \( F \),
where \( \text{last}_\text{ppl}_\text{result}(F) \).

This formula is valid: \( p(v,x) \land \forall w (r(v,w) \rightarrow \forall y (e(w,y) \rightarrow (p(w,y) \rightarrow w = v \land y = x))) \leftrightarrow \text{last}_\text{result} \).

The following definition now renders \( \text{D2} \) as definition of the predicate \( \text{ess} \) in terms of \( \text{val}_\text{ess} \).

Macro 18 \( \text{def}_2(V,P) \) is defined as
\[
\text{world}(V) \rightarrow \forall X (\text{ess}(V,P',X) \leftrightarrow \forall Q \text{val}_\text{ess}(V,P,X,Q)),
\]
where \( P' := \hat{P} \).

In [31] it is shown that two observations about \( \text{essence} \) mentioned as \( \text{NOTE} \) in Scott’s version [25] of Gödel’s proof can be derived in this modeling.

The following two macros render the right-to-left direction of axioms \( \text{A1} \) and \( \text{A4} \). The original axioms involve a universally quantified predicate that appears only in argument role. In the macros, it is represented by the parameter \( P \).

Macro 19 \( ax_1^{-}(V,P) \) is defined as
\[
\text{world}(V) \rightarrow (\neg \text{pos}(V,P') \rightarrow \text{pos}(V,N')),
\]
where \( N' := \neg \hat{P} \),
\( P' := \hat{P} \).

Macro 20 \( ax_4(V,P) \) is defined as
\[
\text{world}(V) \rightarrow (\text{pos}(V,P') \rightarrow \forall W (r(V,W) \rightarrow \text{pos}(W,P'))),
\]
where \( P' := \hat{P} \).

Theorem \( \text{T2} \) is rendered by the following macro with \( \text{ess} \) unfolded, which permits expansion into a universal second-order formula without occurrence of a predicate in argument position.
Macro 21 \( \text{proto\_thm}_2(V, X) \) is defined as

\[
\text{world}(V) \to (e(V, X) \to (g(V, X) \to \forall Q \text{val\_ess}(V, g, X, Q))).
\]

Macro 22 \( \text{pre\_proto\_thm}_2(V, X, Q) \) is defined as

\[
\begin{align*}
ax_1 & (V, Q) \\
\forall W (r(V, W) & \to \forall X (e(W, X) \to def_1(W, X, Q))) \\
\text{def} & (V, X, Q) \\
ax_4 & (V, Q).
\end{align*}
\]

This formula is valid: \( \forall q \exists q \exists \neg q \text{pre\_proto\_thm}_2(v, x, q) \to \text{proto\_thm}_2(v, x) \).
In this implication on the left side the constants \( q \) and \( \neg q \), which represent predicates \( q \) and \( \lambda vx. \neg q(v, x) \) in argument positions, are existentially quantified.

3.4 Necessarily, God Exists – Proving Theorem T3

The definiens of necessary existence, which is defined in Definition D3, is rendered here by the following macro \( \text{val\_ne} \), expressed in terms of \( \text{val\_ess} \), the representation of the definiens of essence, to avoid the occurrence of a predicate representative in argument position.

Macro 23 \( \text{val\_ne}(V, X) \) is defined as

\[
\forall P (\forall Q \text{val\_ess}(V, P, X, Q)) \to \\
\forall W (r(V, W) \to \exists Y (e(W, Y) \wedge P(W, Y))).
\]

Eliminating the quantified predicates shows another view on necessary existence.

Input: \( \text{val\_ne}(v, x) \).

Result of elimination:

\[
\forall y (r(v, y) \to y = v) \wedge \forall y (r(v, y) \to e(y, x)).
\]

The elimination result can be brought into a more clear form.

This formula is valid: \( \forall w (r(v, w) \to w = v \wedge e(w, x)) \leftrightarrow \text{last\_result} \).

In analogy to the definition of the predicate \( \text{ess} \) in Macro 18 we define the predicate \( \text{ne} \) in terms of \( \text{val\_ne} \).

Macro 24 \( \text{def}_3(V, X) \) is defined as

\[
\text{world}(V) \to (e(V, X) \to (\text{ne}(V, X) \leftrightarrow \text{val\_ne}(V, X))).
\]

The following formula renders a fragment of the definition of necessary existence on a “shallow” level, that is, in terms of just the predicates \( \text{ess} \) and \( \text{ne} \), without referring to their definiens \( \text{val\_ess} \) and \( \text{val\_ne} \).

Macro 25 \( \text{def}_3^{-1}(V, X, P) \) is defined as

\[
\begin{align*}
\text{world}(V) & \to (e(V, X) \\
(\text{ne}(V, X) & \to \text{ess}(V, P', X) \\
\forall W (r(V, W) & \to \exists Y (e(W, Y) \wedge P(W, Y))))),
\end{align*}
\]

where \( P' \equiv \hat{P} \).
Correctness of $\def_{3}^{-}$ can be established by showing that it follows from the definitions of $\text{ess}$ and $\text{ne}$.

**This formula is valid:** $\def_{3}(v, p) \land \def_{3}(v, x) \rightarrow \def_{3}^{-}(v, x, p)$.

The following macro renders $\text{T2}$, in contrast to Macro 21 now expressed in terms of the predicate $\text{ess}$ instead of its definiens $\text{val}_{ess}$.

**Macro 26** $\text{thm}_{2}(V, X)$ is defined as

$$\text{world}(V) \rightarrow (\exists X (e(V, X) \land g(V, X)) \rightarrow \forall W (r(V, W) \rightarrow \exists Y (e(W, Y) \land g(W, Y)))).$$

Axiom A5 ($\pos(\text{ne})$) is represented as follows.

**Macro 27** $\text{ax}_{5}(V)$ is defined as

$$\text{world}(V) \rightarrow \pos(V, \text{ne}).$$

Scott’s version [25] shows theorem $\text{T3}$ via the lemma $\text{L}$, rendered as follows.

**Macro 28** $\text{lemma}_{2}(V)$ is defined as

$$\text{world}(V) \rightarrow (\exists X (e(V, X) \land g(V, X)) \rightarrow \forall W (r(V, W) \rightarrow \exists Y (e(W, Y) \land g(W, Y)))).$$

**Macro 29** $\text{pre}_{.}\text{lemma}_{2}(V, X)$ is defined as

$$\text{ax}_{5}(V) \land \def_{3}^{-}(V, X, \text{ne}) \land \def_{3}^{-}(V, X, g) \land \text{thm}_{2}(V, X).$$

**This formula is valid:** $\forall v (\forall x \text{pre}_{.}\text{lemma}_{2}(v, x) \rightarrow \text{lemma}_{2}(v)).$

The following formula states theorem $\text{T3}$, the overall result to show.

**Macro 30** $\text{thm}_{3}(V)$ is defined as

$$\text{world}(V) \rightarrow \forall W (r(V, W) \rightarrow \exists Y (e(W, Y) \land g(W, Y))).$$

**Macro 31** $\text{pre}_{.}\text{thm}_{3}(V)$ is defined as $\text{r}_{.}\text{world}_{1} \land \forall v \text{lemma}_{2}(v) \land \text{coro}(V)$.

**Macro 32** $\text{euclidean}$ is defined as $\forall x \forall y \forall z (r(x, y) \land r(x, z) \rightarrow r(z, y))$.

**Macro 33** $\text{symmetric}$ is defined as $\forall x \forall y (r(x, y) \rightarrow r(y, x))$.

**This formula is valid:** $\text{symmetric} \lor \text{euclidean} \rightarrow (\text{pre}_{.}\text{thm}_{3}(v) \rightarrow \text{thm}_{3}(v))$.

As observed in [7], $\text{T3}$ can not be just proven in the modal logic $\text{S5}$, but also in $\text{KB}$, whose accessibility relation is just constrained to be symmetric. We have shown here in a single statement that the proof is possible for a Euclidean as well as a symmetric accessibility relation by presupposing the disjunction of both properties. Precondition $\text{pre}_{.}\text{thm}_{3}$ includes $\text{coro}$ instantiated with just the current world and $\text{lemma}_{2}$ with a universal quantifier upon the world parameter. In fact, as shown now, using $\text{lemma}_{2}$ there just instantiated with the current world would not be sufficient to derive $\text{thm}_{3}$.

**This formula is not valid:** $\text{symmetric} \lor \text{euclidean} \rightarrow (\text{r}_{.}\text{world}_{1} \land \text{lemma}_{2}(v) \land \text{coro}(v) \rightarrow \text{thm}_{3}(v))$.

In [31] further aspects of Gödel’s proof are modeled, in particular modal collapse and monotheism.
4 On Weakening the Frame Condition for Theorem T3

In the proof of \textit{thm}$_3$ from \textit{pre\_thm}$_3$ we used the additional frame condition \textit{euclidean} ∨ \textit{symmetric}. The observation that the weaker \textit{KB} instead of \textit{S5} suffices to prove \textbf{T3} was an important finding of [7]. Hence, the question arises whether the precondition on the accessibility relation can be weakened further.

In general, the \textit{weakest sufficient condition} [19,11,28] of a formula \textit{G} on a set \textit{Q} of predicates within a formula \textit{F} can be expressed as the second-order formula $\forall p_1 \ldots \forall p_n (F \rightarrow G)$, where $p_1, \ldots, p_n$ are all predicates that occur free in $F \rightarrow G$ and are not members of \textit{Q}. This formula denotes the weakest (with respect to entailment) formula \textit{H} in which only predicates in \textit{Q} occur free such that $H \rightarrow (F \rightarrow G)$ is valid. Second-order quantifier elimination can be applied to this formula to “compute” a weakest sufficient condition, that is, converting it to a first-order formula, which, of course, is inherently not possible in all cases.

