Automated Reasoning in Quantified Non-Classical Logics


Proceedings
Preface

This volume contains the proceedings of the Second International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2016), held July 1st, 2016, in Coimbra, Portugal. The workshop was affiliated and co-located with the International Joint Conference on Automated Reasoning (IJCAR 2016). The aim of the ARQNL 2016 Workshop has been to foster the development of proof calculi, automated theorem proving (ATP) systems and model finders for all sorts of quantified non-classical logics. The ARQNL workshop series provides a forum for researchers to present and discuss recent developments in this area.

Non-classical logics — such as modal logics, conditional logics, intuitionistic logic, description logics, temporal logics, linear logic, dynamic logic, fuzzy logic, paraconsistent logic, relevance logic — have many applications in AI, Computer Science, Philosophy, Linguistics, and Mathematics. Hence, the automation of proof search in these logics is a crucial task. For many propositional non-classical logics there exist proof calculi and ATP systems. But proof search is significantly more difficult than in classical logic. For first-order and higher-order non-classical logics the mechanization and automation of proof search is even more difficult. Furthermore, extending existing non-classical propositional calculi, proof techniques and implementations to quantified logics is often not straightforward. As a result, for most quantified non-classical logics there exist no or only few (efficient) ATP systems. It is in particular the aim of the ARQNL workshop series to initiate and foster practical implementations and evaluations of such ATP systems for non-classical logics.

The ARQNL 2016 Workshop received 6 paper submissions. Each paper was reviewed by at least three referees, and following an online discussion, 5 research papers were selected to be included in the proceedings. The ARQNL 2016 Workshop also included an invited talk by Revantha Ramanaya.

We would like to sincerely thank the invited speaker and all authors for their contributions. We would also like to thank the members of the Program Committee of ARQNL 2016 for their professional work in the review process. Furthermore, we would like to thank the Workshop Chair Reinhard Kahle and the Organizing Committee of IJCAR 2016. Finally, many thanks to all active participants of the ARQNL 2016 Workshop.

Berlin and Oslo, July 2016

Christoph Benzmüller
Jens Otten
Organization

Program Committee

Carlos Areces  
Universidad Nacional de Córdoba, Argentina

Christoph Benzmüller  
Freie Universität Berlin, Germany – co-chair

Walter Carnielli  
Centre for Logic, Epistemology and the History of Science, Brazil

Christian Fermüller  
TU Wien, Austria

Rajeev Goré  
The Australian National University, Australia

Andreas Herzig  
IRIT-CNRS, France

Stephan Merz  
INRIA Nancy, France

Till Mossakowski  
University of Magdeburg, Germany

Aniello Murano  
Università di Napoli “Federico II”, Italy

Hans De Nivelle  
University of Wroclaw, Poland

Jens Otten  
University of Oslo, Norway – co-chair

Valeria De Paiva  
University of Birmingham, UK

Giselle Reis  
INRIA Saclay, France

Julian Richardson  
Google Inc., USA

Luca Viganò  
King’s College London, UK

Workshop Chairs

Christoph Benzmüller  
Freie Universität Berlin  
Arnimallee 7, 14195 Berlin, Germany  
E-mail: c.benzmueller@fu-berlin.de

Jens Otten  
University of Oslo  
PO Box 1080 Blindern, 0316 Oslo, Norway  
E-mail: jeotten@ifi.uio.no
Contents

From Axioms to Proof Rules, Then Add Quantifiers 1–8
Revantha Ramanayake

Non-clausal Connection-based Theorem Proving in Intuitionistic First-Order Logic 9–20
Jens Otten

Sequent Calculi for Indexed Epistemic Logics 21–35
Giovanna Corsi and Eugenio Orlandelli

A Dynamic Logic for Configuration 36–50
Ching Hoo Tang and Christoph Weidenbach

TPTP and Beyond: Representation of Quantified Non-Classical Logics 51–65
Max Wisniewski, Alexander Steen and Christoph Benzmüller

Optimizing Inconsistency-tolerant Description Logic Reasoning 66–80
Mokarrom Hossain and Wendy MacCaull
FROM AXIOMS TO STRUCTURAL RULES, THEN ADD QUANTIFIERS.

REVANTHA RAMANAYAKE

We survey recent developments in the program of generating proof calculi for large classes of axiomatic extensions of non-classical logics by translating each axiom into a set of structural rules, starting from a base calculus. We will introduce three proof formalisms: the sequent calculus, the hypersequent calculus and the display calculus. The calculi that are obtained derive exactly the theorems of the logic and satisfy a subformula property which ensures that a proof of a theorem only contains statements that are ‘related to the conclusion’. These calculi can be used as a starting point for developing automated reasoning systems and to facilitate a proof-theoretic investigation of the logic (e.g. to prove interpolation, consistency, decidability, complexity). In the final section we discuss how first-order quantifiers may be added to these propositional calculi.

Much of the content here can be found in an extended form in the survey paper [10]. The main purpose of this abstract is to provide a concise ‘hands-on’ description of the methods. We present a subjective selection of problems while directing the reader to the references for a more exhaustive exposition.

1. SEQUENT CALCULUS

Let $L$ denote the intuitionistic language. The formulae from this language are given by the grammar:

\[ A, B ::= \text{propositional variable} \ p \mid \bot \mid \top \mid A \land B \mid A \lor B \mid A \rightarrow B \]

The set of theorems of propositional intuitionistic logic is denoted by $\text{Int}$. Define a sequent to be a tuple (denoted $X \vdash A$) where $X$ (antecedent) is a multiset of formulae and $A$ (succedent) is a formula. A sequent calculus $\text{slnt}$ for propositional intuitionistic logic $\text{Int}$ is given below.

\[
\begin{align*}
\bot & \vdash C & X & \vdash \top & p & \vdash p & X, Y \vdash C & (w) & X, Y \vdash C & (c) \\
A & \rightarrow B, X, Y \vdash C & \rightarrow l & A, X \vdash B & \rightarrow r & X \vdash A \rightarrow B & \rightarrow r & A_1, A_2, X \vdash C & \land l & A_1 \land A_2, X \vdash C \\
X & \vdash A & X & \vdash B & \land r & X \vdash A_1 & \lor r & A, X \vdash C & \lor l & A \lor B, X \vdash C
\end{align*}
\]

A derivation in the sequent calculus is defined recursively in the usual way as either an initial sequent or the object obtained by applying a rule to the sequents concluding a derivation. Given a multiset $A_1, \ldots, A_n$, let $\land X$ denote $A_1 \land \ldots \land A_n$ if $n > 0$ else $\top$. Then the relationship between the sequent calculus and the logic can be described as follows.
Theorem 1 (Gentzen [14]). Let \( X \) be a multiset and \( A \) a formula. Then \( X \vdash A \) is derivable in \text{sInt} if the formula \( \land X \rightarrow A \) is a theorem of \( \text{Int} \). Also \( \vdash A \) is derivable in \text{sInt} iff \( A \) is a theorem of \( \text{Int} \).

The theorem reveals that sequents represent a certain normal form for formulae, specifically conjunction of formulae \( \rightarrow \) formula. The point of note here is not that we have a proof calculus for the logic. After all, (axiomatic) Hilbert calculi for \( \text{Int} \) are well-known (see e.g. [5]). Instead, the point is that this proof calculus has the subformula property: every formula occurring in the premises occurs as a subformula of some formula in the conclusion. This means that any formula occurring in the derivation must occur in the conclusion.

From an automated reasoning perspective, this immediately suggests a backward proof-search procedure. The idea is to repeatedly applying proof-rules backwards. If a derivation is obtained, then the sequent is derivable. The subformula property tells us that the premise(s) are completely determined by the conclusion. Of course, further pruning and optimisation on backward proof-search is required to obtain a terminating efficient reasoning system.

Decidability is immediate: because of the weakening and contraction rules, we can replace the multisets in the antecedent with sets. Then every sequent appearing in backward proof search from \( \vdash A \) has the form \( Y \vdash B \) where \( Y \in \mathcal{P}(\Omega) \) and \( B \in \Omega \) where \( \Omega \) is the set of all subformulae in \( X \vdash A \) and \( \mathcal{P} \) is the powerset operator. There are \( |\mathcal{P}(\Omega)| \cdot |\Omega| \) possibilities. Now enumerate the trees with nodes labelled by such sequents (each node is permitted one or two children) such that the height of the tree is \( \leq |\mathcal{P}(\Omega)| \cdot |\Omega| \). If one of these trees is a derivation of \( \vdash A \) then \( A \) is a theorem. If not, there cannot possibly be a derivation of \( \vdash A \) so \( A \) is not a theorem. Moreover, it is possible to amend the rules such that every premise antecedent is \( \geq \) the conclusion antecedent. This corresponds to a pruning of the backward proof search procedure and leads to the tight PSPACE upper-bound for \( \text{Int} \).

Now suppose we wish to obtain a sequent calculus for the axiomatic extension of \( \text{Int} \) with the axiom \(( p \rightarrow q ) \lor (q \rightarrow p) \) (denoted \( \text{Int} + ( p \rightarrow q ) \lor (q \rightarrow p) \)). This formula is not a theorem of \( \text{Int} \) (e.g. argue that \( \vdash ( p \rightarrow q ) \lor (q \rightarrow p) \) is not derivable).

To obtain a sequent calculus for \( \text{Int} + ( p \rightarrow q ) \lor (q \rightarrow p) \), it would be tempting to add the sequent \( \vdash ( p \rightarrow q ) \lor (q \rightarrow p) \) to \text{sInt}. Unfortunately not every theorem of \( \text{Int} + ( p \rightarrow q ) \lor (q \rightarrow p) \) is derivable using \( \text{sInt} + \vdash ( p \rightarrow q ) \lor (q \rightarrow p) \). The addition of the (cut) rule below rectifies this: \( \vdash A \) derivable in \text{sInt} + (cut) \iff \( A \) is a theorem of \( \text{Int} + ( p \rightarrow q ) \lor (q \rightarrow p) \).

\[
\begin{align*}
X &\vdash A \\
A, Y &\vdash B
\end{align*}
\]

Unfortunately the calculus \( \text{sInt} + (\text{cut}) + \vdash ( p \rightarrow q ) \lor (q \rightarrow p) \) lacks the subformula property since the cut-formula \( A \) in (cut) need not appear in the conclusion. The key to applications is having a calculus with the subformula property so we need another solution.

2. Hypersequents

Theorem 1 reveals that every sequent can be read as a formula of the form conjunction of formulae \( \rightarrow \) formula. Let us call such a formula an implicational formula. It turns out that the obstacle for constructing a calculus for \( \text{Int} + ( p \rightarrow q ) \lor (q \rightarrow p) \) is that the sequent-representation is too restrictive. A solution is to move from an implicational formula to a finite disjunction of implicational formulae.
A hypersequent \([1, 23]\) is a non-empty multiset of sequents denoted as below. In particular, each \(X_i\) is a multiset of formulae and \(A_i\) is a formula.

\[
X_1 \vdash A_1 \quad \mid \quad X_2 \vdash A_2 \quad \mid \quad \ldots \quad \mid \quad X_{n+1} \vdash A_{n+1}
\]

A sequent in the hypersequent is called a component. The notion of derivability of a hypersequent is defined analogously to the sequent case. The rules of the hypersequent calculus \(h\text{Int}\) are obtained from \(s\text{Int}\) by appending \(g\) to each sequent and adding the rules below left and centre. The \(g\) is a schematic variable that can be omitted or instantiated with a hypersequent. Define the \((\text{cut})\) rule for the hypersequent calculus as below right.

\[
\frac{g \mid X \vdash B}{g \mid X \vdash B \mid Y \vdash A} \quad \text{ew} \quad \frac{g \mid X \vdash B \mid X \vdash B}{g \mid X \vdash B} \quad \text{ec} \quad \frac{g \mid X_1 \vdash A \mid h \mid A, Y \vdash B}{g \mid h \mid X, Y \vdash B} \quad \text{(cut)}
\]

**Theorem 2.** The hypersequent \(h\) in (1) is derivable in \(h\text{Int} + (\text{cut})\) iff the formula \((\wedge X_1 \rightarrow A_1) \vee \ldots \vee (\wedge X_{n+1} \rightarrow A_{n+1})\) (the interpretation \(h^I\) of \(h\)) is a theorem of \(\text{Int}\). Also \(\vdash A\) is derivable in \(h\text{Int} + (\text{cut})\) iff \(A\) is a theorem of \(\text{Int}\).

2.1. Translating suitable axioms into rules. The method introduced in [7] transforms suitable axiom into a structural rule (i.e. a rule which contains no logical connectives) and ultimately yields a calculus with the subformula property.

Starting from Theorem 2 it may be argued that \(h\) is derivable in \(h\text{Int} + (\text{cut}) + g \mid \vdash p \rightarrow q \mid \vdash q \rightarrow p\) if \(h\) is a theorem of \(\text{Int}\). Also \(\vdash A\) is derivable in \(h\text{Int} + (\text{cut})\) iff \(A\) is a theorem of \(\text{Int}\).

**Lemma 2.1** (Ackermann’s lemma). Let \(C\) be a hypersequent calculus extending \(s\text{Int} + (\text{cut})\). Let \(r_1, \ldots, r_4\) be the rules defined below where \(S\) is a set of hypersequents and the variables \(Y\) and \(\Pi\) do not appear other than where indicated. Then \(C + r_1\) and \(C + r_2\) (also \(C + r_3\) and \(C + r_4\)) derive the same hypersequents.

\[
\begin{align*}
S & \quad g \mid X \vdash A \quad r_1 \quad \frac{S}{g \mid A, X \vdash B} \quad l_1 \quad \frac{S}{g \mid X, Y \vdash B} \quad r_2 \\
& \quad S \quad g \mid A, Y \vdash B \quad r_2 \\
& \quad g \mid X \vdash B \quad l_1 \\
& \quad g \mid X, Y \vdash B \quad r_2 \\
& \quad g \mid A, X \vdash B
\end{align*}
\]

Setting \(g \mid p \vdash q \mid q \vdash p\) as the zero-premise rule \(\rho_0\), a single application of Ackermann’s lemma tells us that \(h\text{Int} + (\text{cut}) + \rho_0\) and \(h\text{Int} + (\text{cut}) + \rho_1\) derive the same hypersequents. Three further applications of Ackermann’s lemma yield that \(h\text{Int} + (\text{cut}) + \rho_2\) also derives the same hypersequents (equivalent calculus).

\[
\begin{align*}
& \frac{g \mid X_1 \vdash p \quad g \mid X_1 \vdash q \mid q \vdash p}{g \mid Y_1, X_1 \vdash B_1 \mid X_2 \vdash p} \quad \rho_1 \\
& \frac{g \mid X_1 \vdash p \quad g \mid X_2 \vdash q \quad g \mid Y_1, q \vdash B_1}{g \mid Y_1, X_1 \vdash B_1 \mid X_2 \vdash p} \quad \rho_3 \\
& \frac{g \mid X_1 \vdash p \quad g \mid X_2 \vdash q \quad g \mid Y_1, q \vdash B_1 \quad Y_2 \vdash B_2}{g \mid Y_1, X_1 \vdash B_1 \mid Y_2, X_2 \vdash B_2} \quad \rho_4
\end{align*}
\]
The hypersequent calculus $h\text{Int} + (\text{cut}) + \rho_4$ does not have the subformula property because the propositional variables $p$ and $q$ in the premise do not appear in the conclusion. To rectify this there is one final step: take the cut-closure on the premises of the rule. In effect we delete those premises which contain propositional variables which appear in either the antecedent or succedent but not both, and we apply cut in all possible ways to the remaining premises. This operation may not terminate in general (consider the situation when the same propositional variable appears in the same component of the antecedent and succedent). In the case of $\rho_4$ it does terminate to yield the rule

$$
\frac{g\mid X_1, Y_2 \vdash B_2 \quad g\mid X_2, Y_1 \vdash B_1}{g\mid Y_1, X_1 \vdash B_1 \mid Y_2, X_2 \vdash B_2} \quad \text{(com)}
$$

When cut-closure terminates it can be shown that it yields an equivalent structural rule. I.e. $h\text{Int} + (\text{cut}) + (\text{com})$ and $h\text{Int} + (\text{cut}) + \rho_4$ are equivalent. It remains to show that $h\text{Int} + (\text{cut}) + (\text{com})$ has cut-elimination i.e. $h\text{Int} + (\text{com})$ is an equivalent calculus. In turns out that the structural rules obtained by the above procedure satisfies sufficient conditions for cut-elimination (such rules are called analytic rules) so we are done.

The four steps are summarised below.

1. Given the axiom $\vdash A_1 \lor \ldots \lor A_{n+1}$ apply all possible invertible rules backwards to the hypersequent $g\mid \vdash A_1 \mid \ldots \mid \vdash A_{n+1}$
2. Use Ackermann’s lemma on each formula in the conclusion in order to obtain a rule where the conclusion contains no formulae.
3. Apply all possible invertible rules backwards to the premises of the rule. (Fail if one of the resulting hypersequents contain a compound formula which cannot be made propositional by the invertible rules.)
4. Delete premises containing propositional variables appearing only on one side. Apply all possible cuts to the remaining premises. (Fail if this step does not terminate.)

As we would expect, not all axioms can be handled by this method. If the nesting depth of logical connectives invertible in the antecedent/succedent is too great then item 3 will fail. In the context of intuitionistic logic cut-closure always terminates due to the presence of weakening and contraction. However this is not always the case in substructural logics.

2.2. Some open problems. Hypersequents in the calculus $h\text{Int}$ were built using components $X \vdash A$ where $X$ is a multiset. If we take $X$ as a list of formulae, then the exchange rule (ex) below left, which states that formulae in the list can be permuted, must be stated explicitly. Deleting structural properties of the comma such as exchange (ex), weakening (w), contraction (c) and associativity lead to hypersequent calculi for substructural logics. The substructural logics typically contain additional language connectives. For example, in the absence of (w) and (c), a commutative operator $\otimes$ distinct from $\land$ can be defined using the rules below left and right. An identity 1 for the $\otimes$ operator may also be defined.

$$
\frac{g\mid X, A, B, Y \vdash C}{g\mid X, A, B, Y \vdash C} \quad \text{(ex)} \quad \frac{g\mid X, A \vdash B \quad g\mid h \mid X, Y \vdash B}{g\mid h \mid X \otimes B \vdash B} \quad \text{\otimes r}
$$

Let $h\text{FL}_e$ denote the substructural hypersequent calculus lacking the weakening and contraction properties (and containing the rules for $\otimes$). The corresponding logic is the Full Lambek calculus with exchange (denoted $\text{FL}_e$).
• A hypersequent for the fuzzy logic MTL [13] (monoidal T-norm based logic) can be constructed by the procedure above: add the (w) and (com) rules to $hFL_e$. This logic is known to be decidable via a semantic proof. It would be interesting to obtain a proof by arguing directly on the hypersequent calculus. This in turn would yield an upper bound on the logical complexity. Meanwhile it is unknown if Involutive MTL is decidable. A hypersequent for this calculus can be obtained by amending the rules of the calculus to use a list of formulae in the succedent rather than a single formula. A syntactic proof of decidability for MTL may provide a pathway for obtaining a proof of decidability for IMTL.

• A hypersequent calculus for IUL [20] (involutive uninorm logic) can be obtained by the addition of (com) to $hFL_e$. A result of interest to the fuzzy logic community is if this logic is standard complete [16]. This in turn would follow by showing that whenever $g \vdash X, p \vdash U, V$ is derivable (propositional variable $p$ occurs only where indicated) then so is $g \vdash X, U \vdash Y, V$. Such an argument is called density elimination [20, 8]. Some automated solutions to this problem are described in [3]. See [21] for further details on these two problems. A more general open problem concerns the handling of axioms which fail this methodology. A solution is to venture to a more expressive proof formalism as described in the following section. In the context of modal logics, alternative approaches [19] to generating hypersequent calculi from axioms have also been investigated.

3. Display Calculus

Since item 3 above fails when the invertible rules cannot reduce compound formulae to propositional variables, it follows that the more invertible rules in the calculus, the more axioms that can be presented. Essentially, the hypersequent calculus was able to invert top-level disjunctions. The display calculus formalism [4] extends [24] the hypersequent formalism: the formula corresponding to a display sequent has a normal form which is broader and also more general in the sense that the normal form is based on the algebraic semantics of the logic.

Below we introduce the display calculus $\deltaBilnt$ [28] for bi-intuitionistic logic $Bilnt$. The language of $Bilnt$ extends the intuitionistic language with the coimplication $\leftarrow_d$. This is forced because the display calculus formalism requires that the logical connectives come in residuated pairs. To make the disjunction invertible on the right we added the semicolon on the right. Residuation then necessitated the addition of the structural connective $<$ standing for $\leftarrow_d$. An attractive feature of the display calculus is the general sufficient conditions [4] for cut-elimination.

\[
\begin{align*}
\forall p \vdash p & \quad \frac{\Gamma \vdash X \quad \top \vdash \top}{\top \vdash \top} \quad \frac{X \vdash I \quad \bot \vdash \bot}{I \vdash \bot} \quad \frac{I \vdash \top}{\top \vdash I} \\
A, B \vdash Y & \quad \frac{A \land B \vdash Y \quad \land}{X \vdash A \land B} \quad \frac{X \vdash A \rightarrow B \quad \rightarrow}{X \vdash A} \quad \frac{B \leftarrow_d A \vdash Y \quad \leftarrow}{X \vdash A; B} \quad \frac{X \vdash A \land B \quad \land}{X \vdash A \lor B} \quad \frac{X \vdash A \lor B \quad \lor}{X \vdash A \land B} \\
X \vdash A & \quad \frac{X \vdash A \land B \quad \land}{X \vdash A} \quad \frac{X \vdash B \quad \lor}{X \vdash A \land B} \quad \frac{X \vdash B \land Y \quad \land}{X \vdash A \land B} \quad \frac{X \vdash B \lor Y \quad \lor}{X \vdash A \land B} \\
A \rightarrow B & \quad \frac{A \rightarrow B \vdash X \rightarrow Y \quad \rightarrow}{X \vdash A \rightarrow B} \quad \frac{A \vdash Y \quad \leftarrow}{X \vdash A \land B} \quad \frac{A \vdash Y \quad \leftarrow}{X \vdash A \land B} \\
X \vdash B & \quad \frac{A \rightarrow B \vdash X \vdash Y \quad \rightarrow}{X \vdash A \land B} \quad \frac{A \vdash Y \quad \leftarrow}{X \vdash A \land B} \quad \frac{A \vdash Y \quad \leftarrow}{X \vdash A \land B}
\end{align*}
\]
The double lines indicate rules that can be applied in both directions. This calculus has the subformula property. The large number of structural connectives “∀”, “⊥”, “>”, “;” and “<” reflect the flexible normal form of the corresponding formulae.

**Theorem 3** ([9]). Let \( \mathcal{C} \) be a display calculus for the logic \( L \) satisfying the conditions in [9]. Also suppose that \( \{ r_i \} \in I \) are the analytic structural rules computed from the finite set \( \{ \alpha_j \} \in J \) of axioms using the analogous procedure to the one in Section 2. Then \( \mathcal{C} + \{ r_i \} \in I \) is a display calculus with the subformula property for \( L + \{ \alpha_j \} \in J \).

For example, the logic \( \text{BInit} + (p \to \bot) \lor (p \to \bot) \to \bot \) can be presented in this manner. Since \( \text{BInit} + (p \to \bot) \lor (p \to \bot) \) is conservative over \( \text{Int} + (p \to \bot) \lor (p \to \bot) \) (argue via the Kripke or algebraic semantics), we can obtain a display calculus for the latter by deleting the logical rules (not the structural rules!) introducing \(<\). Incidentally, proving the conservativity directly on the display calculus appears to be surprisingly difficult. The issue is with the interaction of the display rules introducing \(<\) and contraction.

It should be noted that a version of the method of extracting analytic structural rules from axioms appeared rather early on [17] in the context of display calculi for tense logics. Recently, the tools of unified correspondence theory have been applied [15] to extend that work in order to provide a new and uniform perspective on the axioms to rules paradigm.

## 4. First-order quantifiers

A cut-free hypersequent calculus \( \text{hln}^{fo} \) for first-order intuitionistic logic \( \text{ln}^{fo} \) is obtained by adding to \( \text{hln} \) the following rules for quantifiers [2, 6, 22]:

\[
\begin{align*}
g \vdash A(t), X \vdash B & \quad \forall l & g \vdash A(t) & \quad \forall r \\
g \vdash \forall x A(x), X \vdash B & \quad \forall l & g \vdash X \vdash \forall x A(x) & \quad \forall r \\
g \vdash A(a), X \vdash B & \quad \exists l & g \vdash X \vdash \exists x A(x) & \quad \exists r \\
g \vdash \exists x A(x), X \vdash B & \quad \exists l & g \vdash X \vdash \exists x A(x) & \quad \exists r
\end{align*}
\]

where the rules \((\forall r), (\exists l)\) have an eigenvariable condition: the free variable \( a \) must not occur in the lower hypersequent.

The addition of quantifiers to the hypersequent calculus can have some unexpected effects. For example, if we add the (com) rule to \( \text{hln}^{fo} \) then \( \vdash \forall x (A(x) \lor B) \to (\forall x A(x) \lor B) \) is derivable whenever \( x \) is not free in \( B \). The corresponding formula is known as the quantifier-shift or constant-domains formula and is not a theorem of \( \text{ln}^{fo} + (p \to q) \lor (q \to p) \) ! A non-standard hypersequent calculus for the latter logic (known as Corsi’s logic [12] or Gödel-Dummett logic with non-constant domains) has been introduced [25] where the eigenvariables are incorporated into the hypersequent syntax. These hypersequents have no formula-interpretation in the logic and it is not clear how to generalise the calculus to capture other logics.
4.1. Some open problems. The hypersequent calculus $\text{hInt}_{\text{fo}}^+ (\text{com})$ presents first-order Gödel logic. The question of interpolation for this logic is open. It is conceivable to investigate this problem via the hypersequent proof-theory although it has defied all such attempts thus far.

Meanwhile, the interaction between the (com) rule and the first-order quantifiers witnessed in first-order Gödel logic can be generalised to a new question: let $L$ be a propositional axiomatic extension of $\text{Int}$ and suppose that $\mathcal{C}$ is the hypersequent calculus for it, obtained by the methods of the previous section. Let $L_1$ be the first-order axiomatic extension of $L_1$ and let $L_2$ consist of those formulae $A$ such that $\vdash A$ is derivable in the extension of $\mathcal{C}$ by the first-order quantifier rules above. What then is the relationship between $L_1$ and $L_2$?

The only investigation of first-quantifiers for display calculi appears in [27]. Adding the obvious correspondents of the quantifier rules above to $\delta \text{BiInt}$ derives the quantifier-shift formula. Meanwhile classical predicate logic may be treated as a kind of propositional tense logic [26, 18], where the quantifiers $\exists$ and $\forall$ are treated as a residuated pair analogous to $\Diamond$ and $\Box$ (see [17]). However the resulting display calculus fails cut-elimination due to the presence of certain rules. A decidable minimal first-order system may be obtained by dropping these rules.

References

Non-clausal Connection-based Theorem Proving in Intuitionistic First-Order Logic

Jens Otten

Department of Informatics, University of Oslo
PO Box 1080 Blindern, 0316 Oslo, Norway
jeotten@leancop.de

Abstract

This paper introduces a non-clausal connection calculus for intuitionistic first-order logic. It is an extension of the non-clausal connection calculus for classical logic by prefixes and an additional prefix unification, which encode the Kripke semantics of intuitionistic logic. nanoCoP-i is a first implementation of this intuitionistic non-clausal connection calculus. Details of the compact Prolog code are presented, which extends the non-clausal connection prover nanoCoP for classical logic. Experimental evaluations on the ILTP and the TPTP problem libraries show a solid performance of the nanoCoP-i prover. In comparison to the ileanCoP prover, the resulting non-clausal proofs are not only shorter, but can also be more easily translated into, e.g., sequent proofs.

1 Introduction

Intuitionistic (or constructive) logic is one of the most popular non-classical logics. It is often used when constructing provably correct software, for example within the NuPRL [6] proof development system. Other interactive proof assistants, such as Coq [3], use a constructive logic as well. Hence, automated reasoning in intuitionistic logic is an important task and many applications would benefit from more powerful reasoning tools.

During the last decade the development of automated theorem proving (ATP) systems for classical first-order logic has made significant progress. Extending these ATP systems to intuitionistic logic is usually not (easily) possible, as many of the underlying calculi and techniques can not be adapted to intuitionistic logic. Most leading ATP systems for classical logic require the translation of the input formulae into a clausal form, i.e. into disjunctive or conjunctive normal form. For intuitionistic logic there is no validity-preserving translation into such a (simple) clausal form.

Nevertheless, ileanCoP, one of the fastest ATP systems for intuitionistic first-order logic, uses a clausal connection calculus and additional prefixes to capture the semantics of intuitionistic logic [16]. While the use of a clausal form technically simplifies the proof search, the standard translation as well as the definitional translation [22] into clausal form introduce a significant overhead into the proof search [17]. Furthermore, a translation into clausal form modifies the structure of the original formula and the translation of the clausal form proof back into one of a more readable proof of the original formula is not straightforward [24, 25]. On the other hand, fully automated theorem provers that use non-clausal calculi, such as standard tableau or sequent calculi, have a significant lower performance [16].

The paper is structured as follows. Section 2 introduces the non-clausal connection calculus for intuitionistic first-order logic, which works on prefixed non-clausal matrices. The non-clausal connection calculus and the prefix unification that is additionally required are presented. An implementation of the intuitionistic non-clausal connection calculus is described in Section 3. It provides details about the non-clausal prefixed matrices, the source code of the non-clausal proof search and the prefix unification algorithm. Section 4 presents an evaluation of the implementation on the ILTP and the TPTP problem libraries. The paper concludes with a short summary and an outlook on future work in Section 5.
2 The Intuitionistic Connection Calculus

The prefixed non-clausal matrix is the main concept used in the intuitionistic connection calculus. The intuitionistic connection calculus uses prefixes and requires an additional prefix unification algorithm.

2.1 Preliminaries

The standard notation for first-order formulae is used. Terms (denoted by $t$) are built up from functions (denoted by $f$), constants and (term) variables (denoted by $x$). An atomic formula (denoted by $A$) is built up from predicate symbols and terms. A \textit{(first-order) formula} (denoted by $F, G, H$) is built up from atomic formulae, the connectives $\neg, \land, \lor, \Rightarrow$, and the standard first-order quantifiers $\forall$ and $\exists$. A literal $L$ has the form $A$ or $\neg A$. Its complement $\overline{L}$ is $A$ if $L$ is of the form $\neg A$; otherwise $\overline{L}$ is $\neg L$.

A \textit{connection} is a set $[A, \neg A]$ of literals with the same predicate symbol but different polarity. A \textit{term substitution} $\sigma_Q$ assigns terms to variables.

Intuitionistic logic [7] and classical logic share the same \textit{syntax}, i.e. formulae in both logics use the same connectives and quantifiers, but their \textit{semantics} is different. For example, the formula

$$\text{man(Socrates)} \lor \neg \text{man(Socrates)}$$

(1)

is valid in classical logic, but not in intuitionistic logic. This property holds for all formulae of the form $A \lor \neg A$ for any atomic formula $A$. In classical logic this formula is valid as either $A$ is true or $\neg A$ is true whether $A$ is true or not true. The semantics of intuitionistic logic requires a proof for $A$ or for $\neg A$. As this property neither holds for $A$ nor for $\neg A$, the formula is not valid in intuitionistic logic. Formally, the semantics of intuitionistic logic is specified by a Kripke semantics [30].

2.2 Prefixed Non-clausal Matrices

The intuitionistic non-clausal connection calculus is based on Wallen’s \textit{matrix characterization} [29, 30] and uses prefixes to encode the Kripke semantics for intuitionistic logic.

\textbf{Definition 2.1 (Prefix).} A prefix (denoted by $p$) is a string, i.e., a sequence of characters over an alphabet $\Phi \cup \Psi$, in which $\Phi$ is a set of prefix variables ($V_1, \ldots$) and $\Psi$ is a set of prefix constants ($a_1, \ldots$).

Semantically, a prefix encodes a sequence of worlds in a Kripke model. Proof-theoretically, prefixes and variables represent applications of the rules $\neg$-right, $\Rightarrow$-right, $\forall$-right, and $\neg$-left, $\Rightarrow$-left, $\forall$-left in the sequent calculus [8], respectively. The prefix $p$ of a subformula $G$, denoted $G : p$, specifies the sequence of rules that have to be applied (bottom-up) to obtain $G$ in the sequent. In order to preserve the atomic formulae that form an axiom in the intuitionistic sequent calculus, their prefixes need to unify under an intuitionistic substitution $\sigma_J$. Hence, in the matrix characterization for intuitionistic logic, it is required that the prefixes of the literals in every connection unify under $\sigma_J$ [30].

\textbf{Definition 2.2 (Intuitionistic substitution, $\sigma$-complementary).} An intuitionistic substitution $\sigma_J : \Phi \rightarrow (\Phi \cup \Psi)^*$ maps elements of $\Phi$ to strings over $\Phi \cup \Psi$. For a combined substitution $\sigma := (\sigma_Q, \sigma_J)$, a connection $[L_1 : p_1, L_2 : p_2]$ is \textit{$\sigma$-complementary} iff $\sigma_Q(L_1) = \sigma_Q(L_2)$ and $\sigma_J(p_1) = \sigma_J(p_2)$.

