LOGIC JOURNAL of the IGPL

Volume 10 Number 5 September 2002

Interest Group in Pure and Applied Logics

OXFORD UNIVERSITY PRESS

ISSN 1367-0751
www.oup.co.uk/igpl
Printed in Great Britain by Watkiss Studio Ltd.

Subscription Information

Volume 10, 2002 (bimonthly) Full: Europe pounds sterling 275; Rest of World US$ 450. Personal: pounds sterling 138 (US$ 225). Please note that personal rates apply only when copies are sent to a private address and payment is made by a personal cheque or credit card.

Order Information

Subscriptions can be accepted for complete volumes only. Prices include air-speeded delivery to Australia, Canada, India, Japan, New Zealand, and the USA. Delivery elsewhere is by surface post. Payment is required with all orders and may be made in the following ways:

Cheque (made payable to Oxford University Press)
National Girobank (account 500 1056)
Credit card (Access, Visa, American Express)
UNESCO Coupons
Bankers: Barclays Bank plc, PO Box 333, Oxford, UK. Code 20-65-18, Account 00715654.

Requests for sample copies, subscription enquiries, orders and changes of address should be sent to the Journals Subscriptions Department, Oxford University Press, Great Clarendon Street, Oxford OX2 6DP, UK. Tel: +44 (0) 1865 267907. Fax: +44 (0) 1865 267485. Email: jnl.orders@oup.co.uk

Advertisements

Advertising enquiries should be addressed to Peter Carpenter, PRC Associates, The Annexe, Fitznells Manor, Chessington Road, Ewell Village, Surrey KT17 1TF, UK. Tel: +44 (0) 181 786 7376. Fax: +44 (0) 181 786 7262.

Copyright

Logic Journal of the IGPL (ISSN 1367-0751) is published bimonthly in January, March, May, July, September and November by Oxford University Press, Oxford, UK. Annual subscription price is US$ 450.00. *Logic Journal of the IGPL* is distributed by M.A.I.L. America, 2323 Randolph Avenue, Avenel, NJ 07001. Periodical postage paid at Rahway, New Jersey, USA and at additional entry points.
US Postmasters: Send address changes to *Logic Journal of the IGPL*, c/o Mercury International, 365 Blair Road, Avenel, NJ 07001, USA.

Back Issues

The current plus two back volumes are available from Oxford University Press. Previous volumes can be obtained from the Periodicals Service Company, 11 Main Street, Germantown, NY 12526 USA. Tel: +1 (518) 537 4700, Fax: +1 (518) 537 5899.

Logic Journal of the IGPL

Volume 10, Number 5, September 2002

Contents

Original Articles

Please visit the journal's World Wide Web site at
http://www.oup.co.uk/igpl

Logic Journal of the Interest Group in Pure and Applied Logics

Scope of the Journal

The *Journal* is the official publication of the International Interest Group in Pure and Applied Logics (IGPL), which is sponsored by The European Foundation for Logic, Language and Information (FoLLI), and currently has a membership of over a thousand researchers in various aspects of logic (symbolic, computational, mathematical, philosophical, etc.) from all over the world.

The *Journal* is published in hardcopy and in electronic form six times per year. Publication is fully electronic: submission, refereeing, revising, typesetting, checking proofs, and publishing, all is done via electronic mailing and electronic publishing.

Papers are invited in all areas of pure and applied logic, including: pure logical systems, proof theory, model theory, recursion theory, type theory, nonclassical logics, nonmonotonic logic, numerical and uncertainty reasoning, logic and AI, foundations of logic programming, logic and computation, logic and language, and logic engineering.

The *Journal* is an attempt to solve a problem in the logic (in particular, IGPL) community:

- Long delays and large backlogs in publication of papers in current journals.
- Very tight time and page number limits on submission.

Papers in the final form should be in LaTeX. The review process is quick, and is made mainly by other IGPL members.

Submissions

Submissions are made by sending a submission letter to the e-mail address: jigpl@dcs.kcl.ac.uk, giving the title and the abstract of the paper, and informing: of how to obtain the file electronically or, by sending 5 (five) hardcopies of the paper to the Editor-in-Chief.