For \textbf{T3}, the weakest sufficient condition on the accessibility relation \textit{r} and the domain membership relation and \textit{e} is the second-order formula

$$\forall g \forall v (\textit{pre\_thm}$_3$(v) → \textit{thm}$_3$(v)).$$

Unfortunately, elimination of the second-order quantifier upon \textit{g} fails for this formula (at least with the current version of \textit{PIE}). But elimination succeeds for a simplified variant of the problem, which considers just propositional modal logic and combines two instances of Lemma \textit{lemma}$_2$ with an unfolding of \textit{C}. The way in which this simplification was obtained is outlined in [31].

\textbf{Macro 34} \textit{lemma}$_2$\_simp($V$) is defined as

$$g(V) \rightarrow \forall W (r(V,W) \rightarrow g(W)).$$

\textbf{Macro 35} \textit{pre\_thm}$_3$\_simp\_inst($V$) is defined as

$$\textit{lemma}$_2$\_simp($V$) \wedge \exists W (r(V,W) \wedge g(W) \wedge \textit{lemma}$_2$\_simp(W)).$$

\textbf{Macro 36} \textit{thm}$_3$\_simp($V$) is defined as

$$\forall W (r(V,W) \rightarrow g(W)).$$

This formula is valid: \textit{euclidean} ∨ \textit{symmetric} → (\textit{pre\_thm}$_3$\_simp\_inst($v$) → \textit{thm}$_3$\_simp($v$)).

\textbf{Input:} $\forall g \forall v (\textit{pre\_thm}$_3$\_simp\_inst(v) → \textit{thm}$_3$\_simp(v)).$

\textbf{Result of elimination:}

$$\forall x \forall y \forall z (r(x,y) \land r(x,z) \rightarrow r(y,x) \lor r(y,z) \lor x = y \lor y = z).$$

We write the resulting first-order formula in a slightly different form, give it a name, verify equivalence to the original form and show some of its properties.

\textbf{Macro 37} \textit{frame\_cond\_simp} is defined as

$$\forall x \forall y \forall z (r(x,y) \land r(x,z) \land y \neq x \land y \neq z \rightarrow r(y,x) \lor r(y,z)).$$

This formula is valid: \textit{frame\_cond\_simp} ↔ \textit{last\_result}.
Macro 38  reflexive is defined as $\forall x \, r(x, x)$.

This formula is valid: reflexive $\rightarrow$ (symmetric $\lor$ euclidean $\leftrightarrow$ frame_cond_simp).

This formula is valid: symmetric $\lor$ euclidean $\rightarrow$ frame_cond_simp.

This formula is not valid: frame_cond_simp $\rightarrow$ symmetric $\lor$ euclidean.

Thus, the obtained frame condition frame_cond_simp is under the assumption of reflexivity equivalent to symmetric $\lor$ euclidean, and without that assumption strictly weaker. The following statement shows that this weaker frame condition also works for our original problem, proving T3.

This formula is valid: frame_cond_simp $\rightarrow$ (pre_thm$_3$(v) $\rightarrow$ thm$_3$(v)).

Hence, via the detour through elimination applied to a simplified problem, we have found a strictly weaker frame condition for T3 than symmetric $\lor$ euclidean, but, since elimination has just been performed on the second-order formula representing the simplified problem, we do not know whether it is the weakest one.

5 Conclusion

We reconstructed Gödel’s ontological proof in an environment that integrates automated first-order theorem proving, second-order quantifier elimination, a formula macro mechanism and LATEX-based formula pretty-printing, supplementing a number of previous works that render Gödel’s proof in other automated theorem proving environments. Particular observations of interest for the study of Gödel’s proof that became apparent through our modeling include the following:

1. The presentation of the derivation of theorem T1 exhibits the few actually used instantiations of axioms A1 and A2. The derivation is via a lemma, which might be useful as internal interface in the proof because it can be justified in alternate ways.

2. Corollary C can be shown independently from the actual definition of God-like (D1) just on the basis of the assumption that T1 applies to God-like.\(^4\)

3. In the whole proof, definition D1 is only used in the left-to-right direction.\(^5\)

4. Second-order quantifier elimination yields first-order representations of essence (definition D2) and necessary existence (definition D3).

5. Lemma L can be derived independently from the definiens of essence. Here the predicate ess appears in the respective expanded formula passed to the reasoner, but not its definiens.

6. For the derivation of theorem T3 an accessibility relationship is sufficient that, unless reflexivity is assumed, is strictly weaker than the disjunction of the Euclidean property and symmetry.

If non-experts in automated reasoning are addressed, the syntactical presentation of Gödel’s argument is of particular importance [8]. We approached this

\(^4\) This is also apparent in [5, Fig. 4, line 20].

\(^5\) This applies if A3 is given as in Scott’s version, but not if it is derived from further general properties of positive, as in Gödel’s original version and in [5, Fig. 4, line 19].
requirement by means of formula macro definitions with the representation of input formulas by Prolog terms and \LaTeX \ pretty-printing for output formulas.

Most automated formalizations of metaphysical arguments, e.g., [12,24,7,8,5], seem closely tied to a particular system or combination of systems. Of course, processing a \textit{PIE} document similarly depends on the \textit{PIE} system with specific embedded reasoners. However, a system-independent view on the formalization is at least obtainable: The underlying target logic of the macro expansion is just the well-known classical first-order logic extended with predicate quantification. Reasoning tasks are only performed on the expanded formulas. The \textit{PIE} system can output these explicitly (see, e.g., [31]), providing a low-level, but system-independent logical representation of the complete formalization. As a further beneficial aspect, such an explicit low-level formalization may prevent the unnoticed interaction with features of involved special logics.

A limitation of our approach might be that there is no automated support for the passage from the low to the high level, i.e., folding into formula macros. If trust in proofs is an issue, steps in the overall workflow for which no proof representations are produced may be objectionable. This concerns macro expansion, formula normalization (see, however, [21]), pre- and postprocessing of formulas, and in particular second-order quantifier elimination, for which the creation of proofs seems an unexplored terrain. A practical makeshift is comparison with the few other elimination systems [2, Sect. 4].

In principle it should be possible to integrate second-order quantifier elimination as used here also into automated reasoning environments based on other paradigms, in particular the heterogeneous environments that involve forms of higher-order reasoning and are applied in [7,8,6,5].

Concerning second-order quantifier elimination, an issue that might be worth further investigation is the generalization of the method applied here ad-hoc to weaken the precondition on the accessibility relation: We started from an elimination problem that could not be solved (at least with the current implementation of \textit{PIE}), moved to a simpler problem and then verified that the solution of the simpler problem is also applicable to the original problem, where it does not represent the originally desired unique \textit{weakest} sufficient condition, but nevertheless a condition that is weaker than the condition known before.

\textbf{Acknowledgments.} The author thanks Christoph Benzmüller and anonymous reviewers for helpful remarks. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 457292495.

\textbf{References}


1 Introduction

*Second-Order Propositional Modal Logic (SOMPL).* Modal logic with propositional quantifiers has been considered in the literature since Kripke [13], Bull [2], Fine [8, 9], and Kaplan [7]. This language is of high complexity: its satisfiability problem is not decidable, and indeed not even analytical. In Kaminski and Tionkin [12], the authors showed that the expressive power for SOMPL whose modalities are S4.2 or weaker is the same as second-order predicate logic. However, not every second-order formula is equivalent to an SOMPL-formula, since SOMPL-formulas are preserved under generated submodels (see van Benthem [16]). In ten Cate [15], the author proved the analogues of the van Benthem-Rosen theorem (on the model level) and Goldblatt-Thomason theorem (on the frame level) for SOMPL. Therefore, a natural question is: on the frame level, can we find a natural fragment of SOPML-formulas such that each formula in this fragment corresponds to a first-order formula, in the sense of Sahlqvist theory (see [14, 16])? This is what we will answer in the paper.

*Correspondence Theory.* Typically, modal correspondence theory [16] concerns the correspondence of modal formulas and first-order formulas over Kripke frames, via the tools of standard translation. Syntactic classes (e.g. Sahlqvist formulas [14], inductive formulas [11], etc.) of modal formulas are identified to have first-order correspondents and are canonical, i.e. their validity are closed under taking canonical extensions. In the present paper, we identify the Sahlqvist formulas of SOMPL, which cover and properly extend the Sahlqvist fragment in basic modal logic. We show that there is an SOMPL Sahlqvist formula which corresponds to $\forall x \forall y (Rxy \land Ryx \rightarrow Rxx)$, which is not modally definable, and that the SOMPL Sahlqvist formula $\forall q (\forall p (p \rightarrow \Diamond p \lor q) \rightarrow q)$ is not canonical, which is in contrast to the basic modal logic setting where each Sahlqvist formula is canonical. The present paper use the same methodology as [6, 3]. The Sahlqvist fragment of SOPML is defined in a step-by-step way, and we give an algorithm ALBA_{SOPML} (Ackermann Lemma Based Algorithm) which can successfully reduce Sahlqvist formulas in SOPML to first-order formulas and is sound with respect to Kripke semantics.

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
2 Preliminaries

2.1 Language and semantics

In the present paper we consider the unimodal language. Given a set $\text{Prop}$ of propositional variables, the second-order propositional modal formulas are defined as follows:

$$\varphi ::= p \mid \bot \mid \top \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varphi \rightarrow \varphi \mid \Box \varphi \mid \Diamond \varphi \mid \forall p \varphi \mid \exists p \varphi$$

where $p \in \text{Prop}$. We use the boldface notation $\vec{p}$ to denote a set of propositional variables and use $\varphi(\vec{p})$ to indicate that the propositional variables occur in $\varphi$ are all in $\vec{p}$. We say that an occurrence of a propositional variable $p$ in a formula $\varphi$ is positive (resp. negative) if it is in the scope of an even (resp. odd) number of negations (here $\alpha \rightarrow \beta$ is regarded as $\neg \alpha \lor \beta$).