An additional \textit{domain condition} on $\sigma$ ensures that $\sigma_Q$ and $\sigma_J$ are mutually consistent [30]. The \textit{irreflexivity test} of the reduction ordering [30] is realized by the \textit{occurs check} during the term and prefix unification. To this end, the \textit{skolemization} technique, originally used to eliminate \textit{eigenvariables} in classical logic, is extended and also used for prefix constants in intuitionistic logic [15]. For the extended skolemization, the same skolem function symbol is used for instances of the same subformula, a technique that is similar to the \textit{liberalized $\delta^+$-rule} in classical tableau calculi [10].
Table 1: The definition of the prefixed non-clausal matrix for intuitionistic logic

<table>
<thead>
<tr>
<th>Type</th>
<th>$F^{\text{pol}} : p$</th>
<th>$M(F^{\text{pol}} : p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic</td>
<td>$A^0 : p$</td>
<td>${{A^0 : pa^+}}$</td>
</tr>
<tr>
<td></td>
<td>$A^1 : p$</td>
<td>${{A^1 : pV^+}}$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\neg(G^0) : p$</td>
<td>$M(G^1 : pa^+)$</td>
</tr>
<tr>
<td></td>
<td>$(G \land H)^1 : p$</td>
<td>${{M(G^1 : p)}}, {{M(H^1 : p)}}$</td>
</tr>
<tr>
<td></td>
<td>$(G \lor H)^0 : p$</td>
<td>${{M(G^0 : p)}}, {{M(H^0 : p)}}$</td>
</tr>
<tr>
<td></td>
<td>$(G \implies H)^0 : p</td>
<td>$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$(G \land H)^0 : p$</td>
<td>${{M(G^0 : p), M(H^0 : p)}}$</td>
</tr>
<tr>
<td></td>
<td>$(G \lor H)^1 : p$</td>
<td>${{M(G^1 : p), M(H^1 : p)}}$</td>
</tr>
<tr>
<td></td>
<td>$(G \implies H)^1 : p</td>
<td>$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$\forall x G^0 : p$</td>
<td>$M(G[x \land t^1] : pV^+)$</td>
</tr>
<tr>
<td></td>
<td>$(\exists x G)^0 : p$</td>
<td>$M(G[x \land t^0] : p)$</td>
</tr>
<tr>
<td>$\delta$</td>
<td>$\forall x G^1 : p$</td>
<td>$M(G[x \land t^1] : pa^+)$</td>
</tr>
<tr>
<td></td>
<td>$(\exists x G)^1 : p$</td>
<td>$M(G[x \land t^0] : p)$</td>
</tr>
</tbody>
</table>

An intuitionistic non-clausal matrix is a set of prefixed clauses, which consist of prefixed literals and prefixed (sub)matrices. The polarity 0 or 1 is used to represent negation in a matrix, i.e. literals of the form $A$ and $\neg A$ are represented by $A^0$ and $A^1$, respectively.

Definition 2.3 (Intuitionistic non-clausal matrix). Let $F$ be a formula, $\text{pol}$ be a polarity, and $p$ be a prefix. The prefixed (non-clausal) matrix $M(F^{\text{pol}} : p)$ of a prefixed formula $F^{\text{pol}} : p$ is a set of prefixed clauses, in which a prefixed clause is a set of prefixed literals and prefixed (non-clausal) matrices, and defined inductively according to Table 1. $G[x \land t]$ denotes the formula $G$ in which all free occurrences of $x$ are replaced by $t$. $x^*$ is a new term variable, $t^+$ is the Skolem term $f^*(x_1, \ldots, x_n)$ in which $f^*$ is a new function symbol and $x_1, \ldots, x_n$ are all free term and prefix variables in $(\forall x G)^0 : p$ or $(\exists x G)^1 : p$. $V^+$ is a new prefix variable, $a^*$ is a prefix constant of the form $f^*(x_1, \ldots, x_n)$ in which $f^*$ is a new function symbol and $x_1, \ldots, x_n$ are all free term and prefix variables in $A^0 : p$, $(\neg(G \land H))^0 : p$, $(G \implies H)^0 : p$, or $(\forall x G)^0 : p$. The intuitionistic (non-clausal) matrix $M(F)$ of $F$ is the prefixed matrix $M(F^0 : \epsilon)$.

In the graphical representation of a matrix, its clauses are arranged horizontally, while the literals and (sub-)matrices of each clause are arranged vertically.

For example, the formula

$$\forall x (\text{man}(Socrates) \implies \text{man}(Socrates))$$

$$\land ((\text{man}(Plato) \land \forall x (\text{man}(x) \implies \text{mortal}(x))) \implies \text{mortal}(Plato))$$

(2)

has the (simplified, i.e. redundant brackets are removed) intuitionistic non-clausal matrix

$$\{\{\{\text{man}(Socrates)^0 : a_1 V_1\}\}, \{\{\text{man}(Socrates)^1 : a_1 a_2\}\}, \{\{\text{man}(Plato)^1 : a_3 V_2\}\}, \{\{\text{man}(x)^0 : a_3 V_3 V_4 a_4 V_3, x, V_4\}, \{\text{mortal}(x)^1 : a_3 V_3 V_4 V_5\}, \{\text{mortal}(Plato)^0 : a_3 a_5\}\}\}$$

(3)

and the following graphical representation

$$\begin{bmatrix}
[\text{man}(Socrates)^0 : a_1 V_1] & [\text{man}(Socrates)^1 : a_1 a_2] \\
[\text{man}(Plato)^1 : a_3 V_2] & [\text{man}(x)^0 : a_3 V_3 V_4 a_4 V_3, x, V_4] \\
& [\text{mortal}(x)^1 : a_3 V_3 V_4 V_5] & [\text{mortal}(Plato)^0 : a_3 a_5]
\end{bmatrix}.$$

11
2.3 An Intuitionistic Non-clausal Connection Calculus

The non-clausal connection calculus for intuitionistic logic extends the non-clausal connection calculus for classical logic [18] and generalizes the clausal connection calculus for intuitionistic logic [15]. According to the matrix characterization of logical validity [4, 5, 30] a formula $F$ is valid, if and only if all paths through its matrix $M(F)$ (with added clause copies) contain a $\sigma$-complementary connection. The (non-clausal) connection calculus uses a connection-driven search strategy. In each reduction and extension step of the calculus a $\sigma$-complementary connection is identified and only paths that do not contain this connection are investigated afterwards. If every path contains a $\sigma$-complementary connection, the proof search succeeds and the given formula is valid. In contrast to sequent calculi, connection calculi permit a more goal-oriented proof search. This leads to a significantly smaller search space and, thus, to a more efficient proof search. A non-clausal connection proof can be illustrated within the graphical matrix representation.

For example, the proof of formula (2) and its matrix (3) consists of three connections, which are represented by a line in the following graphical matrix representation. The three connections are $\sigma$-complementary with $\sigma = (\sigma_Q, \sigma_J)$ and $\sigma_Q(x) = Plato$, $\sigma_J(V_1) = a_2$, $\sigma_J(V_2) = a_4(\varepsilon, Plato, \varepsilon)$, $\sigma_J(V_3) = \varepsilon$, $\sigma_J(V_4) = \varepsilon$, $\sigma_J(V_5) = a_5$ (where $\varepsilon$ is the empty string).

The intuitionistic matrix of formula (1) is

$$\{\{man(Socrates)^0 : a_1\}, \{man(Socrates)^1 : a_2\}\}.$$  \hspace{1cm} (4)

There is only one connection $\{man(Socrates)^0 : a_1, man(Socrates)^1 : a_2\}$ which is $\sigma$-complementary if the two prefixes $a_1$ and $a_2$ can be unified. There is no substitution $\sigma_J$ with $\sigma_J(a_1) = \sigma_J(a_2)$ and, hence, no connection proof of this matrix. Therefore, the formula $man(Socrates) \lor \neg man(Socrates)$ is not valid in intuitionistic logic.

A few additional concepts are required as follows in order to specify which clauses can be used within the generalized non-clausal extension rule.

**Definition 2.4** (alpha-related, parent clause, clause copy). A clause $C$ contains a literal $L$ (or clause $C'$) iff $L \in C$ or $C'$ contains $L$ ($C = C'$ or $C'$ contains $C'$) for a matrix $M' \in C$ with $C' \in M$. A clause $C$ is $\alpha$-related to a literal $L$ iff it occurs besides $L$ in the graphical matrix representation; more precisely, $C$ is $\alpha$-related to a literal $L$ iff $\{C', C''\} \subseteq M'$ for some matrix $M'$ such that $C'$ contains $L$ and $C''$ contains $C$. $C'$ is a parent clause of $C$ iff $M' \in C'$ and $C \in M'$ for some matrix $M'$. In the copy of a clause $C$ all free variables in $C$ are replaced by fresh variables. $M[C_1 \setminus C_2]$ denotes the matrix $M$, in which the clause $C_1$ is replaced by the clause $C_2$.

For example, in the matrix (3) the clauses $\{man(Plato)^1 : a_3V_2\}$ and $\{mortal(Plato)^0 : a_3a_5\}$ are $\alpha$-related. $\{\{man(Socrates)^1 : a_1V_1\}, \{man(Socrates)^1 : a_2a_2\}\}$, $\{\{man(Plato)^1 : a_3V_2\}, \{man(x)^0 : a_3V_3a_4(V_3, x, V_4)\}, \{mortal(x)^1 : a_3V_3V_4V_5\}, \{mortal(Plato)^0 : a_3a_5\}\}$ is the parent clause of all other clauses in the matrix, e.g., of the clause $\{man(Socrates)^0 : a_1V_1\}$.
Axiom (A) \[ \{\}, M, \text{Path} \]

Start (S) \[ C_2, M, \{\} \] and \( C_2 \) is copy of \( C_1 \in M \)

Reduction (R) \[ C, M, \text{Path} \cup \{L_2 : p_2\} \quad \text{and} \quad \{L_1 : p_1, L_2 : p_2\} \text{ is } \sigma\text{-complementary} \]

Extension (E) \[ C_3, M[C_1 \setminus C_2], \text{Path} \cup \{L_1 : p_1\} \quad C, M, \text{Path} \]

Decomposition (D) \[ C \cup C_1, M, \text{Path} \quad \text{and} \quad C_1 \in M_1 \]

\begin{figure}[h]
\centering
\begin{tabular}{|c|}
\hline
Axiom (A) \{\}, M, Path \\
\hline
Start (S) \[ C_2, M, \{\} \] and \( C_2 \) is copy of \( C_1 \in M \)
\hline
Reduction (R) \[ C, M, Path \cup \{L_2 : p_2\} \quad \text{and} \quad \{L_1 : p_1, L_2 : p_2\} \text{ is } \sigma\text{-complementary} \]
\hline
Extension (E) \[ C_3, M[C_1 \setminus C_2], Path \cup \{L_1 : p_1\} \quad C, M, Path \]
\hline
Decomposition (D) \[ C \cup C_1, M, Path \quad \text{and} \quad C_1 \in M_1 \]
\hline
\end{tabular}
\caption{The intuitionistic non-clausal connection calculus}
\end{figure}

Definition 2.5 (Extension clause, \( \beta\)-clause). \( C \) is an extension clause (e-clause) of the matrix \( M \) with respect to a set of literals \( \text{Path} \) iff either (a) \( C \) contains a literal of \( \text{Path} \), or (b) \( C \) is \( \alpha\)-related to all literals of \( \text{Path} \) occurring in \( M \) and if \( C \) has a parent clause, it contains a literal of \( \text{Path} \). In the \( \beta\)-clause of \( C_2 \) with respect to \( \text{Path} \), denoted by \( \beta\)-clause \( \text{Path} \cup \{L_1 : p_1\} \), \( C_2 \) contains \( L_2 : p_2 \), \( \{L_1 : p_1, L_2 : p_2\} \) is \( \sigma\)-complementary.

The non-clausal connection calculus for intuitionistic logic has the same axiom, start rule, and reduction rule as the clausal connection calculus [15]. The extension rule is slightly modified and a decomposition rule is added. It is an extension of the non-clausal connection calculus for classical logic [16], in which a prefix is added to each literal and an additional intuitionistic substitution is used to identify \( \sigma\)-complementary connections.

Definition 2.6 (Intuitionistic non-clausal connection calculus). The axiom and the rules of the intuitionistic (non-clausal) connection calculus are given in Fig. 1. It works on tuples \( "C, M, \text{Path}" \), where \( M \) is a prefixed non-clausal matrix, \( C \) is a prefixed (subgoal) clause or \( \varepsilon \) and (the active) \( \text{Path} \) is a set of prefixed literals or \( \varepsilon \). \( \sigma = (\sigma_Q, \sigma_J) \) is a combined term and prefix substitution. An intuitionistic (non-clausal) connection proof of a prefixed matrix \( M \) is an intuitionistic connection proof of \( \varepsilon, M, \varepsilon \).

The non-clausal connection calculus for intuitionistic logic is correct and complete, i.e. a formula \( F \) is valid in intuitionistic logic if and only if there is an intuitionistic non-clausal connection proof of its intuitionistic non-clausal matrix \( M(F) \). It follows from the correctness and completeness of the non-clausal connection calculus [16] and the clausal connection calculus for intuitionistic logic [15].

The analytic, i.e., bottom-up proof search in the non-clausal calculus is carried out in the same way as in the clausal calculus. Additional backtracking might be required when choosing the clause \( C_1 \) in the decomposition rule; no backtracking is required when choosing the matrix \( M_1 \). The rigid term and intuitionistic substitutions \( \sigma_Q \) and \( \sigma_J \) are calculated whenever a reduction or extension rule is applied. The term substitution is calculated by one of the well-known algorithms for term unification. The intuitionistic substitution is calculated by a prefix unification algorithm (see Section 2.4). Optimization techniques, such as positive start clauses, regularity, lemmata and restricted backtracking, can be employed in a similar way as in the clausal connection calculus for intuitionistic logic [16].
2.4 Prefix Unification

The intuitionistic substitution \( \sigma_j \) is calculated by a prefix unification algorithm [15]. For a given set of prefix equations \( \{p_1 = q_1, \ldots, p_n = q_n\} \), an appropriate substitution \( \sigma_j \) is a unifier such that \( \sigma_j(p_i) = \sigma_j(q_i) \) for all \( 1 \leq i \leq n \). General algorithms for string unification exist, but the following unification algorithm is more efficient, as it takes the prefix property of the prefixes \( p_1, p_2, \ldots \) of a formula into account: for two prefixes \( p_i = u_i X w_i \) and \( p_j = u_j X w_j \) with \( X \in \Phi \cup \Psi \) the property \( u_1 = u_2 \) holds. This reflects the fact that prefixes correspond to sequences of connectives and quantifiers within the same formula.

**Definition 2.7.** The prefix unification for the prefixes equation \( \{p = q\} \) is carried out by applying the rewriting rules R1 to R10 in Figure 2. It is \( V, V', V'' \in \Phi \) with \( V \neq V' \), \( V' \) is a new prefix variable, \( a, b \in \Psi \), \( X \in \Phi \cup \Psi \), and \( u, w, v \in (\Phi \cup \Psi^*) \). For rule 10 the restriction \( \{\} \neq a \in v \) or \( w \neq a \) or \( V \in \Psi \) applies. \( \sigma_j(V) = u \) is written \( \{V, u\} \). The unification starts with the tuple \( \{(p = q), \{\}\} \). The application of a rewriting rule \( E \to E' \), \( \tau \) replaces the tuple \( (E, \sigma_j) \) by the tuple \( (E', \tau(\sigma_j)) \). \( E \) and \( E' \) are prefix equations, \( \sigma_j \) and \( \tau \) are substitutions. The unification terminates when the tuple \( \{(\), \( \sigma_j\} \) is derived. In this case, \( \sigma_j \) represents a most general unifier. Rules can be applied non-deterministically and lead to a minimal set of most general unifiers.

![Figure 2: The prefix unification algorithm for intuitionistic logic](image)

In the worst-case, the number of unifiers grows exponentially with the length of the prefixes \( p \) and \( q \). To solve a set of prefix equations \( E = \{p_1 = p_1, \ldots, q_n = q_t\} \), the equations in \( E \) are solved one after the other and each calculated unifier is applied to the remaining prefix equations in \( E \).

For example, for the prefix equation \( \{a_1 V_2 V_3 = a_3 a_3\} \), there are the two possible derivations:

1. \( \{a_1 V_2 V_3 = e|a_3 a_3\}, \{\} \xrightarrow{R3} \{V_2 V_3 = e|a_3\}, \{\} \xrightarrow{R6} \{V_3 = e|a_3\}, \{V_2 \backslash e\} \xrightarrow{R10} \{V_3 = a_3 | e\}, \{V_2 \backslash e\} \)
2. \( \{a_1 V_2 V_3 = e|a_1 a_3\}, \{\} \xrightarrow{R3} \{V_2 V_3 = e|a_3\}, \{\} \xrightarrow{R10} \{V_2 V_3 = a_3 | e\}, \{\} \xrightarrow{R5} \{V_3 = e|e\}, \{V_2 \backslash a_3\}, \{V_2 \backslash a_3\} \)

They yield the most general unifiers \( \{V_2 \backslash e, V_3 | a_3\} \) and \( \{V_2 \backslash a_3, V_3 \backslash e\} \).

3 An Intuitionistic Connection Prover

The following implementation of the intuitionistic non-clausal connection calculus of Fig. 1 follows the lean methodology [2], which is already used for the clausal connection provers leanCoP [21] and ileanCoP [16]. It uses very compact Prolog code to implement the basic calculus and adds a few essential optimization techniques in order to prune the search space. The resulting natural nonclausal connection prover for intuitionistic logic nanoCoP-i is available under the GNU General Public License and can be downloaded at [http://www.leancop.de/nanocopi/](http://www.leancop.de/nanocopi/).
3.1 Prefixed Non-clausal Matrices

In a first step the input formula $F$ is translated into its intuitionistic non-clausal matrix $M(F)$ according to Table 1; redundant brackets of the form “$\{\ldots\}$” are removed [18]. Additionally, every (sub-)clause $(I, V, FV): C$ and (sub-)matrix $J : M$ are marked with unique indices $I$ and $J$; clauses $C$ are also marked with a set $V$ of (free) term and prefix variables and a set $FV$ of prefixed (free) term variables of the form $x:p$ that are newly introduced in $C$ but not in any subclause of $C$. Atomic formulae are represented by Prolog atoms, term and prefix variables by Prolog variables and the polarity 1 by “−”.

Sets, e.g. clauses and matrices, are represented by Prolog lists (representing multisets); prefixes are also represented by Prolog lists and marked with the polarity of the corresponding literal.

For example, the matrix 3 from Sec. 2.2 is represented by the Prolog term

\[
\begin{align*}
(2^K)[1]\cdot[j1]\cdot[i1]: & \quad \begin{cases}
4^K:\{(5^K)[1]\cdot[j1]: \neg (\text{man(socrates)}): -([3^1])\}, \\
7^K:\{(7^K)[1]: \text{man(socrates):}[3^1]\} , \\
10^K:\{(10^K)[1]: \neg (\text{man(plato)}): -([8^1])\}, \\
13^K:\{8^K]\cdot[w,v]\cdot[x]: 
\{\text{man}(x):[8^1],w,v\}, \\
19^K:\{19^K][1]: \{\text{mortal}(\text{plato}):[8^1]\} \}
\end{cases}
\end{align*}
\]

in which the Prolog variable $K$ is instantiated later on in order to enumerate clause copies; as an optimization, prefix characters introduced by atomic formulae are only considered during the unification. In the second step the matrix $M(F)$ is written into Prolog’s database. For every literal $\text{Lit}$ in $M$ the fact

\[
\text{lit(Lit,ClaB,ClaC,Grnd)}
\]

is asserted into the database where $\text{ClaC} \in M$ is the (largest) clause in which $\text{Lit}$ occurs and $\text{ClaB}$ is the $\beta$-clause of $\text{ClaC}$ with respect to $\text{Lit}$. $\text{Grnd}$ is set to $n$ if the smallest clause in which $\text{Lit}$ occurs is ground, i.e. does not contain any variables; otherwise $\text{Grnd}$ is set to $n$. Storing literals of $M$ in the database in this way is called lean Prolog technology [17] and integrates the advantages of the Prolog technology approach [27] into the lean theorem proving framework. No other modifications or simplifications of the original formula (structure) are done during these two preprocessing steps.

3.2 Intuitionistic Non-clausal Proof Search

The nanoCoP-i source code is shown in Fig. 3. It is an extension of the non-clausal connection prover nanoCoP [20] for classical first-order logic. The underlined text was added to the source code of nanoCoP: (1) prefixes are added to all literals, (2) the sets $\text{PreS}$ and $\text{VarS}$ are added, which contain prefix equations and free (prefixed) term variables, respectively, and (3) a prefix unification is added.

First, nanoCoP-i performs a classical proof search, in which the prefixes of each connection are stored in $\text{PreS}$. If the search succeeds, the domain condition is checked and the prefixes in $\text{PreS}$ are unified (line 4), using the predicates $\text{domain_cond}$ and $\text{prefix_unify}$ (see Section 3.3), respectively.

The predicate $\text{prove(Mat,PathLim,Set,Proof)}$ implements the start rule (lines 1–4) and iterative deepening on the length of the active path (lines 5–9). $\text{Mat}$ is the prefixed matrix generated in the preprocessing step, $\text{PathLim}$ is the maximum size of the active path used for iterative deepening, $\text{Set}$ is a list of options used to control the proof search, and $\text{Proof}$ contains the returned intuitionistic (non-clausal) connection proof. Start clauses are restricted to positive clauses (line 2); after member selects a start clause, $\text{positiveC(Cla,Cla1)}$ returns the clause $\text{Cla1}$ in which all clauses that are not positive in $\text{Cla}$ are deleted. A clause is positive if all of its elements (matrices and literals) are positive; a matrix is positive if it contains at least one positive clause; a literal is positive if its polarity is 0.

The predicate $\text{prove(Cla,Mat,Path,PathI,PathLim,Lem,PreS,VarS,Set,Proof)}$ implements the axiom (line 10), the decomposition rule (lines 11–16), the reduction rule (lines 17–20, 24–26, 37–38), and the extension rule (lines 17–20, 28–49) of the non-clausal connection calculus in Fig. 1.
Non-clausal Connection-based Theorem Proving in Intuitionistic First-Order Logic

J. Otten

% start rule
prove(Mat,PathLim,Set,[(I^0)\text{V}:Clai\{Proof\}]):= (1)
  member((I^0)\text{V}:VS:Clai,Mat), positiveC(Clai,Clai1), (2)
  prove(Clai,Mat,{},[(I^0),PathLim,{}],Pres,VarS,Set,Proof). (3)
append(VarS,VS,VarS1), domain_cond(VarS1), prefix_unify(PreS). (4)
prove(Mat,PathLim,Set,Proof):= (5)
  retract(pathlim) -> (6)
  ( member(comp(PathLim),Set) -> prove(Mat,1,{},Proof) ; (7)
    PathLim1 is PathLim+1, prove(Mat,PathLim1,Set,Proof) ) ; (8)
  member(comp(_),Set) -> prove(Mat,1,{},Proof). (9)

% axiom
prove({{}},...,{},[],[],[],[]). (10)

% decomposition rule
prove([J^K:Mat1:Clai],MI,Path,PI,PathLim,Lem,PreS,VarS,Set,Proof):= !, (11)
member(I^0:TV:Clai,Mat1), (12)
prove(Clai,MI,Path,[I,J^K|PI],PathLim,LEM,Pres1,VarS1,Set,Proof1), (13)
prove(Clai,MI,Path,PI,PathLim,Lem,Pres2,VarS2,Set,Proof2), (14)
append(Pres2,Pres1,Pres), append(FV,VarS1,VarS3), (15)
append(Pres3,VarS3,VarS), append(Proof1,Proof2,Proof). (16)

% reduction and extension rules
prove([Lit:Pre|Clai],MI,Path,PI,PathLim,Lem,Pres,VarS,Set,Proof):= (17)
  Proof=[[I^V:[NegLit:ClaiB1]|Proof1]|Proof2], (18)
  \+( member(LitC,[Lit:Pre|Clai]), member(LitP,Path), LitC=LitP), (19)
  \(-NegLit=Lit;Lit=NegLit -> (20)
    ( member(Lit,Lem), Lit:Pre==LitL, PreS3=[], VarS3=[]), (21)
    ClaB1=[], Proof1=[] ; (22)
    ; (23)
    member(NegLit:Pres,Path), unify_with_occurs_check(NegL,NegLit), (24)
    \+ prefix_unify([Pre=PreN], PreS3=[Pre=PreN], VarS3=[]), (25)
    ClaB1=[], Proof1=[] ; (26)
    ; (27)
    lit(NegLit:PreN,ClaiB,ClaB1,M1I), (28)
    (Grndl=q -> true ; length(Path,K), K<PathLim -> true ; (29)
     \+ pathlim -> assert(pathlim), fail ), (30)
     \+ prefix_unify([Pre=PreN]), (31)
     prove_ec(ClaiB,Clai,MI,PI,I^V:TV:ClaiB1,M1I), (32)
     prove(ClaiB1,MI,[Lit:Pre|Path],I^V:PI,PathLim,Lem,Pres1,VarS1, (33)
     Set,Proof1), Pres3=[Pre=PreN|Pres1], append(VarS1,FV,VarS3), (34)
     ), (35)
     ( member(cut,Set) -> ! ; true ), (36)
     prove(Clai,MI,Path,PI,PathLim,[Lit:Pre|Lem],Pres2,VarS2,Set,Proof2) \+ (37)
     append(Pres2,PresS,Pres), append(VarS3,VarS,VarS3). (38)

% extension clause (e-clause)
prove_ec((I^K)^V:ClaiB,IV:Clai,MI,PI,ClaiB1,M1I):= (39)
  append(MIA,[{I^K}:V1:Clai1(M1B),MI]), length(PI,K), (40)
  ( Clai=[I^K:Clai2]|ClaiB2, member(J^K1,PI), (41)
    unify_with_occurs_check(V1,V1), Clai=[Clai2]|ClaiB2), (42)
    append(Clai,B,J^K1:M12|Clai,E,Clai), (43)
    prove_ec(ClaiB2,Clai2,M12,PI,ClaiB1,M1I), (44)
    append(Clai,B,J^K1:M13|Clai,E,Clai), (45)
    append(MIA,[I^K}:V1:Clai3(M1B),MI1) (46)
  ; (47)
  \+(member(I^K1,PI);\(\backslash\)=V1;\(\backslash\)=[]), (48)
  ClaiB1=(I^K)^V:ClaiB, append(MIA,[I^K:Clai1(M1B),MI1]). (49)

Figure 3: Source code of the nanoCoP-i prover

16
Cla, Mat, and Path represent the subgoal clause C, the prefixed matrix M and the (active) Path.

The indexed path PathI contains the indices of all clauses and matrices that contain literals of Path; it is used for calculating extension clauses. The list Lem is used for the lemmata rule and contains all literals that have been “solved” already [17]. PreS and VarS are lists of prefixes and free (prefixed) term variables, respectively. Set is a list of options and may contain the elements “cut” and “comp(I)” for I ∈ IN, which are used to control the restricted backtracking technique [17]. This prove predicate succeeds iff there is an intuitionistic connection proof for the tuple (Cla, Mat, Path) with |Path| < PathLim. In this case Proof returns a compact intuitionistic connection proof. The prefixed input matrix Mat has to be stored in Prolog’s database (as explained above). The substitution σ is stored implicitly by Prolog and also applied to the variables returned in Proof. The predicate prove_ec(ClaB, Cla1, Mat, ClaB1, Mat1) is used to calculate extension clauses (lines 39–49).

nanoCoP uses additional optimization techniques that are already used in the classical connection provers leanCoP [17] and nanoCoP [20]: regularity (line 19), lemmata (line 21), and restricted backtracking (line 36). Restricted backtracking is a very effective (but incomplete) technique for pruning the search space in connection calculi [17]. It is switched on if the list Set contains the element “cut”. If it also contains “comp(I)” for I ∈ IN, then the proof search restarts again without restricted backtracking if the path limit PathLim exceeds I.

3.3 Prefix Unification

The source code of the prefix unification is shown in Figure 4. Each clause R1 to R10 corresponds to one of the rewrite rules defined in Figure 2. The predicate tunify(S, [], T) succeeds if the two prefixes S and T can be unified. The second argument contains the left part of the right prefix. The prefix variables are instantiated with a most general unifier; alternative unifiers are calculated via backtracking. As the skolemization technique is applied to prefix constants as well, a term unification with unify_with_occurs_check is required whenever a prefix constant is unified with another prefix constant or variable. The predicate prefix_unify(G) solves a set of prefix equations (lines a–d).

(a) prefix_unify([]).
(b) prefix_unify([S=T|G]) :- (-S2=S -> T2=T ; -S2=T, T2=S),
   flatten([S|_],S1), flatten(T2,T1),
   tunify(S1,[],T1), prefix_unify(G).
(c) tunify([],[],[]).
(d) tunify([],[],[X|T]) :- tunify([X|T],[],[]).
(e) tunify([X1|S],[],[X2|T]) :- (var(X1) -> (var(X2), X1==X2);
   \+var(X2), unify_with_occurs_check(X1,X2)),
   !, tunify(S,[],T).
(f) tunify([C|S],[],[V|T]) :- \+var(C), \+, var(V), tunify([V|T],[],[C|S]).
(g) tunify([V|S],X,[]) :- unify_with_occurs_check(V,Z), tunify(S,[],[]).
(h) tunify([V|S],[],[C1|T]) :- \+var(C1), \+, var(V), append([C1],V1),
   unify_with_occurs_check(V,V1),
   tunify(S,[],[C1|T]).
(i) tunify([V|S],Z,[C1,C2|T]) :- var(V1),
   unify_with_occurs_check(V1,T),
   tunify([V|S],Z,[C1,C2|T]).
(j) tunify([V1|T],[],[V|S]) :- var(V1),
   unify_with_occurs_check([V|T],[],[V|S]).
(k) tunify([V1|T],[],[V|S]) :- var(V1),
   append([V1|T],[],[V|S]),
   unify_with_occurs_check(V,V2),
   tunify([V|S],[],[V|S]).
(l) tunify([V|S],Z,[X|T]) :- (S=[]; T]); \+var(X) ->
   append(Z,[X],Z1), tunify([V|S],Z1,T).

Figure 4: Source code of the prefix unification
4 Experimental Evaluation

The following evaluations were conducted on a 3.4 GHz Xeon system with 4 GB of RAM running Linux 3.13.0 and ECLiPSe Prolog 5.10. The CPU time limit was set to 10 seconds.

4.1 ILTP Library

The ILTP problem library [23] contains 2550 first-order formulae in various problem domains. Table 2 shows the number of proved problems of the ILTP library v1.1.2 for the intuitionistic theorem provers JProver, itizenTAP, ft, ileanCoP, and nanoCoP-i.

JProver [25] is based on a (simple) prefixed non-clausal connection calculus for intuitionistic first-order logic [12]; it is implemented in OCaml. itizenTAP [14] implements a prefixed free-variable tableau calculus for intuitionistic first-order logic; it is implemented in Prolog. ft [26] is a C implementation of an analytic tableau calculus for intuitionistic first-order logic and uses many additional optimization techniques. ileanCoP [15, 16] implements a prefixed clausal connection calculus for intuitionistic first-order logic and is implemented in Prolog. In order to make the results comparable to nanoCoP-i, the (Prolog) core prover of ileanCoP was used with the standard translation (“[nodef]”) and the definitional translation (“[def]”) into the (prefixed) clausal form. nanoCoP-i was tested with and without restricted backtracking technique, i.e. Set=[] and Set=[cut,comp(6)], respectively.

Table 2: Results on ILTP library v1.1.2

<table>
<thead>
<tr>
<th></th>
<th>JProver</th>
<th>itizenTAP</th>
<th>ft</th>
<th>ileanCoP</th>
<th>nanoCoP-i</th>
</tr>
</thead>
<tbody>
<tr>
<td>proved</td>
<td>205</td>
<td>1.17</td>
<td>1.23</td>
<td>1.2</td>
<td>1.0</td>
</tr>
<tr>
<td>0 to 1sec.</td>
<td>239</td>
<td>295</td>
<td>323</td>
<td>487</td>
<td>509</td>
</tr>
<tr>
<td>1 to 10sec.</td>
<td>11</td>
<td>8</td>
<td>5</td>
<td>42</td>
<td>51</td>
</tr>
</tbody>
</table>

nanoCoP-i proves more problems than both clausal translations of ileanCoP. The “full” version of ileanCoP1.2 (using the TPTP syntax translation of nanoCoP-i and a shell script to implement the strategy scheduling) proves 717 problems. The proofs found by nanoCoP-i are in general shorter than those found by ileanCoP. The proof size is the number of connections (and applications of the lemma rule). Compared to ileanCoP [nodef], 95% of the nanoCoP-i proofs are on average 30% shorter; 4% of the proofs are larger, 1% have the same size. Compared to ileanCoP [def], 97% of the nanoCoP-i proofs are on average 33% shorter; 2% of the proofs are larger, 1% have the same size.

4.2 TPTP Library

Table 2 shows the results on all 3644 first-order (FOF) problems of the TPTP library v3.3.0 [28].

Table 3: Results on TPTP library v3.3.0

<table>
<thead>
<tr>
<th></th>
<th>JProver</th>
<th>itizenTAP</th>
<th>ft</th>
<th>ileanCoP</th>
<th>nanoCoP-i</th>
</tr>
</thead>
<tbody>
<tr>
<td>proved</td>
<td>177</td>
<td>251</td>
<td>260</td>
<td>636</td>
<td>652</td>
</tr>
<tr>
<td>0 to 1sec.</td>
<td>171</td>
<td>248</td>
<td>258</td>
<td>566</td>
<td>572</td>
</tr>
<tr>
<td>1 to 10sec.</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>70</td>
<td>80</td>
</tr>
</tbody>
</table>

In the TPTP Library, the results showed that nanoCoP-i again performed better than ileanCoP, with 717 problems being proved. The proof size is still relevant, with the average size of nanoCoP-i proofs being 33% shorter than the ileanCoP proofs, while 2% of the nanoCoP-i proofs are larger or the same size as the ileanCoP proofs.
Again, nanoCoP-i proves more problems than both clausal translations of ileanCoP. The “full” version of ileanCoP 1.2 proves 932 problems. Compared to ileanCoP [nodef], 92% of the nanoCoP-i proofs are on average 28% shorter; 7% of the proofs are larger, 1% have the same size. Compared to ileanCoP [def], 94% of the nanoCoP-i proofs are on average 34% shorter; 5% of the proofs are larger, 1% have the same size.