URL: www.oup.co.uk/igpl

Formalizing Belief Revision in Type Theory

Tijn Borghuis, *Mathematics and Computing Science, Technische Universiteit Eindhoven, P.O.Box 513, 5600 MB Eindhoven, the Netherlands,* v.a.j.borghuis@win.tue.nl

Fairouz Kamareddine, *Mathematical and Computer Sciences, Heriot-Watt Univ., Riccarton, Edinburgh EH14 4AS, Scotland. E-mail: fairouz@macs.hw.ac.uk*

Rob Nederpelt, *Same address as Borghuis. E-mail: r.p.nederpelt@tue.nl*

Abstract

This paper formalizes belief revision for belief states in type theory. Type theory has been influential in logic and computer science but as far as we know, this is the first account at using type theory in belief revision. The use of type theory allows an agent's beliefs as well as his justifications for these beliefs to be explicitly represented and hence to act as first-class citizens. Treating justifications as first-class citizens allows for a deductive perspective on belief revision. We propose a procedure for identifying and removing "suspect" beliefs, and beliefs depending on them. The procedure may be applied iteratively, and terminates in a consistent belief state. The procedure is based on introducing explicit justification of beliefs. We study the belief change operations emerging from this perspective in the setting of typed λ-calculus, and situate these operations with respect to standard approaches.

Keywords: Belief Revision, Type Theory, Explicit Justifications, Propositions as Types.

1 Introduction

An agent who keeps expanding his belief state with new information may reach a stage where his beliefs have become inconsistent, and his belief state has to be adapted to regain consistency. Usually, in studying this problem of "belief revision", the justifications an agent has for his beliefs are not considered to be first-class citizens.

The two main approaches in the belief revision literature regarding justifications [17] are:

1. "Foundations theory", in which one needs to keep track of justifications for one's beliefs; propositions that have no justification should not be accepted as beliefs.
2. "Coherence theory", in which one needs not consider justifications; what matters is how a belief coheres with the other beliefs that are accepted in the present state.

In foundations theory, beliefs are held to be justified by one or several other beliefs (and some beliefs are justified by themselves). However, in this view, justifications are only *implicitly* present as relations between beliefs, rather than as objects in their own right which are *explicitly* represented in the formalisation of belief states and belief change operations. Hence, justifications are not first-class citizens in foundations

theory, and not considered at all in coherence theory.

However, experience in the past decades shows that when building automated systems and theorem provers, explicit representation is absolutely necessary. This is the case for example in the theorem prover Automath (for automating mathematics, [31]) where definitions (the heart of mathematics) are made explicit. This is also the case in the implementation of programming languages where contexts and environments are made explicit. It turns out also that treating justifications explicitly (hence as first-class citizens), allows for a deductive perspective of belief revision which can be automated.

In this paper, we explore belief revision for belief states in which justifications are first-class citizens represented explicitly. Our motivation for investigating belief revision along these lines stems from working on knowledge representation in Pure Type Systems [4] in the DenK-project [9]. Type theory was chosen due to its excellent success in the field of theorem proving (Automath [31] and Coq [5]) and programming languages (ML [30]). See also [6] where type theory has been shown to be useful for knowledge representation. In the DenK-project a formal model was made of a specific communication situation, and used to implement a human-computer interface. Both in the model and in the system, belief states of agents were formalised as type theoretical contexts. This means that an agent's beliefs are represented in a binary format, where one part is the proposition believed by the agent and the other the justification the agent has for this belief. Both parts are syntactic objects in their own right, and can be calculated upon by means of the rules of the type theory. This way of representing beliefs turns justifications into first-class citizens, and proved to be very fruitful for the purposes of the project.

At that time mechanisms for belief revision were not investigated or implemented, but it became clear that given this formalisation of belief states there is a straightforward deductive approach to the problem: since every belief is accompanied by its justification (and the rules of the calculus operate on both), every inconsistency that surfaces in the agent's belief state has its own (complex) justification containing the justifications of the beliefs that together cause the inconsistency. This makes it easy to identify and remove the "suspects" among the beliefs in the agent's belief state. Although, technically speaking, this is a direct consequence of the so-called Propositions As Types-principle (cf. sections 3 and 4), this simple idea seems not to have been explored before. We feel that this is of a more general interest for two reasons:

1. Our type theoretical case study shows that explicitly represented justifications have clear advantages: a number of drawbacks traditionally associated with foundational approaches disappear. As such, it may serve as a precursor to a more general account in the setting of Labelled Deductive Systems [15], of which typed λ-calculi are a simple case.[1]
2. It may contribute to a more computational account of belief revision, one which is applicable to agents that have finite information and finite reasoning powers.