The semantics of the second-order propositional modal formulas are defined as follows:

**Definition 1.** A Kripke frame is a pair $F = (W, R)$ where $W \neq \emptyset$ is the domain of $F$, the accessibility relation $R$ is a binary relation on $W$. A Kripke model is a pair $M = (F, V)$ where $V : \text{Prop} \rightarrow P(W)$ is a valuation on $F$. $V^p$ denote a valuation which is the same as $V$ except that $V^p(p) = X \subseteq W$.

Now the satisfaction relation can be defined as follows: given any Kripke model $M = (W, R, V)$, any $w \in W$, the basic and Boolean cases are standard, and for modalities and propositional quantifiers,

- $M, w \models \Box \varphi$ iff for any $v$ such that $Rwv$, $M, v \models \varphi$;
- $M, w \models \Diamond \varphi$ iff there exists $v$ such that $Rwv$ and $M, v \models \varphi$;
- $M, w \models \forall p \varphi$ iff for all $X \subseteq W$, $(W, R, V^p_X), w \models \varphi$;
- $M, w \models \exists p \varphi$ iff there exists $X \subseteq W$ such that $(W, R, V^p_X), w \models \varphi$.

In order to use the algorithm to compute the first-order correspondents of Sahlqvist SOPML formulas, we will need the following expanded modal language which is defined as follows:

$$\varphi ::= p \mid i \mid \bot \mid \top \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varphi \rightarrow \varphi \mid \Box \varphi \mid \Diamond \varphi \mid \Boxi \varphi \mid \Diamondi \varphi \mid \forall i \varphi \mid \exists i \varphi \mid l(\varphi, \varphi)$$

where $p \in \text{Prop}$, $i \in \text{Nom}$ is a nominal, $\Box$ and $\Diamond$ are the backward-looking box and diamond respectively, $\forall i$ and $\exists i$ are nominal quantifiers, and $l$ is a binary modality. We call a formula pure if it does not contain propositional variables or propositional quantifiers (it can contain nominals, nominal quantifiers and the binary modality $l$).

The interpretation of the expanded modal language is given as follows: For a valuation $V$, it is defined as $V : \text{Prop} \cup \text{Nom} \rightarrow P(W)$ such that $V(i)$ is a singleton for all $i \in \text{Nom}$. The additional satisfaction clauses are given as follows (here $V^i_v$ denote a valuation which is the same as $V$ except that $V^i_v(i) = \{v\} \subseteq W.$):
\( M, w \models i \quad \text{iff } V(i) = \{ w \}; \)
\( M, w \models \Box \varphi \quad \text{iff for any } v \text{ such that } Rvw, M, v \models \varphi; \)
\( M, w \models \Diamond \varphi \quad \text{iff there exists } v \text{ such that } Rvw \text{ and } M, v \models \varphi; \)
\( M, w \models \forall i \varphi \quad \text{iff for all } v \in W, (W, R, V_i^v), w \models \varphi; \)
\( M, w \models \exists i \varphi \quad \text{iff there exists } v \in W \text{ such that } (W, R, V_i^v), w \models \varphi; \)
\( M, w \models l(\varphi, \psi) \quad \text{iff for all } v \in W \text{ (if } M, v \models \varphi, \text{ then } M, v \models \psi). \)

We can extend \( V \) to a map from the set of formulas to \( P(W) \) in the natural way.

### 2.2 Inequalities and complex inequalities

We will find it convenient to use the inequality notation \( \varphi \leq \psi \) where \( \varphi \) and \( \psi \) are formulas. We use \( \text{Ineq} \) to denote the set of all inequalities in the expanded modal language. We define complex inequalities as follows:

\[
\text{Comp} ::= \text{Ineq} \mid \text{Comp} \& \text{Comp} \mid \text{Comp} \Rightarrow \text{Comp} \mid \forall p \text{Comp} \mid \exists p \text{Comp} \mid \forall i \text{Comp} \mid \exists i \text{Comp}
\]

Here we assume that the quantifiers have a higher precedence than \( \& \), and \( \& \) is higher than \( \Rightarrow \).

Complex inequalities are interpreted in models \( M = (W, R, V) \) instead of pointed models \( (M, w) \). The semantics of complex inequalities is defined as follows:

- An inequality is interpreted as follows:
  
  \( (W, R, V) \models \varphi \leq \psi \iff \) 
  
  \( \text{(for all } w \in W, \text{ if } (W, R, V), w \models \varphi, \text{ then } (W, R, V), w \models \psi); \)

- \( (W, R, V) \models \text{Comp}_1 \& \text{Comp}_2 \iff (W, R, V) \models \text{Comp}_1 \text{ and } (W, R, V) \models \text{Comp}_2; \)

- \( (W, R, V) \models \text{Comp}_1 \Rightarrow \text{Comp}_2 \iff ((W, R, V) \models \text{Comp}_1 \implies (W, R, V) \models \text{Comp}_2); \)

- \( (W, R, V) \models \forall p \text{Comp} \iff \text{for all } X \subseteq W, (W, R, V_X^p) \models \text{Comp}; \)

- \( (W, R, V) \models \exists p \text{Comp} \iff \text{there exists an } X \subseteq W \text{ such that } (W, R, V_X^p) \models \text{Comp}; \)

- \( (W, R, V) \models \forall i \text{Comp} \iff \text{for all } v \in W, (W, R, V_i^v) \models \text{Comp}; \)

- \( (W, R, V) \models \exists i \text{Comp} \iff \text{there exists an } v \in W \text{ such that } (W, R, V_i^v) \models \text{Comp}. \)

### 2.3 Standard translation

In the correspondence language which is second-order due to the existence of propositional quantifiers in \( \text{SOPML} \), we have a binary predicate symbol \( R \) corresponding to the binary relation, a set of constant symbols \( i \) corresponding to each nominal \( i \), a set of unary predicate symbols \( P \) corresponding to each propositional variable \( p \).

**Definition 2.** The standard translation of the expanded \( \text{SOPML} \) language is defined as follows (for the basic and Boolean case, it is standard):
Proposition 1. For any Kripke model $\mathcal{M}$, any $w \in W$ and any expanded SOPML formula $\varphi$,

$$\mathcal{M}, w \models \varphi \iff \mathcal{M} \models ST_x(\varphi)[x := w].$$

For inequalities and complex inequalities, the standard translation is given in a global way:

Definition 3. – $ST(\varphi \leq \psi) := \forall x(ST_x(\varphi) \to ST_x(\psi))$;
– $ST(\text{Comp}_1 \& \text{Comp}_2) = ST(\text{Comp}_1) \wedge ST(\text{Comp}_2)$;
– $ST(\text{Comp}_1 \Rightarrow \text{Comp}_2) = ST(\text{Comp}_1) \to ST(\text{Comp}_2)$;
– $ST(\forall p(\text{Comp})) := \forall P(ST(\text{Comp}))$;
– $ST(\exists p(\text{Comp})) := \exists P(ST(\text{Comp}))$;
– $ST(\forall i(\text{Comp})) := \forall i(ST(\text{Comp}))$;
– $ST(\exists i(\text{Comp})) := \exists i(ST(\text{Comp}))$.

Proposition 2. For any Kripke model $\mathcal{M}$, any inequality $\text{Ineq}$, any complex inequality $\text{Comp}$,

$$\mathcal{M} \models \text{Ineq} \iff \mathcal{M} \models ST(\text{Ineq});$$

$$\mathcal{M} \models \text{Comp} \iff \mathcal{M} \models ST(\text{Comp}).$$

3 Sahlgqvist formulas in second-order propositional modal logic

In this section, we define Sahlgqvist formulas of second-order propositional modal logic step by step.

We first define (quantifier-free) positive formulas $\text{POS}(\vec{p})$ whose propositional variables are among $\vec{p}$:

$$\text{POS}(\vec{p}) := p \mid \bot \mid \top \mid \text{POS}(\vec{p}) \wedge \text{POS}(\vec{p}) \mid \text{POS}(\vec{p}) \vee \text{POS}(\vec{p}) \mid \square\text{POS}(\vec{p}) \mid \Diamond\text{POS}(\vec{p})$$

where $p$ is in $\vec{p}$. These positive formulas have similar roles to the positive consequent part in Sahlgqvist formulas in basic modal logic, which are going to receive minimal valuations. The reason why we do not allow propositional quantifiers in positive formulas is that we want the formula after receiving the minimal valuations to be translated into a first-order formula, while propositional quantifiers will make it second-order.
3.1 The $\Pi_1$-fragment: Sahlqvist formulas in basic modal logic

We define the $\Pi_1$-Sahlqvist antecedent $\text{Sahl}_1(\vec{p})$ whose propositional variables are among $\vec{p}$:

$$\text{Sahl}_1(\vec{p}) ::= \Box^n p \mid \bot \mid \top \mid \neg \text{POS}(\vec{p}) \mid \text{Sahl}_1(\vec{p}) \land \text{Sahl}_1(\vec{p}) \mid \Diamond \text{Sahl}_1(\vec{p})$$

where $p$ is in $\vec{p}$.

Then the $\Pi_1$-Sahlqvist formulas are defined as $\forall \vec{p}(\text{Sahl}_1(\vec{p}) \to \text{POS}(\vec{p}))$. Indeed, Sahlqvist formulas\(^1\) in the basic modal logic setting can be treated as universally quantified by propositional quantifiers which bind all occurrences of propositional variables, so in this sense the $\Pi_1$-Sahlqvist formulas can be taken as the Sahlqvist formulas in basic modal logic.