5 Conclusion

This paper introduces a non-clausal connection calculus for intuitionistic first-order logic and nanoCoP-i, a compact implementation of this calculus. Using prefixed non-clausal matrices, the proof search works directly on the original structure of the input formula; no translation steps to any clausal or other normal form are required. This combines the advantages of more natural non-clausal tableau or sequent provers with the goal-oriented efficiency of connection provers.

Even though the non-clausal inferences introduce a slight overhead, nanoCoP-i outperforms both clausal form translations of the ileanCoP core prover on the ILTP and the TPTP problem library. It is expected that the integration of strategy scheduling into nanoCoP-i will also outperform the “full” ileanCoP prover. More than 90% of the returned non-clausal proofs are on average about 30% shorter than their clausal counterparts.

Both, the standard translation as well as a definitional translation [22] into clausal form not only increase the size of the formula, but also modify the structure of the original formula. This makes it difficult to translate the resulting proof back into a proof of the original formula, an effect that has already been observed for classical logic [24].

By considering the intuitionistic substitution, the returned connection proof can be translated into an intuitionistic sequent proof [8], making nanoCoP-i an ideal tool to be used within interactive proof systems, such as Coq [3], Isabelle [13], HOL [9] or NuPRL [6].

In contrast to the calculus used in nanoCoP-i, the non-clausal connection calculus used in JProver [25] does not add clause copies dynamically during the proof search. Instead they are added iteratively, which introduces a huge redundancy into the proof search. Hence, the performance of nanoCoP-i is significantly higher than the performance of JProver.

Future work includes the adaption of the non-clausal connection calculus and the nanoCoP-i prover to other non-classical logics, such as modal or description logics, for which so far only clausal connection calculi exist [19]. Integrating search techniques into nanoCoP-i in order to obtain a decision procedure is another important task. Furthermore, optimization techniques that are used for classical logic, such as strategy scheduling [17], learning [11] and variable splitting [1], could be integrated into an intuitionistic non-clausal connection calculus as well.

References

Non-clausal Connection-based Theorem Proving in Intuitionistic First-Order Logic  

10. Otten, J.:  
Indexed epistemic logics constitute a well-structured class of quantified epistemic logics with great expressible power and a well-behaved semantics based on the notion of epistemic transition model. It follows that they generalize term-modal logics. As to proof theory, the only axiomatic system for which we have a completeness theorem is the minimal system $\mathbf{Q.K}_{\omega}$, whether with classical or with free quantification. This paper proposes a different approach by introducing labelled sequent calculi. This approach turns out to be very flexible and modular: for each class of epistemic transition structures $C^\ast$ considered in the literature, we introduce a $\mathbf{G3}$-style labelled calculus $\mathbf{GE}_{\ast}$. We show that these calculi have very good structural properties insofar as all rules are height-preserving invertible (hp-invertible), weakening and contraction are height-preserving admissible (hp-admissible) and cut is admissible. We will also prove that each calculus $\mathbf{GE}_{\ast}$ characterizes the class $C^\ast$ of indexed epistemic structures.

1 Introduction

Indexed epistemic logics (IEL) are quantified multi-agent epistemic logics characterized by the fact that epistemic formulas are expressions like

$$\langle t : \ast \rangle Px$$

meaning ‘the agent (denoted by) $t$ knows of (the denotation of) $s$ that it is a $P$’, see [1, 2]. As in term-modal logics (TML), [3], agents are denoted by terms, and therefore it is possible to quantify on agents and reason about groups of agents, e.g. ‘every $Q$ knows of $s$ that it is $P$’ is expressed as $\forall y(Qy \to \langle y : \ast \rangle Px)$, and ‘every $Q$ knows that $Ps$’ is expressed as $\forall y(Qy \to \langle y : \ast \rangle Ps)$. IEL are more general than TML in many respects: (i) they allow for non-rigid designators, i.e. the denotation of a term can vary from world to world, thus the logic doesn’t impose that the agents know every true identity; (ii) they are based on a counterpart-theoretic semantics, which, as shown in [1], permits a better treatment of de re modalities.

In [1, 2] many interesting IEL are semantically introduced, but almost no complete proof-theoretic characterization is given, the only exception is an axiomatic characterization of the formulas valid on all structures. As is well known, axiomatic systems are not well suited for automated reasoning and, in the case of quantified modal logics, they do not allow modular proofs of completeness. For these aspects the labelled $\mathbf{G3}$-style calculi studied in [4, 5, 7] behave...
better: they allow cut-free sequent calculi for a wide class of modal logics and to give modular proofs of completeness obtained by a mechanical procedure that determines a derivation for every valid sequent and an appropriate countermodel for every invalid one. This paper applies the methodology of labelled calculi to IEL: for every IEL semantically defined in [1, 2] a labelled calculus is introduced and proved complete. A constructive proof that the structural rules of inference are admissible is given, thus solving a problem left open in [1, p. 1182], where it was raised the question of finding ‘a cut free sequent calculus of the classically quantified epistemic logic $Qe.K + GF$’.

In the sequel to this section the language and the semantics of IEL are sketched. Section 2 introduces labelled sequent calculi for IEL. Section 3 shows that the calculi introduced have the good structural features that are distinctive of G3-style calculi: all rules are hp-invertible, the structural rules of weakening and contraction are hp-admissible and the rule of cut is admissible. Section 4 deals with soundness and completeness.

**Language.** Consider a first-order language $\mathcal{L}$ whose signature contains individual constants and predicate symbols including identity. Let $Var$ be an infinite set of variables. Terms are defined as usual. The logical symbols are $\bot, \rightarrow, \forall$, and, for $n \geq 0$, $[t : x_1 \ldots x_n]$, where $x_1, \ldots, x_n$ are pairwise distinct variables and $t, x_1, \ldots, x_n$ are terms; when $n = 0$, we write $[t : \star]$. $[t : x_1 \ldots x_n]$ stands for $[t : x_1^1 \ldots x_n^1]$. We will also use $x$ and $t$ for $n$-tuples of variables and terms, respectively.

**Definition 1.1.** The notions of formula, $A$, and of its free variables, $fv(A)$, are defined simultaneously as follows:

- If $t_1, \ldots, t_n$ are terms and $P$ is an $n$-ary predicate, then $P t_1, \ldots, t_n$ is a(n atomic) formula whose free variables are all the variables among $t_1, \ldots, t_n$;
- $\bot$ is a formula with no free variables;
- If $B$ and $C$ are formulas, then $(B \rightarrow C)$ is a formula and $fv(B \rightarrow C) = fv(B) \cup fv(C)$;
- If $B$ is a formula and $x$ a variable, then $\forall xB$ is a formula and $fv(\forall xB) = fv(B) - \{x\}$;
- If $B$ is a formula whose free variables are among $x_1, \ldots, x_n$, then $[t : s_1^1 \ldots s_n^1]B$ is a formula, where $t, s_1, \ldots, s_n$ are terms. The free variables of $[t : s_1^1 \ldots s_n^1]B$ are all (and only) the variables occurring in $t, s_1, \ldots, s_n$.

We use $\equiv$ for syntactical identity. For the sake of simplicity, it will always be assumed that the bound variables occurring in a formula are different from the free ones. This can be achieved by renaming bound variables. Formulas that differ only in the name of the bound variables are taken to be identical.

**Definition 1.2.** The expression $s[t/x]$ stands for $t$ if $s \equiv x$, else it is $s$.

The notion of substitution of a term $t$ for a variable $x$ in a formula $A$, $A[t/x]$, is defined thus:

\[
\begin{align*}
(Ps_1, \ldots, s_n)[t/x] &= df Ps_1[t/x], \ldots, s_n[t/x] \\
\bot[t/x] &= df \bot \\
(B \rightarrow C)[t/x] &= df B[t/x] \rightarrow C[t/x] \\
(\forall yB)[t/x] &= df \begin{cases} 
\forall yB & \text{if } y \equiv x, \\
\forall z((B[z/y])[t/x]) & \text{if } y \not\equiv x \text{ and } y \equiv t, \\
\forall y(B[t/x]) & \text{if } y \not\equiv x \text{ and } y \not\equiv t,
\end{cases} \\
(|s : s_1^1 \ldots s_n^1|B)[t/x] &= df |s[t/x] : s_1^1[t/x] \ldots s_n^1[t/x]|B.
\end{align*}
\]
Two comments on the notion of substitution. Having identified formulas that differ only in the name of bound variables, we are able to circumvent problems related to ‘captured’ variables by first renaming the bound variables and then by performing the problematic substitution. Second, in epistemic formulas the substitution is carried out inside the epistemic operator and not inside the formula that follows it as it is usually done. This is one of the reasons why the lower tuple of an indexed epistemic operator contains all the free variables of the formula that follows it. In this way we gain a better control of substitutions in epistemic contexts, see [1, 2].

We will also make extensive use of simultaneous substitutions of the terms $t_1, \ldots, t_m$ for the free variables $x_1, \ldots, x_n$. This notion can be defined in terms of sequences of simple substitutions as follows:

$$A[s_1, \ldots, s_n, t/x_1, \ldots, x_n, y] \equiv (A[s_1, \ldots, s_n/x_1, \ldots, x_n])[t/y], \text{ where } y \neq x_i \text{ and } y \neq s_i.$$  

**Semantics.** Given the epistemic formula $|t : \exists y Px|$ considered in (1), its truth conditions are as follows: where $a$ and $b$ are the denotations of $t$ and $s$, respectively, in a world $w$, the formula above is true in $w$ if in every world that is compatible with $a$’s knowledge every object that is a counterpart of $b$ w.r.t. $a$’s knowledge satisfies the open formula $Px$; in other words, if every way that $b$ may be that is compatible with $a$’s knowledge satisfies $Px$. In order to render such semantical conditions formally, we need to introduce a few notions.

Let $W$ be a non-empty set of possible worlds, an *epistemic transition model* is, intuitively speaking, a family of extensional double-domain models $\{(U_w, D_w, I_w) : w \in W\}$ whose elements are related in two different ways: (i) by a *compatibility relation* between elements of the domain of some model and the other models: $a \prec b$ means that the world $v$ is compatible with $a$’s knowledge. (ii) By a *counterpart relation* between elements of the domains of (not necessarily) different models, modulo an agent: $b \xrightarrow{\sigma} c$ means that according to the knowledge of agent $a$, if $c \in U_v$ and $a \prec v$, $c$ represents in $v$ the object that is $b$ in $w$.

**Definition 1.3 (e-model).** An e-model is a tuple $\mathcal{M} = \langle W, U, D, \prec, \rightarrow, I \rangle$ where

- $W$ is a non-empty set of worlds.
- $U = \{U_w : w \in W\}$ is a family of pairwise disjoint non-empty sets indexed by members of $W$: the outer domains.
- $D = \{D_w : D_w \subseteq U_w\}$ is a family of inner domains indexed by members of $W$.
- $\prec \subseteq U \times W$ is the compatibility relation between agents and worlds: $a \prec w$ means that the world $w$ is compatible with $a$’s knowledge.
- $\rightarrow = \{\xrightarrow{\sigma} : a \in U\}$ is a family of counterpart relations indexed by agents:
  - $\xrightarrow{\sigma} = \{a \times U_w \times U_v : a \in U_w \text{ and } a \prec v\}$ intuitively $b \xrightarrow{\sigma} c$ means that $c$ is a counterpart of $b$ according to $a$’s knowledge.
- $I$ is a function associating to every $w \in W$ a first-order interpretation $I_w$, defined over $U_w$, in particular:
  - $I_w(c) \in U_w$;
  - $I_w(P^n) \subseteq (U_w)^n$; and $I_w(\neg) = \{(a, a) : a \in U_w\}$.

We say that the e-model $\mathcal{M} = \langle W, U, D, \prec, \rightarrow, I \rangle$ is based on the e-frame $\mathcal{F} = \langle W, U, D, \prec, \rightarrow \rangle$.

Assignments are defined world by world: a $w$-assignment $\sigma$ is a mapping from the set $\text{Var}$ of variables to $U_w$. Given a $w$-assignment $\sigma$ and $a \in U_w$, we use $\sigma^{\text{Var}}_w$ for the $w$-assignment that maps $x$ to $a$ and behaves like $\sigma$ on all other variables. When no ambiguity arises, we use $\sigma(t)$
to denote not only the object assigned by the \(w\)-assignment \(\sigma\) to the variable \(t\), but also the object assigned by \(I_w\) to the constant \(t\).

**Definition 1.4.** Satisfaction of a formula \(A\) at \(w\) under \(\sigma\) in \(\mathcal{M}\), \(\sigma \models^\mathcal{M}_w A\), is so defined:

\[
\begin{align*}
\sigma &\models^\mathcal{M}_w \bot \\
\sigma &\models^\mathcal{M}_w Pt_1, \ldots, t_n \quad \iff \quad \langle \sigma(t_1), \ldots, \sigma(t_n) \rangle \in I_w(P) \\
\sigma &\models^\mathcal{M}_w B \rightarrow C \\
\sigma &\models^\mathcal{M}_w \forall x B \\
\sigma &\models^\mathcal{M}_w \langle t : x_1, \ldots, x_n \mid B \rangle \\
\sigma &\models^\mathcal{M}_w \langle x \mid A \rangle \\
\end{align*}
\]

for all \(a \in D_w, a \models^\mathcal{M}_w B\)

for all \(v\) s.t. \(\sigma(t) < v\) and for all \(v\)-assignment \(\tau\) s.t. 

\[\sigma(s_i) \rightarrow^\tau(x_i), 1 \leq i \leq n, \text{we have that } \tau \models^\mathcal{M}_v B\]

The notions of truth in a world, \(\models^\mathcal{M}_w A\), truth in a model, \(\models^\mathcal{M} A\), and validity in a frame, \(\mathcal{F} \models A\), are defined in the standard way.

**Relevant classes of \(e\)-frames.** Some results of correspondence between classes of \(e\)-frames and epistemic formulas are listed here, see [1, Sect. 5] for their proofs.

**Proposition 1.5.**

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
<th>is valid on</th>
<th>(e)-frames such that</th>
</tr>
</thead>
</table>
| \(D_e\) | \(|t : x|A \rightarrow \neg|t : x|A\) | serial | (a) \(\forall a \in U_w \exists v \in W (a < v)\)  
(b) \(\forall a, b \in U_w \exists c \in U_v (b \rightarrow c)\) |
| \(T_c\) | \(|t : x|A \rightarrow A\) | reflexive | (a) \(\forall a \in U_w (a < w)\)  
(b) \(\forall a, b \in U_w (b \rightarrow b)\) |
| \(4_e\) | \(|t : x|A \rightarrow |t : x|A \mid |y : x|A\) | transitive | (a) \(\forall a \in U_w \forall b \in U_v (a \rightarrow b \land b < u \supset a < u)\)  
(b) \(\forall a, b \in U_w \forall c, d \in U_v \forall v \in U_w (a \rightarrow d \land b \rightarrow c \land c \rightarrow e \supset b \rightarrow e)\) |
| \(B_e\) | \(A \rightarrow |t : x|A \mid \neg|y : x|A\) | symmetric | (a) \(\forall a \in U_w \forall b \in U_v (a \rightarrow b \supset b < w)\)  
(b) \(\forall a, b \in U_w \forall c, d \in U_v (a \rightarrow d \land b \rightarrow c \land c \rightarrow d \supset b)\) |

**\(CB_{BF}\)** | \(|t : x|yA \rightarrow \forall y|t : x|yA\) | \(D\)-preservative | \(\forall a \in U_w \forall b \in D_w \forall c \in U_v\)  
(b) \(a \rightarrow c \supset c \in D_w\)  
(c) \(a \rightarrow b\) |
| **\(BF_{c}\)** | \(\forall y|t : x|yA \rightarrow |t : x|yA\) | \(D\)-surjective | \(\forall a \in U_w \forall b \in D_v\)  
(a) \(\forall v \supset \exists c \in D_w (c \rightarrow b)\) |
| **\(GF_{c}\)** | \(\exists y|t : x|yA \rightarrow |t : x|yA\) | \(D\)-total | \(\forall a \in U_w \forall b \in D_w\)  
(a) \(\forall v \supset \exists c \in D_v (b \rightarrow c)\) |
| **\(SHRT_{F}\)** | \(|t : x|yA \rightarrow |t : x|A\) | \(U\)-total | \(\forall a, b \in U_w\)  
(a) \(\forall v \supset \exists c \in U_v (b \rightarrow c)\) |
| **\(N_{I_{c}}\)** | \(t_1 \neq t_2 \rightarrow |s : t_1 \mid x = y\) | \(U\)-functional | \(\forall a, b \in U_w \forall c, d \in U_v\)  
(b) \(a \rightarrow c \land b \rightarrow d \supset c = d\)  
(d) \(b \rightarrow d \land c \rightarrow d \supset b = c\) |
| **\(N_{D_{c}}\)** | \(t_1 = \neq t_2 \rightarrow |s : t_1 \mid x \neq y\) | \(U\)-injective | \(\forall a, b, c \in U_w \forall d \in U_v\)  
(a) \(\forall a \rightarrow d \land c \rightarrow d \supset b = c\)  
(d) \(b \rightarrow d \land c \rightarrow d \supset b = c\) |

We will use \(CN\) to denote the class of all \(e\)-frames satisfying the condition(s) corresponding to the schema \(X_e\) of Prop 1.5. We talk of a single domain \(e\)-frame if \(\forall w \in W, D_w = U_w\).
Definition 1.6. Let \( c \) be any individual constant, we say that
1. \( c \) is rigid iff for all \( a \in U_w \), \( a \not\prec v \) implies \( I_w(c) \models a \rightarrow I_a(c) \);
2. \( c \) is stable iff for all \( a \in U_w \) for all \( b \in U_v \), if \( a \not\prec v \) and \( I_w(c) \models a \rightarrow b \) then \( b = I_v(c) \).

Proposition 1.7. Let \( c \) be any individual constant, it holds that
1. If \( c \) is rigid in \( M \), then \( \models^M t : x | A \rightarrow | t : A[c/x] \) ;
2. If \( c \) is stable in \( M \), then \( \models^M t : x | A[c/x] \rightarrow | t : c | A \).

We talk of rigid/stable \( c \)-model if all individual constants are rigid/stable. As we can see from the next proposition, term-modal logics are a particular case of indexed epistemic ones:

Proposition 1.8. For every term-modal model \( \langle W, D, \rightarrow, I, \models \rangle \), see [3], there is a pointwise equivalent rigid and stable \( c \)-model \( M \) which is based on a single-domain \( c \)-frame that is \( D \)-total, \( D \)-injective and \( D \)-functional, and vice versa.

2 Labelled Sequent Calculi

In this section we introduce labelled sequent calculi in order to characterize the indexed epistemic logics defined by the properties of Props. 1.5 and 1.7. These calculi are given by rules meant to internalize the semantics into the syntax in the style of [4, 7]. In order to do it the language \( L \) is modified as follows:

- a countable new set of variables \( w, v, u \ldots \) called world labels is added,
- for each term \( t \) of \( L \) and each world label \( w \), a labelled term \( t^w \) is added,
- for each wff \( A \) of \( L \) and each world label \( w \), a labelled formula \( A^w \) is added,
- a new set of atomic formulas, called ancillary formulas, is added to the language:
  - existence formulas \( t^w \in D(w) \), \( t^w \) is an element of the inner domain of \( w \):
  - compatibility formulas \( t^w(v) \), \( v \) is compatible with \( t^w \)'s knowledge: \( t^w \not\prec v \);
  - counterpart formulas \( s^w(t^w, r^v) \), according to the agent \( s^w \): \( t^w \overset{s^w}{\rightarrow} r^v \),

Notice that ancillary formulas are not labelled formulas even if they contain labelled terms. Labelled formulas are formulas of \( L \) decorated with some label.

Given a labelled formula \( A^w \) and a term \( t \), we want to pin down those occurrences of \( t \) in \( A^w \), if any, which are, so to speak, in the scope of the world label \( w \). To this end we introduce the notion of \( w \)-ground occurrence.

Definition 2.1. Given a labelled formula \( A^w \), a \( w \)-ground occurrence of a term \( t \) in \( A^w \) is defined by induction on \( A^w \):

- \( t \) is a \( w \)-ground occurrence in \( (P^w t_1 \ldots t_n)^w \) iff \( t \equiv t_i \) for some \( i, 1 \leq i \leq n \),
- \( t \) is a \( w \)-ground occurrence in \( (B \rightarrow C)^w \) iff \( t \) is a \( w \)-ground occurrence in \( B^w \) or \( t \) is a \( w \)-ground occurrence in \( C^w \),
- \( t \) is a \( w \)-ground occurrence in \( (\forall x B)^w \) iff \( t \equiv x \) or \( t \) is a \( w \)-ground occurrence in \( B^w \),
- \( t \) is a \( w \)-ground occurrence in \( |s : x_1 \ldots x_n| B \) iff \( t \equiv s \) or \( t \equiv s_i \), for some \( i, 1 \leq i \leq n \).
Any occurrence of a term $t$ in $A^w$ which is a $w$-ground occurrence is replaced by $t^w$. If $x^w$ is free in $A^w$, the $w$-ground occurrences of $x^w$ are exactly those occurrences eligible for being substituted. Let us denote by $E_{<w}$ the language so modified. The definition of substitution 1.2 applies (with minor changes due to the labels) to labelled and ancillary formulas and to labelled terms. Whenever convenient, we allow ourselves to write $(A[t/x])^w$ instead of $(A^w)[t^w/x^w]$. By $E[w/t]$ we denote the labelled or ancillary formula obtained by replacing each occurrence of the world label $v$ in $E$ with an occurrence of $w$. Finally, we use $s^w(t^w, r^w)$ to abbreviate the multiset \{\(s^w(t^w_i, r^w_i): t^w_i \in t^w\) and \(r^w_i \in r^w\)}.

Sequents are expressions of the form

$$\Omega, \Gamma \Rightarrow \Delta$$

where $\Omega$ is a multiset of ancillary formulas, and $\Gamma, \Delta$ are multisets of labelled formulas.

We said that the rules of the calculus are meant to internalize the semantic clauses. In order to see how this is done, let us look at a very simple example.

$$\sigma \models^M_w t : s \vdash Px \iff \text{for all } v \text{ s.t. } \sigma(t) \prec v \text{ and for all } v\text{-assignment } \tau \text{ s.t. } \sigma(s) \models^v \tau(x),$$

it holds that $\tau \models^v Px$.

Reading from right to left, we get the following right introduction rule:

$$\frac{t^w(v), t^w(s^w, x^w), \Omega \Rightarrow \Delta, (P_x)^w}{\Omega, \Gamma \Rightarrow \Delta, ([t^w : s^w]_x^w)^w} \quad \text{Repl}$$

where $v$ is an eigenvariable, i.e. it doesn’t occur free in the conclusion. This entails that $x^w$ is an eigenvariable too.

Reading from left to right, we get the following left introduction rule:

$$\frac{t^w(v), t^w(s^w, r^w), (P_x)^w[r^w/x^w], ([t^w : s^w]_x^w)^w, \Omega \Rightarrow \Delta}{t^w(v), t^w(s^w, r^w), ([t^w : s^w]_x^w)^w, \Omega \Rightarrow \Delta} \quad \text{L_{\Box}}$$

The rules of labelled calculus GE.K for the minimal indexed epistemic logic, i.e. for the formulas valid on the class of all $e$-frames, are given in Table 1. Identity is treated by means of mathematical rules so that the structural rules are admissible, see e.g. [7, Chap. 6]. Note that identity formulas are labelled and that the rule of replacement, Repl, is not a world-independent one as it would be for standard quantified modal systems: since terms need not be rigid designators, identities are world-bound. Thus the fact that $t$ and $s$ denote the same object in $w$ doesn’t imply anything about their denotation in other worlds.

Thanks to the non-logical rules in Table 2, we can define a calculus for each class of $e$-frames considered in Props. 1.5 and 1.7: it is enough to add, for each and every semantic condition (Condition), holding in that class of $e$-frames, the appropriate rule (Cond). See the cut-free proof of GF$_k$ (with rule Dtot) given in Table 3. Roughly, we call mathematical (geometrical) a non-logical rule if it involves no (some) variable condition, see [7, Chaps. 6 and 8] for a precise definition. We will use GE.* to talk of any labelled calculus considered in this paper.

Two observations are in order. First, contrary to what normally happens with non-logical rules, we don’t have to close them under contraction in order to prove the hp-admissibility of contraction, see the proof of Theorem 3.6 for the details. Second, the rules Rig and Stab, which capture respectively the conditions of rigidity and stability, don’t follow exactly the general structure of mathematical rules since they are given with respect to an arbitrary individual.
Table 1: The sequent calculus GE.K.

- **Initial sequents** \( \Omega, P^w, \Gamma \Rightarrow \Delta, P^w \) (\( P^w \) is an atomic labelled formula)

- **Propositional rules**
  \[
  \frac{\Omega, \top, \Gamma \Rightarrow \Delta}{\frac{\Omega, \Gamma \Rightarrow \Delta, A^w, \Gamma \Rightarrow \Delta}{\Omega, (A \rightarrow B)^w, \Gamma \Rightarrow \Delta} L_D} R_D \\
  \frac{\Omega, A^w, \Gamma \Rightarrow \Delta, B^w}{\Omega, \Gamma \Rightarrow \Delta, (A \rightarrow B)^w} R_r
  \]

- **Quantifier rules** (in the rule \( R_e \), \( y^w \) is not free in the conclusion).
  \[
  \frac{t^w \in D(w), \Omega, (A[t/x])^w, (\forall x A)^w, \Gamma \Rightarrow \Delta}{t^w \in D(w), \Omega, (\forall x A)^w, \Gamma \Rightarrow \Delta} L_q
  \]
  \[
  \frac{y^w \in D(w), \Omega, \Gamma \Rightarrow \Delta, (A[y/x])^w}{\Omega, \Gamma \Rightarrow \Delta, (\forall x A)^w} R_q
  \]

- **Modal rules** (in rule \( R_e \), \( v \) is not free in the conclusion).
  \[
  \frac{t^w(v), t^w(s^w, r^w), \Omega, (A[r/x])^w, \{(s : x^w[A]^w, \Gamma \Rightarrow \Delta}{t^w(v), t^w(s^w, x^w), \Omega, (t : x^w[A]^w, \Gamma \Rightarrow \Delta}{\Omega, \Gamma \Rightarrow \Delta, A^w} R_\square
  \]

- **Identity rules** (\( E \) is an atomic labelled formula or an ancillary one.)
  \[
  \frac{\Omega, (t = t)^w, \Gamma \Rightarrow \Delta}{\Omega, \Gamma \Rightarrow \Delta} Selfid
  \]
  \[
  \frac{\Omega, E[s^w/x^w], E[t^w/x^w], (t = s)^w, \Gamma \Rightarrow \Delta}{\Omega, E[t^w/x^w], (t = s)^w, \Gamma \Rightarrow \Delta} Repl
  \]

constant \( c \), and not with respect to an arbitrary term \( t \) as would be the case with mathematical rules. This reflects the fact that these are not conditions on \( e \)-frames, but conditions on \( e \)-models governing the behaviour of closed terms. This difference does not impair the admissibility of the structural rules of inferences. To characterize the indexed epistemic logics based on single domain \( e \)-frames we have introduced the rule Singdom (see Table 2).

### 3 Structural Properties

In this section we show that the calculi GE.\( \ast \) have the same good structural properties of G3c:

- all rules are hp-invertible, the rules of weakening and contraction are hp-admissible, and cut is admissible. A few notions are needed in order to prove the results above. In the rules in Tables 1 and 2 (i) the multisets \( \Omega, \Gamma \) and \( \Delta \) are called contexts, (ii) the other formulas occurring in the conclusion are called principal, and (iii) the formulas of the premisses not occurring in the conclusion are called active. As measures for inductive proofs we use the notions of height of a formula and of height of a derivation. The height of a formula \( E \), \( h(E) \), is the length of the longest branch of its construction tree. The height of a derivation \( D \), \( h(D) \), is the length of its longest branch. We write GE.\( \ast \) \( \vdash n \Omega, \Gamma \Rightarrow \Delta \) if the sequent \( \Omega, \Gamma \Rightarrow \Delta \) is derivable in GE.\( \ast \) with a derivation of height at most \( n \).

**Definition 3.1.** A rule of inference is (height-preserving) admissible in GE.\( \ast \) if, whenever its premisses are derivable (with height \( n \)), also its conclusion is derivable (with at most height \( n \)).

We assume that the free and bound variables occurring in formulas of a sequent are disjoint.

Given that formulas that differ only by a renaming of bound variables are considered identical, we don’t have to prove a lemma of \( \alpha \)-conversion, see [6, Lemma 4.1.1]. We also assume, without loss of generality, that the bound variables occurring in a derivation are such that we never have to rename them when applying a substitution.

27
Table 2: Non-logical rules expressing semantic conditions

- **Rules for** $D_e, T_e, A_e, B_e$
  In the rule $Scr_a$ ($Scr_b$) $v (x^n)$ resp. is not free in the conclusion.

\[
\frac{t^w(v), \Omega, \Gamma \Rightarrow \Delta}{\Omega, \Gamma \Rightarrow \Delta} \hspace{1cm} Scr_a
\]
\[
\frac{t^w(s^w, x^n), \Omega, \Gamma \Rightarrow \Delta}{\Omega, \Gamma \Rightarrow \Delta} \hspace{1cm} Scr_b
\]
\[
\frac{t^w(u), \Omega, \Gamma \Rightarrow \Delta}{t^w(u), t^w(s^w, s^u), \Omega, \Gamma \Rightarrow \Delta} \hspace{1cm} Trans_a
\]
\[
\frac{t^w(r_1^w, r_2^w), t^w(t^w(s^w), t^w(r_1^w, r_2^w)), \Omega, \Gamma \Rightarrow \Delta}{t^w(t^w(s^w), s^w(r_1^w, r_2^w), \Omega, \Gamma \Rightarrow \Delta} \hspace{1cm} Trans_b
\]

- **Rules for** $N_e, ND_e, CBF_e, BF_e, SHR_e, GF_e$
  In the rule $Dsurj$ ($Dtot$ and $Utot$) $x^n$ ($x^s$) resp. is not free in the conclusion.

\[
\frac{t^w(s^n), r^w, t^w(s^n, s^n), r^n, \Omega (r_1 = r_2) \Rightarrow \Delta}{t^w(s^n, s^n), t^w(s^n), r^n, \Omega \Rightarrow \Delta} \hspace{1cm} Dpres
\]
\[
\frac{t^w(r_1^w, r_2^w), t^w(r_1^w, r_2^w, s^n), r^n, \Omega \Rightarrow \Delta}{t^w(s^n, x^n), t^w(v), \Omega \Rightarrow \Delta} \hspace{1cm} Dsurj
\]
\[
\frac{t^w(s^n, x^n), t^w(v), \Omega \Rightarrow \Delta}{t^w(s^n, x^n), t^w(v), \Omega \Rightarrow \Delta} \hspace{1cm} Utop
\]
\[
\frac{t^w(s^n, x^n), t^w(v), \Omega \Rightarrow \Delta}{t^w(s^n, x^n), t^w(v), \Omega \Rightarrow \Delta} \hspace{1cm} Dtot
\]

- **Rules for rigidity, stability, and single domain c-frames**
  In the rules $Rig$ and $Stab$ $c^n$ and $c^s$ are labelled individual constants.

\[
\frac{t^w(c^n), t^w(v), \Omega, \Gamma \Rightarrow \Delta}{t^w(v), \Omega, \Gamma \Rightarrow \Delta} \hspace{1cm} Rig
\]
\[
\frac{t^w(v), t^w(c^n), t^w(v), \Omega, (c = c^s) \Rightarrow \Delta}{t^w(c^n), t^w(v), \Omega, \Gamma \Rightarrow \Delta} \hspace{1cm} Stab
\]
\[
\frac{t^w(u), \Omega, \Gamma \Rightarrow \Delta}{\Omega, \Gamma \Rightarrow \Delta} \hspace{1cm} Singdom
\]

Table 3: Example: a cut free proof of $GF_e$ (using rule $Dtot$)

\[
\frac{z^n \in D(v), t^w(y^n, z^n), t^w(v), y^n \in D(w), (Pz[x/z])^w \Rightarrow (\exists xPx)^w, (Pz[x/z])^w}{z^n \in D(v), t^w(y^n, z^n), t^w(v), y^n \in D(w), (Pz[x/z])^w \Rightarrow (\exists xPx)^w} \hspace{1cm} R_3
\]
\[
\frac{z^n \in D(v), t^w(y^n, z^n), t^w(v), y^n \in D(w), (Pz[x/z])^w \Rightarrow (\exists xPx)^w}{t^w(v), y^n \in D(w), (t \cdot (z^n)Pz)^w \Rightarrow (\exists xPx)^w} \hspace{1cm} L_3
\]
\[
\frac{y^n \in D(w), (t \cdot (z^n)Pz)^w \Rightarrow (\exists xPx)^w}{(\exists x[t \cdot x]Px)^w \Rightarrow (\exists x[t \cdot x]Px)^w} \hspace{1cm} L_3
\]
\[
\frac{(\exists x[t \cdot x]Px)^w \Rightarrow (\exists x[t \cdot x]Px)^w}{(\exists x[t \cdot x]Px)^w \Rightarrow (\exists x[t \cdot x]Px)^w} \hspace{1cm} R_3
\]
Lemma 3.2.
1. The rule of substitution of labelled terms is hp-admissible in $GE_*:
   \begin{align*}
   & GE_* \vdash^n \Omega, \Gamma \Rightarrow \Delta \text{ implies } GE_* \vdash^n \Omega[t^w/x^w], \Gamma[t^w/x^w] \Rightarrow \Delta[t^w/x^w] \\
   \end{align*}

2. The rule of substitution of world labels is hp-admissible in $GE_*:
   \begin{align*}
   & GE_* \vdash^n \Omega, \Gamma \Rightarrow \Delta \text{ implies } GE_* \vdash^n \Omega[w/v], \Gamma[w/v] \Rightarrow \Delta[w/v] \\
   \end{align*}

Proof. (3.2.1) The proof is by induction on the height of the derivation $D$ of $\Omega, \Gamma \Rightarrow \Delta$. If $h(D) = 1$, then $\Omega, \Gamma \Rightarrow \Delta$ is an initial sequent or an instance of $L_\bot$ and also the result of the substitution is an initial sequent or an instance of $L_\bot$. If $h(D) > 1$ we distinguish various cases according to the last rule $R$ applied in them. Let us consider just the following two.