In developing the idea, we will come across other well-known issues in this field of research. For instance the question whether belief states should be taken to be log-

[1] Note that in the conclusion, [16] discusses the possibility of a general theory of inconsistency where an account of belief revision would fall out as a special case. However, as far as we know, this general theory of inconsistency in LDS has not yet been materialized.

ically closed sets or rather a base set of beliefs which is not closed under logical consequence [19], and the question whether an agent should always accept new information (prioritized versus non-prioritized revision [21]). In addition, we question a number of assumptions that are traditionally made such as the assumption that an agent has infinite reasoning powers, and that an agent has to solve the revision problem "in splendid isolation", i.e. without going back to his sources of information via observation and communication.

The paper is structured as follows: in Section 2 we review type theory and its untyped basis (the type-free λ-calculus), the propositions-as-types principle and introduce the extension of type theory with definitions that will be used for belief revision. In Section 3, we explain how belief states can be captured in type theory. Section 4 shows how type theoretical belief states develop as new information becomes available, and gives an informal statement of the problem of revision in type theory. This account of type theoretical revision is formalised in Section 5. In Sections 6 and 7 we situate our approach with respect to standard approaches from the literature, and make a comparison on the level of belief change operations. As it turns out, our revision procedure is particularly close to the so-called consolidation operations. This is shown in Appendix A. We conclude in Section 8.

2 Type theory

2.1 Informal introduction

Judgements

The basic relation in type theory is the *judgement*

$$\Gamma \vdash a : T$$

which can be read as 'term a has type T in context Γ'. Here 'a' and 'T' are both formulas written according to a well-defined syntax (on the basis of λ-calculus). The expression $a : T$ is called a *statement*, term a is *the subject* of the statement. One also says that term a is an *inhabitant* of type T.

The context Γ is a list of statements with *variables* as subjects, e.g. $x_1 : T_1, \ldots, x_n : T_n$. The above judgement can then be read as follows: "If x_1 has type T_1, ..., and x_n has type T_n, then term a has type T". Note that a may *contain* x_1, ..., x_n, so a *depends on* x_1 to x_n. The set $\{x_1, \ldots, x_n\}$ is called the *domain* of Γ, or $\mathtt{dom}(\Gamma)$.

Statements

The intuitive notion 'has type' has a direct counterpart in naive set theory, viz. 'is element of'. For example, consider the statement '$a : \mathbf{N}$' ('term a *has type* $\mathbf{N}$'). Assuming that $\mathbf{N}$ is a symbol representing the set of natural numbers, this statement can be interpreted as '$a \in \mathbf{N}$' ('the object represented by a *is element of* the naturals').

The notion of having a type, however, is more general than the notion of set-theoretical elementhood. This is because a type T can represent not only some kind of set, but also *a proposition*. In the latter representation, the statement $a : T$ expresses: 'a is (a term representing) a *proof* of the proposition T'. One speaks of 'propositions as types and proofs as terms' (together abbreviated as *PAT*) in order

to emphasize this special usage of types. Section 2.2 below gives more details.

The advantage of PAT is that proofs belong to the object language, not the meta-language. That is, proofs are 'first class citizens' in the syntactical world of type theory. This, combined with the strength of the standard λ-calculus operations, makes type theory a powerful mechanism.

Contexts

The context Γ in a judgement $\Gamma \vdash a : T$ contains the 'prerequisites' necessary for establishing the statement $a : T$. A context Γ is a list of statements with *distinct* variables as subjects, like $x_1 : T_1, \ldots, x_n : T_n$. A context statement $x_i : T_i$ can express several kinds of prerequisites, the simplest being:

1. x_i is an element of the set T_i,
2. T_i is an assumption (a proposition) and x_i is its atomic justification.

However, in type theory there are different 'levels' of typing: a type can have a type itself. Statements expressing the typing of types are concerned with the well-formedness of these types. For the T_i occurring in 1. and 2. above, such statements have the form:

1. T_i : `set`, to express that T_i is a well-formed formula representing a set,
2. T_i : `prop`, to express that T_i is a well-formed formula representing a proposition.

The last-mentioned statements can also be part of a context in the special case that T_1 and T_2 are variables. So a context could look like: $T_1 : \texttt{prop}, T_2 : \texttt{set}, x_1 : T_1, x_2 : T_2$ (to be read as: "let T_1 be a proposition, T_2 a set, x_1 a justification for T_1 and x_2 for T_2"). The terms `set` and `prop` are examples of so-called *sorts*, predefined constants on which the type system is based. Every type system has a specific set of sorts, which we denote by $\mathcal{S}$.