3.2 The $\Pi_2$-fragment

We define the PIA formula $\text{PIA}(\vec{q}, \vec{p})$ as follows:

$$\text{PIA}(\vec{q}, \vec{p}) ::= p \mid \Box \text{PIA}(\vec{q}, \vec{p}) \mid \text{PIA}(\vec{q}, \vec{p}) \land \text{PIA}(\vec{q}, \vec{p}) \mid \text{POS}(\vec{q}) \lor \text{PIA}(\vec{q}, \vec{p})$$

where $p$ is in $\vec{p}$. Here the PIA formula has two bunches of propositional variables: $\vec{q}$ is to receive minimal valuations for $\vec{q}$ from somewhere else, and $\vec{p}$ is used to compute minimal valuations for $\vec{p}$. Then it is easy to see that $\text{PIA}(\vec{q}, \vec{p})$ is equivalent to the form $\bigwedge \Box(\text{POS}(\vec{q}) \lor \Box(\text{POS}(\vec{q}) \lor \ldots p))$, where $p$ is in $\vec{p}$.

Now we can define $\Pi_2$-Sahlqvist antecedents as follows:

$$\text{Sahl}_2(\vec{p}) ::= \text{Sahl}_1(\vec{p}) \mid \forall \vec{q}(\text{Sahl}_1(\vec{q}) \to \text{PIA}(\vec{q}, \vec{p})) \mid \text{Sahl}_2(\vec{p}) \land \text{Sahl}_2(\vec{p}) \mid \Diamond \text{Sahl}_2(\vec{p})$$

Then $\Pi_2$-Sahlqvist formulas are defined as $\forall \vec{p}(\text{Sahl}_2(\vec{p}) \to \text{POS}(\vec{p}))$.

It is easy to see that formulas of the form $\forall \vec{p}(\text{Sahl}_1(\vec{p}) \land \forall \vec{q}(\text{Sahl}_1(\vec{q}) \to \text{PIA}(\vec{q}, \vec{p})) \to \text{POS}(\vec{p}))$ are in the $\Pi_2$-hierarchy.

3.3 The $\Pi_n$-fragment

Now for the $\Pi_n$-fragment, assume that we have already defined $\Pi_{n-1}$-Sahlqvist antecedents $\text{Sahl}_{n-1}(\vec{p})$ and $\Pi_{n-1}$-Sahlqvist formulas $\forall \vec{p}(\text{Sahl}_{n-1}(\vec{p}) \to \text{POS}(\vec{p}))$, then we can define $\Pi_n$-Sahlqvist antecedents as follows:

$$\text{Sahl}_n(\vec{p}) ::= \text{Sahl}_{n-1}(\vec{p}) \mid \forall \vec{q}(\text{Sahl}_{n-1}(\vec{q}) \to \text{PIA}(\vec{q}, \vec{p})) \mid \text{Sahl}_n(\vec{p}) \land \text{Sahl}_n(\vec{p}) \mid \Diamond \text{Sahl}_n(\vec{p})$$

Then $\Pi_n$-Sahlqvist formulas are defined as $\forall \vec{p}(\text{Sahl}_n(\vec{p}) \to \text{POS}(\vec{p}))$.

\(^1\) In [1, Chapter 3], what we call Sahlqvist formulas are called Sahlqvist implications.
4 The Algorithm ALBA^SOMPL

In the present section, we define the correspondence algorithm ALBA^SOMPL for second-order propositional modal logic, in the style of [4, 5]. The algorithm receives a $\Pi_n$-Sahlqvist formula $\forall \vec{p}(Sahl_n(\vec{p}) \rightarrow POS(\vec{p}))$ as input and goes in three stages.

1. Preprocessing and first approximation:
   The algorithm receives a $\Pi_n$-Sahlqvist formula $\forall \vec{p}(Sahl_n(\vec{p}) \rightarrow POS(\vec{p}))$ as input, and then apply the rewriting rule:
   \[
   \forall \vec{p}(Sahl_n(\vec{p}) \rightarrow POS(\vec{p})) \rightarrow \forall \vec{p}(Sahl_n(\vec{p}) \leq POS(\vec{p}))
   \]

   Then apply the first-approximation rule:
   \[
   \forall \vec{p}(Sahl_n(\vec{p}) \leq POS(\vec{p})) \rightarrow \forall \vec{p}(\forall i_0(i_0 \leq Sahl_n(\vec{p}) \Rightarrow i_0 \leq POS(\vec{p})))
   \]

2. The reduction stage:
   In this stage, we aim at reducing $i \leq Sahl_n(\vec{p})$ to a complex inequality in which $p$ occurs either in the form $\varphi \leq p$ where $\varphi$ is pure or in the form $j \leq \neg POS(\vec{p})$.
   (a) The commutativity rule and the associativity rule for $\&$;
   (b) The rules for nominals:
      i. Splitting rule:
         \[
         i \leq \alpha \land \beta \rightarrow i \leq \alpha \land i \leq \beta \quad (Spl - Nom)
         \]
      ii. Separation rule:
         \[
         i \leq \alpha \rightarrow \beta \rightarrow i \leq \beta \quad (Sep - Nom)
         \]
      iii. Quantifier rule:
         \[
         i \leq \forall q \alpha \rightarrow \forall q(i \leq \alpha) \quad (Quant - Nom)
         \]
      iv. Approximation rule:
         \[
         i \leq \Diamond \alpha \rightarrow \exists j(j \leq \alpha \land i \leq \Diamond j) \quad (Approx - Nom)
         \]
      The nominals introduced by the approximation rule must not occur in the whole complex inequality before applying the rule.
   (c) The residuation rules:
      \[
      \alpha \leq \Box \beta \quad (Res - \Box) \quad \alpha \leq \beta \lor \gamma \rightarrow \alpha \land \neg \beta \leq \gamma \quad (Res - \lor)
      \]
(d) The splitting rule:
\[
\alpha \leq \beta \land \gamma \\
\alpha \leq \beta \land \alpha \leq \gamma \quad (\text{Splitting})
\]

(e) The quantifier rules:
\[
\exists j (\text{Comp}_1) \& \text{Comp}_2 \quad (\text{Scope} - \&) \\
\exists j (\text{Comp}_1 \land \text{Comp}_2) \quad (\text{Scope} - \Rightarrow)
\]

where \(\text{Comp}_2\) does not have free occurrences of \(j\).

\[
\forall q \forall p (\text{Comp}) \quad (\text{Ex} - p q) \\
\forall i \forall p (\text{Comp}) \quad (\text{Ex} - p i)
\]

\[
\forall i \forall p (\text{Comp}) \quad (\text{Ex} - i p) \\
\forall j \forall i (\text{Comp}) \quad (\text{Ex} - j i)
\]

\[
\forall p (\text{Comp}_1 \Rightarrow (\text{Comp}_2 \& \text{Comp}_3)) \\
\forall p (\text{Comp}_1 \Rightarrow \text{Comp}_2) \land \forall p (\text{Comp}_1 \Rightarrow \text{Comp}_3) \quad (\text{Spl} - \text{Quant} - p)
\]

\[
\forall i (\text{Comp}_1 \Rightarrow (\text{Comp}_2 \& \text{Comp}_3)) \\
\forall i (\text{Comp}_1 \Rightarrow \text{Comp}_2) \land \forall i (\text{Comp}_1 \Rightarrow \text{Comp}_3) \quad (\text{Spl} - \text{Quant} - i)
\]

(f) The Ackermann rule:
In this step, we compute the minimal valuation for propositional variables and use the Ackermann rule to eliminate all the propositional variables.

\[
\forall q (\alpha_1 \leq \beta_1 \& \ldots \& \alpha_n \leq \beta_n \& \psi_1 \leq q \& \ldots \& \psi_m \leq q \Rightarrow \alpha \leq \beta) \\
\alpha_1 [\bigvee \psi/q] \leq \beta_1 [\bigvee \psi/q] \& \ldots \& \alpha_n [\bigvee \psi/q] \leq \beta_n [\bigvee \psi/q] \Rightarrow \alpha [\bigvee \psi/q] \leq \beta [\bigvee \psi/q]
\]

where:

i. \(\varphi[\theta/p]\) means uniformly replace occurrences of \(p\) in \(\varphi\) by \(\theta\);
ii. \(\bigvee \psi = \psi_1 \lor \ldots \lor \psi_m\);
iii. Each \(\alpha_i\) is positive, and each \(\beta_i\) negative in \(q\), for \(1 \leq i \leq n\);
iv. \(\alpha\) is negative in \(q\) and \(\beta\) is positive in \(q\);
v. Each \(\psi_i\) is pure (therefore \(q\) does not occur in \(\psi_i\)).

(g) The packing rule:

\[
\forall i (\alpha_1 \leq \beta_1 \& \ldots \& \alpha_n \leq \beta_n \Rightarrow \alpha \leq \beta) \\
\exists i (\mathbb{I}(\alpha_1, \beta_1) \land \ldots \land \mathbb{I}(\alpha_n, \beta_n) \land \alpha) \leq \beta
\]

where \(\beta\) does not contain occurrences of \(i\).

3. **Output**: By the execution of the algorithm, we can guarantee that given a \(\Pi_n\)-Sahlqvist formula as input, we can rewrite it into a pure complex inequality. Then we use standard translation to translate it into a first-order formula.
Theorem 1 (Soundness and Success).