If $R \equiv L_\bot$, we apply the inductive hypothesis (IH) to its premiss and then $L_\bot$ to obtain a derivation $D[t^w/x^w]$ of $\Omega[t^w/x^w], \Gamma[t^w/x^w] \Rightarrow \Delta[t^w/x^w]$ with same derivation height of $D$. If $R \equiv R_C$ we proceed analogously, but we apply IH twice: the first time to replace the eigenvariables of $R$ with some variables occurring neither in $D$ nor in $t^w$, and the second time to apply the substitution $[t^w/x^w]$.

(3.2.2) We proceed as above by induction on $h(D)$. We show the interesting case of $R_\forall$ with eigenvariable $y^w$. We transform $D$:

\[
\begin{align*}
  \frac{y^w \in D(v), \Omega, \Gamma \Rightarrow \Delta, (A[y/x])^w}{\Omega, \Gamma \Rightarrow \Delta, (\forall x A)^w} & \quad R_\forall \\
  \frac{y^w \in D(v), \Omega, \Gamma \Rightarrow \Delta, (A[y/x])^w}{\Omega, \Gamma \Rightarrow \Delta, (A[y/x])^w} & \quad IH \\
  \frac{z^w \in D(w), \Omega[w/v], \Gamma[w/v] \Rightarrow \Delta[w/v], (A[z/x])^w}{\Omega[w/v], \Gamma[w/v] \Rightarrow \Delta[w/v], (\forall x A)^w} & \quad R_\forall
\end{align*}
\]

where both $x^w$ and $z^w$ do not occur in $D$. The steps by Lemma 3.2.1 and by IH are hp-admissible, and therefore $D[w/v]$ has the same height of $D$.

Lemma 3.3. All sequents $\Omega, A^w, \Gamma \Rightarrow \Delta, A^w$, with $A^w$ arbitrary labelled formula, are derivable in $GE_*$. 

Proof. By an easy induction on $h(A^w)$. 

Weakening and contraction. In the following $E$ stands for an arbitrary labelled or ancillary formula and $A^w$ for an arbitrary labelled formula.

Theorem 3.4. The left and right rules of weakening are hp-admissible in $GE_*:

\[
\begin{align*}
  & \frac{\Omega, \Gamma \Rightarrow \Delta}{\Omega, E, \Gamma \Rightarrow \Delta} \quad L-W \\
  & \frac{\Omega, \Gamma \Rightarrow \Delta}{\Omega, \Gamma \Rightarrow \Delta, A^w} \quad R-W
\end{align*}
\]

Proof. If $E (A^w)$ contains world labels used in the proof $D$ of the premiss as eigenvariables, then by Lemma 3.2 we replace those eigenvariables by new variables occurring neither in $D$ nor in $E (A^w)$. Then the proof proceeds in the standard way by induction on the height of the derivation of the premiss.

Lemma 3.5. All rules of $GE_*$ are hp-invertible.

Proof. The proof is by induction on the height of the derivation $D$ of the conclusion of the rule $R$ we are considering. Here is a paradigmatic case. Suppose that a proof of $\vdash^n \Omega, \Gamma \Rightarrow \Delta, (|t : x^w A|^w$ is given and that the last rule applied in that proof is $Ser_a$. So we have:

\[
\begin{align*}
  & (1) \vdash^{n-1} t^w (v), \Omega, \Gamma \Rightarrow \Delta, (|t : x^w A|^w, \text{ and then by } Ser_a \\
  & (2) \vdash^n \Omega, \Gamma \Rightarrow \Delta, (|t : x^w A|^w)
\end{align*}
\]
Our aim is to show that
\[(3) \vdash_n t^w(v), t^w(s^w, x^v), \Omega, \Gamma \Rightarrow \Delta, A^w, \text{ where } v \text{ is the eigenvariable of the instance of } S_{e_n} \text{ we are considering.}\]

By applying Lemma 3.2 to (1) (w.r.t. to some u not occurring in D) we get
\[(4) \vdash_n t^w(u), \Omega, \Gamma \Rightarrow \Delta, (|t : s^w|A)^w, \text{ then by IH to (4)}\]
\[(5) \vdash_n t^w(u), t^w(v), t^w(s^w, x^v), \Omega, \Gamma \Rightarrow \Delta, A^w, \text{ and by } S_{e_n} \text{ we conclude}\]
\[(3) \vdash_n t^w(v), t^w(s^w, x^v), \Omega, \Gamma \Rightarrow \Delta, A^w.\]

**Theorem 3.6.** The left and right rules of contraction are hp-admissible in GE.*:
\[
\begin{align*}
\Omega, E, E, \Gamma & \Rightarrow \Delta & \Omega, E, E, \Gamma & \Rightarrow \Delta & L-C \\
\Omega, \Gamma & \Rightarrow \Delta, A^w, \Omega & \Rightarrow \Delta, A^w & R-C
\end{align*}
\]

**Proof.** The proofs are by simultaneous induction on the height of the derivation D of the premiss for left and right contraction. The base case is straightforward. For the inductive cases, we have two subcases for each possible last rule R in D, depending on whether or not an instance of the formula we are contracting is principal in R. If no instance is principal, two instances occur in the premiss(ess) and we can apply IH to it and then the rule R. Else, one instance of the contraction formula is principal in R and we have three subcases.

(i) If R is a rule with repetition of the principal formulas in the premiss (i.e. either L= or L□) or a non-logical rule, we can simply apply IH to the premiss and then the rule. Observe that the non-logical rules we are considering are such that we don’t have to close them under contraction: for Transₚ and Symₚ we have only to apply IH twice, and for Repl, Ufunc and Usurj any contracted instance can be obtained by using IH and then rule Selfid. Thus, if the last step in D is by rule Ufunc with E \(\equiv t^w(s^w, r^v)\), we transform
\[
\begin{align*}
\Omega, t^w(s^w, s^v), t^w(s^w, r^v), (r = r^v), \Gamma & \Rightarrow \Delta & \Omega, t^w(s^w, r^v), (r = r)^v, \Gamma & \Rightarrow \Delta & Ufunc \\
\Omega, t^w(s^w, r^v), t^w(s^w, s^v), (r = r)^v, \Gamma & \Rightarrow \Delta & \Omega, t^w(s^w, r^v), (r = r^v), \Gamma & \Rightarrow \Delta & L-C
\end{align*}
\]
which has the same derivation height of D. All other non-logical rules cannot have instances with a duplicated principal formula.

(ii) If R is a rule where all active formulas are proper subformulas of the principal one (i.e. any propositional rule), we proceed as for G3c.

(iii) If R is a rule where active formulas are (a) proper subformulas of the principal one and (b) ancillary formulas (i.e. one of Rₚ and R□), we start by using the hp-invertibility – Lemma 3.5 – of that rule, then we apply IH as many times as needed and finally the rule R; see [4, Theorem 4.12] for the details. □

**Admissibility of cut.**

**Theorem 3.7.** The rule of cut is admissible in GE.*:
\[
\begin{align*}
\Omega, \Gamma & \Rightarrow \Delta, A^w & \Omega', A^w, \Pi & \Rightarrow \Sigma & \Sigma \\
\Omega, \Omega', \Gamma, \Pi & \Rightarrow \Delta, \Sigma & C_{cut}
\end{align*}
\]

**Proof.** The proof is by induction on the height of the cut formula with a subinduction on the cut-height, i.e. the sum of the heights of the derivations of the two premisses. The cases of (i) initial sequents, (ii) cut formula not principal in at least one of the two premisses, and (iii) cut formula principal in both premisses but not of shape \( (|t : s^w|B)^w \) are treated as for G3Kq*, see [7, Theorem 12.9], and are therefore omitted.
If the cut formula is \(|t: x|B|w\) and it is principal in both premisses, we have a derivation of the form

\[
\frac{\Omega, t^w(u), t^w(s^w, x^w), \Gamma \Rightarrow \Delta, B^w}{\Omega, \Gamma \Rightarrow \Delta, (|t: x|B)|w} \quad R_C  \quad \frac{\Omega, t^w(v), t^w(s^w, r^v), (B[r/x])^v, (|t: x|B)|w, \Pi \Rightarrow \Sigma}{\Omega, \Omega', t^w(v), t^w(s^w, r^v), (B[r/x])^v, \Gamma, \Pi \Rightarrow \Delta, \Sigma} \quad \text{Cut}
\]

which can be transformed into a derivation of the same conclusion having two cuts that are admissible by IH: first we construct the derivation \(D_3\) which has a cut on \(|t: x|B|w\) of lesser cut-height,

\[
\frac{\Omega, t^w(u), t^w(s^w, x^w), \Gamma \Rightarrow \Delta, B^w}{\Omega, \Gamma \Rightarrow \Delta, (|t: x|B)|w} \quad R_C  \quad \frac{\Omega, \Omega', t^w(v), t^w(s^w, r^v), (B[r/x])^v, \Gamma, \Pi \Rightarrow \Delta, \Sigma}{\Omega, \Omega', t^w(v), t^w(s^w, r^v), (B[r/x])^v, \Gamma, \Pi \Rightarrow \Delta, \Sigma} \quad \text{Cut}
\]

Second, we make use of the lp-admissibility of substitutions, Lemma 3.2, to apply the following substitutions to the derivation \(D_1\): first \([r^w/x^w]\) and then \([v/u]\). Observe that it is essential to apply the substitutions in this order since the \(x^w\)'s, but not the \(x^s\)'s, satisfy the variable condition and therefore don’t occur in \(\Omega, \Gamma, \Delta, t^w, s^w\). Now we can apply a cut, which has a cut formula of lesser height and therefore is admissible by IH, to \((D_1[r^w/x^w])[v/u]\) and \(D_3\) as follows

\[
\frac{\Omega, t^w(v), t^w(s^w, r^v), \Gamma \Rightarrow \Delta, (B[r/x])^v}{\Omega, \Omega', t^w(v), t^w(s^w, r^v), (B[r/x])^v, \Gamma, \Pi \Rightarrow \Delta, \Delta, \Sigma} \quad \text{Cut}
\]

The only case where the cut formula can be principal in a non-logical rule is when the cut formula is atomic and we are dealing with a rule for identity. Thus the presence of non-logical rules from Table 2 has no role w.r.t. the admissibility of \text{Cut}.

We are now going to show that the rules \text{Self} and \text{Repl} are enough to capture the logic of identity and, by using the admissibility of cut, that the rule that generalizes \text{Repl} to arbitrary \(L^{ext}\)-formulas is admissible.

\textbf{Lemma 3.8.}

1. \(\text{GE}\.\star \vdash (t = t)^w\)
2. \(\text{GE} \.\star \vdash (r_1 = r_2)^w, (A[r_1/y]^w \Rightarrow (A[r_2/y])^w\)
3. The following rule, where \(E\) is an arbitrary \(L^{ext}\)-formula, is admissible in \(\text{GE}\.\star\)

\[
\frac{\Omega, E[s^w/x^w], E[t^w/x^w], (t = s)^w, \Gamma \Rightarrow \Delta}{\Omega, E[t^w/x^w], (t = s)^w, \Gamma \Rightarrow \Delta} \quad \text{Repl}_A
\]

\textbf{Proof.} 3.8.1. The sequent \(\Rightarrow (t = t)^w\) is derivable by applying \text{Self} to the initial sequent \((t = t)^w \Rightarrow (t = t)^w\).

3.8.2. By induction on \(h(A^w)\). The only interesting case is \(A^w \equiv (|t: x|B|w\) where, instead of IH, we have to use Lemma 3.3. This happens because formulas with an indexed epistemic operator as principal behave like atomic formulas with respect to substitutions.

3.8.3. If \(E \equiv A^w\) with \(A^w\) not atomic, we proceed as follows:

31
3.8.2 \[ \Omega, (A[t/x])^w, (A[s/x])^w, (t = s)^w, \Gamma \Rightarrow \Delta \]
\[ \Omega, (A[t/x])^w, (A[t/x])^w, (t = s)^w, (t = s)^w, \Gamma \Rightarrow \Delta \]
\[ \Omega, (A[t/x])^w, (t = s)^w, \Gamma \Rightarrow \Delta \]

C\text{ut}
L-C, 2 times

\[ \Box \]

4 Soundness and Completeness

Soundness. The proof of soundness is structured as in [5, 7]; we proceed, in brief, by defining what it means for a sequent to be valid on a class of $e$-frames, and then we show that initial sequents are valid on any $e$-frames and that each rule of GE.\textbullet preserved validity over the appropriate class of $e$-frames $C^e$.

Definition 4.1. Let $W^e$ be the set of all world labels occurring in a sequent $S, \mathcal{M} = (\mathcal{W}, \mathcal{U}, \mathcal{D}, <, \Rightarrow, \mathcal{I})$ a $t$-model, $f$ be a mapping from $W^e$ to $\mathcal{W}$, and $f^e$ be a function associating to each $f(w)$ some $f(w)$-assignment $\sigma$ that we agree to indicate with $\sigma_{f(w)}$. We say that:

- $(f, f^e, \mathcal{M})$ satisfies $A^w (\in S)$ if $\sigma_{f(w)} \models^{\mathcal{M}} A$;
- $(f, f^e, \mathcal{M})$ satisfies $t^w(v) (\in S)$ if $\sigma_{f(w)}(t) < f(v)$;
- $(f, f^e, \mathcal{M})$ satisfies $r^w(t^w(s^w)) (\in S)$ if $\sigma_{f(w)}(t) \rightarrow \sigma_{f(w)}(s)$;
- $(f, f^e, \mathcal{M})$ satisfies $t^w \in D(w) (\in S)$ if $\sigma_{f(w)}(t) \in D_w$.

Definition 4.2 ($C^e$-validity). A sequent $\Omega, \Gamma \Rightarrow \Delta$ is said to be $C^e$-valid if every triple $(f, f^e, \mathcal{M})$ where $\mathcal{M}$ is (rigid and/or stable and) based on a $e$-frame in $C^e$ is such that:

- if $(f, f^e, \mathcal{M})$ satisfies all formulas in $\Omega, \Gamma$, then it satisfies some formula in $\Delta$.

Theorem 4.3 (Soundness). If $\text{GE.\textbullet} \vdash \Omega, \Gamma \Rightarrow \Delta$, then $\Omega, \Gamma \Rightarrow \Delta$ is $C^e$-valid (on rigid and/or stable models if the corresponding rules are in $\text{GE.\textbullet}$).

Proof. The proof is by induction on the height of the derivation $D$ of the sequent $\Omega, \Gamma \Rightarrow \Delta$. The base case holds trivially since either some $(Pt_1 \ldots t_n)^w$ occurs both in $\Gamma$ and in $\Delta$, or $\bot^w$ occurs in $\Gamma$.

For the inductive step, we distinguish cases according to the last rule applied in $D$. We omit the cases of the propositional rules and of the rules for $\forall$, see [7, Theorem 12.13].

If the last step of $D$ is
\[
\frac{t^w(v), t^w(v^r, s^v), \Omega, (A[r/x])^v, (t : s^v | A)^w, \Gamma \Rightarrow \Delta}{t^w(v), t^w(s^w, r^v), \Omega, (t : s^v | A)^w, \Gamma \Rightarrow \Delta}
\]
we know by IH that any triple $(f, f^e, \mathcal{M})$ satisfying all the formulas in the antecedent of the premiss satisfies also some formula in $\Delta$. The antecedent of the conclusion differs from the antecedent of the premiss insofar as the formula $(A[r/x])^v$ is missing. Nevertheless, any triple $(f, f^e, \mathcal{M})$ satisfying $t^w(v), t^w(s^w, r^v), (t : s^v | A)^w$ satisfies also $(A[r/x])^v$.

If the last step is
\[
\frac{t^w(v), t^w(s^w, x^v), \Omega, \Gamma \Rightarrow \Delta, A^v}{\Omega, \Gamma \Rightarrow \Delta, (t : s^v | A)^w}
\]

we know by IH that any triple $(f, f^e, \mathcal{M})$ satisfying all the formulas in the antecedent of the premiss satisfies also some formula in $\Delta$. The antecedent of the conclusion differs from the antecedent of the premiss insofar as the formula $(A[r/x])^v$ is missing. Nevertheless, any triple $(f, f^e, \mathcal{M})$ satisfying $t^w(v), t^w(s^w, x^v), (t : s^v | A)^w$ satisfies also $(A[r/x])^v$.

32
where $s^w$ and $x^v$ are tuples of length $n$, we know by IH that the premiss is $C^*$-valid. Let \( \langle f, f^*, M \rangle \) be a generic triple that is defined w.r.t. the language of the conclusion and such that it satisfies all formulas in $\Omega, \Gamma$. Either there are some $u \in W$ such that $\sigma_{f(w)}(|u|) \prec u$ and some $o_1, \ldots, o_n \in U_u$ such that, for all $s^w_i \in s^w, \sigma_{f(w)}(s^w_i) \rightarrow o_i$, or not. If not, it can be seen that \( \langle f, f^*, M \rangle \) trivially satisfies \( (|t : \delta | A)^w \). Otherwise, we extend $f$ and $f^*$ to $f'$ and $(f')^*$ such that: $f'(v) = u$ and $(f')^*(u) = \sigma_u$ for some $\sigma_u$ such that: $\sigma_u(x^v_i) = o_i$ for all $x_i \in x^v$ (this extension is feasible thanks to the variable condition of $R_0$). The triple \( \langle f', (f')^*, M \rangle \) satisfies all the formulas in the antecedent of the premiss and, therefore, it satisfies also some formula in $\Delta$ or it satisfies $A^w$. In the former case we have that the non-extended triple \( \langle f, f^*, M \rangle \) satisfies some formula in $\Delta$ and in the latter that it satisfies \( (|t : \delta | A)^w \).

If the last step is by some mathematical rule, it can at once be seen that the theorem holds since $M$ is (rigid and/or stable and) based on a $c$-frame in $C^*$ and $= is interpreted as real identity.

Suppose that the last step is by a geometrical rule, say
\[
\frac{x^w \in D(w), t^w(x^w, s^v), t^w(v), s^v \in D(v), \Omega, \Gamma \Rightarrow \Delta}{D_{surj}, x^w \text{ fresh}}
\]
Take any triple \( \langle f, f^*, M \rangle \) such that (i) it is defined w.r.t. the language of the conclusion, (ii) it satisfies all formulas in $t^w(v), s^v \in D(v), \Omega, \Gamma$, and (iii) $M$ is based on a $D$-surjective $c$-frame. Thanks to (i), (ii) and (iii), we can extend $f$ and $f^*$ to obtain a triple \( \langle f', (f')^*, M \rangle \) satisfying all formulas in the antecedent of the premiss. By IH this triple satisfies also some formula in $\Delta$. We conclude that also \( \langle f, f^*, M \rangle \) satisfies some formula in $\Delta$ because $x^w$ doesn’t occur in the conclusion.

**Completeness.** We follow the pattern of [7, Theorem 12.14] for $G3.Kq^*$: we give a constructive proof of (weak) completeness by defining a root-first proof search procedure. The procedure is such that if it terminates, the sequent is derivable and therefore valid; otherwise the tree generated by the proof search has at least one infinite branch. Such an infinite branch contains all the information needed to construct a countermodel for $\Omega, \Gamma \Rightarrow \Delta$ based on a $c$-frame in $C^*$. In order to set up the procedure, it is expedient to add for each $w$ an infinite set of new individual constants $c^w$, and to consider sequents $\Omega, \Gamma \Rightarrow \Delta$ containing only closed formulas.

**Definition 4.4 (⋆-reduction tree).** Given a sequent $\Omega, \Gamma \Rightarrow \Delta$ containing only closed formulas and a calculus $GE.\star$, we define the following procedure for constructing a $\star$-reduction tree $T$:

**Stage 0.** We write $\Omega, \Gamma \Rightarrow \Delta$ as root of $T$.

**Stage $n + 1$** Two cases need to be distinguished.

**Case (i).** Each topmost sequent of the $n$-th stage of $T$ is an initial sequent or an instance of $L_\perp$. The construction ends.

**Case (ii).** Else we continue the construction by applying the following $8 + k + 1$ substages ($k$ is the number of non-logical rules of $GE.\star$) to all the leaves of the tree generated at the previous (sub)stage which are not initial sequents or instances of $L_\perp$.

**Substage 1.** We reduce all new formulas of the form $(A \rightarrow B)^w$ occurring in the antecedent as follows: if the leaf is $\Omega, (A_1 \rightarrow B_1)^{w_1}, \ldots, (A_n \rightarrow B_n)^{w_n}, \Gamma \Rightarrow \Delta$

we write over it the new $2^n$ topmost sequents:

\[
\Omega, (B_1)^{w_1}, \ldots, (B_n)^{w_n}, \Gamma \Rightarrow \Delta, (A_{i_k})^{w_{i_k+1}}, \ldots, (A_{i_n})^{w_{i_n}}
\]
where \( \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\} \) and \( \{i_{k+1}, \ldots, i_n\} = \{1, \ldots, n\} - \{i_1, \ldots, i_k\} \).

Substage 2. We reduce all formulas \((A \rightarrow B)^w\) occurring in the succedent of the leaf by applying, root-first, all possible instances of \(R_\rightarrow\).

Substage 3. We apply, root-first, all possible instances of \(L_\gamma\): for any pair of \(L^{ext}\)-formulas \((\forall x.A)^w\) and \(t^w \in D(w)\) occurring in the antecedent of the leaf, we add to the new topmost sequent the formula \((A[t/x])^w\).

Substage 4. For each formula \((\forall x.A)^w\) occurring in the succedent, we apply, root-first, rule \(R_\forall\) by using a fresh constant \(c^w\).

Substage 5. We apply, root-first, all possible instances of \(L_\Box\): for any set of \(L^{ext}\)-formulas \(t^w(v), t^w(s^w, r^v), (t : s^w[A])^w\) occurring in the antecedent of the leaf, we add to the new topmost sequent the formula \((A[r/x])^v\).

Substage 6. For each formula \((t : s^w[A])^w\) occurring in the succedent, we apply, root-first, rule \(R_{\forall}\) by using a fresh world label \(v\).

Substage 7. We apply, root-first, rule \(Selfid\) for every labelled constant occurring in the leaf.

Substage 8. We apply, root-first, all possible instances of rule \(Repl\).

Substage 8+j. We apply, root-first, all possible instances of the \(j\)-th non-logical rule \(R\) of \(GE.*\). If \(R\) is a mathematical rule, we apply it w.r.t. all constants occurring in the topmost sequent for which we can apply it. If \(R\) is a geometrical rule, we apply it w.r.t. some new labelled constant.

Substage 8+k+1. If at no previous substage we have introduced some new topmost sequent, we write a copy of the topmost sequent on top of itself.

**Definition 4.5.** Given an infinite branch \(B\) of a \(*\)-reduction tree,

- \(LB\) \((RB)\) denotes the set of all formulas occurring in the antecedents (succedents) of \(B\).
- if \(t^w\) and \(s^w\) are constants occurring in formulas of sequents of \(B\),
  \[t^w \approx s^w \iff (t = s)^w \in LB\]

Given the \(*\)-reduction procedure and Lemma 3.8, \(\approx\) is an equivalence relation. By \([t^w]\) we denote the equivalence class of \(t^w\) modulo \(\approx\).

**Definition 4.6.** Let \(B\) be an infinite branch of a \(*\)-reduction tree. We define the model \(M_B = (\mathcal{W}_B, \mathcal{U}_B, \mathcal{D}_B, \prec_B, \rightarrow_B, I_B)\) as:

- \(\mathcal{W}_B\) is the set of world labels occurring in \(LB \cup RB\);
- \(\mathcal{U}_B\) is the family of all sets \(U_w\) where \(w \in \mathcal{W}_B\) and \(U_w\) is the set of all equivalence classes \([c^w]\), where \(c^w\) is a labelled constant occurring in \(LB \cup RB\);
- \(\mathcal{D}_B\) is the family of the sets \(D_w = \{[c^w] : c^w \in D(w)\} \subseteq LB\};
- \(\prec_B\) is such that \([t^w] \prec_B v\) iff \(t^w(v)\) occurs in \(LB\);
- \(\rightarrow_B\) is such that, for all \(w \in \mathcal{W}_B\) and all \([t^w] \in U_w\), \([s^w] \rightarrow [r^v] \iff t^w(s^w, r^v)\) occurs in \(LB\);
- \(I_B\) maps each \(w \in \mathcal{W}_B\) to an interpretation \(I_w\) such that (i) \(I_w(c^w) = [c^w]\) for every \(c^w\) occurring in \(LB \cup RB\); (ii) \(I_w(P^n) = \{([t^w], \ldots, [t^w]) : (P^n t_1, \ldots, t_n)^w\} \subseteq LB\).

**Theorem 4.7** (Completeness). Any sequent \(\Omega, \Gamma \Rightarrow \Delta\) not containing free variables is such that either it is derivable in \(GE.*\) or there is some \(M\), which is (rigid/stable and) based on a member of \(C*\), that satisfies all unlabelled version of the formulas in \(\Gamma\) and no one in \(\Delta\).
Proof. We build a ⋆-reduction tree $T$ of $\Omega, \Gamma \Rightarrow \Delta$. If $T$ is a finite tree, then $\Omega, \Gamma \Rightarrow \Delta$ is derivable and the theorem is proved. Otherwise the proof search fails and, by König’s Lemma, $T$ has an infinite branch $B$ out of which we can construct a model $M^B$ as in Defn. 4.6. We know that

$$\text{if } A^w \text{ occurs in } T^B (R^B), \text{ then } \models_{M^B} A \ (\not\models_{M^B} A) \quad (4)$$

This claim, which can be proved by an easy induction on $h(A^w)$, is enough to conclude that $M^B$ satisfies all (unlabelled) members of $\Gamma$ and no member of $\Delta$. We prove just one case of (4), if $A^w \equiv ([t : x] B)^w$ occurs in $R^B$, the ⋆-reduction is such that that formula has been reduced at some stage. Thus the antecedent of some sequent of $B$ (and $L^B$) contains $t^w(v)$ and $t^w(s^w, c^y)$ and its succedent (and $R^B$) contains $(B[c/x])^v$. By IH, this last fact implies that $\not\models_{M^B} B[c/x]$, and therefore $\not\models_{M^B} [t : x] B$, since $[t^w] \not\models^B v$ and $[s^w_t] \mapsto [c^y_t]$, for all $s^w_t \in s^w$ and $c_t^y \in c^y$.

To show that $M^B$ is (rigid/stable and) based on a $c$-frame in $C^*$, just notice that the ‘if’-clause of instance of any semantic Condition, which holds in $C^*$, can be satisfied in $M^B$ only if the corresponding ancillary formulas occur in some node of $B$. Thus, at substage $8 + j$ of some stage of the construction of the ⋆-reduction tree, the formulas corresponding to the ‘then’-clause of Condition have been added to the antecedent (by rule Condl), and therefore they occur in $L^B$. Given the way we have constructed $M^B$, this means that the ‘then’-clause of the given instance of Condition holds in $M^B$, and by generalization we conclude that $M^B$ is (rigid/stable and) based on a member of $C^*$.

Concluding remarks. In this paper we have introduced a labelled sequent calculus for every IEL considered in [1, 2]. These calculi have good structural properties insofar as weakening and contraction are hp-admissible and cut is admissible. In this way we have answered the question of finding a cut-free calculus for logics with $GF_c$, [1, p.1182]. Then we have shown that each calculus $\mathsf{GE}_1^\star$ is sound and complete w.r.t. the corresponding class of $c$-frames $C^*$. In particular the proof of completeness is based on a procedure that gives a proof in $\mathsf{GE}_1^\star$ of every $C^*$-valid sequent and a countermodel (based on a $c$-frame in $C^*$) of every sequent which is not $C^*$-valid. Given Prop. 1.8, we have also implicitly introduced a labelled calculus for every term-modal logic considered in [3], as well as for the TML based on symmetric structures, which weren’t considered in [3]. The completeness of the calculus gives us also a semantic proof that cut is admissible therein, proof which backs up the constructive proof given in Theorem 3.7.

References

A Dynamic Logic for Configuration

Ching Hoo Tang and Christoph Weidenbach

Max Planck Institute for Informatics, Saarbrücken, Germany
chtang, weidenbach@mpi-inf.mpg.de

Abstract

We define the new dynamic logic PIDL+ that extends our previously developed logic PIDL (Propositional Interactive Dynamic Logic) with arithmetic constraints. The language of PIDL+ is motivated by real world configuration systems, in particular, configuration systems for power plants. A PIDL+ specification consists of the description of an initial state, global constraints, and actions. Its semantics are the possible worlds starting from the initial state, spanned by the actions and restricted by the constraints. It distinguishes user actions from rule actions. Any user action is followed by a unique fixed point, called rule-terminal state, generated through exhaustive application of rule actions from the specification. The built in rule action fixpoint semantics and arithmetic constraints distinguish PIDL+ from known dynamic or action logics. Correctness of a PIDL+ specification as well as reachability of a particular state are decidable. We provide sound and complete algorithms.

1 Introduction

The dynamic logic PIDL+ is motivated by real-world configuration systems, e.g., configuration systems for steel or power plants as they are productive at Siemens [8]. A run of such a configuration system is a dialog between its user picking components or setting parameters inside certain bounds and the system reacting by adjusting the current configuration such that eventually a buildable product is the result. Today’s implementations of such configuration systems are not based on formal specifications but are typically built on top of general-purpose programming languages. Hence, overall soundness or completeness of the configuration system cannot be automatically shown.

PIDL+ copes with the needed expressiveness of real-world configuration systems on one side and on the other side enables automatic verification of relevant system properties. The above form of dialog between the system and the user needs to distinguish variables controlled by the user and variables controlled by the system. Soundness is defined by one quantifier alternation: for all inputs to variables controlled by the user, the system can react by adjusting variables under its control such that eventually a buildable product is the result. If for any final product there is a sequence of user inputs reaching a state representing this product, the configuration system is complete.

From the user perspective, the system should terminate after each input and always produce the same result on the same inputs. In PIDL+ the user and the system are modeled by rules, called transitions (Definition 1). A user or system transition comes in a “precondition \(\rightarrow\) update” form: if the precondition of a transition is implied by the current state, its update is applied to the current state and yields the next state. After the application of a user transition, the system transitions, later called rules, are exhaustively applied until a unique state is reached that is a fixpoint with respect to the system transitions. We call such a state a rule-terminal state. The system is sound if whatever the user does inside the bounds specified eventually a rule-terminal state representing a product can be reached. Overall requirements to products are represented by constraint formulas in PIDL+. They have to be satisfied by any state and, in particular, by states representing a final product (Definition 17).
In real-world applications it is necessary that the set of variables controlled by the user dynamically changes. For example, if the user picks a certain component but subsequent user choices require an update to the component by the system, the user decision is overwritten and hence the component is no longer under the control of the user. In PIDL+ this is represented by a set of user controlled variables attached to each state (Definition 4). Eventually a state is represented by a set of literals and a set of user controlled variables.

Components are presented in PIDL+ by propositional variables and parameters by integer variables. For an admissible PIDL+ specification (Definition 3) all integer variables are finitely bounded. Therefore, the number of all reachable states is finite and properties of the specification become decidable. A priori bounded integer variables are typical for a real-world configuration system, e.g., out of a component box no car company is able to build an engine with arbitrarily high throughput. Still, result variables that are needed to represent the eventual product may be of type real in PIDL+ and computed by complex expressions. In order to preserve decidability, they must not occur in any transition (Definition 3).

Semantically, a system state is a possible world where all propositional and integer variables have fixed values. Then a user or system transition leads to the next world. We call this semantics the small step semantics. It is easy to understand but not well-suited for computing properties as the number of possible worlds becomes the product over the combination of all variable ranges. In this paper we present a big step semantics where a state of the big step semantics represents up to exponentially many states of the small step semantics. In any state the integer variables are bounded, but do not need to have a fixed single value. The semantics is existential for user and universal for all other variables: the precondition of a transition is satisfied in a state, if for some instance of the user variables satisfying the bounds of a state and all instances of all other variables satisfying the bounds of a state the precondition is implied. The bounds either come from the initial state or are updated by transition applications. With respect to the above semantics, a transition depending on a user variable might only be applicable to some of the potential values of the variable, but not to all. This makes it necessary to partition the values of variables in states with respect to a transition in order to compute the next state of the big step semantics. We call such partitions meeting the precondition of a transition with respect to a state a selection (Definition 5). Finite selections always exist because the applicability of transitions only depends on bounded integer variables.