Note that the statements in the context are *ordered*: first arbitrary set T_1 and proposition T_2 are proposed, before their inhabitants x_1 and x_2 are introduced. This is a general principle in contexts: every variable (except the sorts) used in a type must be introduced as the subject of a preceding statement. As a matter of fact, a similar consideration applies to *judgements*: in $\Gamma \vdash a : T$ all variables and (free) constants used in a *and* T must be introduced as subjects in Γ.

2.2 PAT: Propositions As Types

The idea of PAT originates in the formulation of intuitionistic logic where frequently occurring constructions in intuitionistic mathematics have a logical counterpart. One of these constructions is the proof of an implication. Heyting [24] describes the proof of an implication $a \Rightarrow b$ as: Deriving a solution for the problem b from the problem a. Kolmogorov [28] is even more explicit, and describes a proof of $a \Rightarrow b$ as the construction of a method that transforms each proof of a into a proof of b. This means that a proof of $a \Rightarrow b$ can be seen as a *(constructive) function* from the proofs of a to the proofs of b. In other words, the proofs of the proposition $a \Rightarrow b$ form exactly the set of functions from the set of proofs of a to the set of proofs of b. This suggests to identify a proposition with the set of its proofs. Now *types* are used to

represent these sets of proofs. An element of such a set of proofs is represented as a *term* of the corresponding type. This means that propositions are interpreted as *types*, and proofs of a proposition a as *terms of type* a.

PAT was, independently from Heyting and Kolmogorov, discovered by Curry and Feys [13]. Howard [25] follows the argument of Curry and Feys [13] and combines it with Tait's discovery of the correspondence between cut elimination and β-reduction of λ-terms [32]. Howard's discovery dates from 1969, but was not published until 1980. Independently of Curry and Feys and Howard, we find a variant of PAT in AUT-68, the first Automath system of De Bruijn [31]. Though De Bruijn was probably influenced by Heyting, his ideas arose independently from Curry, Feys and Howard. This can be clearly seen in Section 2.4 of [8], where propositions as types (or better: Proofs as terms) was implemented in a different way to that of Curry and Howard.

The Propositions as Types and Proofs as Terms (PAT) principle has opened the possibility to use Type Theory not only as a restrictive method (to prevent paradoxes) but also as a constructive method. Many proof checkers and theorem provers, like Automath [31], Coq [5] and LF [23], use the PAT principle (see [29] for more details).

"Proofs as terms" already suggests an important advantage of using type theory as a logical system: In this method proofs are first-class citizens of the logical system, whilst for many other logical systems, proofs are rather complex objects outside the logic (for example: derivation trees), and therefore cannot be easily manipulated.

The fact that PAT was discovered independently by many different people, and its use in various logical frameworks and theorem provers, is an evidence to the usefulness of such notion in logic and compoutation. For our purpose of belief revision, PAT allows to store the developmental history of the justifications of a belief and hence, to retrace back this histoy and to restore inconsistent belief states.

2.3 Theories

A 'proof' is generally considered to be a mathematical notion, but in the PAT-style a proof is anything *justifying* a proposition. This can be a proof in the mathematical sense, but also any other acceptable justification. Let T represent a proposition and let $a : T$. Then:

- If a is an *atomic term* (think of a constant or a variable), then a encodes a justification which cannot be further analysed:
  - It can stand for an axiomatic justification of a proposition: T is an *axiom* and a expresses that the axiom 'holds'.
  - The validity of proposition T can also come from a *reliable source*. In this case the proof a itself cannot be inspected, but the reliability of the source is enough guarantee to accept the proof. The origin of the knowledge can be any source, either virtual: e.g. a knowledge base, or real: a reliable (community of) person(s).
  - Proposition T can also be justified by *observational evidence*. For example, the proposition that a certain body is yellow can be justified by an atomic term representing the observation that this is the case.
  - Finally, proposition T can be an *assumption*. This case is dealt with in type theory by introducing a variable (say x) as an arbitrary (but fresh) inhabitant for the proposition: the statement $x : T$ then expresses: 'Let x be a proof of T'.

Since x is an unspecified variable, this amounts to: 'Assume T' (albeit that the proof x can be called upon later).

- If a is a *composite term*, composed according to the (type-theoretical) syntax, it embodies a complex justification. In this case the precize structure of a expresses how the evidence for T is constructed. For example, under the PAT-interpretation a complete mathematical proof (of a theorem) is coded in one, possibly large, composite term. But also a justification that combines knowledge obtained from observing a certain object with general rules about its behaviour, will lead to a composite term.