- If ALBA\textsuperscript{SOPML} runs successfully on an input $\Pi_n$-Sahlqvist formula $\forall \vec{p}(\text{Sahl}_n(\vec{p}) \rightarrow \text{POS}(\vec{p}))$ and outputs a first-order formula $\text{FO}(\forall \vec{p}(\text{Sahl}_n(\vec{p}) \rightarrow \text{POS}(\vec{p})))$, then for any Kripke frame $\mathbb{F} = (W,R)$,

$$\mathbb{F} \models \forall \vec{p}(\text{Sahl}_n(\vec{p}) \rightarrow \text{POS}(\vec{p})) \text{ iff } \mathbb{F} \models \text{FO}(\forall \vec{p}(\text{Sahl}_n(\vec{p}) \rightarrow \text{POS}(\vec{p}))).$$

- There is an algorithm such that for any $\Pi_n$-Sahlqvist formula $\varphi$, it can be transformed into an equivalent first-order formula.

5 Examples

We give three examples of $\Pi_2$-Sahlqvist formulas to show how the ALBA\textsuperscript{SOPML} algorithm works:

Example 1. $\forall p(\square p \land \forall q(\square q \rightarrow \square(\square q \lor \square p)) \rightarrow \square \square p)$

$$\forall p \forall i(i \leq \square p \land \forall q(\square q \rightarrow \square(\square q \lor \square p)) \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(i \leq \square p \land i \leq \forall q(\square q \rightarrow \square(\square q \lor \square p)) \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(i \leq \square p \land \forall q(i \leq \square q \rightarrow \square(\square q \lor \square p)) \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(i \leq \square p \land \forall q(i \leq \square q \land \forall j(i \leq j \land j \leq \square q \Rightarrow i \leq \square(\square q \lor \square p)) \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(i \leq \square p \land \forall j(i \leq j \Rightarrow i \leq \square(\square j \lor \square p)) \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(i \leq \square p \land \forall j(i \leq j \Rightarrow \square i \land \forall j \leq \square p) \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(i \leq \square p \land \forall j(i \leq j \Rightarrow \square i \land \forall j \leq p) \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(i \leq \square p \land \forall j(i \leq j \land i \leq \square j \land \forall j \leq \square p) \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(i \leq \square p \land \forall j(i \leq j \land i \leq \square j \land \forall j \leq \square p) \Rightarrow i \leq \square \square p)$$

Now denote $\exists j(l(i,\square j) \land \square (\square i \land \square j))$ as $\varphi$, then

$$\forall p \forall i(i \leq \square p \land \varphi \leq p \Rightarrow i \leq \square \square p)$$

$$\forall p \forall i(k(i \leq \square k \land k \leq \square p \land \varphi \leq p \Rightarrow i \leq \square \square p)$$

$$\forall i(k(i \leq \square k \land \varphi \leq p \Rightarrow i \leq \square \square p)$$

$$\forall i(k(i \leq \square k \Rightarrow i \leq \square \square (\square k \lor \varphi))$$

Then we can use standard translation to get its first-order correspondence.

Example 2. The following example resembles the irreflexivity rule of Gabbay [10]:

$$\forall q(\forall p(\rightarrow \square p \lor q) \rightarrow q)$$

$$\forall q \forall i(i \leq \forall p(\rightarrow \square p \lor q) \Rightarrow i \leq q)$$

$$\forall q \forall i(\forall p(i \leq \rightarrow \square p \lor q) \Rightarrow i \leq q)$$

$$\forall q \forall i(\forall p(i \leq p \Rightarrow i \leq \rightarrow p \lor q) \Rightarrow i \leq q)$$

$$\forall q \forall i(i \leq \rightarrow \square q \Rightarrow i \leq q)$$
∀q∀i(i ∧ ¬3i ≤ q ⇒ i ≤ q)
∀i(i ≤ i ∧ ¬3i)
∀i(i ≤ ¬3i)
∀x¬Rxx.

By [1, Example 2.58], the irreflexive property is not preserved under taking ultrafilter extensions, which means that the validity of ∀q(∀p(p → ♢p ∨ q) → q) is not preserved under taking canonical extensions, which means that ∀q(∀p(p → ♢p ∨ q) → q) is not canonical.

Example 3. The following example is not equivalent to any Sahlqvist formula in the basic modal language:

∀p(□p ∧ ∀q(q → □♢q ∨ p) → p)
∀p∀i(i ≤ □p ∧ ∀q(q → □♢q ∨ p) ⇒ i ≤ p)
∀p∀i(i ≤ □p & i ≤ ∀q(q → □♢q ∨ p) ⇒ i ≤ p)
∀p∀i(◇i ≤ p & i ≤ ∀q(q → □♢q ∨ p) ⇒ i ≤ p)
∀p∀i(◇i ≤ p & ∀q(i ≤ q → □♢q ∨ p) ⇒ i ≤ p)
∀p∀i(◇i ≤ p & ∀q(i ≤ q ⇒ i ≤ ◇q ∨ p) ⇒ i ≤ p)
∀p∀i(◇i ≤ p & i ≤ ◇□i ⇒ i ≤ p)
∀p∀i(◇i ≤ p & i ∨ ¬□□i ≤ p ⇒ i ≤ p)
∀p∀i(◇i ∨ (i ∨ ¬□□i) ≤ p ⇒ i ≤ p)
∀i(i ≤ ◇i or i ≤ i ∧ ¬□□i)
∀i(i ≤ ◇i or i ≤ ¬□□i)
∀i(i ≤ ◇□i → ◇i)
∀x∀y(Rxy ∧ Ryx → Rxx)

One can show that this property is not modally definable:

Consider \( F_1 = (W_1, R_1) \) where \( W_1 \) is the set of all integers, \( R_1 = \{(x, x+1) | x \in W_1\} \), \( F_2 = (W_2, R_2) \) where \( W_2 = \{w_0, w_1\} \), \( R_2 = \{(w_0, w_1), (w_1, w_0)\} \), then \( F_2 \) is a bounded morphic image of \( F_1 \), \( F_1 \models ∀x∀y(Rxy ∧ Ryx → Rxx) \), while \( F_2 \not\models ∀x∀y(Rxy ∧ Ryx → Rxx) \).

References

Metadata-based Term Selection for Modularization and Uniform Interpolation of OWL Ontologies

Xinhao Zhu\textsuperscript{2}, Xuan Wu\textsuperscript{1,2}, Ruiqing Zhao\textsuperscript{2}, Yu Dong\textsuperscript{2}, and Yizheng Zhao\textsuperscript{1,2}

\textsuperscript{1} National Key Laboratory for Novel Software Technology, Nanjing University, China
\textsuperscript{2} School of Artificial Intelligence, Nanjing University, China

Abstract. This paper explores the problem of selecting good terms as seed signatures for abstraction of OWL ontologies. Existing methods generate seed signatures based on geographic connections, which is far from sufficient to produce a satisfactory abstract. This restricts the reusability of OWL ontologies from the aspect of knowledge management. In this paper, we propose a signature extension approach to generate seed signatures for modularization and uniform interpolation of OWL ontologies, both of which are ontology abstraction techniques. The approach establishes the semantic relevance of terms by taking into account as much as possible metadata information of an OWL ontology, and computes a numerical value to measure the relevance of terms using their embedding transformed based on a so-called OWL2Vec* framework. An empirical evaluation of the approach shows that the proposed method significantly outperforms other term selection baselines in making accurate selections. Besides, a case study on ontology abstraction tasks shows that modularization tools can make more complete and precise abstractions using the signature extended by our method.

Keywords: OWL Ontology · Term Selection · Modularization · Uniform Interpolation

1 Introduction

Because of the heterogeneous nature of web resources, ontologies developed for the semantic web are typically large, sometimes monolithic, and knowledge modelled therein is rich and covers multiple topics. This may however restrict the reusability and interoperability of ontologies in real-world application scenarios, since large ontologies can be difficult to manage, unwieldy to manipulate, and moreover costly to reason about.

Consider an ontology reuse use case where an ontologist wants to import a football ontology into a growing sports knowledge base. Currently the only well-established ontology concerning football is the BBC Sports Ontology\textsuperscript{3}, which, however, publishes data about all types of competitive physical activities, pertaining not only to the topic of football. Importing the whole ontology into the knowledge base is not difficult from an engineering perspective, but as one can expect, many web services upon the knowledge base such as search, querying, retrieval, which typically involve extensive reasoning,

\textsuperscript{3} https://www.bbc.co.uk/ontologies/sport

Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
may become problematic, as too much irrelevant information has been, automatically yet unnecessarily, introduced. Such information makes no contribution to the formalization of the information about football but increases the computational cost.

A straightforward way to tackle these challenges of reusability and interoperability is to extract a fragment of an ontology that can behave in the same way as the original ontology in a specific context, but is significantly smaller. In the above case, this means to extract from the BBC Sports Ontology a fragment that contains sufficiently many logical statements to summarize all knowledge about football. Ideally, this fragment should be as small as possible.

Two logic-based approaches have been developed for computing fragments of ontologies. One is based on modularization [5,9,12,8,2,14], which seeks to identify from an ontology a subset (module) that preserves several reasoning tasks for a sub-vocabulary of the ontology, namely a seed signature.\(^4\) The other is uniform interpolation [17,15,6,16], which computes a more compact representation of a module of an ontology which preserves the underlying logical definitions of the terms in the seed signature.

As one could expect, the quality of extracted fragments depends largely on the seed signature fed to modularization and uniform interpolation procedures. We may say that a fragment is complete if it covers all essential information about the topic of interest, and a fragment is precise if it is complete and in addition, it does not include too much irrelevant information about the topic of interest. More specifically, if we selected as seed signature too few terms to summarize all materials of the topic, we would lose important information that a user may be interested in, and if we selected as seed signature too many terms with some of them not strongly relevant to the topic, we would include too much additional information. Importing more information can also change the definitions of the terms in the original ontology [9].