PIDL+ is the extension of PIDL [9] with arithmetic constraints. Similar approaches that deal with logics formalizing change naturally arise from modal logics, originally introduced to describe the behavior of programs, prominently represented by Propositional Dynamic Logic [10] and its variants [2]. Boolean games [12] and variants thereof [17] are another mechanism to model a changing system, in this case by considering situations as they appear in game theory [14], where players play games by setting assignments of Boolean variables. These approaches are different from PIDL+ insofar that they do not have the inherent features to describe the characteristic user-rule relationship with rule-terminal states of the configuration system as mentioned earlier, or that they, in the case of Boolean games, focus on equilibria depending on certain strategies, as opposed to an exhaustive unfolding of user and rule actions in PIDL+. Also, our logic supports arithmetic constraints through selections. The same holds for approaches based on temporal logic [15, 4] and model checking [5], where solutions, for example, deal with the verification of configuration product variants [6] modeled as feature models [13], whose logics have been studied extensively [7]. Further work that explicitly centers on the analysis of configuration systems usually provides stage-wise verification, that is, it aims to analyze the configuration after each user input and guide the user through the configuration process, often through constraint satisfaction [1, 16, 3].
In Section 2 we introduce the above described syntax and semantics of PIDL+ in full detail. The section ends with an example specification illustrating the features of the logic (Example 20). Section 3 presents algorithms checking soundness and completeness and we end with a short conclusion, Section 4.

2 The Logic PIDL+

We give a detailed description of our logic PIDL+ in this section. Its design is geared towards modeling the dynamics of an interactive configuration system with the help of a states-and-transitions semantics. The states describes the possible worlds reachable from the initial state via defined sets of user and rule transitions. The base logic underlying the formulas used to express single states and transitions is propositional logic in combination with the theory of arithmetic over the reals. Relevant concepts connected with the base logic and general notions essential for the design of PIDL+ are given in the preliminaries subsection, which is followed by the definition of the actual syntax of PIDL+. Its semantics of states and transitions is contained in the final part of this section.

2.1 Preliminaries

The basic units of PIDL+ are terms and formulas derived from the fragment of first-order logic that is composed of propositional variables and the theory of reals. We provide the central notions needed in this paper.

We fix the order-sorted signature \( \Sigma := ([R, Z], \mathbb{R} \cup \{+, -, \cdot, <, \leq, \geq, =, \neq\}) \). It has sorts \( R \) and \( Z \), where \( Z \) is a subset of \( R \), \( Z \subset R \), constants \( \mathbb{R} \), where the sort of each \( c \in R \) is \( R \), written as \( \text{sort}(c) = R \), function symbols \( +, -, \cdot \), and \( \cdot \), where it holds that \( \text{sort}(f) = R \times R \rightarrow R \), \( f \in \{+, -, \cdot\} \), and predicate symbols \( <, \leq, \geq, =, \neq \), where it holds that \( \text{sort}(\varphi) = R \times R, \varphi \in \{<, \leq, \geq, =, \neq\} \).

We say that a set of variables \( X \) is \( \Sigma \)-sorted if the sort of each variable \( x \in X \) is one of the sorts specified in \( \Sigma \): \( \text{sort}(x) \in \{R, Z\} \). We choose to have two sorts \( R \) and \( Z \), to be interpreted as the sets \( R \) and \( Z \), because, as is delineated later in the section, we want to interpret numerical expressions that have an effect on the action dynamics as bounded integers, which is a crucial requirement for achieving decidability of the logic. Additionally, we allow those expressions which do not affect the action flow to be over the reals. They represent results that are purely restricted to the single worlds, holding information values of the configurations.

A term \( t \) over the signature \( \Sigma \) and a \( \Sigma \)-sorted variable \( X \) is called \( \Sigma \)-term and is defined by the usual rules known from first-order logic using variables, constants and function symbols. \( T_\Sigma(X) \) denotes the set of all terms over the signature \( \Sigma \) and the variable set \( X \). We use the customary infix notation when writing \( \Sigma \)-terms. For example, we write \( x + 9 \) instead of \( +(x, 9) \).

An atom over \( \Sigma \) is an expression of the form \( t \circ t' \), a simple atom over \( \Sigma \) is an atom of the form \( x \circ t \), and a simple bound over \( \Sigma \) is a simple atom of the form \( x \circ c \), where \( x \in X, t, t' \in T_\Sigma(X), c \in Z \) and \( \circ \in \{<, \leq, \geq, =, \neq\} \).

A formula over \( \Sigma \), \( \Sigma \)-sorted variable set \( X \) and a set of propositional variables \( \Pi \), also called \( \Sigma \)-formula, is a first-order formula constructed in the usual way using the terms from \( T_\Sigma(X) \), the variables from \( \Pi \), the atoms over \( \Sigma \), the usual Boolean connectives \( \neg, \land, \lor, \rightarrow \) and \( \leftrightarrow \), and the quantifier symbols \( \forall \) and \( \exists \). We write \( F_\Sigma(X, \Pi) \) to denote the set of all formulas over \( \Sigma, X \) and \( \Pi \).

Let \( \text{var}(F) \) denote the set of variables that occur in a formula \( F \). Analogously, \( \text{var}(N) \) denotes the set of variables that occurs in a set \( N \) of formulas. Also, \( \text{var}(x \circ t) \) denotes the
A Dynamic Logic for Configuration Tang and Weidenbach

variable that is the left operand of the atom \( x \circ t \), that is, \( \text{varl}(x \circ t) = x \). If \( N \) is a set containing
atoms over \( \Sigma \), then \( \text{varl}(N) \) denotes the set of variables that are the left operands of the atoms
in \( N \): \( \text{varl}(N) = \{ x | x \circ t \in N \} \). We use \( \text{varl} \) because the relevant terms in the semantics
take the form of simple atoms \( x \circ t \) in which the relevant variables are always on the left side
of the operation by design. Likewise, we define the intersection \( N \restriction M \) of a set \( N \) of formulas
with a set \( M \) of variables as the set \( N \restriction M := \{ F \in N | \text{varl}(F) \cap \text{varl}(M) \neq \emptyset \} \). For example,
\( \{ x \geq w + 4, y \approx 23, z < y, C \} \restriction \{ x, z \} = \{ x \geq w + 4, z < y \} \).

The objects of the signature \( \Sigma \) are interpreted in the usual way as done in the theory of
reals: The sorts \( \mathbb{R} \) and \( \mathbb{Z} \) are interpreted as \( \mathbb{R} \) and \( \mathbb{Z} \), respectively, the symbols are interpreted
as the usual operations and comparisons in \( \mathbb{R} \). We use \( I_\Sigma \), called \( \Sigma \)-interpretation, to denote
both the valuation of a term from \( T_\Sigma(X) \) and the truth value of a formula from \( F_\Sigma(X, \Pi) \).
Again, valuations and truth values are defined as one expects from the first-order logic theory
of reals.

We say that a \( \Sigma \)-interpretation satisfies a \( \Sigma \)-formula \( F \), written as \( I_\Sigma \models F \), if \( I_\Sigma(F) = 1 \).
A formula \( F \) is satisfiable if there is a \( \Sigma \)-interpretation \( I_\Sigma \) with \( I_\Sigma(F) = 1 \), and \( F \) is valid if
\( I_\Sigma \models F \) for all \( \Sigma \)-interpretations \( I_\Sigma \). Formula \( F \) entails formula \( F' \), written as \( F \models F' \), if the
follows holds: If \( I_\Sigma \models F \) and satisfies \( F \), then \( I_\Sigma \) also satisfies \( F' \). A set of
\( \Sigma \)-formulas entails a \( \Sigma \)-formula \( F' \), written as \( N \models F' \), if the following holds: If \( I_\Sigma \models F \) is a
\( \Sigma \)-interpretation and satisfies each \( F \in N \), then \( I_\Sigma \) also satisfies \( F' \).

We consider tuples of integer intervals, which appear in the semantics of PIDL+. As ex-
pected, intersection of a tuple \( t = (t_1, \ldots, t_n) \) with another \( t' = (t'_1, \ldots, t'_n) \) is defined component-
wise: \( t \cap t' := (t_1 \cap t'_1, \ldots, t_n \cap t'_n) \). An intersection is empty, written \( t \cap t' = \emptyset \), if \( t_i \cap t'_i = \emptyset \)
for some \( i \). For an interval \( I = [v_1, v_2] \), the atomic representation \( at(I, x) \) of \( I \) with respect to
a variable \( x \) is the set of arithmetic atoms \( at(I, x) := \{ x \geq v_1, x \leq v_2 \} \).

2.2 Syntax

We now define the syntax of PIDL+. It takes the form of a specification tuple whose components
comprehensively describe a configuration system including its variables, rules and possible user
actions.

Definition 1. A PIDL+ specification is a tuple \( \mathcal{G}_+ = (\Pi, X, S_I, U_I, C, T_U, T_R) \), where the
components are as follows: \( \Pi \) is a finite set of propositional variables, \( X \) is a finite set of \( \Sigma \)-
sorted variables, \( (S_I, U_I) \) is called the initial state, \( C \) is a finite set of \( \Sigma \)-formulas in \( F_\Sigma(X, \Pi) \),
called the constraints, and \( T_U \) and \( T_R \) are finite sets of user and rule transitions \( \Lambda \cap \Pi \setminus F \sim E \),
respectively, where (1) \( S_I \) is a finite set of simple bounds \( x \circ c \) in \( F_\Sigma(X, \Pi) \) and propositional
literals over \( \Pi \) with
\[
\forall \vec{x}_u \exists \vec{y} S_I \mid U_I \rightarrow (S_I \setminus S_I \mid U_I) \cup C
\]
binding being satisfiable, \( \vec{x}_u = x_1 \ldots x_k \) being all the variables in \( U_I \) and \( \vec{y} = y_1 \ldots y_l \) being all the
variables in \( X \setminus U_I \), (2) \( U_I \subseteq X \), called the set of initial user variables, (3) \( \Lambda \) is a conjunction
of simple atoms \( x \circ t \) over \( \Sigma \), called the arithmetic condition, (4) \( F \) is a conjunction of literals
over \( \Pi \), called the propositional condition, (5) \( E \) is called the update set and is a satisfiable
set of simple bounds \( x \circ c \) over \( \Sigma \) and propositional literals over \( \Pi \) for user transitions and is a
satisfiable set of simple atoms \( x \circ t \) over \( \Sigma \) and propositional literals over \( \Pi \) for rule transitions.

The set \( \Pi \) provides the propositional variables needed to describe all the Boolean statements
about the worlds occurring in a configuration process, such as “part \( \Lambda \) is active”, whereas the
set \( X \) represents the numerical variables of the configuration, such as the weight of a component
in kg. The initial state describes the starting world of the modeled system. The set \( S_I \) contains
simple bounds and propositional literals that represent the initial configuration state, together with $U_I$ which indicates which of the system’s variables are set by the user. The models of the specification, as explained in the next subsection, consider further states that arise from this initial state. The requirement that $\forall \vec{x}_u \exists y \ S_I \mid U_I \rightarrow (S_I \setminus S_I \mid U_I) \cup C$ must be satisfiable basically says that the world is consistent: The simple bounds $x \circ c$ in $S_I \mid U_I$ represents the range of user input with respect to that state, and they are consistent with the other simple bounds and the formulas in $C$, the constraints. This consistency requirement is used for all the other states induced by the system, as can be seen in the semantics section. The general and domain-specific constraints of the configuration system are encoded in $C$. For example, it typically contains formulas that express "option A and option B cannot be simultaneously active". Finally, we have two different sets of of transitions, namely user transitions $T_U$ and rule transitions $T_R$, whose elements are constructs $\Lambda \land F \rightsquigarrow E$. The conjunction $\Lambda \land F$ form the condition part of a transition and express what must be met in the current state in order to have a transition from it to another state. There is a clear separation between the arithmetic conditions $\Lambda$ and the propositional conditions $F$. The set $E$ determines how the next state is defined by updating the current state with the atoms and literals in $E$ if a transition is indeed possible. It expresses user actions in the case of $T_U$ and rule actions in the case of $T_R$. User actions can set simple bounds while rules can update with simple atoms, which is a natural representation of a configuration system.

Definition 1 defines specifications in a broad sense. We work with a restricted class of specifications that have certain properties that, in particular, enables us to achieve finiteness of the system and thus decidability. First, we identify the set of all variables involved in transitions, which plays an important role in this.

**Definition 2.** The transition variables $X_T$ of a specification $\mathcal{S}_+$ are the set

$$X_T := \{x \mid x \in \text{var}(\Lambda \land F \rightsquigarrow E), \Lambda \land F \rightsquigarrow E \in (T_U \cup T_R)\}.$$ 

Now the desired kind of specification, which we call admissible specifications, is defined as follows:

**Definition 3.** A specification $\mathcal{S}_+$ is admissible if the following holds. (1) For each $x \in X_T$, sort$(x) = \mathcal{Z}$, (2) the constraints set $C$ has the form

$$\left( \bigwedge_{x \in X_T} x \geq c_{x_1} \land x \leq c_{x_2} \right) \land \left( \bigwedge_{x \in X_T} x \approx t_x \right) \land \Phi,$$

where $c_{x_1}, c_{x_2} \in \mathcal{Z}$ for all $x \in X_T$, $X_r \subseteq X \setminus X_T$, $t_x \in T_\Sigma(X_T)$ for all $x \in X_r$, and $\Phi$ is a propositional formula over $\Pi$, and (3) for each $\Lambda \land F \rightsquigarrow E \in (T_U \cup T_R)$, it holds that if $x \circ t \in E$, then $x \notin \text{var}(t)$.

The composition of the constraints $C$ in admissible specifications is fixed. There, variables occurring in transitions are set to be over the integers and bounded by integer constants. They do not appear anywhere else in the constraints. As is explained later in the semantics, the atoms $x \circ c$ in a state with $x \in U$ of a state stand for the possible instances of the state modulo user choices. By bounding the variables relevant to the transition flow we achieve decidability of the transition system of PIDL+. Variables not in $X_T$ do not affect transitions and function only as holders of result values specific to the current state. Their evaluations are derived from the values of terms over the transition variables. Moreover, admissible specifications may have constraints that are purely propositional, which are encoded in the formula $\Phi$. 
2.3 Semantics

Given a PIDL+ specification, the semantics is a possible-worlds semantics with states and transitions between them, starting from the initial state and adhering to the constraints and transition conditions of the specification. For the rest of the paper, we assume that we only work with admissible specifications. While the meaning of the initial state is briefly given in the syntax section because the initial state is part of the specification, we now define states generally.

**Definition 4.** A *state* is a pair \((S, U)\) of a set of *user variables* \(U \subseteq X\), and a set of literals \(S\) containing simple atoms over \(\Sigma\) and propositional literals over \(\Pi\). The subset \(S|_U\) is always a set of simple bounds over \(\Sigma\).

A state in PIDL+ contains the description of a corresponding state in the configuration system. As mentioned in the syntax section, propositional literals and arithmetic atoms fulfill this task. The user variables \(U\) tell us what variable has been set by the user during the configuration process so far. The simple bounds \(x \circ c\) with \(x \in U\) describe the range of user choices in that state. For example, the state \(\{(x > 4, x \leq 21, A, \{x\})\}\) implicitly corresponds to a set of worlds. It includes an instance in which \(A\) holds and the user has chosen \(x \approx 5\), another instance in which \(A\) holds and the user has set \(x \approx 6\), or one where \(A\) holds and \(x \approx 18\) by a user decision.

We adopt the following convention. Let \((S, U)\) be a state. We write \(\bar{x}_i\) to mean all the variables of \(U\) and \(\bar{y}\) to mean all the variables in \(X \setminus U\). If we use this notation, it should be clear from the context that we refer to the \(U\) of a certain state.

The view that a state is an aggregation of all the possible instances modulo the user decisions, embodied by the simple bounds in \(S|_U\), has far-reaching implications for our semantics. We define transitions later, but we already want to say at this point that different instances of the user variable valuations can imply different behaviors concerning transitions. Consequently, it is necessary to divide a state with respect to its user choices. We do this by considering *selections*.

**Definition 5.** Let \((S, U)\) be a state. Furthermore, let \(\Lambda \land F \sim E \in (T_U \cup T_R)\) be a transition. Then, a *selection* \(\gamma\) with respect to \((S, U)\) and \(\Lambda \land F \sim E\) is defined as follows: (1) If \(U\) is a non-empty set and \(U = \{x_1, \ldots, x_n\}\), then \(\gamma\) is a tuple of interval integers \((I_1, \ldots, I_n)\), where \(\forall \bar{y} (S|_{I_U} \land (S \setminus S|_{I_U} \cup C \rightarrow \Lambda \land F))\sigma\) is valid for all \(\sigma = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}\) with \(v_i \in I_i\) and \(i = 1, \ldots, n\), and (2) if \(U = \emptyset\), then \(\gamma\) is the empty tuple () and \(\forall \bar{y} S \cup C \rightarrow \Lambda \land F\) is valid.

We write \(\gamma(i)\) to denote the \(i\)-th component \(I_i\) of \(\gamma = (I_1, \ldots, I_n)\). The intervals \(I_1, I_2, \ldots, I_n\) of a selection contain values of the user variables \(x_1, x_2, \ldots, x_n\) for which \(\forall \bar{y} (S|_{I_U} \land (S \setminus S|_{I_U} \cup C \rightarrow \Lambda \land F))\) is valid, expressed by all substitutions that map the variables from \(U\) to the values of the intervals. These intervals exist if there is at least one valuation of the user variables that makes the above statement true. If this is the case, we consider the condition of the corresponding transition \(\Lambda \land F \sim E\) to be fulfilled. It can be read as “all assignments done by the user that are based on the values occurring in the selection and that are consistent with the simple atoms of \(S|_U\) entail the transition condition \(\Lambda \land F\), thus that transition is possible”. As is defined later in this section, all transitions are with respect to some selections. If the set of user variables is empty, the criterion for allowing a transition reduces to essentially checking the validity of \(S \cup C \rightarrow \Lambda \land F\).

In our setting, considering subsets of user choices and their properties makes sense. We then talk about *subselections*.
Definition 6. Let $\gamma$ and $\gamma'$ be two selections of the same length. We say that $\gamma'$ is a subselection of $\gamma$, written as $\gamma' \subseteq \gamma$, if $\gamma'(i) \subseteq \gamma(i)$ for all $i = 1, \ldots, n$. We say that $\gamma'$ is a proper subselection of $\gamma$, written as $\gamma' \subset \gamma$, if $\gamma'(i) \subset \gamma(i)$ for an $i \in \{1, \ldots, n\}$.

In general, there can be more than one possible selection with respect to a transition. In our semantics, we restrict the possibilities to consider to maximal selections. Maximal selections are defined with the help of subselections.

Definition 7. A selection $\gamma$ with respect to $(S, U)$ and $\Lambda \land F \rightsquigarrow E$ is maximal if either $\gamma = ()$ or there is no selection $\gamma'$ with respect to $(S, U)$ and $\Lambda \land F \rightsquigarrow E$ such that $\gamma \subset \gamma'$.

Information about current selections may have to be included in the states. This means we have to write the intervals occurring the selections as simple bounds. Atomic representations denote those simple bounds that corresponds to the intervals of the respective selection. They are based on the atomic representations of integer intervals mentioned in the preliminaries of this section.

Definition 8. Let $\gamma$ be a selection with respect to a state $(S, U)$ and a transition. The atomic representation $at(\gamma)$ of $\gamma$ is defined as follows: (1) If $U = \{x_1, \ldots, x_n\}$, then

$$at(\gamma) := \bigcup_{i=1}^{n} at(\gamma(i), x_i),$$

and (2) if $U = \emptyset$, then $at(\gamma) := \emptyset$.

In a transition from a state to a new state, the new state is the result of an update of the former state. The updates are based on the update sets $E$ of the transitions $\Lambda \land F \rightsquigarrow E \in (T_U \cup T_R)$. The way how updates are done depends on the type of the transition: Updates with respect to rule transitions are different from updates with respect to user transitions. We first define updates of rule transitions.

Definition 9. The rule update operator $\triangleleft_R$ takes a state $(S, U)$ and a pair $(E, \gamma)$ as arguments, where (1) $E$ is an update set and (2) $\gamma$ is a selection with respect to $(S, U)$ and a rule transition $\Lambda \land F \rightsquigarrow E \in T_R$. It is defined as $(S, U) \triangleleft_R (E, \gamma) := (S', U')$, where

- $U' := U \setminus varl(E)$,
- $S' := \{L | L \text{ literal over } \Pi, L \in S, \not\in E\} \cup \{L | L \text{ literal over } \Pi, L \in E\} \cup \{x \circ t | x \circ t \in S, x \in X \setminus U, x \not\in varl(E)\} \cup \{x \circ t | x \circ t \in \text{at}(\gamma), x \in U, x \not\in varl(E)\} \cup \{x \circ t | x \circ t \in E\}$.

The propositional literals and simple atoms of the update set $E$ replaces the propositional literals and simple atoms of the old state literal set $S$ if those are of the same propositional variables and left-hand variables, respectively, to form the new state literal set $S'$. We fix the variables on the left-hand sides of simple atoms $x \circ t$ to be the relevant variables determining the updates of atoms. Expressions of $E$ with variables new to the old state are added to $S'$ too. Expressions in $S$ whose propositional variables or left-hand variables do not occur as propositional variables or left-hand variables in $E$ are preserved in $S'$. However, there is a distinction in what is preserved with respect to whether the left-hand variables $x$ of the simple atoms $x \circ t$ are in $U$. If $x$ is not a user variable, then the simple atoms $x \circ t$ from $S$ are just carried over in the next state. If $x \in U$, then it is not the simple atoms from $S$ that are preserved but the simple atoms in the atomic representation $at(\gamma)$ where $x$ appears on the left sides of the atoms. The general rationale behind this is the following: The selection $\gamma$ can
be seen as a subset of the user choices defined by \( S|U \) that makes the transition possible, as described earlier with the definition of selections. It is crucial to keep track of what selection is responsible for the transition. Therefore, a simple atom \( x \circ t \) of \( at(\gamma) \) is contained in the new state by definition unless \( x \) occurs in \( varl(E) \). In that case, the corresponding atom from \( E \) takes precedence and is contained in \( S' \) instead of the one from \( at(\gamma) \). We consider the expressions appearing in the update set \( E \) of a rule transition to be “overwriting” existing user decisions. That is why, after the update, we do not see those variables of \( U \) is responsible for the transition. Therefore, a simple atom \( x \) described earlier with the definition of selections. It is crucial to keep track of what selection \( S \) are not subselections of selections having the same property, analogously to the definition of maximal selections.

In general, if \( (S', U') \neq (S, U) \), we also say that the update or transition changes or alters the state \( (S, U) \).

We can now formally define rule transitions. If a state is consistent and there is an appropriate maximal selection with respect to a transition, then there is a transition from that state to a new state with respect to the corresponding transition, with the new state being the result of an update by the rule update operator \( \varsigma_R \).

**Definition 10.** A rule transition from a state \( (S, U) \) to a state \( (S', U') \) with respect to a rule transition \( \Lambda_i \land F_i \rightsquigarrow E_i \in T_R \) and a selection \( \gamma \) is written as

\[
(S, U) \xrightarrow{\gamma_i(\gamma)} (S', U'),
\]

where (1) \( \forall x \exists \gamma \exists y \ S|U \rightarrow (S \setminus S|U) \cup C \) is satisfiable, (2) \( \gamma \) is a maximal selection with respect to \( (S, U) \) and \( \Lambda_i \land F_i \rightsquigarrow E_i \), and (3) \( (S', U') = (S, U) \setminus (E_i, \gamma) \).

The criterion that \( \forall x \exists \gamma \exists y \ S|U \rightarrow (S \setminus S|U) \cup C \) must be satisfiable is the same as the one used for the initial state \( (S_i, U_i) \) in the definition of specifications. It is the general criterion determining the consistency of a state.

In the configuration systems we consider, as mentioned earlier, all the rules are applied whenever possible until a fixed point is reached. Then, a user decision may happen after which another round of rule applications takes place. As long as rules can be applied that can change a state, no user action is allowed. To define user updates and user transitions, we therefore need the notion of rule-terminal states to denote states that are indeed “ready” for user transitions.

**Definition 11.** Let \( \gamma \) be a maximal selection with respect to a state \( (S, U) \) and a user transition \( \Lambda \land F \rightsquigarrow E \in T_U \). (1) If \( \gamma \neq () \), then we call a selection \( \gamma' \) with \( \gamma' \subseteq \gamma \) a rule-terminal subselection of \( \gamma \) if the following holds: (a) there is no selection \( \gamma^* \) with respect to \( (S, U) \) and a rule transition \( \Lambda^* \land F^* \rightsquigarrow E^* \in T_R \) such that \( \gamma^* \cap \gamma \neq () \) and \( (S, U) \neq (S, U) \setminus (E, \gamma^*) \), and (b) there is no other selection \( \gamma^\# \) that fulfills (a) and \( \gamma' \subset \gamma^\# \). (2) If \( \gamma = () \), then we call \( \gamma \) rule-terminal if (1) is not a maximal selection with respect to \( (S, U) \) and a rule transition \( \Lambda^* \land F^* \rightsquigarrow E^* \in T_R \) such that \( (S, U) \neq (S, U) \setminus (E, ()) \). In the respective cases, we say that \( (S, U) \) is rule-terminal with respect to \( \gamma' \) or () and to \( \Lambda \land F \rightsquigarrow E \).

A state is thus rule-terminal with respect to a selection \( \gamma' \) and a user transition if \( \gamma' \) is such a subselection of a maximal selection \( \gamma \) with respect to the state and the user transition that it does not have a non-empty intersection with another selection that can be used for a rule transition changing the state. A non-empty intersection would mean that the current selection making the user transition possible also contains instances of the user decisions that enable rule transitions that alter the state. Therefore, the state cannot be considered to be a fixed point yet with respect to that selection. What we are interested in is a subselection \( \gamma' \) for which the state becomes a fixed point. To have uniqueness, we only consider those \( \gamma' \) that are not subselections of selections having the same property, analogously to the definition of maximal selections.
The definition of the user update operator is very similar to that of the rule update operator. The difference is that rule updates may reduce the set of user variables, while user updates may increase it. This is because user transitions represent user actions, so the corresponding update set E can introduce new user variables.

**Definition 12.** The user update operator \( \triangleright_U \) takes a state \((S, U)\) and a pair \((E, \gamma)\) such that (1) \(E\) is an update set, (2) \(\gamma\) is a rule-terminal selection with respect to \((S, U)\) and a user transition \(A \land F \sim E \in T_U\), and is defined as \((S, U) \triangleright_U (E, \gamma) := (S', U')\), where \(U' := U \cup \text{varl}(E)\) and \(S'\) is defined in the exact same way as in the case of the rule update operator \(\triangleright_R\).

User transitions are then defined analogously to rule transitions, with respect to rule-terminal states.

**Definition 13.** A user transition from a state \((S, U)\) to a state \((S', U')\) with respect to a user transition \(\Lambda_i \land F_i \sim E_i \in T_U\) and a selection \(\gamma\) is written as 
\[ (S, U) \rightarrow_{(i, \gamma)} (S', U') \],
where (1) \(\forall x \exists \gamma S|U \rightarrow (S \setminus \text{varl}(E) \cup C)\) is satisfiable, and (2) \((S, U)\) is rule-terminal with respect to \(\gamma\) and \(\Lambda_i \land F_i \sim E_i\), and (3) \((S', U') = (S, U) \triangleright_U (E_i, \gamma)\).

A classic notion connected to state transition systems is that of reachability. We give the corresponding concepts in our semantics involving paths and reachability. A path tells us, in particular, what user actions and rule actions were responsible for reaching a certain state, starting from the initial state \((S_0, U_0)\) of a specification.

**Definition 14.** A path from a state \((S, U)\) to a state \((S', U')\) is a tuple \(((i_1, \gamma_1), \ldots, (i_n, \gamma_n))\) such that \((S_0, U_0) \rightarrow_{(i_1, \gamma_1)} (S_1, U_1) \rightarrow_{(i_2, \gamma_2)} \cdots \rightarrow_{(i_n, \gamma_n)} (S_{n-1}, U_{n-1}) \rightarrow_{(i_n, \gamma_n)} (S_n, U_n)\) is a sequence of transitions, where for all \(j = 1, \ldots, n\) it holds that \(\gamma_j\) is a maximal selection with respect to \((S_{j-1}, U_{j-1})\) and a \(\Lambda_{i_j} \land F_{i_j} \sim E_{i_j} \in T_R\) or \((S_{j-1}, U_{j-1})\) is rule-terminal with respect to \(\gamma_j\) and a \(\Lambda_{i_j} \land F_{i_j} \sim E_{i_j} \in T_U\), and moreover \((S_0, U_0) = (S, U)\) and \((S_n, U_n) = (S', U')\).

**Definition 15.** A state \((S', U')\) is reachable from a state \((S, U)\) if there is a path from \((S, U)\) to \((S', U')\). A state \((S, U)\) is always reachable from itself via the empty path.

The question whether a specification is consistent is determined by whether it has a model. A model is basically the set of all states and transitions between them when starting from the initial state, where no inconsistent world according to the definition of \(\Sigma\)-interpretations as described in the preliminaries subsection is reachable.

**Definition 16.** An interpretation of an admissible specification \(\Sigma_+\) is a tuple \((V_{\Phi+}, T_{\Phi+}, I_{\Phi+})\) with (1) the state space \(V_{\Phi+} := \{(S, U) | (S, U)\) reachable from \((S_I, U_I)\}\), (2) the transition space 
\[ T_{\Phi+} := \{(i, \gamma, (S', U')) | (S, U), (S', U') \in V_{\Phi+}, (S, U) \rightarrow_{(i, \gamma)} (S', U'), \Lambda_i \land F_i \sim E_i \in (T_U \cup T_R)\} \],
and (3) the state interpretations 
\[ I_{\Phi+} := \{((S, U), I_S)| (S, U) \in V_{\Phi+}, I_S \models S\} \].

Note that indeed \((S_I, U_I) \in V_{\Phi+}\) according to our definition of reachability. An interpretation is a model if the constraints \(C\) are satisfied in each world.

**Definition 17.** An interpretation \((V_{\Phi+}, T_{\Phi+}, I_{\Phi+})\) is a model of an admissible specification \(\Sigma_+\) if \(I_S \models C\) for each \(((S, U), I_S) \in I_{\Phi+}\).

**Definition 18.** An admissible interpretation \(\Sigma_+\) is sound if it has a model. Moreover, \(\Sigma_+\) is complete with respect to a set \(X_U\) of user variables if for every \(\sigma : X_U \rightarrow \mathbb{Z}\) with \(\mathbb{C}\) being satisfiable, there is \((S, U) \in V_{\Phi+}\) such that \((S \cup C)\sigma\) is satisfiable and for each \(x \in X_U\) there is a user transition \(\Lambda_i \land F_i \sim E_i \in T_U\) with \(x \in \text{varl}(E_i)\), \(((S_{j-1}, U_{j-1}), i, \gamma, (S_j, U_j)) \in T_{\Phi+}\).
Soundness and completeness can be effectively checked, once a finite representation of the transition graph is computed. Such a graph is the result of the algorithm BUILDINTERPRETATION, Algorithm 1.

From a user’s perspective, soundness means that he/she cannot navigate the system into an inconsistent state. Completeness means that within the initial bounds with respect to the overall constraints, any product can be configured by respective user actions for a given set of user controllable variables.

The following theorem follows from the finiteness of the components of admissible specifications and the fact that the transition variables in them are over bounded integers.

**Theorem 19.** The components $V_{\mathcal{G}_+}$ and $T_{\mathcal{G}_+}$ of each interpretation of an admissible specification $\mathcal{G}_+$ are finite. Soundness and completeness are decidable.

We conclude this section with an example.

**Example 20.** Assume the following admissible specification:

$\Pi = \{A, B, C, D, E\}$,

$X = \{x_1, x_2, y_1\}$,

$S_1 = \{\neg A\}$,

$U_1 = \emptyset$,

$C = \{x_1 \geq 0, x_1 \leq 100,$

$x_2 \geq 0, x_2 \leq 100,$

$y_1 \geq 0, y_1 \leq 1000,$

$C \rightarrow D\}$,

$T_U = \{\neg A \sim u_1 \{A, B, x_1 \geq 20, x_1 \leq 90\},$  

$x_1 < 74 \land B \sim u_2 \{x_2 > 5, x_2 < 20\}\}$,

$T_R = \{x_1 \leq 30 \sim r_1 \{\neg C\},$  

$x_2 \geq 10 \land D \sim r_2 \{y_1 \approx x_1 \cdot x_2, C\},$  

$y_1 \geq 600 \land y_1 \leq 800 \sim r_3 \{\neg E\}\}$.

In the following, we describe how the set of states that are reachable from the initial state can be derived. We refer to transitions $A_i \land F_i \sim E_i$ by their indices $i$. From the initial state $(S_1, U_1) = (\{\neg A\}, \emptyset)$, we see that user transition $u_1$ is possible using the empty selection ($\emptyset$), since $U_1$ is empty. With that, the relevant criterion for ($\emptyset$) to be the right selection is that $\forall y[\neg A \cup C \rightarrow \neg A]$ is valid, which is obviously the case. We get a transition $(S_1, U_1) \rightarrow (u_1, (\emptyset)) (S_1, U_1)$, where $S_1 = \{A, B, x_1 \geq 20, x_1 \leq 90\}$ and $U_1 = \{x_1\}$, according to the update $(S_1, U_1) \triangleleft_U (\{A, B, x_1 \geq 20, x_1 \leq 90\})$ as defined in Definition 12. Note that the update overwrites the literal $\neg A$ from the initial state and replaces it with the literal $A$. The new state $(S_1, U_1)$ symbolizes user choices of the variable $x_1$ with integer values between 20 and 90.