The PAT-interpretation enables a well-established connection between mathematics and type theory, as has been shown already in the Automath project [31], in which large parts of mathematics have been formalized in type theory: an entire mathematical theory was rendered as a list of judgements. The great importance of such a type-theoretical formalization is that it makes it possible to check whether a given proof of a certain theorem does indeed *prove* the theorem. In fact, it turns out that *syntactical correctness* of the list of judgements is enough to establish the mathematical correctness of the mathematical theory. And the check on syntactical correctness is relatively easy, since the question whether a certain term is of a certain type in a certain context is decidable. This check on syntactical correctness can be performed by man, but also by a straightforward computer program. In the Automath project, this has already been done with the computer technology of the seventies.

A second advantage is the long-standing connection between logic and type theory. The 'reasoning power' of logic finds a natural counterpart in the operations of λ-calculus underlying type theory. A well-known result is that logics of arbitrarily high order can be expressed in type theory. In the PAT-interpretation of logic, terms capture the full proof *process*: from a proof term one can reconstruct not only the premisses used in the proof, but also the order in which they were used and the logical rules used to combine them.

2.4 The type free λ-calculus

Modern type theory is based on the λ-calculus. This section intrtoduces the type free λ-calculus.

Definition 2.1 (Syntax of λ-terms) The set of classical λ-terms or λ-expressions $\mathcal{M}$ is given by: $\mathcal{M} ::= \mathcal{V} \mid (\lambda\mathcal{V}.\mathcal{M}) \mid (\mathcal{M}\mathcal{M})$ where $\mathcal{V} = \{x, y, z, \dots\}$ is an infinite set of *term variables*. We let $v, v', v'', \cdots$ range over $\mathcal{V}$ and $A, B, C \cdots$ range over $\mathcal{M}$.

Example 2.2 $(\lambda x.x)$, $(\lambda x.(xx))$, $(\lambda x.(\lambda y.x))$, $(\lambda x.(\lambda y.(xy)))$, and $((\lambda x.x)(\lambda x.x))$ are all classical λ-expressions.

This simple language is surprisingly rich. Its richness comes from the freedom to create and apply functions, especially higher order functions to other functions (and even to themselves). To explain the intuitive meaning of these three sorts of expressions, let us imagine a model where every λ-expression denotes an element of that model (which is a function). In particular, the variables denote a function in the model via an interpretation function or an *environment* which maps every variable into a specific element of the model. Such a model by the way was not obvious for more

than forty years. In fact, for a domain D to be a model of λ-calculus, it requires that the set of functions from D to D be included in D. Moreover, as the λ-calculus represents precisely the recursive functions, we know from Cantor's theorem that the domain D is much smaller than the set of functions from D to D. Dana Scott was armed by this theorem in his attempt to show the non-existence of the models of the λ-calculus. To his surprise, he proved the opposite of what he set out to show. He found in 1969 a model which has opened the door to an extensive area of research in computer science. We will not go into the details of these models in this paper.

Definition 2.3 (Meaning of Terms) Here is now the intuitive meaning of each of the three λ-expressions given in the syntax:

Variables Functions denoted by variables are determined by what the variables are bound to in the *environment*. Binding is done by λ-abstraction.

Function application If A and B are λ-expressions, then so is (AB). This expression denotes the result of applying the function denoted by A to the function denoted by B.

Abstraction If v is a variable and A is an expression which may or may not contain occurrences of v, then $\lambda v.A$ denotes the function that maps the input value B to the output value $A[v := B]$, that is: the expression A in which B has been substituted for v.

Example 2.4 $(\lambda x.x)$ denotes the identity function. $(\lambda x.(\lambda y.x))$ denotes the function which takes two arguments and returns the first.

As parentheses are cumbersome, we will use the following notational convention:

Definition 2.5 (Notational convention) We use these notational conventions:

1. Functional application associates to the left. So ABC denotes $((AB)C)$.
2. The body of a λ is anything that comes after it. So, instead of $(\lambda v.(A_1 A_2 \dots A_n))$, we write $\lambda v.A_1 A_2 \dots A_n$.
3. A sequence of λ's is compressed to one, so $\lambda xyz.t$ denotes $\lambda x.(\lambda y.(\lambda z.t))$.