Nevertheless, very little attention has been paid to the problem of term selection for ontology extraction. Chen et al. [2] have proposed a signature extension algorithm to generate seed signatures for ontology modularization. The idea is to (1) fix a primitive seed signature \(\Sigma\), often containing several domain expert-suggested terms, and (2) extend \(\Sigma\) with new terms collected from the axioms which contain the current \(\Sigma\)-terms. This step is iterated until no new terms can be added to \(\Sigma\). One may understand this as: if two people \(p_1\) and \(p_2\) live together in a house \(h_1\) on an island, then they are relevant and team up as \(\Sigma = \{p_1, p_2\}\), and if there exists a road connecting \(h_1\) with another house \(h_2\), then the people living in \(h_2\) are collected into \(\Sigma\). Iteratively, the same strategy applies to the entire island, and in the end, \(\Sigma\) will probably have collected all habitants on the island. However, a person who lives on another island will never be collected by \(\Sigma\) since there is no road connecting two islands; islands are geographically isolated.

Evidently, following this signature extension strategy one must obtain a larger seed signature with which, a more informative fragment will be produced, but we may argue that the seed signature obtained in this way, i.e., using a signature extension algorithm based merely on geographic connections, could hardly yield a complete fragment. Our argument is that: the relevance between a term and the expanding seed signature should be evaluated based on a consideration of all metadata of the participating terms in the context of the host ontology, rather than based merely on their geographic connections.

---

\(^4\) A signature of an ontology is the set of all concept and role names in the ontology.
Consider a scenario where an ontologist wants to extract from a multi-domain ontology a fragment that describes football and closely related information; see Figure 1. With the central term “Football” being selected as a single seed in the primitive signature, an extension \( \Sigma = \{ \text{Football}, \text{BallGame}, \text{Sports}, \text{Player}, \text{FootballPlayer} \} \) is obtained using the above signature extension algorithm. Terms in other domains such as MentholSpray will not be collected in \( \Sigma \), because it is geographically isolated from the domain of Sports. However, the annotated information of MentholSpray explains that “MentholSpray can be used as pain reliever for sports players”. In this sense the term MentholSpray is supposed to be strongly relevant to the topic. Collecting MentholSpray in the extended signature may enable the expanded knowledge base to answer queries regarding the treatment of an injury in a football match. This is a good example showing that the relevance between a term and the expanding seed signature in the context of the host ontology could be established based on important metadata of the participating terms, for example, based on their lexical information.

In this paper, we propose a novel term selection approach to discovering semantic relationships between two isolated groups of terms. The idea is to measure the relevance of non-\( \Sigma \) terms with \( \Sigma \) terms based on their D-dimensional vector representation computed from important metadata of the ontology using OWL2Vec* [3], a random walk- and word embedding-based OWL ontology embedding framework that encodes the semantics of OWL ontologies in a vector space by taking into account their graph structure, lexical information, as well as the logical constructors used therein. The work is intended to enhance existing logic-based ontology abstraction techniques as practical tools for many ontology-based knowledge processing tasks by exploiting non-logical approaches to facilitate this transfer. Previously, not much work has considered tightly coupled logical and data-driven techniques and exploited the complementary strengths of them to open up an application pipeline. Our empirical evaluation showed that the proposed approach significantly outperformed other term selection baselines in recommending good seed signatures, and with this approach, more precise fragments could be produced using one modularization and one uniform interpolation tool.
2 Metadata-based Term Selection

For space reasons, we have to assume readers’ familiarity with the notions of ontology modularization [9] and uniform interpolation [16]. Our term selection approach accommodates ontologies described in OWL 2, which are based on the description logic $SROIQ$ [11]; see the Description Logic Handbook [1] for a detailed description of the syntax and semantics of description logics.

Arguably, most topics can satisfactorily be summarized or defined by a set of concept names, but do not depend too much on role names. Hence, in this paper, we only consider the seed signature to be a set of concept names.

The signature $\Sigma$ of an ontology $\mathcal{O}$ is the set of all concept names in $\mathcal{O}$. Given an ontology $\mathcal{O}$ and a seed signature $\Sigma \subseteq \Sigma(\mathcal{O})$ containing a single or a few concept names suggested by domain experts or simply selected by users, which are believed to be the central term or terms that can best summarize the topic of interest, our approach computes an extension $\Sigma'$ of $\Sigma$ in three steps, namely concept representation learning, computing relevance value, and signature extension based on relevance value. $\Sigma'$ is the seed signature to be fed to modularization and uniform interpolation procedures.

2.1 Concept Representation Learning

The first step is to transform all concept names $A$ in $\mathcal{O}$ into D-dimensional vectors in a vector space where the relevance of each concept name (to $\Sigma$) is computed based on important metadata of $\mathcal{O}$.

Our concept representation learning model is based on OWL2Vec* [3], an ontology embedding framework, which computes the vector representations for concept names in OWL ontologies as expressive as $SROIQ$. OWL2Vec* computes the embedding of an OWL ontology based on a corpus of sequences of tokens, which are encoded from the metadata of the ontology. Such metadata includes the graph structure of the ontology, i.e., an RDF graph (a set of RDF triples) converted from the OWL ontology by OWL2Vec*, the so-called lexical information about the ontology, i.e., annotations, and the so-called logical information about the concepts and roles in the ontology, i.e, subsumption, equivalence, disjointness, etc.

We note that OWL2Vec* was not meant for term selection tasks, so we make modifications to the original OWL2Vec* model to maximize the performance of the downstream term selection models. In particular, we designed a fine-tuning process to further improve ontology embedding, which was task-specific and further discussed in section 3. In the end, every concept name $A$ is represented as a D-dimensional vector $e_A$.

2.2 Computing Relevance of Concept Names w.r.t $\Sigma$

The second step is to compute the relevance value of every (non-$\Sigma$) concept name $A$ in $\mathcal{O}$ w.r.t. $\Sigma$. The computation is based on the relative distance of $e_A$ to its nearest seed neighbour (the nearest seed name) in the vector space. The range of the relevance value is $[0, 1]$ with 1 standing for the strongest relevance and 0 for the weakest relevance. The relevance value is computed by a newly developed algorithm called Nearest Neighbor Ranking algorithm (NN-RANK), shown in Algorithm 1.
Algorithm 1 Nearest Neighbour Ranking

Input: A set of concepts $N_C$, A set of seed signatures $\Sigma$ s.t. $\Sigma \subseteq N_C$, A set of concept embedding $\{e_A : A \in N_C\}$, A distance function $d : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0, \infty]$. 

Output: A relevance function $f : N_C \rightarrow [0, 1]$. 

1: Let $g$ be a mapping of $N_C \rightarrow [0, \infty]$. 
2: for all $A \in N_C$ do 
3: \hspace{1em} $g(A) := \infty$ 
4: \hspace{1em} for all $A' \in \Sigma$ do 
5: \hspace{2em} \hspace{1em} $g(A) := \min(d(e_A, e_{A'}), g(A))$. 
6: \hspace{1em} end for 
7: end for 
8: Let $f$ be a mapping of $N_C \rightarrow [0, 1]$. 
9: for all $A \in N_C$ do 
10: \hspace{1em} Find $i$, s.t. $A$ has the $i$-th smallest $g(A)$ in $N_C$. 
11: \hspace{1em} $f(A) := 1 - (i - 1)/|N_C|$. 
12: end for 
13: return $f$

NN-RANK first computes the distance from each concept name to each seed name in the vector space. In principle, many distance functions $d : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0, \infty]$ can be used to achieve this, but the Cosine distance, formulated as

$$d(e_A, e_B) = 1 - \frac{e_A \cdot e_B}{\|e_A\|_2 \|e_B\|_2}$$

has made the best measure of relevance in our experiments. $|\Sigma|$ distance values are computed in this way for each concept name $A$, while the smallest distance value, which denotes the shortest distance, is identified as a valid distance value of $A$ to $\Sigma$. NN-RANK then sorts all concepts names in $\mathcal{O}$ by their valid distance value. Concept names with smaller valid distance values are considered to be semantically more relevant to the seed signature, and thus to the central topic. These valid distance values (and the corresponding concept names) are then uniformly distributed between 0 and 1. The result is the relevance value of each $A$ w.r.t. $\Sigma$.

2.3 Relevance-based Seed Signature Extension

A natural question arises: how to use the computed relevance values to guide the selection of terms for ontology abstraction? Upon different application demands, the strategies may vary. Without a well-acknowledged gold standard, a feasible solution could be to measure the “degree” of relevance and define to what degree the relevance is a concept name can be thought of as “relevant” to the seeds in $\Sigma$. In this work, we use a threshold $\sigma$ at the scale of 0 to 1 to denote the “degree” of relevance. Our approach extends the primitive seed signature $\Sigma$ by adding to $\Sigma$ the concept names with relevance value no less than $\sigma$. The result is $\Sigma' = \Sigma \cup \{A \mid A \in \text{sig}(\mathcal{O}) \land f(A, \Sigma) \geq \sigma\}$. 

126 Xinhao Zhu, Xuan Wu, Ruiqing Zhao, Yu Dong and Yizheng Zhao
Computing $|\text{sig}(O)| \times |\Sigma|$ distances requires linear time to $|\text{sig}(O)|$, and the subsequent sorting requires linear time to $|\text{sig}(O)| \cdot \log(|\text{sig}(O)|)$. Hence, we have the following lemma regarding the time complexity of NN-RANK.

**Lemma 1.** Given any OWL ontology $O$ in SROIQ and a primitive seed signature $\Sigma \subseteq \text{sig}(O)$ with $n = |\text{sig}(O)|$ and $k = |\Sigma|$, our term selection approach always computes an extended seed signature $\Sigma'$ such that $\Sigma \subseteq \Sigma'$ in $O(n \log n + kn)$ time.

### 3 Empirical Evaluation of NN-RANK

In this experiment, we used NN-RANK to predict SNOMED CT Refset components. The aim was to show that the algorithm could enrich a given primitive seed signature $\Sigma$ with concept names highly relevant to the initial seeds (in a vector space). The experiment was conducted on a work station with an Intel Xeon CPU @2.60GHz and 32 GB memory.