From $(S_1, U_1)$, there are two possible transitions, one of which is a rule transition and the other is a user transition. First, rule transition $r_2$ is applicable for the maximal selection $\{20, 30\}$. Note that indeed $\forall y[S_1\mid U_1 \land (S_1 \setminus S_1\mid U_1 \cup C \rightarrow x_1 \leq 30)]$ is valid for all $\sigma = \{x_1 \mapsto v\}, v \in [20, 30]$ as required in Definition 5. We register a transition $(S_1, U_1) \rightarrow (r_2, \{20, 30\}) (S_2, U_2)$, where $S_2 = \{A, B, \neg C, x_1 \geq 20, x_1 \leq 30\}$ and $U_2 = \{x_1\}$, according to $\triangleleft_{r_2}$ as defined in Definition 9. On the other hand, there is a rule-terminal selection with respect to $(S_1, U_1)$ and user transition $u_2$. The maximal selection with respect to $(S_1, U_1)$ and user transition $u_2$ is $[31, 73]$. The subselection that is rule-terminal with respect to that is $[31, 73]$. No other rule transition that changes the state is possible in this subselection. Thus, we have
A Dynamic Logic for Configuration

Tang and Weidenbach

\((S_1, U_1) \rightarrow \gamma_{u_2, ([31, 73])} (S_3, U_3)\), where

\(S_3 = \{A, B, x_1 \geq 31, x_1 \leq 73, x_2 > 5, x_2 < 20\}\) and \(U_3 = \{x_1, x_2\}\),

which represents that the user has now additionally set \(x_2\). State \((S_3, U_3)\) warrants rule
transition \(r_2\) with the selection \(([31, 73], [10, 19])\). We get

\((S_3, U_3) \rightarrow \gamma_{u_2, ([31, 73], [10, 19])} (S_4, U_4)\), where

\(S_4 = \{A, B, C, x_1 \geq 31, x_1 \leq 73, x_2 \geq 10, x_2 \leq 19, y_1 = x_1 \cdot x_2\}\) and \(U_4 \approx \{x_1, x_2\}\).

Finally, there are more than one possible selections to enable rule transition \(r_3\). If we choose
the selection \(([50, 61], [12, 13])\), we get

\((S_4, U_4) \rightarrow \gamma_{r_3, ([50, 61], [12, 13])} (S_5, U_5)\), where

\(S_5 = \{A, B, C, \neg E, x_1 \geq 50, x_1 \leq 61, x_2 \geq 12, x_2 \leq 13, y_1 \approx x_1 \cdot x_2\}\) and \(U_5 = \{x_1, x_2\}\).

Again, note that all selections given in this example are maximal in the sense that they are
not subselections of any other selections with the same properties demanded in the definition
of transitions and rule-terminal states.

3 Algorithms

The algorithm \textsc{buildInterpretation}, Algorithm 1, computes an abstract partial interpretation
according to Definition 16 of an admissible specification if it exists. It takes an admissible
specification \(\Theta_+\) and computes the components \(V_{\Theta_+}\) and \(T_{\Theta_+}\) of a possible interpretation of
\(\Theta_+\), named \(V\) and \(T\) in the algorithm. It aborts and returns “inconsistent” if it encounters
an inconsistent state, which means the specification is inconsistent. Otherwise, it returns the
resulting state graph \(G = (V, T)\) that can serve as the basis for further analyses of the properties
of the specification and thus of the modeled configuration system.

A central element is the computation of selections, that is, integer intervals as defined in
the semantics (Definition 5). Since the intervals represent valuations of bounded user variables,
as explained in the previous section, this task is decidable and can be carried out in the most
naive way by enumerating the finitely many possibilities. A more efficient alternative to this is
an appropriate use of interval arithmetic [11]. In this paper, we restrict ourselves to assuming
that selections relevant to transitions as defined in the semantics (Definitions 10 and 13)
are available in the form of \(\text{maxSelections}_{\Theta_+}\), given a state and a transition.

**Definition 21.** Let \((S, U)\) be a state, and \(\Lambda \land F \sim E \in (T_L \cup T_R)\) be a transition. Then
\(\text{maxSelections}_{\Theta_+}(S, U, \Lambda \land F \sim E)\) is the set of all maximal selections \(\gamma\) with respect to \((S, U)\)
and \(\Lambda \land F \sim E\).

\textsc{buildInterpretation} starts with the initial state of the input specification, as shown by
the initialization of \(N\), which is the set of states not yet processed by the algorithm. The set
\(H\) stores all the states that the algorithm has computed so far. The algorithm runs as long as
there are states to be processed (line 5). In each iteration, consistency of the current state is
checked (line 7) first, using an external solver, according to the same consistency criterion as
stated in the transition semantics (Definitions 10 and 13). The algorithm stops if the state is
not consistent, otherwise it continues. Then, for each suitable maximal selection with respect
to the current state and a rule transition, a new state is computed according to \(\gamma_H\) (line 11).
The new transition is registered in line 12. If the new state differs from the current one, the
current selection is one of the selections under which the state is truly changed, and these are
collected in a set \(\Gamma\) (line 14). Also, if the newly generated state has not appeared before, it is
added to \(V\), \(H\) and \(N\) (lines 15-18). This makes sure that we do not process states more than
once.
After that, the user transitions with respect to the current state are investigated. If \( \Gamma = \{ (i) \} \) (check in line 19), it means that there has been a selection, namely \((i)\), that changes the state with the set of user variables being empty (Definition 5). This means the state is not rule-terminal in any case by Definition 11. If this is not the case, it is checked if the current state is rule-terminal with respect to any selections. Given the current maximal selection \( \gamma \) with respect to the current state and current user transition, and the set \( \Gamma \) of all selections where rule transitions alter the state, \textsc{reduceSelection} (Algorithm 2) is used to find all rule-terminal subselections of \( \gamma \) (line 22). The subselections \( \gamma' \) returned by \textsc{reduceSelection} that are sound and complete (Theorem 22), have the following property, as explained further below: If \( \gamma' \) does not include an empty interval or \( \gamma' \) is the empty selection \((\cdot)\), then the current state is rule-terminal with respect to \( \gamma' \) and the current user transition (line 23). \textsc{buildInterpretation} then creates a new state with \( \gamma' \) and does the same steps as in the case of rule transitions. At the end of the while loop, we remove the current state from the set \( N \) of states to be processed (line 33). Provided that the algorithm has not encountered an inconsistent state, it returns \( V \) and \( T \), which are the components of every interpretation and, in particular, every model of the consistent specification.

We give an overview on the recursive algorithm \textsc{reduceSelection} in the following. It takes as arguments a selection \( \gamma \) and a set \( \Gamma \) of selections of the same length, and it returns a set of selections \( \Delta \) such that all elements of \( \Delta \) are (1) subselections of \( \gamma \) and (2) have only empty intersections with the selections in \( \Gamma \). In addition, each selection \( \delta \in \Delta \) is maximal in the sense that there is no other selection \( \delta' \) with \( \delta \subset \delta' \) and that fulfills (1) and (2). The basic idea is that, in each recursive step, \( \gamma \) is reduced to a subselection so that it just about has an empty intersection with one selection \( \gamma' \) from \( \Gamma \). That selection \( \gamma' \) is removed from \( \Gamma \) for the next recursion step, and once \( \Gamma \) has become empty, we get a reduced \( \gamma \) that has the above properties. This is why the current \( \gamma \) in line 2 is returned as a singleton. If \( \Gamma \) is not empty, we keep removing selections \( \gamma' \) from \( \Gamma \) until we have removed everything or we have found a selection \( \gamma' \) that has a non-empty intersection with the current \( \gamma \) (lines 3 to 7). Again, \textsc{reduceSelection} returns \( \{\gamma\} \) if it turns out that it has no non-empty intersections with the rest of \( \Gamma \) (line 9). If we now assume we have found a \( \gamma' \) with non-empty intersection with \( \gamma \), the next step is to reduce \( \gamma \) so that there is an empty intersection. There is potentially more than one possibility to do the reduction. At each position \( i \) of \( |\gamma| \) positions, we consider the difference \( \gamma(i) \setminus \gamma'(i) \) (line 12) and replace \( \gamma(i) \) with that difference, thus making \( \gamma \) having an empty intersection with \( \gamma' \). Another branching action can be required at this point: If the set difference consists of two intervals that are not connected (line 13), which is the case if \( \min(\gamma) < \min(\gamma') \leq \max(\gamma') < \max(\gamma) \), then we consider two separate recursive calls of \textsc{reduceSelection} with \( \gamma(i) \) being replaced by each one of the intervals respectively (line 14). Otherwise, we simply replace \( \gamma(i) \) with the difference (line 16) in one recursive call. We collect all the possible solutions into a result set \( R \).

Theorem 22 says that the use of \textsc{reduceSelection}(\( \gamma, \Gamma \)) in line 22 of \textsc{buildInterpretation} gives us the correct rule-terminal subselections as defined in the semantics in a complete way. Note that the results of \textsc{reduceSelection} can include “extra” subselections that have been so much reduced from \( \gamma \) that they contain empty components. These selections are discarded by \textsc{buildInterpretation} because they are no longer selections with respect to the current state and user transition (line 23). The case \( \gamma' = (\cdot) \) means that there are no user variables (Definition 5) and that \( \Gamma = \emptyset \), since otherwise \( \Gamma = \{ (i) \} \), which is not the case (line 19). Consequently, \textsc{reduceSelection} immediately returns \( \{ (i) \} \), a singleton containing a rule-terminal selection.

**Theorem 22.** Let \( \gamma \) be a maximal selection with respect to a state \( (S,U) \) and a user transition
Algorithm 1: buildInterpretation(S+)

\begin{algorithmic}
\State $N := \{(S_I, U_I)\}$
\State $H := \{(S_I, U_I)\}$
\State $V := \{(S_I, U_I)\}$
\State $T := \emptyset$
\While{$N \neq \emptyset$}
\State $\forall S, U \in N$
\If{$\exists S, U \to (S \setminus S_U) \cup C$ is satisfiable}
\State $\Gamma := \emptyset$
\For{each $\Lambda_i \land F_i \leadsto E_i \in T_R$}
\For{each $\gamma \in \text{maxSelections}_{\Theta_+}(S, U, \Lambda_i \land F_i \leadsto E_i)$}
\State $(S', U') := (S, U) \triangleleft (E_i, \gamma')$
\State $T := T \cup \{(S, U, i, \gamma, (S', U'))\}$
\EndFor
\EndFor
\EndIf
\If{$\Gamma \neq \emptyset$}
\For{each $\Lambda_i \land F_i \leadsto E_i \in T_U$}
\For{each $\gamma' \in \text{reduceSelection}(\gamma, \Gamma)$}
\If{$\gamma'(j) \neq \emptyset$ for all $j$ or $\gamma' = ()$}
\State $(S', U') := (S, U) \triangleleft (E_i, \gamma')$
\State $T := T \cup \{(S, U, i, \gamma', (S', U'))\}$
\EndIf
\EndFor
\EndFor
\If{$(S', U') \notin H$}
\State $V := V \cup \{(S', U')\}$
\State $H := H \cup \{(S', U')\}$
\State $N := N \cup \{(S', U')\}$
\EndIf
\Else
\State return inconsistent
\EndIf
\State $N := N \setminus \{(S, U)\}$
\EndWhile
\State return $(V, T)$
\end{algorithmic}

A selection $\gamma^*$ with $\gamma^* \subseteq \gamma$ and $\gamma^*(i) \neq \emptyset$ for all $i$ is a rule-terminal subselection of $\gamma$ if and only if $\gamma^* \in \text{reduceSelection}(\gamma, \Gamma)$, where $\Gamma$ is the set of all selections $\gamma'$ such that $\gamma'$ is a maximal selection with respect to $(S, U)$ and a rule transition $\Lambda' \land F' \leadsto E' \in T_R$ with $(S, U) \triangleleft (E', \gamma')$.

Proof. (Idea) Using associativity of set differences, the fact that all selections occurring as the first argument in reduceSelection are subselections of the input selection $\gamma$ and induction over the recursion depth.

Apparently, each state computed by buildInterpretation implies a path of transitions as indicated by the set $T$ in the algorithm. It is easy to see they are analogous to the paths defined
Algorithm 2: reduceSelection(γ, Γ)

1. if Γ = ∅ then
2. return {γ}
3. let γ′ ∈ Γ
4. Γ := Γ \ {γ′}
5. while Γ ≠ ∅ and γ ∩ γ′ = ∅ do
6. let γ′ ∈ Γ
7. Γ := Γ \ {γ′}
8. if γ ∩ γ′ = ∅ then
9. return {γ}
10. R := ∅
11. for i = 1 to |γ| do
12. s := γ(i) \ γ′(i)
13. if s = I1 ∪ I2, I1, I2 intervals and I1 ∩ I2 = ∅ then
14. R := R ∪ reduceSelection(γ[i/I1], Γ) ∪ reduceSelection(γ[i/I2], Γ)
15. else
16. R := R ∪ reduceSelection(γ[i/s], Γ)
17. return R

in the semantics in Definition 14. Theorem 23 then says buildInterpretation is sound and complete with respect to the semantics of PIDL+.

Theorem 23. Let $\mathcal{S}_+ = (\Pi, X, S_I, U_I, C, T_U, T_R)$ be an admissible specification. A state (S, U) is reachable from the initial state (S_I, U_I) via a path $\tau$ if and only if (S, U) ∈ V and we have a sequence $((S_0, U_0), i_1, \gamma_1, (S_1, U_1)), \ldots, ((S_{n-1}, U_{n-1}, i_n, \gamma_n, (S_n, U_n))$, where $((S_{j-1}, U_{j-1}), i_j, \gamma_j, (S_j, U_j)) ∈ T$ and $\tau(j) = (i_j, \gamma_j)$ for all $j = 1, \ldots, n$, $|\tau| = n$, $(S_0, U_0) = (S_I, U_I)$, $(S_n, U_n) = (S, U)$ and buildInterpretation($\mathcal{S}_+$) = (V, T).

Proof. (Idea) By induction over the length of the path $\tau$. □

4 Conclusions

We have presented PIDL+, a logic suited for formalizing configuration systems whose dynamics stem from the effects of user interaction and rule actions. Boolean variables and arithmetic expressions of the configurations are considered. The choice of polynomials is motivated by the real-world systems we have already investigated for the design of PIDL+. Of course, PIDL+ can be extended to other suitable choices of closed arithmetic expressions. What distinguishes PIDL+ from similar logics is that it takes the characteristic features of such configuration systems into account by incorporating two types of transitions and the notion of rule-terminal states. This, in principle, forms the basis for a systematic analysis of the whole configuration process, making an a priori verification of those systems possible. Natural next steps include the implementation of the shown decision procedures and conducting corresponding experiments on real-world data. Related to that is the development and implementation of further decision procedures to investigate properties of interests of configuration systems, as we have done in the case of PIDL [9].
References


TPTP And Beyond:
Representation of Quantified Non-Classical Logics*

Max Wisniewski¹, Alexander Steen¹, and Christoph Benzmüller²¹

¹ Freie Universität Berlin, Institute of Computer Science
    {m.wisniewski,a.steen}@fu-berlin.de
² Stanford University, CSLI
    c.benzmueller@gmail.com

Abstract

The practical employment of automated deduction systems requires the user to input
problem statements in a well-formed string representation. While this presentation is usu-
ally fixed by the respective system, the various language dialects of the TPTP library
are meanwhile accepted as a de-facto standard for all current automated theorem provers
based on classical logics. In the context of reasoning in non-classical logics, however, only a
few limited standardization approaches exist, with QMLTP being the most notable excep-
tion. To move standardization forward, we outline conservative extensions to the TPTP
language that allow systematic syntax definitions for various expressive, non-classical log-
ics. These logics include higher-order versions of modal logics, conditional logics, hybrid
logics, free logics, and many-valued logics. We are convinced that a standard syntax for
prominent non-classical logics will not only facilitate their deployment but also support
the development and comparability of corresponding theorem proving systems.

1 Introduction

Computer-assisted reasoning in non-classical logics is of increasing interest to enable and
support applications in e.g. computer science, mathematics and philosophy. Several pow-
erful automated and interactive theorem proving systems have been developed over the
past decades. However, when it comes to quantified logics, most of the available systems
focus on classical logic only. Amongst the notable exceptions is MleanCoP [26] which
automates first-order modal logic.

Orthogonal to the development of specialized provers, the semantical embedding ap-
proach [7] allows for a quick adaptation of existing higher-order reasoning systems to a
broad variety of expressive, non-classical logics. In fact, for each logic discussed in this
paper, we already have a new theorem provers in place [7, 5, 9, 8, 32]. These reasoners
have been implemented by utilizing the embeddings approach on top of systems such as
Leo-II [4] or Isabelle/HOL [22]. Recent experiments show that this approach indeed offers
a surprisingly effective automation of the embedded non-classical logics. However, from the
users perspective the utilization of the embeddings approach can become rather involved
and distracting. Hence, system users may eventually not want to be exposed to the embed-
dings at all. Moreover, a comprehensive evaluation of systems based on the embeddings
approach against systems based on the direct approach is currently hardly feasible. One
reason, in addition to the fact the very few systems in the direct approach are available to
date, is the lack of commonly agreed input formats.

In order to amplify the practical development, deployment and comparison of auto-
mated reasoning in quantified non-classical logics, we therefore outline problem represent-
ation formats for various (quantified) non-classical logics, primarily for use in automated

---

*This work has been supported by the DFG under grant BE 2501/11-1 (Leo-III) and grant BE 2501/9-2
(Computational Metaphysics).
theorem proving (ATP) systems. More specifically, we present conservative extensions to the well-known TPTP [34] syntax representations. We display proposals on how to represent logical problems in quantified versions of multi-modal logics, hybrid logics, conditional logics, free logics and some propositional many-valued logics. Additionally, we include means of adding meta-logical information to the problem statement that specifies details regarding the assumed semantics of the respective problem and logic. We briefly introduce each of the mentioned logics and describe the necessary modifications of the already existing languages.

**TPTP and QMLTP.** The Thousands of Problems for Theorem Provers problem library (TPTP) [34] provides a coherent environment for testing automated theorem provers for their correctness and performance. To that end, it postulates a standardized and stable formula representation syntax for most classical logic languages (e.g. FOF for first-order formulas or THF [35] for typed higher-order formulas). We will base our problem representation format on the THF dialect and moderately extend the existing syntax definitions to match the requirements of the particular non-classical logic in question.

A closely related project is QMLTP [27] which provides a syntax and a collection of problems for first-order modal logic. The QMLTP syntax is designed as an extension of the TPTP FOF language, introducing special symbols for the box and the diamond operators of modal logic (cf. §3.1). Regarding modal logics (or logics based on modal logic), we will also re-use existing syntax representations of the QMLTP project. Moreover, we adopt and extend the QMLTP approach for the specification of meta-logical information.

## 2 Classical Higher-Order Logic

We primarily address quantified versions of non-classical logics in this paper. Since we do not intend to (artificially) restrict these logics to be first-order only, their representation formats will quite naturally be given as extensions of (classical) higher-order logic (HOL) [13]. The syntax and semantics of HOL is now briefly introduced as it serves as a basis for later, when logic-specific definitions of the syntax and semantics of non-classical higher-order logics are depicted. The brief introduction to HOL is mainly borrowed from [33] which, in turn, adapts the simplified notation of [21] for HOL.

HOL is a typed logic. The set of simple types $\mathcal{T}$ contains all types that are freely generated using the binary function type constructor $\to$ and a set of base types, usually chosen to be $\{o, \iota\}$ for Booleans and individuals, respectively. Terms of HOL are given by the following grammar:

$$s, t ::= c_\tau \mid X_\tau \mid (\lambda X_\tau. s_\tau)_{\tau \to o} \mid (s_\tau t_\tau)_{\nu}$$

where $c_\tau \in \Sigma_\tau$ is a constant symbol from the (typed) signature $\Sigma := \bigcup_\tau \Sigma_\tau$ and $X_\tau$ is a variable. The type of a term is explicitly stated as subscript but may be dropped for legibility reasons if obvious from the context. Terms $s_o$ of type $o$ are formulas.

In general, we require $\Sigma$ to contain a complete logical signature. To that end, we choose $\Sigma$ to consist at least of the primitive logical connectives for disjunction, negation, and, for each type, equality and universal quantification. Hence, we have $\{\lor, \land, \neg, =_{\tau, \tau \to o}, \Pi_{\tau \to o}(\lambda \xi. s_o)_{\tau \to o}, \Pi_{\tau \to o} s_o\} \subseteq \Sigma$ for all $\tau \in \mathcal{T}$.\(^1\) Optionally, we add choice operators and definite description operators $\iota$ for all types. Depending on the logics we are addressing in the following, the concrete set of constants (hence also connectives, quantifiers, etc.) $\Sigma$ will actually vary. Often, we will only add further constants to the above ones. In all other cases, we will specify $\Sigma$ explicitly.

\(^1\) The remaining logical connectives can be defined as usual, e.g. conjunction by $\land := \lambda s_o. \lambda t_o. \neg (\neg s \lor \neg t)$. 

---

52
The semantics of HOL is now briefly addressed. A frame \( \{D_r\}_{r \in T} \) is a collection of non-empty sets \( D_r \) such that \( D_c = \{T,F\} \) (for truth and falsehood, respectively) and \( D_{r \rightarrow o} \subseteq D_o^{D_r} \) is a collection of functions from \( D_r \) to \( D_o \). An interpretation is a pair \( M = (\{D_r\}_{r \in T}, \mathcal{I}) \) where \( \{D_r\}_{r \in T} \) is a frame and \( \mathcal{I} \) is a function mapping each constant \( c_r \) to some denotation in \( D_r \). We assume that the primitive logical connectives are assigned their usual denotation. Given a variable assignment \( g \) we can define a valuation \( \|\cdot\|_{M,g} \) by

\[
\begin{align*}
\|c_r\|_{M,g} &= \mathcal{I}(c_r) \\
\|X_r\|_{M,g} &= g(X_r) \\
\|s_{r \rightarrow o}\|_{M,g} &= \|s_{r \rightarrow o}\|_{M,g} \\
\|\lambda X_r. s_o\|_{M,g} &= \left\{ f : z \mapsto s_{M,g}[z/X_r] \right\} \in D_{r \rightarrow o}
\end{align*}
\]

where \( g[z/X_r] \) denotes the variable assignment that maps \( X_r \) to \( z \) and every other variable \( Y_o \) to \( \sigma(Y_o) \), where \( Y_o \neq X_r \).

A formula \( s_o \) is called valid, iff \( \|s_o\|_{M,g} = T \) for every variable assignment \( g \) and every interpretation \( M \). We call \( M \) a standard model iff \( D_{r \rightarrow o} = D_r^{D_o} \). As a consequence of Gödel’s Incompleteness Theorem [17], HOL with standard semantics is necessarily incomplete. However, if we allow \( D_{r \rightarrow o} \) to be a proper subset of \( D_r^{D_o} \) with the constraint that \( \|\cdot\| \) remains total, a meaningful notion of completeness can be achieved [19]. We assume this so-called Henkin semantics in the following.

A de-facto standard representation of HOL problems for automated theorem provers is given by the THF dialect [35] of the TPTP syntax [34]. This representation syntax is supported by most current HOL ATP, including Satallax [11], LEO-II [4], agsyHOL [20], Isabelle/HOL [22] and many others. A small example problem encoded in THF is displayed in Fig. 1. The circumflex \(^\wedge\) and the \( @ \) denote \( \lambda \)-abstraction and function application, respectively. Types can be stated explicitly (cf. first line of Fig. 1), where \( > \) denotes the function type constructor \( \rightarrow \). Most remaining operations are standard TPTP syntax as used in first-order syntax.

We will use the THF dialect as a starting point for the development of specific representations of quantified non-classical logics in the following section.

3 Representation of Non-Classical Logics

In this section, we outline possible conservative extensions to the TPTP THF dialect in order to capture various quantified non-classical logics, to be used as input language of suitable ATP systems. The here discussed logics are modal logics, hybrid logics, conditional logics and free logics, each of them in a higher-order quantified version. Also, we briefly discuss means of representation for many-valued logics.

3.1 Modal Logics

"Modal logic" refers to a family of non-classical logics that are used to express and reason about modal qualities of truth. To that end, the operators \( \Box \) and \( \Diamond \) are added to the usual
classical logic language and characterized by appropriate rules and axiomatizations. Notions of necessity and possibility are probably the most prominent of such modal concepts represented by the new operators, but many further related systems and interpretations (e.g. focusing on temporal or deontic aspects) exist. Modal logics are not only of strong interest for the interpretation of philosophical arguments, but have also become increasingly important to mathematics and computer science [18].

**Syntax and semantics.** We now briefly sketch the syntax and semantics of higher-order modal logics (HOML) [21] by augmenting the appropriate definitions of HOL as given in §2. We here assume a multi-modal logic, that is a modal logic consisting of multiple, different box operators \( \Box^i \), \( i \in I \) (and corresponding diamond operators), for some index set \( I \).

The syntax definition is nearly identical to that of HOL. We merely add the box operators \( \Box_i \) (for all \( i \in I \)) to the set of constants \( \Sigma \). Their duals, the diamond operators \( \Diamond_i \), can be defined by \( \Diamond_i := \lambda \Phi. \neg(\Box^i \neg \Phi) \).

For the semantics of HOML, we augment the concept of a HOL model with Kripke (possible world) semantics, yielding a HOML model structure \( M \):

\[
M = \left\{ W, \{ R^i \}_{i \in I}, \{ D_i \}_{i \in T}, \{ I_w \}_{w \in W} \right\}
\]

where \( W \) is a set of worlds, the \( R^i \subseteq W \times W \) are accessibility relations between the worlds of \( W \), and each \( I_w \) is an interpretation function (similar to \( I \) of §2) for world \( w \). We assume that connectives are always given the standard interpretation by \( I_w \) in each world \( w \in W \).

In a final step, we augment the valuation function \( \| \|^{M, \sigma} \) for HOL models to a valuation function \( \| \|^{M, \sigma, w} \) for HOML models \( M \), a variable assignment \( g \) and a world \( w \in W \) by

\[
\| \Box^i s_o \|^{M, \sigma, w} = T \text{ iff for all } v \in W \text{ such that } wR^i v \text{ it holds that } \| s_o \|^{M, \sigma, v} = T
\]

The semantics definition stated here are only adequate for constant domain semantics in which we assume the domains \( D_i \) to be the same for all worlds \( w \in W \). However, if we assume varying domain semantics (or their restricted forms of cumulative or decreasing domains), we need to further augment the above model. Instead of a single frame \( D := \{ D_i \}_{i \in I} \) we employ a family of frames \( \{ D_w \}_{w \in W} \), one for each world. Additionally, the valuation of universal quantification is appropriately adjusted. We refer to the literature for details (cf. e.g. [15]). Regarding the quantification constants \( \Pi^\tau, \tau \in T \), we might want to allow mixed-semantics quantification statements, i.e. formulas where multiple quantifications are contained, each possibly with different semantics.\(^2\) This can simply be done by adding different quantification constants for the respective different quantification semantics to the signature, yielding \( \Sigma = \{ \ldots, \Pi^{\tau, w}, \Pi^{\tau, inc}, \Pi^{\tau, dec} \} \) for each type \( \tau \in T \) for constant, varying, cumulative and decreasing domain quantification semantics, respectively.

We are in a higher-order setting. Hence, bridge rules (e.g. \( \Box \phi \Rightarrow \Box^i \phi \)) can simply be postulated as axioms (e.g. \( \forall \phi (\Box^i \phi \Rightarrow \Box^i \phi) \)). Consequently, we can avoid a specific representation for bridge rules below. However, such axioms should possibly be marked specifically so that provers can easily recognize them (similar to TPTP definitions) and apply special techniques where possible.

The above definitions give us modalities with logic \( K \) properties. In order to obtain stronger logics, such as \( KB, KD, S4 \) and \( S5 \), we e.g. could, analogous to the above bridge rules, postulate respective axioms. However, it seems to us that this approach would be impractical and too verbose. Hence, we include a special syntax for postulating frame conditions to the modalities below.

\(^2\) Mixed uses of constant and varying domain quantifiers occur for example in variants of the ontological argument for the existence of God; cf. Anderson [1], footnote 14.
**Representation.** We adopt the representation of QMLTP for representing the box and diamond operators, i.e. by writing \#box for □ and \#dia for ◊. Since we are in a multi-modal setting, we qualify these connectives with an appropriate identifier, called index: \#box(i) for □ i and \#dia(i) for ◊ i. The modal operators are then used similar to quantifiers in TPTP: \#box(a): t represents the formula □ a t to. The remaining syntax is standard THF.

A short example (where a and b are identifiers from I) is given by:

```
hmf(1, type, (p: $i > $o)).
hmf(2, conjecture, ((#box(a): (! [X: $i]: p @ X)) => (#dia(b): (? [X: $i]: p @ X)))).
```

The ! quantifier respects some quantification semantics that is chosen by the user. Additionally, we add four new quantifiers !*, !~, !+, !- that always denote constant, varying, cumulative and decreasing domain quantification semantics, respectively, regardless of the default setting for !.

**Global parameters.** There are several parameters that adjust the exact meaning of modal logic problems. One of this parameters was already mentioned above, namely whether we use constant domain or variants of varying domain semantics. Another important point is the rigidity of constant symbols: Does every symbol denote the same object in every world? The global parameters for the problem input considered here are:

- **quantification** Sets the quantification semantics for the ! and ? symbols of the language.
  - Valid values: constant, varying, cumulative, or decreasing.
  - Default value: constant.
- **constants** Sets the default interpretation constraint for constant symbols, i.e. whether constant symbols have the same denotation in every world (called rigid) or not.
  - Valid values: rigid or dependent.
  - Default value: rigid.
- **consequence** Specifies the precise meaning of the logical consequence relation S |= t where S = {s1,...,sn} is a set of formulas. In the global case we have: S |= t if ∀M, g.∀w.(∥s1∥M,g,w = T and ... and ∥sn∥M,g,w = T) implies ∥t∥M,g,w = T. In the local case we instead have S |= t if ∀M, g.(∀w.∥s1∥M,g,w = T and ... and ∀w.∥sn∥M,g,w = T) implies ∀w.∥t∥M,g,w = T).
  - Valid values: local or global.
  - Default value: local.
- **modalities** Sets which different modalities are defined within the problem. For each indexed modality, the respective index name is given. If not stated, a mono-modal logic is assumed where the default box and diamond operators (i.e. \#box and \#dia without name qualification) are used in the problem.
  - Example: (a,s5) defines an indexed modality named a with S5 axiomatization.

These parameters need to be included in the problem description using the TPTP process instruction language (TPI) which, amongst other aspects, allows adding meta statements about the problem setting. An exemplary multi-modal setting with cumulative domain semantics, rigid constant interpretation, and a global consequence relation is given by

```
tpi(1, set_logic, modal(['quantification' = 'cumulative', 'constants' = 'rigid', 'consequence' = 'global', 'modalities' = [(a, s5), (b, kb), (c, k)]])).
```

\(^3\)A proposal for the TPI language can be found at [http://www.cs.miami.edu/~tptp/TPTP/Proposals/TPILanguage.html](http://www.cs.miami.edu/~tptp/TPTP/Proposals/TPILanguage.html).
Here, three different indexed modalities $a$, $b$ and $c$ are introduced with the given axiomatizations S5, KB and K, respectively. Valid axiomatization schemes for modalities include $k$, $kb$, $k4$, $k5$, $d$, $n$, $b$, $s4$ and $s5$. There have been more systems presented in the literature. Hence, this list could/should be appropriately extended.

**Per-Symbol Options.** As a convenience feature, we allow per-symbol specification of rigidity, allowing some symbols to be rigid and some symbols to be world-dependent. More specifically, all symbols introduced using type statements have the default rigidity as stated by the constants option of the set_logic statement, unless overridden by another set_logic statement specifically for that new constant. In the following example, the constant symbol $q$ is a rigid symbol while $p$ is world-dependent as stated by statement 4:

$$
\text{tpi}(1, \text{set_logic}, \text{modal}([\text{'quantification'} = '\text{cumulative}',
\text{'constants'} = '\text{rigid}',
\text{'modalities'} = [(a, s5), (b, kb), (c, k)]]).
\text{hmf}(2, \text{type}, (q: \$i)).
\text{hmf}(3, \text{type}, (p: \$i)).
\text{tpi}(4, \text{set_logic}, \text{hmf}(p, '\text{dependent}')).
$$

### 3.2 Hybrid Logics

Hybrid logic [10] is a general term for extensions of ordinary modal logics that introduce a new sort of atomic formulas – the so-called nominals. Nominals introduce one convenient feature to modal logic, that is referencing and arguing about worlds. In natural language it is a common construct to refer to a specific point in time or the knowledge of one particular person. Although modal logic is used to model these problem domains, it is not possible to refer to the underlying world structure. Hybrid logic allows to evaluate a formula in a specific world – with the satisfaction operator $@$ – and to bind the current world to a variable – with the shift operator $\downarrow$.

**Syntax and semantics.** Classically, nominals are introduced in propositional and first-order logic by introducing new cases into the syntax BNF and adjusting the models accordingly. Additional to HOML we need to introduce the nominals, the satisfaction operator $@$, and the shifter $\downarrow$ to the BNF over a set $\text{NOM}$.\(^4\)

$$
n, t ::= \ldots | n_o | @ (n_o) s_o | \downarrow s_{o \rightarrow o} | \ldots \quad n \in \text{NOM}
$$

The semantics is build on the same model as ordinary modal logic.

$$
\mathcal{M} = (W, \{R^i\}_{i \in I}, \{D^r\}_{r \in T}, \{I^w\}_{w \in W})
$$

The variable assignment $g = (g', g^n)$ is adorned with an extra variable assignment $g^n$, that maps nominals to worlds. The assignment $g'$ is the variable assignment from standard higher-order modal logic.

The valuation $\parallel \cdot \parallel$ is then augmented for formulas containing nominals, given by

$$
\parallel n \parallel^{\mathcal{M}, g, w} = g^n(n) \equiv w, \quad n \in \text{NOM}
\parallel @ n \varphi \parallel^{\mathcal{M}, g, w} = \parallel \varphi \parallel^{\mathcal{M}, g, g^n(n)}
\parallel \downarrow \varphi_{o \rightarrow o} \parallel^{\mathcal{M}, g, w} = \parallel \varphi \parallel^{\mathcal{M}, g, w} \quad \text{where } n \text{ is free in } \varphi
$$

\(^4\)Higher-Order Hybrid Logics have not yet been researched. They can be conceived as a straightforward adaptation of propositional and first-order hybrid logic to HOL. Another possibility is to introduce a new type for nominals. This would yield a solution, that resembles the embedding approach [36].

56
Since hybrid logic is an extension of ordinary modal logic, we can still obtain the stronger logics stated by the frame conditions $K_B$, $K_D$, $S4$, $S5$ as usual. On the other hand, nominals allow the formulation of frame conditions, that were not previously expressible in ordinary modal logic. For example, the condition $\square n \Rightarrow \neg n$ for $n \in NOM$, for instance, corresponds to an irreflexivity condition. As in the last section we allow to name the new frame conditions explicitly.

**Representation.** We extend the $\text{hmf}$ syntax for ordinary modal logic described before and add the two new syntax features described above. To distinguish nominals from ordinary Boolean constants, we introduce a new type, called $\text{nominalType}$. In a formula the nominal is grounded to Boolean type, by introducing a new predicate $\#nom$. A nominal $n$ has then to be written as $\#nom(n)$ inside a formula. We propose to adopt the satisfaction and shifter operator with an explicit binding mechanism: $\#at(n) : p$ for $\forall n p$, and $\#shift[n] : p$ for $\downarrow (\lambda X.p)$. Thereby, in both cases, the operator takes as first argument only nominals. The remaining syntax containing modal operators and higher-order features are the above described $\text{hmf}$ based on the standard THF. A short example (where $n_1$ is a nominal) is given by:

```
hhf(1, type, p : $i > $o).
hhf(2, type, n_1 : $nominalType).
hhf(3, conjecture, [\#shift[n] : (\#at(n_1) : \neg \exists X : $i \Rightarrow \#at(n_1) : p(X))].)
```

**Global parameters.** Hybrid logic is an extension to ordinary modal logic, with the same parameters. The same options for quantification, constants, and consequence can be given, as well as giving each symbol a rigid or dependent. In theory a range of additional frame conditions can be given for the modalities, but in practice hybrid logic is used in common frame settings. Hence we suggest to use the exact same (resp. suitably adapted) notation as for ordinary higher-order modal logic.

### 3.3 Conditional Logics

Conditional logics [24] have many applications including action planning, counter-factual reasoning, default reasoning, deontic reasoning, metaphysical modeling and reasoning about knowledge. A new operator for so-called conditionality, denoted $\rightarrow$, is added to the basic logical language which is not to be confused with material implication ($\Rightarrow$).

First-order conditional logics have been studied in [14, 16] and extended (to include propositional quantification) and embedded in HOL in [3]. We here consider a higher-order quantified version of conditional logic (HCL).

**Syntax and semantics.** Terms of HCL are defined as ordinary HOL terms, except that we add a new constant symbol for conditionality to the signature, i.e. $\Sigma = \{\ldots \rightarrow_o a_o a_o\}$. As for modal logics above, we may add quantifiers for different domain conditions. Monomodal logics are subsumed by HCL since $\square a_o$ can be introduced as an abbreviation for $\neg a_o \rightarrow a_o$. Syntactically, HCL can be seen as a generalization of HOML where the index of modality $\rightarrow$ is a formula of the same language. For instance, in $(s_0 \rightarrow t_0) \rightarrow t_0$ the subformula $s \rightarrow t$ is the index of the second occurrence of $\rightarrow$.

---

5 The literature on conditional logics often uses $\Rightarrow$ for conditionality and $\rightarrow$ for material implication. Our choice here is pragmatically motivated, since the TPTP already reserves $\Rightarrow$ for material implication.

6 The extension of quantified conditional logic to full higher-order conditional logic as presented here is ad hoc and straightforward. Whether there are any particular complications arising from that extension still needs to be inspected.
An adequate semantics is achieved by adapting selection function semantics [31, 12]. We modify the HOL model structure by adding possible worlds (similar to HOML) and a selection function $f : W \times 2^W \to 2^W$, yielding a HCL model structure

$$\mathcal{M} = (W, f, \{D_t\}_{t \in \tau}, \{I_w\}_{w \in W})$$

where $W$ is a set of worlds, $\{D_t\}_{t \in \tau}$ is a frame and $\{I_w\}_{w \in W}$ a collection of interpretation functions.

Together with a variable assignment $g$ and a world $w \in W$ we can then refine the valuation function $\|\|_{\mathcal{M}, g, w}$ (only the valuation of conditionality is shown, the remaining cases are straight-forward adaptions of the HOL case):

$$\|s_o \to t_o\|_{\mathcal{M}, g, w} = T \text{ iff } \|t\|_{\mathcal{M}, g, t} \text{ for all } t \in W \text{ s.t. } t \in f(w, [s])$$

where $[s_o] := \{ u \mid \|u\|_{\mathcal{M}, g, u} = T \}$ is the so-called proof set of $s$.

Like in the case of modal logics, where we distinguish between logics such as K, B, D, S4, S5, there are many different conditional logics, which differ regarding the particular axioms/conditions associated with the conditionality operator $\to$. These logics are based on the axioms ID, MP, CS, CEM, AC, RT, CV, CA; see Fig. 2.

**Representation.** Representation of HCL is straight-forward: The syntax is exactly the same as for ordinary HOL problems. We merely add a new implication $\to$ which denotes the conditional implication whereas the included implication of THF, $\Rightarrow$, still denotes material implication (for the sake of consistency). Also, we add the different new quantification symbols $\forall^+$, $\forall^-$, $\exists^+$ and $\exists^-$ for the respective quantification semantics and denote by $t$ the default quantification semantics as chosen by the user. The following example presents a formula that is valid in MP but not in ID:

```
1. hcf(1, type, (f : $i > $o)).
2. hcf(2, type, (g : $i > $o)).
3. hcf(3, conjecture, (! [X: $i]: ((f @ X) \to (g @ X)) \to ((f @ X) \to (g @ X)))).
```
Parameters. The global parameters quantification and constants are the same as for in HOML. We also allow per-symbol rigidity specification as done for HOML. We do not need the specification of modalities here, instead we only need to specify the logic (axiomatization) under consideration. In contrast to modal logic where there exist mostly standardized naming conventions for important logical systems, this is not as evident for conditional logic. However, the axiom names themselves seem standard enough, hence we can describe the logical system by enumerating the included axioms.

This is done by the parameter logic Sets the semantics for the conditional logic under consideration. More precisely, collects a list of axioms that is to be considered. If omitted, a base conditional logic with none additional axioms is assumed.

Valid values: All of the axiom names of Fig. 2.

In the following example the logical axioms ID, MP and CEM (cf. Fig 2) are assumed for the remainder of the conditional logic problem under consideration.

\[
\text{tpi}(1, \text{set}, \text{logic}, \text{hcl}([\{'quantification' = 'cumulative', 'constants' = 'rigid', 'logic' = ['id', 'mp', 'cem']\}])).
\]

Analogous to HOML, the above representations could be extended to support multi-conditional logics, i.e. indexed operators \(\rightarrow\). We do not pursue this further here, since were are not aware yet of applications.

3.4 Free Logics

Classical logic is only mildly suited for handling undefinedness and partiality in an appropriate way. There are two related reasons: (i) terms denote, without exemptions (e.g. for undefined terms), entities in a non-empty domain of “existing” objects \(D\), and (ii) the quantifiers range over this entire set \(D\).

An elegant alternative to remedy these shortcomings is free logic [23, 28], which distinguishes between a raw domain of possibly non-existing objects \(D\) and a particular sub-domain \(E\) of \(D\), containing only the “existing” entities. Free variables range over \(D\) and quantified variables only over \(E\). Each term denotes in \(D\), but not necessarily in \(E\). This is the case, for example, for improper definite descriptions which can now be mapped to a distinguished non-existing object, denoted \(* \in D\).

Moreover, the domain \(E\) may be empty (this special case is called inclusive logic). Unfortunately, no theorem provers have been available so far for free logic. Nevertheless, free logic can be embedded in HOL [8], allowing indirect automation via HOL ATP systems.

Syntax and semantics. The syntax of free higher-order logic is the same as for ordinary HOL. The non-trivial semantics definitions concern universal quantification and definite description (denoted by \(\iota\)). A model \(M\) in this context distinguishes (for all types \(\tau\)) between a raw domain \(D_\tau\) and a set \(E_\tau \subseteq D_\tau\) of existing objects. A valuation function for these cases can be formulated as

\[
\|\forall X_\tau.s_\eta\|^{M,g} = T \text{ if for all } d \in E \text{ holds } \|s_\eta\|^{M,g}[d/X_\tau] = T
\]

\[
\|\iota X_\tau.s_\eta\|^{M,g} = \begin{cases} d & \text{ if } \{d \in E \mid \|s_\eta\|^{M,g}[d/X_\tau] = T\} \text{ is unitary} \\ * & \text{ otherwise} \end{cases}
\]

Representation. Since the syntax is exactly the same as for HOL, we do not need any special representation for free logic formulas. Nevertheless, we need to specify if the domain \(E\) is empty or not.

This is simply stated by the parameter
3.5 Many-valued Logics

Classical logics are based on the bivalence principle, that is, the set of truth-values $V$ has the cardinality $|V| = 2$, usually denoted $V = \{T,F\}$ for truth and falsity. Many-valued logics generalize this requirement to more or less arbitrary sets of truth-values, rather referred to as *truth-degrees* in that context. Popular examples of many-valued logics are Gödel logics, Łukasiewicz and fuzzy logics with (non-)denumerable sets of truth-degrees, and, from the field of finitely-many valued logics, Kleene, Bocvar and Dunn/Belnap logic [2]. The latter logics (Kleene, Bocvar) introduce a third value often denoted $u$ for *unknown* and differ in the interpretation in the presence of the unknown value. The Dunn/Belnap logic introduces two additional values $n,b$ – denoting *none* and *both*, respectively.

Many valued logics have applications in linguistics and philosophy (especially non-western philosophy) for arguing about vagueness, and in computer science for analyzing database and information systems.

**Syntax and semantics.** There is no single unique way for defining the semantics of quantification in many-valued logics. Hence, we are focusing, for the time being, on propositional many-valued logic. The grammar we are considering is formed by

$$a, b ::= t_i | c | \neg a | a C b.$$ 

Where the $t_i$ are the truth-values of the logic, $c \in \Sigma$ is a constant symbol and $C$ is a symbol of the set of binary connectives, containing at least $\{\&, \lor, \land, \to\}$. Depending on the selected logic, additional symbols can be added, and the number of $t_i$ is fixed.

The semantics here is highly dependent on the chosen logic. As usual the semantics of the connectives can be given in a truth-table. In Fig. 3 the semantics of $\to$ for three different three-valued logics is given as an example.

**Representation.** Since we are only considering propositional many-valued logic at this point, it does not fit well into the quantified fragment of the TPTP. But since there exist quantified versions of many-valued logics, we base the propositional case on the FOF fragment of the TPTP. This way, the proposal can be extended towards a first-order version.

In addition to FOF, we first introduce a term for truth degrees $\# t(i)$ for $t_i$ with $i = 1, \ldots, n$, where $n$ is the number of truth constants for the selected logic. To distinguish between weak and strong conjunction, we introduce a new symbol $\\\wedge$ for the standard *weak*...
conjunction ∧. The standard AND-symbol \& is defined to be the strong conjunction to avoid confusion.

The following is an example for axioms of a many-valued logic, relating ∧ and ∨ to the minimal signature of Lukasiewicz or Gödel logics.

| mvf(1, axiom, (a ∧ b) = ((a ⊃ (a → b))) | 
| mvf(2, axiom, (a ∨ b) = ((a → b) → b) ∧ ((b → a) → a)) |

Global Parameters. There are two parameters we can provide to adjust the semantics. The first fixes the exact logic. As described, there are many possible many-valued logics, that fix an interpretation for the logical symbols. The second parameter sets the cardinality of the set of truth values. This parameter is only important for the logics with an adjustable amount of truth values. The exact parameters are:

- **semantics** Sets the exact logic and fixes the interpretation for the logical symbols.
  - Valid values: kleene-weak, kleene-strong, post, lukasiewicz, goedel, bocvar.
- **card** Sets the size of the truth values. Has only an effect on lukasiewicz and goedel.
  - Valid values: any natural number

The parameters can be set as in the previous cases in the instruction language TPI. For example, a Lukasiewicz logic with \{0, \frac{1}{4}, \frac{3}{4}, 1\} as truth values can be declared with:

```
 tpi(1, set_logic, mvl('semantics' = 'lukasiewicz', 'card' = 5)).
```

In theory the cardinality cases \(\aleph_0\) and \(\aleph_1\) were possible, but this would require a different mechanism to name truth values.

### 3.5.1 MVL based on SIXTEEN

There are several sixteen-valued logics based on a lattice denoted SIXTEEN. These logical systems have been developed by Shramko and Wansing as a generalization of the four-valued system of Dunn/Belnap [2] to knowledge bases in computer networks [29] and was subsequently further investigated in various contexts (e.g. [25, 30]). In SIXTEEN, the truth-degrees are given by the power set of Belnap’s truth values, i.e.

\[ V = 2^{\{N, T, F, B\}} = \{\emptyset, N, T, F, B, \ldots, \{N, T, F, B\}\} \]

where \(N, T, F\) and \(B\) are the respective singleton sets containing \(N, T, F\) and \(B\). The remaining truth-degrees are named using a combination of the letters \(N, T, F\) and \(B\), representing the truth-degree that contains the respective elements when regarded as a set (e.g. \(NT\) for the set \(\{N, T\}\)). This generalization is essentially motivated by the observation that a four-valued system cannot express certain phenomena that arise in knowledge bases in computer networks. Further applications in linguistics and philosophy are discussed in the monograph by Shramko and Wansing [30], to which we refer to for a thorough investigation of SIXTEEN, the definitions of logical connectives and their semantics. Briefly speaking, there exists a set of connectives \(\lor, \land, \neg\) for two distinct logics \(L_\star\), with \(\star \in \{t, f\}\) and a logic given by their union, denoted \(L_{tf}\). Additionally, multiple different entailment relations \(\models_{\star}\) for \(\star \in \{t, f, tf\}\) can be considered. An embedding of logics based on SIXTEEN into HOL for use in ATP systems is sketched in [32].

Representation. The representation of the \(L_\star\) is more involved as we need a non-quantified language with different infix operators denoting the different logical connectives. The following syntax representation could be seen as a generalization and restriction of
FOF, where we add new infix connectives but restrict the problem not to contain first-order ingredients (such as quantifiers). Consequently, we add infix operators \( \&t \), \( |t| \), \( \neg t \), \( \Rightarrow t \), \( \&f \), \( |f| \), \( \neg f \) and \( \Rightarrow f \) to the language. The problem statements are then straightforward, an example is given by

\[
\begin{align*}
\text{sf}(1, \text{axiom}, (a \mid t b)). \\
\text{sf}(2, \text{axiom}, (a \mid f b)). \\
\text{sf}(3, \text{conjecture}, (a \Rightarrow t b)).
\end{align*}
\]

where \( a, b \) are ad-hoc introduced individuals symbols as supported by FOF.

## 4 Conclusion

In this paper, we discussed means of representing both problems and meta-logical specification for quantified non-classical logics. To that end we adapt and extend TPTP-THF and QMLTP syntax for problems and the TPI language proposal for fixing semantic parameters. We have outlined specialized syntaxes for higher-order modal logic, hybrid logic, conditional logic, and free logic to be used as input languages of ATP systems. Additionally, we sketched ideas for representing many-valued logics. Further logics can easily be added and addressed along the same lines. For example, due to space restrictions we have omitted the inclusion of (quantified) intuitionistic logic [6].

The suggestions in this paper are, at this stage, not meant to be conclusive. Instead, we want to stimulate discussions, e.g. at the ARQNL event, about further requirements and extensions. Moreover, we envision a close collaboration with the QMLTP and TPTP projects.

A concluding, motivating example is displayed in Appendix A where an encoding of Gödel’s Ontological argument is given in \texttt{hmf} syntax.

**Acknowledgments:** We thank Harold Boley for his comments and for proofreading this document. We also thank the reviewers for the very valuable feedback they provided.

## References


[29] Y. Shramko and H. Wansing. Some useful 16-valued logics: How a computer network...


A Example: Gödel’s ontological argument

The following example is an encoding of Gödel’s ontological argument in hmf syntax as described in this work (cf. §3.1). More precisely, we present here Dana Scott’s variant as a one step proof in which intermediate argumentation steps are omitted. HOL provers such as Leo-II are capable of automating this example in a few seconds (however, not yet for the syntax representation below). Details about the ontological argument, its formalization and its automation can be found in the literature (e.g. in [5]).

```plaintext
tpi(1, set_logic, modal('quantification' = 'varying',
                           'constants' = 'rigid',
                           'consequence' = 'global',
                           'modalities' = [a, s5])).

hmf(positive_const, type, (p: (i>s)>o)).

hmf(A1, axiom, (![Phi: i>s]: ((p @ (\[X:i]: ~(Phi @ X)) <=> ~(P @ Phi)))).

hmf(A2, axiom, (![Phi: i>s, Psi: i>s]: ((((p @ Phi)
             & #box(a): ([X:i]: ((Phi @ X) => (Psi @ X)))))
              => (p @ Psi)))).

hmf(god_const, type, (g: i>s)).

hmf(god, definition, (g = ([X:i]: (![Phi: i>s]: ((p @ Phi) => (Phi @ X)))))).

hmf(essence_const, type, (ess: (i>s)>i>).

hmf(essence, definition, (ess = ([Phi: i>s,X: i]: ((Phi @ X)
             & (![Psi: i>s]: ((Psi @ X) => #box(a): (Psi @ X)))))
              => #box(a): ([Y:i]: (Phi @ Y))))))}

hmf(necessary_existence_const, type, (ne: i>s)).

hmf(necessary_existence, definition, (ne = ([X:i]: (![Phi: i>s]: ((ess @ Phi @ X)
             => #box(a): (?[Y:i]: (Phi @ Y)))))
              => #box(a): (?[Y:i]: (Phi @ Y)))).

hmf(A5, axiom, (p @ ne)).

hmf(T3, conjecture, (#box(a): (?[X:i]: (g @ X)))).
```

65
Optimizing Inconsistency-tolerant Description Logic Reasoning

Mokarrom Hossain and Wendy MacCaull

Department of Mathematics, Statistics and Computer Science
St. Francis Xavier University, Antigonish, NS, Canada
{mokarrom.hossain, wmaccaull}@stfx.ca

Abstract

The study of inconsistency-tolerant description logic reasoning is of growing importance for the Semantic Web since knowledge within it may not be logically consistent. The recently developed quasi-classical description logic has proved successful in handling inconsistency in description logic. To achieve a high level of performance when using tableau-based algorithms requires the incorporation of a wide range of optimizations. In our previous work, we developed a naive inconsistency-tolerant reasoner that can handle inconsistency directly. Here, we investigate a set of well known, state-of-the-art optimization techniques for inconsistency-tolerant reasoning. Our experimental results show significant performance improvements over our naive reasoner for several well-known ontologies. To the best of our knowledge, this is the first attempt to apply optimizations for inconsistency-tolerant tableau-based reasoning.

1 Introduction

Since the environment of the Semantic Web (SW) is open, constantly changing and collaborative, it is unreasonable to expect knowledge bases (KBs) to be always logically consistent [1]. With the explosive growth of the SW the probability of introducing inconsistencies in ontologies is increasing day by day due to ontology merging, ontology evolution, ontologies created using machine learning or data mining, migration from other formalisms, modeling errors, knowledge from distributed sources, etc. [2]. The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies in SW applications. The family of description logics (DLs), decidable fragments of First Order Logic, is the logical foundation of OWL. Classical DLs are unable to tolerate inconsistencies occurring in ontologies due to the principle of explosion [3]. Therefore, it is important to study the ways of dealing with inconsistencies in DL based ontologies [4, 5, 6].

There are several approaches to deal with inconsistencies. These approaches may be roughly categorized into two different classes. One class assumes that inconsistencies indicate errors in data. The main view of this class is that an ontology should not be inconsistent, and thus researchers try to eliminate inconsistency from an ontology in order to obtain a consistent ontology [7]. In this approach the idea, therefore, is, first detect and then, repair, inconsistencies. This class of approaches could be called “removing inconsistencies”. In this approach, inconsistency could be removed by pinpointing the part of the ontology which causes inconsistencies and removing or weakening axioms in these parts to restore consistency. The main assumption of the other class of approaches is that inconsistency is a natural phenomenon of realistic data and should be tolerated in reasoning [3, 8, 9]. In this class of approaches, inconsistency is not simply avoided, rather, the idea is to employ non standard reasoning methods in order to obtain meaningful information from an inconsistent KB. This class of approaches could be called “living with inconsistencies”. Though inconsistency in small ontologies may be dealt with using
the former approach, the latter one is better for large and complex ontologies [10]. Moreover,
in the first class of approaches, we may lose useful information during the process of removing
inconsistencies [3]. In this work we focus on the second class of approaches.

In the last decades, a number of researchers have extended DL in different ways in order to
cope with inconsistency [3, 8, 10]. Nguyen et al. studied three-valued DL based on Kleene’s
paradoxical semantics. Kaminski et al. presented a paraconsistent version of the three-valued
semantics for hybrid knowledge bases which allows full paraconsistent reasoning [12]. Indeed,
three-valued DLs are usually appropriate for handling the inconsistency but not the incom-
pleteness of a KB. In many KBs, information is not only inconsistent but is also incomplete.
Four-valued DL, based on four-valued logic [13], is studied by Ma et al. in [3] and has received a
lot of attention. Though it can handle both inconsistent and incomplete KBs, it is not widely ac-
cepted due to its weak inference power [10]. For example, four-valued DL does not fully support
a few important properties about inference such as modus tollens (MT), disjunctive syllogism
(DS), resolution, etc. Recently, Zhang et al., proposed a paraconsistent version of DL, called
QCDL, in [10], based on quasi-classical propositional logic (QC-logic) originally presented in
[14]. Weak inference power, one of the common problems of the family of paraconsistent logics,
is overcome by QCDL. Most inference rules like MT, DS are valid in QCDL.

In the previous work [15], we presented a sound, complete, and decidable tableau algorithm
for QCDL and implemented a tableau based paraconsistent reasoner called QC-OWL based
on this algorithm. However, we did not address any optimization techniques in QC-OWL.
Most modern reasoners implement a set of state-of-the-art optimization techniques and these
techniques are the keys to the enhanced performance of a modern tableau-based reasoner.
Here, we extend a set of state-of-the-art optimization techniques for classical tableau based
reasoner, namely normalization and simplification, unfolding, absorption, semantic branching,
and dependency directed backtracking, to paraconsistent reasoning (background and motivation
of these well-known techniques may be found in [16]). We implement those techniques on top of
QC-OWL and compare the performance with that of our previous work, where no optimizations
were addressed.

The rest of this paper is organized as follows: Section 2 briefly introduces the syntax and
semantics of QCDL, Section 3 describes the optimization techniques, Section 4 presents the
implementation and evaluation of those optimizations, Section 5 outlines some related works
and future research directions, and finally, Section 6 concludes the paper.

2 Preliminaries

In this section we briefly introduce the syntax and semantics of QCDL. We focus on an expressive
DL fragment, $SHIQ$, and call it quasi-classical $SHIQ$ (in short, QC-$SHIQ$). We assume that
the readers are familiar with the basic DL formalism; for more comprehensive background
knowledge of DLs and QCDL, the reader is referred to [10, 15, 16].

2.1 QCDL syntax and semantics

The syntax and semantics of QCDL is presented in [10] by extending the semantics of quasi-
classical logic (QC-logic) proposed in [14]. The syntax of QC-$SHIQ$ is slightly different from
the syntax of classical $SHIQ$. In QC-$SHIQ$, the negation of a concept, i.e., $\neg C$, is taken
as a different concept from $C$, rather than the complement of $C$. QC-negation, denoted by
is used to represent the complement (set-theoretic complement) concept of C [10]. For a comprehensive background and motivation of QC-negation, the reader is referred to [10, 17].

Let \( N_C, N_R \) and \( N_I \) be non-empty and pair-wise disjoint sets of concept names, role names, and individual names, respectively. Let \( R \) be a set of role names with a subset \( R_+ \subseteq R \) of transitive role names. The set of roles is \( R \cup \{ R^- | R \in R \} \). The function \( Inv() \) is defined on roles such that \( Inv(R) = R^- \) and \( Inv(R^-) = R \), where \( R \) is a role name.

Let \( R_1, R_2 \in R \). A role axiom is a role inclusion of the form \( R_1 \sqsubseteq R_2 \). An RBox or role hierarchy \( R \) is a finite set of role axioms. For a role hierarchy \( R \), the relation \( \sqsubseteq \) is defined to be the transitive-reflexive closure of \( \sqsubseteq \) on \( R \cup \{ Inv(R) \sqsubseteq Inv(S) | R \sqsubseteq S \in R \} \). A role \( R \) is called sub-role (respectively, super-role) of a role \( S \) if \( R \sqsubseteq S \) (respectively, \( S \sqsubseteq R \)). A role \( S \) is simple if it is neither transitive nor has any transitive sub-roles.

The set of complex concepts is the smallest set such that

- each concept name \( A \in N_C \) is a concept;
- if \( C, D \) are concepts, \( R \) is a role, \( S \) is a simple role, and \( n \) is a nonnegative integer, then \( C \sqcap D | C \sqcup D | \neg C | C^{\neg} \ \forall R.C \ \exists R.C \ | \geq nS.C \ | \leq nS.C \) are also concepts.

A general concept inclusion (GCI) is an expression in the form \( C \sqsubseteq D \), where \( C, D \) are concepts. A TBox is a finite set of GCIs. An assertion is of the form \( \langle \Delta I, I \rangle = \langle \{ x \in \Delta I | \# y \in \Delta I: (x, y) \in +S \text{ and } y \in \neg C \} \rangle ; \langle \{ x \in \Delta I | \# y \in \Delta I: (x, y) \in +S \text{ and } y \in +C \rangle > n \rangle \).

Note that each base interpretation maps an object \( X \), when \( X \in \{ A, R, R^\perp \} \), to a pair of sets of elements, unlike classical interpretation where an object is mapped to a set of elements. \(+X \) and \( -X \) are not necessarily disjoint. Intuitively, \(+X \) is the set of elements known to be in \( X \) while \(-X \) is the set of elements known to not be in \( X \).

A weak interpretation \( I \) is a base interpretation \((\Delta^I, I) \) such that the assignment function \( I \) satisfies the following conditions [15], where \#M denotes the cardinality of a set \( M \):

- \( T^I = \{ \Delta^I, \emptyset \} \); \( \perp^I = \{ \emptyset, \Delta^I \} \);
- \((-C)^I = \{ -C, +C \} \); \((\neg C)^I = \{ \Delta^I \ \cup \ C, \Delta^I \ \setminus \ -C \} \);
- \((\forall R.C)^I = \{ \{ x \in \Delta^I \ | \ \forall y \in \Delta^I: (x, y) \in +R \text{ implies } y \in +C \} \), \( \{ x \in \Delta^I \ | \ \exists y \in \Delta^I: (x, y) \in +R \text{ and } y \in -C \} \} \);
- \((\exists R.C)^I = \{ \{ x \in \Delta^I \ | \ \exists y \in \Delta^I: (x, y) \in +R \text{ and } y \in +C \} \), \( \{ x \in \Delta^I \ | \ \forall y \in \Delta^I: (x, y) \in +R \text{ implies } y \in -C \} \} \);
- \((\geq nS.C)^I = \{ \{ x \in \Delta^I \ | \ \# y \in \Delta^I: (x, y) \in +S \text{ and } y \in +C \rangle \geq n \} \), \( \{ x \in \Delta^I \ | \ \# y \in \Delta^I: (x, y) \in +S \text{ and } y \in \neg C \} < n \} \);
- \((\leq nS.C)^I = \{ \{ x \in \Delta^I \ | \ \# y \in \Delta^I: (x, y) \in +S \text{ and } y \in +C \rangle \geq n \} \), \( \{ x \in \Delta^I \ | \ \# y \in \Delta^I: (x, y) \in +S \text{ and } y \in +C \} < n \} \).

68
Let \( I \) be a weak interpretation. A weak satisfaction relation, denoted by \( \models_{w} \), is defined as follows: 
\( I \models_{w} C(a) \) if \( a_{I} \in +C \); 
\( I \models_{w} R(a, b) \) if \( (a^{T}, b^{T}) \in +R \); 
\( I \models_{w} C \sqsubseteq D \) if \( +C \sqsubseteq +D \); 
\( I \models_{w} R_{1} \sqsubseteq R_{2} \) if \( +R_{1} \sqsubseteq +R_{2} \); 
and \( I \models_{w} a \neq b \) if \( a^{T} \neq b^{T} \); where \( X^{T} = \langle +X, -X \rangle \) for any \( X \in \{ C, D, R, R_{1}, R_{2} \} \) [15].

A strong interpretation is as similar to a weak interpretation except that the conjunction and disjunction of concepts are interpreted as follows [15]:
\[
(C \cap D)^{T} = (+C \cap +D, (-C \cup -D) \cap (-C \cup +D) \cap (\neg C \cup -D));
\]
\[
(C \cup D)^{T} = ((+C \cup +D) \cap (\neg C \cup +D) \cap (+C \cup -D), -C \cap -D).
\]

The definition of strong satisfaction relation, denoted by \( \models_{s} \), is the same as that of the weak satisfaction relation except for GCIs [15]. For GCIs, \( \models_{s} \) is defined as \( I \models_{s} C \subseteq D \) if \( -C \subseteq +D \), \( +C \subseteq +D \), \( -D \subseteq -C \).

Let \( K \) be a KB and \( \phi \) be an axiom. \( K \) quasi-classically entails (QC entails) \( \phi \), denoted by \( K \models_{Q} \phi \), if for every base interpretation \( I \), \( I \models K \) implies \( I \models_{w} \phi \). In this case, \( \models_{Q} \) is called QC-entailment. A base interpretation \( I \) is a QC-model of \( K \) if for all axioms \( \phi \) in \( K \), \( I \models_{s} \phi \). \( K \) is QC-consistent if there exists some QC-model \( I \) of \( K \), else it is QC-inconsistent [15].

### 2.2 A tableau algorithm for QC-SHIQ Abox

State of the art DL systems typically use tableau algorithms [18] to decide the consistency of a KB, i.e., to determine whether a given KB has a model. Consistency checking is one of the main inference problems to which all other inferences can be reduced [16]. A sound, complete and decidable tableau algorithm (called QC-tableau) for checking the QC-consistency of an ABox is proposed in [15] by modifying and extending the standard tableau algorithm for SHIQ [19].

We outline the main steps as follows.

First we note that, in order to work efficiently, the TBox is reduced to an empty TBox with the internalization technique and the ABox is transformed into Negation Normal Form (NNF).

**Internalization:** Let \( U \) be a universal role, that is, a transitive super role consisting of all roles occurring in \( T \) together with their respective inverses. The base interpretation of \( U \) is defined as follows: \( U^{T} = \langle \Delta^{T} \times \Delta^{T}, \emptyset \rangle \) for any base interpretation \( I \). Given \( T \), a concept \( C_{T} \) is defined as
\[
C_{T} := \bigcap_{C_{i} \subseteq D_{i} \in T} (-C_{i} \cup D_{i}).
\]
Any individual \( x \) in any model of \( T \) will be an instance of \( C_{T} \). Let \( R_{U} = R \cup \{ R \subseteq U, \text{Inv}(R) \subseteq U \mid R \text{ occurs in } C, D, T, A \text{ or } R \} \). Now, \( A \) is QC-consistent w.r.t. \( R \) and \( T \) iff \( A \cup \{ C_{T} \cap \forall U.CT(a) \mid a \text{ occurs in } A \} \) is QC-consistent w.r.t. \( R_{U} \).

**Negation Normal Form (NNF):** A concept expression is in NNF if all the negations directly precede concept names. Let \( C, D \) be two concepts. We say \( C \) is equivalent to \( D \), denoted by \( C \equiv_{s} D \), if for any strong interpretation \( I \), \( C^{T} = D^{T} \). That is, \( +C = +D \) and \( -C = -D \) where \( C^{T} = \langle +C, -C \rangle \) and \( D^{T} = \langle +D, -D \rangle \). Each QCDL concept is equivalent to a QCML concept in NNF. NNF of a concept expression in QCML can be computed by applying the equivalences from Table 1.

The QC-tableau algorithm works on a data structure called a completion forest. The algorithm starts with the input ABox, \( A \), and applies consistency preserving expansion rules from Table 2 (due to space limitation only five rules are presented here; for the remaining rules the reader is referred to [15, 17]) until no more rules are applicable (the tableau is complete) or an obvious contradiction (called a clash) is found in every branch. If a complete and clash-free completion forest is obtained, \( A \) is QC-consistent; otherwise it is QC-inconsistent. In the completion forest, the label of a node \( x \) is denoted by \( L(x) \).
Example 1 Consider the concept expression \( \exists R. (C \sqcap D) \sqcap \exists R. C \), where \( C \) is an atomic concept. Will be detected only when an atomic concept and its negation occur in the same node label.