SNOMED CT\(^5\) is currently the most comprehensive, multilingual clinical healthcare ontology in the world. A SNOMED CT Refset\(^6\) is a collection of SNOMED CT components sharing specific characteristics (e.g., a specific domain). An example of SNOMED CT Refset is the Malaria refset released by the National Resource Centre for EHR Standards in India, which includes findings, disorders, and organisms related to Malaria. Arguably, the refset published officially by a group of ontology engineers and domain experts, can be considered as a complete and precise standard of an Malaria abstract of SNOMED CT.

Our task was to predict concepts in SNOMED CT Refsets based on a seed signature (randomly or manually) selected from the refsets. This task was designed to fit with realistic scenarios where we needed to develop a new refset with least intervention from domain experts. We assumed that refsets developed by the domain experts were complete and precise fragments, containing concepts that were highly interconnected on the semantic level (e.g., in the same clinical domain). Therefore, the task of predicting SNOMED CT Refset components could be used to evaluate the performance of term selection models.

To better position our algorithm, we compared NN-RANK with two other term selection strategies, namely, a strategy adapted from locality-based modularization [10] (denoted as Star-modularization), and the signature-extension based on geographic connections [2] (denoted as Sig-Ext, configured with depth $d$). We treated them as baselines. The idea of the locality-based modularity strategy was to take all concept names in the computed module as the extended signature of the seed. This may not be ideal but was nevertheless a means to extend the seed signature. In this way, the relevance value $f(A, \Sigma)$ of $A$ was 1 if $A$ was in the signature of the computed module, and 0 otherwise. We also considered a comparison of NN-RANK with Meta-SVDD [7], a model designed for few-shot one-class-classification problems. Using Meta-SVDD, we learnt patterns about refsets from existing refsets, in order to enhance its performance in predicting new refset components.

---

\(^5\) [https://www.snomed.org/](https://www.snomed.org/)

\(^6\) [https://confluence.ihtsdotools.org/display/DOCGLOSS/refset](https://confluence.ihtsdotools.org/display/DOCGLOSS/refset)
We considered the International Edition of SNOMED CT (version July 2020), which contains 354,256 concepts, 355,214 logical axioms, and 1,506,185 description axioms. We used two sets of publicly accessible and in-use term collections, *NHS refsets*\(^7\) and *NRC refsets*\(^8\), as the target refsets.

The NHS refsets, issued by the National Health Service (NHS) in the UK, offered from the full Edition of SNOMED CT a set of components defined by a particular requirement. The NRC refsets were released by the National Resource Centre for EHR Standards (NRCeS) in India, which contained 30 standalone refsets covering concepts related to common diseases.

We adopted two metrics widely used in classification and ranking tasks, namely the Normalized Discounted Cumulative Gain (NDCG) and the Area under the ROC Curve (AUC), to evaluate the performance of term selection models. Both measures returned high values if a model made accurate predictions, i.e. they measured the similarity between the approximations and the refset components.

Ontology embedding generated by OWL2Vec\(^*\) on SNOMED CT was used for the concept embedding, where each concept was represented by a 200-dimensional vector. Different from the original OWL2Vec\(^*\) model, we used a fine-tuning process specially designed for this task, to further improve the ontology embedding. Specifically, refsets in this process were transformed to documents containing (concept_uri, refset_identifier, concept_uri) triples, then a Word2Vec model was used to fine-tune the pre-computed concept embedding on these documents. The fine-tuning process was done in a 10-fold cross validation manner, which meant that evaluations on any refset is based on a concept embedding fine-tuned on 90% refsets other than itself.

For NRC refsets, two seed signatures \(\Sigma_r\) and \(\Sigma_s\) consisted of \(K\) concepts respectively were used throughout the experiment. \(\Sigma_r\) was randomly selected among all the refset concepts, while \(\Sigma_s\) was manually selected with the aim that the \(K\) concepts it contained could describe the topic from different aspects. For NHS refsets, we only used a different set of \(\Sigma_r\) generated in the same way. It was crucial to be able to set the size of the primitive seed signature \(K\) accordingly to the application. In realistic use cases, the seed signature may be manually selected, where smaller \(K\) means less manual cost, so \(K = 5\) is used in the experiments.

We used the OWL API syntactic locality module extraction tool\(^9\) as the implementation of the locality-based module, and the official implementation of Sig-Ext. For Meta-SVDD, our implementation was based on the source code provided by [4].

### 3.1 Results and Analysis

The results (mean value ± standard deviation of the two measures) in Table 1 and 2 show that embedding-based methods outperformed logical approaches in the above settings. This was because logical methods were not designed for this task, and it did not capture lexical information of the ontology, which was crucial in determining the semantic relevance between concepts.

---

\(^7\) https://dd4c.digital.nhs.uk/dd4c/

\(^8\) https://www.nrces.in/resources#snomedct_releases

\(^9\) https://github.com/owlcs/owlapi
Besides, NN-RANK slightly outperformed Meta-SVDD, particularly when using \( \Sigma_s \). We will conduct a case study on the aforementioned Malaria refset to explain the mechanism and effectiveness of NN-RANK in this task.

Table 1: Results on NHS, NRC refsets using \( \Sigma_v \) (the higher the better).

<table>
<thead>
<tr>
<th>Methods</th>
<th>NHS refsets</th>
<th></th>
<th></th>
<th>NRC refsets</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NDCG</td>
<td>AUC</td>
<td></td>
<td>NDCG</td>
<td>AUC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>K=1</td>
<td>K=5</td>
<td>K=1</td>
<td>K=5</td>
<td>K=1</td>
<td>K=5</td>
</tr>
<tr>
<td>Star-modularization</td>
<td>40.93 ± 14.61</td>
<td>47.33 ± 13.36</td>
<td>50.84 ± 1.10</td>
<td>54.73 ± 5.56</td>
<td>49.10 ± 16.23</td>
<td>51.83 ± 14.62</td>
</tr>
<tr>
<td>Sig-Ext (d=1)</td>
<td>-</td>
<td>49.14 ± 10.92</td>
<td>-</td>
<td>54.31 ± 5.58</td>
<td>-</td>
<td>55.68 ± 11.06</td>
</tr>
<tr>
<td>Sig-Ext (d=2)</td>
<td>-</td>
<td>47.99 ± 11.66</td>
<td>-</td>
<td>54.31 ± 5.58</td>
<td>-</td>
<td>54.31 ± 11.83</td>
</tr>
<tr>
<td>Meta-SVDD</td>
<td>-</td>
<td>67.72 ± 33.26</td>
<td>-</td>
<td>91.55 ± 10.41</td>
<td>-</td>
<td>73.65 ± 16.62</td>
</tr>
<tr>
<td>NN-RANK</td>
<td>68.57 ± 20.36</td>
<td>77.93 ± 14.91</td>
<td>92.19 ± 11.19</td>
<td>96.49 ± 5.11</td>
<td>71.32 ± 14.33</td>
<td>77.25 ± 10.63</td>
</tr>
<tr>
<td>NN-RANK + fine-tuning</td>
<td>69.50 ± 20.13</td>
<td>78.76 ± 14.62</td>
<td>93.33 ± 9.57</td>
<td>96.98 ± 4.62</td>
<td>73.57 ± 12.54</td>
<td>80.19 ± 9.14</td>
</tr>
</tbody>
</table>

Table 2: Results on NRC refset using \( \Sigma_a \) (the higher the better).

<table>
<thead>
<tr>
<th>Methods</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NDCG</td>
<td>AUC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>K=1</td>
<td>K=3</td>
<td>K=5</td>
<td>K=1</td>
<td>K=3</td>
<td>K=5</td>
</tr>
<tr>
<td>Star-modularization</td>
<td>48.85 ± 16.68</td>
<td>50.65 ± 15.26</td>
<td>52.25 ± 14.42</td>
<td>50.82 ± 2.01</td>
<td>53.21 ± 5.53</td>
<td>54.68 ± 7.50</td>
</tr>
<tr>
<td>Sig-Ext (d=1)</td>
<td>49.97 ± 15.36</td>
<td>53.42 ± 12.16</td>
<td>55.76 ± 10.48</td>
<td>50.80 ± 1.53</td>
<td>52.14 ± 3.89</td>
<td>53.34 ± 5.81</td>
</tr>
<tr>
<td>Sig-Ext (d=2)</td>
<td>49.56 ± 15.82</td>
<td>52.33 ± 13.06</td>
<td>54.38 ± 11.36</td>
<td>50.87 ± 1.60</td>
<td>52.28 ± 4.01</td>
<td>53.48 ± 5.93</td>
</tr>
<tr>
<td>Meta-SVDD</td>
<td>71.28 ± 12.25</td>
<td>74.91 ± 16.48</td>
<td>75.24 ± 13.73</td>
<td>72.4 ± 16.23</td>
<td>86.83 ± 10.05</td>
<td>92.01 ± 6.45</td>
</tr>
<tr>
<td>NN-RANK</td>
<td>79.77 ± 11.79</td>
<td>83.67 ± 10.74</td>
<td>84.83 ± 9.95</td>
<td>94.07 ± 5.11</td>
<td>96.09 ± 3.73</td>
<td>96.64 ± 3.09</td>
</tr>
<tr>
<td>NN-RANK + fine-tuning</td>
<td>80.39 ± 12.02</td>
<td>84.41 ± 10.95</td>
<td>85.53 ± 10.20</td>
<td>94.65 ± 5.01</td>
<td>96.49 ± 3.73</td>
<td>96.97 ± 3.06</td>
</tr>
</tbody>
</table>

Figure 2 shows the distribution of the Malaria refset components and other SNOMED CT concepts in a 2-dimensional vector space. As illustrated in the figure, refset components tended to form a number of minor clusters, with each containing some highly semantically relevant concepts. The whole refset was composed of several concept clusters instead of a giant cluster. This meant that when two seed concepts \( A_1 \) and \( A_2 \) were given, any concept \( A \) that was similar to \( A_1 \) or \( A_2 \), i.e. \( d(e_{A_1}, e_{A_2}) < \epsilon \) or \( d(e_A, e_{A_2}) < \epsilon \) with \( \epsilon \) being a small value greater than 0, were more likely to be a refset component compared to another \( A \) which was similar to the average of \( e_{A_1} \) and \( e_{A_2} \), i.e., \( d(e_A, (e_{A_1} + e_{A_2})/2) < \epsilon \). NN-RANK was designed to fit in this multi-clusters pattern, and achieved better performance compared to other models utilizing concept embedding.