Table 1: QC-NNF equivalences.

<table>
<thead>
<tr>
<th>( \exists R.C )</th>
<th>( \forall R.C )</th>
<th>( \exists R.\neg C )</th>
<th>( \forall R.\neg C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \equiv s )</td>
<td>( \equiv s )</td>
<td>( \equiv s )</td>
<td>( \equiv s )</td>
</tr>
<tr>
<td>( \neg(C \sqcup D) )</td>
<td>( \equiv s )</td>
<td>( \neg C \sqcap \neg D )</td>
<td>( \equiv s )</td>
</tr>
<tr>
<td>( \neg \exists R.C )</td>
<td>( \equiv s )</td>
<td>( \forall R.\neg C )</td>
<td>( \equiv s )</td>
</tr>
<tr>
<td>( \neg \forall R.C )</td>
<td>( \equiv s )</td>
<td>( \exists R.\neg C )</td>
<td>( \equiv s )</td>
</tr>
<tr>
<td>( \leq n \mathcal{S}.C )</td>
<td>( \equiv s )</td>
<td>( \geq n + 1 \mathcal{S}.C )</td>
<td>( \equiv s )</td>
</tr>
<tr>
<td>( \leq n \mathcal{S}.C )</td>
<td>( \equiv s )</td>
<td>( \geq (n + 1) \mathcal{S}.\neg C )</td>
<td>( \equiv s )</td>
</tr>
</tbody>
</table>

Table 2: QC-tableau expansion rules.

3 Optimizations

The basic algorithm for inconsistency-tolerant reasoning discussed in the previous section is too slow for use in practice. We have investigated and employed a range of optimizations that improve the performance of standard tableau algorithms. These optimizations include: normalization and simplification, unfolding, absorption, semantic branching search, dependency directed backtracking. The first three techniques are performed directly on the input which serve to pre-process and simplify the input into a form more amenable to later processing, while the remaining techniques are applied during the search for a model. In the following subsections, each of these techniques will be revised for QCDL by modifying and extending the techniques presented in [16] for classical DLs.

3.1 Normalization and Simplification

Normalization is an optimization technique performed in pre-processing. It allows the detection of contradictions involving complex concepts early during tableaux expansion. Theoretical descriptions of tableau algorithms generally assume that the concept expression to be tested is in negation normal form. Though it simplifies the algorithm, this means that a contradiction will be detected only when an atomic concept and its negation occur in the same node label.

Example 1 Consider the concept expression \( \exists R. (C \sqcap D) \sqcap \exists R.\neg C \), where \( C \) is an atomic concept.
concept. When the algorithm creates an $R$-successor using the $\exists$-rule and applies the $\exists$-rule to $\exists R.C$, a clash will be detected due to the fact that $\{C, \overline{C}\} \in \mathcal{L}(y)$. However, if $C$ is a concept expression then $C$ will be transformed into NNF. Hence, the clash would not be detected immediately. If $C$ is a complex concept this may cause a lot of unnecessary expansion. It is possible to detect contradictions caused by non-atomic concepts early by transforming all concepts into a syntactic normal form, which we define here.

Logics that include full negation often provide pairs of operators, either one of which can be eliminated in favor of the other, by using negation. In syntactic normal form, all concepts are transformed so that only one of each such pair appears in the KB. In QC-$\mathcal{SHIQ}$, as in classical $\mathcal{SHIQ}$, all concepts could be transformed into atomic concepts, negations, conjunctions, value restrictions and QC-negations. For example, $\exists R.C$ is transformed into $\neg \forall R.\neg C$. It is important to note that within the normalization process, conjunctions are considered as sets; this simplifies the elimination of redundant conjuncts. For example, $\neg C \sqcup \neg D$ is transformed into $\neg \sqcap \{C, D\}$. The normalization process can also include a range of simplifications that detect obvious clashes during the normalization process and also gets rid of redundant elements of a concept expression.

$$
\begin{align*}
\text{Norm}(A) &= A \text{ for an atomic concept } A \\
\text{Norm}(\neg C) &= \text{Simp}(\neg (\text{Norm}(C))) \\
\text{Norm}(\overline{C}) &= \text{Simp}(\text{Norm}(C)) \\
\text{Norm}(\sqcap \{C_1, \ldots, C_n\}) &= \text{Simp}(\sqcap \{\text{Norm}(C_1) \cup \ldots \cup \text{Norm}(C_n)\}) \\
\text{Norm}(\forall R.C) &= \text{Simp}(\forall R.\text{Norm}(C)) \\
\text{Norm}(\exists R.C) &= \text{Norm}(\neg \forall R.\neg C) \\
\text{Norm}(\preceq n R.C) &= \text{Simp}(\preceq n R.\text{Norm}(C)) \\
\text{Norm}(\succeq n R.C) &= \text{Norm}(\neg \preceq (n + 1) R.C)
\end{align*}
$$

$$
\begin{align*}
\text{Simp}(A) &= A \text{ for an atomic concept } A \\
\text{Simp}(\neg C) &= \begin{cases} \text{Simp}(D) & \text{if } C = \neg D \\ \text{Simp}(\neg D) & \text{if } C = \overline{D} \\ \neg C & \text{otherwise.} \end{cases} \\
\text{Simp}(\overline{C}) &= \begin{cases} \text{Simp}(D) & \text{if } C = \overline{D} \\ \overline{C} & \text{otherwise.} \end{cases} \\
\text{Simp}(\sqcap S) &= \begin{cases} \text{Simp}(\sqcap P \cup S \setminus \{\sqcap \{P\}\}) & \text{if } \sqcap \{P\} \in S \\ \sqcap S & \text{otherwise.} \end{cases} \\
\text{Simp}(\forall R.C) &= \forall R.C \\
\text{Simp}(\preceq n R.C) &= \preceq n R.C \quad [S, P \text{ are sets of concepts}]
\end{align*}
$$

Table 3: $\text{Norm}$ and $\text{Simp}$ functions for QC-$\mathcal{SHIQ}$.

Table 3 describes normalization and simplification functions $\text{Norm}$ and $\text{Simp}$ for QC-$\mathcal{SHIQ}$. Normalized and simplified concepts may not be in negation normal form, but they can be dealt with by treating them exactly like their non-negated counterparts. For example, $\neg \sqcap \{C, D\}$ can be treated as $\neg C \sqcup \neg D$ and $\neg \forall R.\neg C$ can be treated as $\exists R.C$ during the tableau expansion. The normalization and simplification procedure is implemented by a recursive function, applied to the concept expression to check. The normalized and simplified concepts for QC-$\mathcal{SHIQ}$ are presented in Table 4.

**Example 2** Consider the Example 1 again. The expression $\exists R.(C \sqcap D) \sqcap \exists R.C$ is transformed into $\sqcap \{\neg (\forall R.\neg \sqcap \{C, D\}), \neg \forall R.\neg C\}$. The term $\neg (\forall R.\neg \sqcap \{C, D\})$ allows the creation of an $R$-
successor using the $\exists$-rule whose label contains both $C$ and $\overline{C}$ (by applying the $\exists$-rule to the term $\neg \forall R.\neg C$). Since the two occurrences of $C$ are in the syntactic normal form, a clash will be detected immediately, regardless of the structure of $C$.

### 3.2 Unfolding

Unfolding is a recursive substitution procedure that allows the testing of the satisfiability of a given concept $C$ w.r.t. $T$ by eliminating from $C$ all concept names occurring in $T$ [16]. If $A \equiv D$ is an axiom in $T$, where $A$ is a non-primitive (defined in $T$) concept name, the procedure simply substitutes (i.e., unfolds) $A$ with $D$ wherever it occurs in $C$, and then recursively unfolds $D$ in the same manner. If $A \subseteq D$ is an axiom in $T$, where $A$ is a primitive concept name, $A$ is substituted by $A' \equiv D$, where $A'$ is a new concept name that does not occur in $T$ or $C$. The concept name $A'$ represents the primitiveness of $A$, i.e., the unspecified characteristics that differentiate $A$ from $D$. $\text{Unfold}(C, T)$ denotes the concept $C$ after unfolding w.r.t. $T$ [16].

Subsumption testing can be made independent of $T$ using the same technique. The problem of determining if $C$ is subsumed by $D$ w.r.t. a TBox $T$ is the same as the problem of determining if $\text{Unfold}(C, T)$ is subsumed by $\text{Unfold}(D, T)$ w.r.t. an empty TBox; in other words, $T \models C \subseteq D$ iff $\emptyset \models \text{Unfold}(C, T) \subseteq \text{Unfold}(D, T)$ [16].

Generally there are two problems regarding concept unfolding: (1) an unrestricted recursive unfolding could possibly produce a resulting concept expression of exponential size; (2) unfolding would not be possible if $T$ contains (i) multiple definitions for some concept name $A$, e.g., if $\{A \equiv C, A \equiv D\} \subseteq T$, or (ii) cyclical axioms, e.g., if $(A \subseteq \exists R.A) \in T$. The former problem can be addressed by a technique called lazy unfolding which unfolds concepts only when required during the progress of the algorithm. In other words, lazy unfolding does not expand the occurrences of concept names which follow $\exists$ or $\forall$. For example, when testing the satisfiability of an expression $\exists R.E$, where $E$ is a concept name, the unfolding of $E$ can be delayed until the $\exists$-rule has created an $R$-successor $y$ with $L(y) = \{E\}$. By imposing this restriction, lazy unfolding may prevent the exponential increase of a concept expression.

**Example 3** Consider, testing the satisfiability of the concept expression: $\exists R.E \land \forall R.\overline{E}$. The optimized algorithm will detect a contradiction immediately when the $\exists$-rule creates an $R$-successor $y$ and applies the $\forall$-rule because $(E, \overline{E}) \subseteq L(y)$. This may save a lot of unnecessary work if unfolding $E$ produces a large and complex expression.

As we have just noticed, all axioms in an arbitrary TBox are not amenable to unfolding. The solution to this problem is to divide the TBox $T$ into two components, a general part $T_g$ and an unfoldable part $T_u$ such that $T = T_u \cup T_g$; where $T_u$ contains unique, acyclical, definition axioms and $T_g$ contains the rest of $T$. This can be achieved easily by initializing $T_u$ to $\emptyset$, then

<table>
<thead>
<tr>
<th>Expression</th>
<th>Normalized &amp; Simplified</th>
<th>Expression</th>
<th>Normalized &amp; Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg \forall C$</td>
<td>$\neg \overline{C}$</td>
<td>$\exists C$</td>
<td>$\exists {C_1, C_2}$</td>
</tr>
<tr>
<td>$C_1 \sqcap C_2$</td>
<td>$\forall C_1 \sqcap C_2$</td>
<td>$\forall R.C$</td>
<td>$\forall R.\neg C$</td>
</tr>
<tr>
<td>$C_1 \sqcup C_2$</td>
<td>$\forall C_1 \sqcup C_2$</td>
<td>$\forall R.C$</td>
<td>$\forall R.\neg C$</td>
</tr>
<tr>
<td>$\geq n , R.C$</td>
<td>$\geq n , R.C$</td>
<td>$\leq n , R.C$</td>
<td>$\leq n , R.C$</td>
</tr>
<tr>
<td>$\geq n , R.C$</td>
<td>$\geq n , R.C$</td>
<td>$\leq n , R.C$</td>
<td>$\leq n , R.C$</td>
</tr>
</tbody>
</table>

Table 4: Normalized and simplified concepts in QC-$S\forall I Q$
for each axiom $X$ in $\mathcal{T}$, adding $X$ to $\mathcal{T}_u$ if $\mathcal{T}_u \cup X$ is still unfoldable, adding $X$ to $\mathcal{T}_g$ otherwise. In this way, reasoning tasks w.r.t. $\mathcal{T}$ can be considered as reasoning tasks w.r.t. $\mathcal{T}_u$ and $\mathcal{T}_g$: use lazy unfolding to deal with $\mathcal{T}_u$ and internalization to deal with $\mathcal{T}_g$ [16].

3.3 Absorption

As we have seen in the previous section, an arbitrary TBox $\mathcal{T}$ is divided into two parts, $\mathcal{T}_u$ and $\mathcal{T}_g$; unfolding is applied to $\mathcal{T}_u$ and internalization is applied to $\mathcal{T}_g$. The reasoning performance for $\mathcal{T}_u$ can be very good, while the reasoning performance for $\mathcal{T}_g$ might be bad, because internalization may introduce many disjunctions which increases the search space exponentially. For example, if a $\mathcal{T}_g$ contains 10 GCIs with 10 nodes, there are already 100 disjunctions, and they can be non-deterministically expanded in $2^{100}$ different ways. Therefore, it is a good strategy to eliminate as many GCIs from $\mathcal{T}_g$ as possible.

Absorption is a technique that tries to eliminate GCIs by absorbing them into primitive definitions. By considering $\mathcal{T}_u$ and $\mathcal{T}_g$, if one can move axioms from $\mathcal{T}_g$ to $\mathcal{T}_u$ while keeping the semantics of $\mathcal{T}$ unchanged, one should be able to improve the reasoning performance. The absorption technique presented here is analogous to that of classical DL described in [16] except for handling the QC-negations. The basic idea is that a GCI of the form $C \sqsubseteq D$, where $C$ may be a non-atomic concept, is transformed into the form of a primitive definition $A \sqsubseteq C'$, where $A$ is an atomic concept, using the axiom equivalences (1) and (2) below. Then, $A \sqsubseteq D'$ together with an existing primitive definition $A \sqsubseteq C'$ may be replaced by $A \sqsubseteq C' \cap D'$.

$$C_1 \sqcap C_2 \sqsubseteq D \iff C_1 \sqsubseteq D \sqcup \neg C_2 \quad (1)$$

$$C \sqsubseteq D_1 \sqcap D_2 \iff C \sqsubseteq D_1 \text{ and } C \sqsubseteq D_2 \quad (2)$$

Given $\mathcal{T}_u$ and $\mathcal{T}_g$, absorbing the axioms from $\mathcal{T}_g$ into the primitive definitions in $\mathcal{T}_u$ can be done according to the following procedure. First, each axiom of the form $C \equiv D$ is replaced by an equivalent pair of axioms $C \sqsubseteq D$ and $\neg C \sqsubseteq \neg D$, and $\mathcal{T}_g'$ is set to $\emptyset$. Then for each axiom $(C \sqsubseteq D) \in \mathcal{T}_g'$ [16]:

(A) Initialize a set $G = \{\neg D, C\}$, which represents the axiom in the form $T \sqsubseteq \neg \sqcap \{\neg D, C\}$ (i.e., $T \sqsubseteq D \sqcup \neg C$).

(B) If there is a primitive definition axiom $(A \sqsubseteq C) \in \mathcal{T}_u$ for some $A \in G$, then absorb the general axiom into the primitive definition axiom so that it becomes $A \sqsubseteq \sqcap \{C, \neg \sqcap \{G \setminus \{A\}\}\}$, and exit.

(C) If there is a primitive definition axiom $(A \equiv D) \in \mathcal{T}_u$ for some $A \in G$, then substitute $A$ with $D$, $G \rightarrow \{D\} \sqcup G \setminus \{A\}$, and return to step (B).

(D) If there is a primitive definition axiom $(A \equiv D) \in \mathcal{T}_u$ for some $\neg A \in G$, then substitute $\neg A$ with $\neg D$, $G \rightarrow \{\neg D\} \sqcup G \setminus \{\neg A\}$, and return to step (B).

(E) If there is a primitive definition axiom $(A \equiv D) \in \mathcal{T}_u$ for some $A \in G$, then substitute $\overline{A}$ with $\overline{D}$, $G \rightarrow \{\overline{D}\} \sqcup G \setminus \{\overline{A}\}$, and return to step (B).

(F) If there is some $C \in G$ such that $C$ is of the form $\sqcap S$, then use associativity to simplify $G$, $G \rightarrow S \sqcup G \setminus \{\sqcap S\}$, and return to step (B).

(G) If there is some $C \in G$ such that $C$ is of the form $\neg \sqcap S$, then for every $D \in S$ determine if $C$ can be absorbed (recursively) in $G$, $\{\neg D\} \sqcup G \setminus \{\neg \sqcap S\}$, and exit.

(H) Otherwise, the axiom could not be absorbed, so add $\neg \sqcap G$ to $\mathcal{T}_g'$, $\mathcal{T}_g' \rightarrow \mathcal{T}_g' \sqcup \neg \sqcap G$, and exit.

In the above absorption technique, step (E) is new and has been added to the technique in [16] to handle QC-negations.
3.4 Semantic branching

Standard tableau algorithms are inherently inefficient because they use a search technique based on syntactic branching. When expanding the label of a node $x$, $\mathcal{L}(x)$, syntactic branching works by choosing an unexpanded disjunction ($C_1 \cup C_2 \cup \ldots \cup C_n$) in $\mathcal{L}(x)$ and searching the different models obtained by adding each of the disjuncts $C_1, C_2, \ldots, C_n$ to $\mathcal{L}(x)$ [16]. Since the alternative branches of the search tree are not disjoint, the recurrence of an unsatisfiable disjunct in different branches can occur. This can lead to a lot of wasted expansions. For example, consider the tableau expansion of a node $x$, where $\mathcal{L}(x) = \{(C \cup D_1), (C \cup D_2)\}$ and $C$ leads to a clash. The syntactic branching technique could lead a wasted expansion as shown in Figure 1, where a clash due to $C$ must be demonstrated twice. This problem can be dealt with by using a semantic branching technique analogous to that of classical DL [16].

With semantic branching, a single disjunct $D$ is chosen from one of the unexpanded disjunctions in $\mathcal{L}(x)$. The two possible sub-trees obtained by adding either $D$ or $\overline{D}$ to $\mathcal{L}(x)$ are then searched (recall, $\overline{D}$ is the QC-negation of $D$, i.e., $D \cap \overline{D} = \emptyset$ w.r.t. the domain, and $\{D, \overline{D}\}$ is the clash). Now we have two disjoint subtrees and the possibility of wasted expansions such as in syntactic branching is avoided. As shown in Figure 2, with the semantic branching, only one exploration of the expression $C$ was needed whereas, with the syntactic branching, two explorations are needed.

3.5 Dependency directed backtracking

If a sub-problem leads to a clash, this clash can be detected only when the sub-problem is expanded. So inherent unsatisfiability concealed in sub-problems can lead to large amounts...
of unproductive backtracking search known as thrashing. The problem becomes worse when blocking is used to guarantee termination, because blocking may require that sub-problems be expanded only after all other forms of expansions have been performed. For example, expanding a node \( x \), where \( L(x) = \{ (C_1 \sqcup D_1), \ldots, (C_n \sqcup D_n), \exists R. (A \sqcap B), \forall R. A \} \) could lead to the fruitless exploration of \( 2^n \) possible \( R \) successors of \( x \) until the inherent unsatisfiability is discovered. The search tree created by tableau algorithm (using semantic branching) is presented in Figure 3. This problem can be addressed by adopting a form of dependency-directed backtracking called backjumping. The technique is essentially the same as that for classical DL presented in [16]; here we review the techniques for handling the QC-negations.

Backjumping is a crucial optimization technique that can effectively prune irrelevant alternatives of non-deterministic branching decisions. If a branching point is not involved in a clash, other alternatives of the branching point may be bypassed, because they cannot eliminate the cause of clash. So the challenge is to locate the cause of a clash which will allow one to downsize the search space. In order to identify the branching point involved in a clash, all concepts are labeled with a dependency set containing information about the branching points on which they depend. A concept \( C \in L(x) \) depends on a branching point if \( C \) was added to \( L(x) \) at the branching point or if \( C \) depends on another concept \( D \) and \( D \) depends on that branching point. A concept \( C \in L(x) \) depends on a concept \( D \) when \( C \) was added to \( L(x) \) by the application of a deterministic expansion rule that used \( D \). For example, if \( A \in L(x) \) was derived from the expansion of \( (A \sqcap B) \in L(x) \), then \( A \in L(x) \) depends on \( (A \sqcap B) \in L(x) \) [16].

When a concept is added to a node by applying the tableau expansion rules, it inherits the dependencies from the concepts it was generated by. If the concept is added by the application of a non-deterministic rule, a dependency from the new branching point is also added. When a clash is discovered, a new dependency set is created by the union of dependency sets of the clashing concepts and backtracking is initiated. In the backtracking, each branching point is checked against the dependency set to see whether it is in the dependency set. If a branching point is not in the dependency set, then the other branching points are ignored and backtracking continues. If the branching point is in the dependency set, and the other branches are not explored yet, then backtracking stops and searching proceeds with the exploration of the other branches. When all branches of a branching point are explored, the union of the dependency sets from the branches is taken and backtracking continues [16].
For example, consider the previous example; when expanding a node $x$, where $\mathcal{L}(x) = \{ (C_1 \sqcup D_1), \ldots, (C_n \sqcup D_n), \exists R.(A \sqcap B), \forall R.\overline{A} \}$, by using backjumping we could reduce the search space dramatically. In the search algorithm, after creating $n$ branches, the label of the $n$th node $x_n$ will contain $\{ \exists R.(A \sqcap B), \forall R.\overline{A} \}$. When $\exists R.(A \sqcap B)$ is expanded, the algorithm creates an $R$-successor $y_1$ with $\mathcal{L}(y_1) = \{ (A \sqcap B) \}$ using the $\exists$-rule, and then applies the $\forall$-rule to $\forall R.\overline{A}$ which results with $\mathcal{L}(y_1) = \{ (A \sqcap B), \overline{A} \}$. Now, applying the $\sqcap$-rule to $A \sqcap B$ leads to a clash, because $\{ A, \overline{A} \} \subset \mathcal{L}(y_1)$. Since neither $A$ nor $\overline{A}$ in $\mathcal{L}(y_1)$ depend on the branching point from $x$ to $x_n$, the algorithm backtracks to the most recent branching point on which one of $A$ or $\overline{A}$ did depend without exploring alternative branches at any branching point between $x$ to $x_n$. We show in Figure 4 how the search tree below $x$ is pruned by backjumping, and thus the number of $R$-successors explored is reduced by $2^n - 1$.

4 Evaluation

To check the effectiveness of the optimization techniques discussed in the previous section for inconsistency-tolerant reasoning, i.e., QC-CDL, we implemented those techniques top of QC-OWL. QC-OWL [17] is an inconsistency-tolerant reasoner that can handle inconsistency directly with reasonable inference power. It can perform reasoning over both consistent and inconsistent ontologies with acceptable performance for the DL $\mathcal{SHIQ}$. It was designed and developed by following the Strategy Pattern, a behavioral design pattern, and is based on the core framework of Pellet [20], a widely used complete OWL-DL reasoner.

In this section, we compare our results with those of our previous version of QC-OWL. The benchmark ontologies used in the experiments are presented in Table 5. The first 12 ontologies (ID# 1 to ID# 12) were found in [15] while 13 (ID# 13) was found in [10]. The ontologies 14, 15, 16, 17, 18 (ID# 14 to ID# 18) were collected from the TONES Ontology Repository [21] while the remaining two (ID# 19 to ID# 20) were collected from ISG Ontology Repository [22]. For the experiments each ontology was processed five times and the time required to perform QC-consistency test was recorded. The average time of the five independent runs is displayed in the table. In the table, the column Con is for the consistency and Opt with optimizations in QC-OWL. All experiments were performed on a Notebook with Intel Core i7 CPU and 8G.
Table 5: Characteristics of the benchmark KBs.

<table>
<thead>
<tr>
<th>ID</th>
<th>KB name</th>
<th>DL expressivity</th>
<th>Concept count</th>
<th>Axiom count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>amino-acid</td>
<td>ALCF(D)</td>
<td>46</td>
<td>563</td>
</tr>
<tr>
<td>2</td>
<td>heart</td>
<td>SHI</td>
<td>75</td>
<td>448</td>
</tr>
<tr>
<td>3</td>
<td>bad-food</td>
<td>ALCO(D)</td>
<td>18</td>
<td>52</td>
</tr>
<tr>
<td>4</td>
<td>buggyPolicy</td>
<td>ALCHE</td>
<td>15</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>tambis-patched</td>
<td>SHIN</td>
<td>395</td>
<td>1090</td>
</tr>
<tr>
<td>6</td>
<td>uma-025-arctan</td>
<td>ALCRI(D)</td>
<td>366</td>
<td>153403</td>
</tr>
<tr>
<td>7</td>
<td>0.01-arctan</td>
<td>ALCRI(D)</td>
<td>366</td>
<td>14816</td>
</tr>
<tr>
<td>8</td>
<td>0.03-arctan</td>
<td>ALCRI(D)</td>
<td>366</td>
<td>26421</td>
</tr>
<tr>
<td>9</td>
<td>0.04-arctan-inc</td>
<td>ALCRI(D)</td>
<td>366</td>
<td>13829</td>
</tr>
<tr>
<td>10</td>
<td>0.03-arctan-inc</td>
<td>ALCRI(D)</td>
<td>366</td>
<td>26421</td>
</tr>
<tr>
<td>11</td>
<td>0.04-arctan-inc</td>
<td>ALCRI(D)</td>
<td>366</td>
<td>32231</td>
</tr>
<tr>
<td>12</td>
<td>0.07-arctan-inc</td>
<td>ALCRI(D)</td>
<td>366</td>
<td>49592</td>
</tr>
<tr>
<td>13</td>
<td>chem-a</td>
<td>ALCRIF(D)</td>
<td>48</td>
<td>196</td>
</tr>
<tr>
<td>14</td>
<td>goslim</td>
<td>AL</td>
<td>161</td>
<td>480</td>
</tr>
<tr>
<td>15</td>
<td>transportation</td>
<td>AlCH</td>
<td>445</td>
<td>2364</td>
</tr>
<tr>
<td>16</td>
<td>economy</td>
<td>AlCH</td>
<td>339</td>
<td>2817</td>
</tr>
<tr>
<td>17</td>
<td>numerics</td>
<td>SHIF(D)</td>
<td>2364</td>
<td>7268</td>
</tr>
<tr>
<td>18</td>
<td>yowl-complex</td>
<td>SHIF(D)</td>
<td>336</td>
<td>2212</td>
</tr>
<tr>
<td>19</td>
<td>uma-025-arctan</td>
<td>SHIF(D)</td>
<td>108411</td>
<td>369895</td>
</tr>
<tr>
<td>20</td>
<td>00786</td>
<td>SH(D)</td>
<td>93413</td>
<td>1212604</td>
</tr>
</tbody>
</table>

The experiments were conducted in two steps. In the first step, QC-consistency tests were performed for the same set of ontologies as found in [15]. The results are presented in Table 6 and Table 7 for consistent and inconsistent ontologies, respectively. As it is shown in Table 6, QC-OWL(Opt) marginally outperforms QC-OWL for consistent ontologies. Since QC-OWL already shows good performance for this set of ontologies, the performance improvement is not

Table 6: QC-consistency test results (consistent ontologies).

<table>
<thead>
<tr>
<th>KB name</th>
<th>Con</th>
<th>QC-OWL</th>
<th>QC-OWL(Opt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>amino-acid.owl</td>
<td>Y</td>
<td>33</td>
<td>26</td>
</tr>
<tr>
<td>0.01-arctan.owl</td>
<td>Y</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>0.03-arctan.owl</td>
<td>Y</td>
<td>39</td>
<td>36</td>
</tr>
<tr>
<td>heart.owl</td>
<td>Y</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>tambis-patched.owl</td>
<td>Y</td>
<td>39</td>
<td>27</td>
</tr>
<tr>
<td>uma-025-arctan.owl</td>
<td>Y</td>
<td>30</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 7: QC-consistency test results (inconsistent ontologies).

<table>
<thead>
<tr>
<th>KB name</th>
<th>Con</th>
<th>QC-OWL</th>
<th>QC-OWL(Opt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad-food.owl</td>
<td>N</td>
<td>34</td>
<td>26</td>
</tr>
<tr>
<td>buggyPolicy.owl</td>
<td>N</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>0.01-arctan-inc.owl</td>
<td>N</td>
<td>134</td>
<td>41</td>
</tr>
<tr>
<td>0.03-arctan-inc.owl</td>
<td>N</td>
<td>10342</td>
<td>38</td>
</tr>
<tr>
<td>0.04-arctan-inc.owl</td>
<td>N</td>
<td>20044</td>
<td>32</td>
</tr>
<tr>
<td>0.07-arctan-inc.owl</td>
<td>N</td>
<td>186803</td>
<td>202</td>
</tr>
</tbody>
</table>
significant. Indeed, the search spaces of these ontologies are small due to the characteristics of these ontologies. However, the results presented in Table 7 show that QC-OWL(Opt) significantly outperforms QC-OWL for inconsistent ontologies. For example, in the case of the $\arctan$-inc ontology, QC-OWL takes around 20 seconds whereas QC-OWL(Opt) takes only 32 milliseconds. It is important to note that the performance improvement through optimization is greater for inconsistent ontologies than consistent ontologies. The reason is, for an inconsistent ontology, every branch must be explored before returning the result. However, in the case of a consistent ontology, a model can be found before exploring all alternative branches. When a model is found in a branch, the algorithm returns immediately without exploring the remaining branches. Therefore in general, the performance improvement for inconsistent ontologies can be expected to be greater than that for consistent ontologies.

In the second step, QC-consistency test were performed for another set of popular ontologies and the results are presented in Table 8. For this experiments, the maximum allocated memory for JVM was 2G and mem-out stands for OutOfMemoryError in Java. The results presented in Table 8 show that optimizations play a significant role for the performance improvement. As an example, for the transportation ontology, QC-OWL takes 470 milliseconds whereas QC-OWL(Opt) takes only 45 milliseconds. It is motivating to note for the 00786 ontology, QC-OWL gets mem-out while QC-OWL(Opt) gets result in 325 milliseconds. The performance improvement in Table 8 is significant because the search spaces of these ontologies are larger than the ontologies in Table 6 (i.e., the ontologies in Table 8 contain more individuals than the ontologies in Table 6).

<table>
<thead>
<tr>
<th>KB name</th>
<th>Con</th>
<th>QC-OWL</th>
<th>QC-OWL(Opt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>chem-a.owl</td>
<td>N</td>
<td>94</td>
<td>32</td>
</tr>
<tr>
<td>goslim.owl</td>
<td>Y</td>
<td>260</td>
<td>44</td>
</tr>
<tr>
<td>transportation.owl</td>
<td>Y</td>
<td>470</td>
<td>45</td>
</tr>
<tr>
<td>economy.owl</td>
<td>Y</td>
<td>661</td>
<td>71</td>
</tr>
<tr>
<td>numerics.owl</td>
<td>Y</td>
<td>2075</td>
<td>100</td>
</tr>
<tr>
<td>yowl-complex.owl</td>
<td>Y</td>
<td>12262</td>
<td>96</td>
</tr>
<tr>
<td>00390.owl</td>
<td>Y</td>
<td>mem-out</td>
<td>32</td>
</tr>
<tr>
<td>00786.owl</td>
<td>Y</td>
<td>mem-out</td>
<td>325</td>
</tr>
</tbody>
</table>

Table 8: QC-consistency test results.

5 Related and Future Work

In the past decades, numerous techniques have been developed for optimizing standard tableau-based reasoning, but none of them are directed to inconsistency-tolerant reasoning. In this section, we outline some work that is related to optimizing the tableau-based reasoning for classical DLs and discuss some future research directions for improving the performance of an inconsistency-tolerant tableau-based reasoner.

Most state-of-the-art optimization techniques in tableau-based DL reasoning have been discussed in [16]. Apart from optimizing the tableau algorithm, a few researchers also attempted to parallelize the tableau algorithm itself by applying a thread-based strategy in a shared-memory environment (see, for example [17]). Although thread-based strategies such as multi-threading in a multi-cored processor are often the easiest and simplest way to achieve high performance, speed gain via thread-level parallelism is limited by the number of available cores. A process-based strategy discussed in [17] is another option for achieving scalable performance.
In order to provide efficiency in reasoning, 3 profiles for OWL 2 offer important advantages depending on the application scenario: OWL 2 EL, OWL 2 QL and OWL 2 RL [23]. For example, OWL 2 EL is useful for ontologies that contain very large numbers of properties and/or classes, while OWL 2 QL is useful for dealing with ontologies with very large volumes of instance data, and where query answering is the most important reasoning task. OWL 2 RL is aimed at applications that require scalable reasoning without sacrificing expressivity [23].

For the experiments, our reasoner has been implemented on top of Pellet using Java. Although Java has many appealing features, it is not strongly recommended for high performance computing due to some design features associated with this language, such as garbage collection, etc. Better performance can be achieved by implementing this algorithm in C or C++. Prolog, a general purpose logic programming language, could be another good choice. Prolog’s backtracking strategy may be well suited implementing the dependency-directed backtracking.

Recently, Faddoul and MacCaull [24] investigated algebraic tableau reasoning for the DL \textit{ALCQ}. Preliminary results motivate the application of algebraic reasoning for paraconsistent reasoning. However, in order to work with an algebraic reasoning component, a standard tableau calculus needs to be modified and extended.

6 Conclusion

In this work, we discuss a set of widely used optimization techniques developed for classical tableau-based reasoners which we implemented on top of QC-OWL, our inconsistency-tolerant reasoner, and compare its performance with our naive implementation. The experimental results show a significant runtime improvement for a wide range of both consistent and inconsistent ontologies. While it is possible to measure the performance of each optimization individually, we did not do so, as these optimizations are benchmarked for classical DLs. In future, we shall investigate other optimizations for QC-OWL, e.g., boolean constraint propagation, heuristic guided search, etc. We also plan to parallelize the QC-tableau algorithm itself incorporating a set of optimization techniques which is hoped to significantly improve the performance of the QC-tableau algorithm.

Acknowledgments: The second author wishes to thank the Natural Sciences and Engineering Research Council of Canada for financial support.

References