The performance of NN-RANK could be significantly enhanced when seed signatures described the topic from different aspects. For a high quality primitive seed signature like \( \Sigma_s \), an increased seed signature size would generally lead to more accurate selection results.

3.2 Time Efficiency

For the current setting of \( N = 354, 256, K = 5, D = 200 \) and using Cosine distance as the distance function, NN-RANK generated \( \Sigma ' \) within 5 seconds. For comparison,
it usually takes minutes to hours for other approaches (e.g., Star-modularization and Sig-Ext) to compute on a large-scale ontology like SNOMED CT, and five minutes for the Meta-SVDD model to converge in the same setting.

It is true that our approach takes hours to build embedding vectors on SNOMED CT, but this cost is acceptable in real-life scenarios since the training is conducted only once but can be meaningfully used many times and forever. Also, the training time can be adjusted. When the ontology contains less than 100K logical and annotation axioms, it is typically less than one hour.

Fig. 2: Distribution of malaria refset components and other SNOMED CT concepts (170 concepts from the malaria refset and 1700 random concepts outside of the refset). Each point corresponds to a SNOMED CT concept, whose colour shows its relevance with the seed signatures computed by NN-RANK (the higher, the deeper), and shape denotes its type (cross for being refset components, and circles for not). Seed concepts are depicted as blue stars accompanied with tags. The mappings between tag and label are: A - Malaria (disorder), B - Allergy to primaquine (finding), C - Accidental pyrimethamine poisoning (disorder), D - Malaria outbreak education (procedure), E - Antimalarial drug adverse reaction (disorder)
4 Case Study: Ontology Abstraction

In this part, we explored how input signature extended by NN-RANK benefits differently between modularization and uniform interpolation in the OWL ontology abstraction task.

As we need ontology having enough metadata to test the effectiveness of term selection method, we considered HeLiS\textsuperscript{10}, an $\mathcal{ALCHIQ}(\mathcal{D})$ ontology integrating knowledge about food and activity from a nutritional point of view. The experiment was based on HeLiS v1.10 which has 172,213 axioms, 277 concepts, and 50 roles.

4.1 Setup Details

First, we randomly generated 10 concept subsets from $\text{sig}(\mathcal{O}_{\text{HeLiS}})$ with the size of subsets ranged from 1 to 5. These randomly generated concept sets, denoted as $\Sigma_r$, could be the approximations of seed signatures around random topics. Then NN-RANK returned the ordered sets $\Sigma_r'$.

As the abstractions in real-life are usually small in size, we chose the top 10% of $\Sigma'$ (i.e., set the threshold as 0.9) to be the input signature for modularization and uniform interpolation. We used UI-FAME\textsuperscript{18} to compute uniform interpolants, and Star-modularization to compute locality-based modules as they are publicly accessible. Both preserved full logical entailments of the input signature $\Sigma'$ in $\mathcal{O}_{\text{HeLiS}}$\textsuperscript{10,13}. Then the abstraction results computed by these two tools with the input of $\Sigma'$ (denoted as $\Sigma'_{+\text{UI-FAME}}$, $\Sigma'_{+\text{Star-modularization}}$) were assessed with four metrics: module size $|M|$, module inherent richness $\text{InhRich}$, module intra distance $\text{IntraDist}$ and module cohesion $\text{Cohesion}$. A module with relative smaller size, higher inherent richness, relative smaller intra distance, and higher cohesion was said to be more compact. We also test $\Sigma_r'_{+\text{Star-modularization}}$ and compared it with $\Sigma_r'_{+\text{Star-modularization}}$.

4.2 Results and Analysis

We compared $\Sigma'_{+\text{UI-FAME}}$ and $\Sigma'_{+\text{Star-modularization}}$ to see the effectiveness of NN-RANK to different abstraction methods. From table 3, we can see that UI-FAME generated more compact abstractions. Besides, UI-FAME was sensitive to the input signature. These results make sense because locality-based modularization introduced other terms which were not in $\Sigma'$ but uniform interpolation stuck to $\Sigma'$. Experiments with thresholds setting as 0.3, 0.5, and 0.7 show that the size of $\Sigma'$ did not affect the compactness of the locality-based module abstraction.

Term selection allowed users to extend the seed signature in an adjustable way. For uniform interpolation, it is a key step to select suitable terms for the specified topic, because the semantics of the topic is mainly captured by the input terms. We observe that once if the input terms were not sufficient enough for uniform interpolation, the module could be very small, containing many meaningless axioms like $A \sqsubseteq \top$ or concept assertion axioms. NN-RANK+UI-FAME generated knowledge highly relative to the topic. For instance, in Table 4, the topic was “SpecialBread”. The related axioms in $\mathcal{O}_{\text{HeLiS}}$

\textsuperscript{10} https://horus-ai.fbk.eu/helis/
Table 3: Module compactness Evaluation (Use top 10% $\Sigma'$ as input. $|M|$ is the sum of quantities of concepts, roles and individuals in $M$. $\text{InhRich}$: the average number of subclasses per class. $\text{IntraDist}$: the overall distance between the entities in the module. $\text{Cohesion}$: the extent to which entities are related to each other in the module.)

<table>
<thead>
<tr>
<th>Metrics</th>
<th>K=1</th>
<th></th>
<th>K=5</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Star-modularization</td>
<td>UI-FAME</td>
<td>Star-modularization</td>
<td>UI-FAME</td>
</tr>
<tr>
<td>$</td>
<td>M</td>
<td>$</td>
<td>171 ± 14</td>
<td>20 ± 7</td>
</tr>
<tr>
<td>$\text{InhRich}$</td>
<td>2.92 ± 0.12</td>
<td>2.1 ± 1.25</td>
<td>4.08 ± 0.17</td>
<td>3.75 ± 0.49</td>
</tr>
<tr>
<td>$\text{IntraDist}$</td>
<td>49683.90 ± 94.61</td>
<td>618.75 ± 617.87</td>
<td>49798.70 ± 278.77</td>
<td>289.50 ± 344.26</td>
</tr>
<tr>
<td>$\text{Cohesion}$</td>
<td>0.08 ± 0.01</td>
<td>0.19 ± 0.09</td>
<td>0.08 ± 0.00</td>
<td>0.15 ± 0.10</td>
</tr>
</tbody>
</table>

Table 4: Term selection for SpecialBread topic in HeLiS

$\Sigma_r$ \{SpecialBread\}

$\mathcal{O}_{\text{fragment}}$ SpecialBread $\sqsubseteq$ Bread
\{SoyBread, OliveBread, MilkBread, OilBread, RyeBread\} $\sqsubseteq$ SpecialBread

$\Sigma'@10$ SpecialBread, Bread, WhiteBread, PizzaAndFocacciaBread, OlivesAndOliveProducts, SoyProducts, LegumesAndLegumeProducts, WheatFlour, WholeWheatFlour, MilkAndDairyProducts

were contained in $\mathcal{O}_{\text{fragment}}$. Clearly, “SpecialBread” had five individuals. Besides, these individuals had no other super-classes except “SpecialBread”. As commonsense knowledge, “OliveBread” can be “OlivesAndOliveProducts”, “SoyBread” can be “SoyProducts”, “MilkBread” can be “MilkAndDairyProducts”, which were missing in $\mathcal{O}_{\text{HeLiS}}$. So without the extension of NN-RANK, these related concepts could not be preserved in $\Sigma_r$ + Star-modularization or $\Sigma_r$ + UI-FAME. While NN-RANK could preserve them according to that “OlivesAndOliveProducts”, “SoyProducts”, and “MilkAndDairyProducts” were lexically close to the individuals of the topic concept “SpecialBread”.

To sum up, with NN-RANK modules and uniform interpolants produced more complete fragments. In addition, $\Sigma'$-uniform interpolation produced more precise fragments than $\Sigma'$-modularization.

5 Conclusion and Future Work

This paper makes a preliminary attempt to address the problem of extending the given seed signature with new terms selected sophisticatedly through embedding-based computation of important metadata of an OWL ontology. An evaluation of the approach on a predication task of a SNOMED CT refset shows that our approach makes accurate selections compared with other term selection baselines. A case study shows that high-quality modules and uniform interpolants of OWL ontologies can be produced using our term selection approach.
The absence of standardized benchmarks remains the main bottleneck in evaluating the performance of term selection methods. Hence, a number of pre-defined question answering instances that are generated based on the input ontology might be helpful in deciding the completeness and precision of the generated abstracts of OWL ontologies. For a problem $Q$ that can be answered by querying an ontology $O$, a satisfactory abstract $M$ of $O$ regarding a input signature $\Sigma$ should be able to answer $Q$ if $Q$ is relevant to $\Sigma$, and should not be able to answer $Q$ if $Q$ is not relevant to $\Sigma$.

Besides, the current experiments only considered concepts. Roles will also be considered in the future work.

References