As a consequence of these notational conventions we get:

1. Parentheses may be dropped: (AB) and $(\lambda v.A)$ are written AB and $\lambda v.A$.
2. Application has priority over abstraction: $\lambda x.yz$ means $\lambda x.(yz)$ and not $(\lambda x.y)z$.

2.4.1 Variables and Substitution

We need to manipulate λ-expressions in order to get values. For example, we need to apply $(\lambda x.x)$ to y to obtain y. To do so, we use the β-rule which says that $(\lambda v.A)B$ evaluates to *the body* A where v is substituted by B, written $A[v := B]$. However, one has to be careful. Look at the following example:

Example 2.6 Evaluating $(\lambda fx.fx)g$ to $\lambda x.gx$ is perfectly acceptable but evaluating $(\lambda fx.fx)x$ to $\lambda x.xx$ is not. By Definition 2.3, $\lambda fx.fx$ and $\lambda fy.fy$ have the same meaning and hence $(\lambda fx.fx)x$ and $(\lambda fy.fy)x$ must also have the same meaning.

Moreover, their values must have the same meaning. However, if $(\lambda fx.fx)x$ evaluates to $\lambda x.xx$ and $(\lambda fy.fy)x$ evaluates to $\lambda y.xy$, then we easily see, according to Definition 2.3, that $\lambda x.xx$ and $\lambda y.xy$ have two different meanings. The first takes a function and applies it to itself, the second takes a function y and applies x (whatever its value) to y.

We define the notions of *free* and *bound* variables which will play an important role in avoiding the problem above. In fact, the λ is a variable binder, just like $\forall$ in logic:

Definition 2.7 (Free and Bound variables) For a λ-term C, the set of free variables $FV(C)$, and the set of bound variables $BV(C)$, are defined inductively as follows:

$$\begin{array}{llll} FV(v) & =_{def} & \{v\} & \qquad BV(v) =_{def} \emptyset \\ FV(\lambda v.A) & =_{def} & FV(A) - \{v\} & \qquad BV(\lambda v.A) =_{def} BV(A) \cup \{v\} \\ FV(AB) & =_{def} & FV(A) \cup FV(B) & \qquad BV(AB) =_{def} BV(A) \cup BV(B) \end{array}$$

An occurrence of a variable v in a λ-expression is free if it is not within the scope of a λv.[2], otherwise it is bound. For example, in $(\lambda x.yx)(\lambda y.xy)$, the first occurrence of y is free whereas the second is bound. Moreover, the first occurrence of x is bound whereas the second is free. In $\lambda y.x(\lambda x.yx)$ the first occurrence of x is free whereas the second is bound. A *closed term* is a λ-term in which all variables are bound.

Here is now the definition of substitution:

Definition 2.8 (Substitution) For any A, B, v, we define $A[v := B]$ to be the result of substituting B for every free occurrence of v in A, as follows:

$$\begin{array}{lcl} v[v := B] & \equiv & B \\ v'[v := B] & \equiv & v \qquad \text{if } v \not\equiv v' \\ (AC)[v := B] & \equiv & A[v := B]C[v := B] \\ (\lambda v.A)[v := B] & \equiv & \lambda v.A \\ (\lambda v'.A)[v := B] & \equiv & \lambda v'.A[v := B] \\ & & \text{if } v' \not\equiv v \text{ and } (v' \notin FV(B) \text{ or } v \notin FV(A)) \\ (\lambda v'.A)[v := B] & \equiv & \lambda v''.A[v' := v''][v := B] \\ & & \text{if } v' \not\equiv v \text{ and } (v' \in FV(B) \text{ and } v \in FV(A)) \end{array}$$

In the last clause, v'' is chosen to be the first variable $\notin FV(AB)$. In the case when terms are identified modulo the names of their bound variables, then in the last clause of the above definition, any $v'' \notin FV(AB)$ can be taken. In implementation however, this identification is useless and a particular choice of v'' has to be made.

Example 2.9 Check that $(\lambda y.yx)[x := z] \equiv \lambda y.yz$, that $(\lambda y.yx)[x := y] \equiv \lambda z.zy$, and that $(\lambda y.yz)[x := \lambda z.z] \equiv \lambda y.yz$.

Lemma 2.10 (Substitution for variable names) Let $A, B, C \in \mathcal{M}$, $x, y, \in \mathcal{V}$. For $x \neq y$ and $x \notin \mathtt{FV}(C)$, we have that: $A[x := B][y := C] \equiv A[y := C][x := B[y := C]]$.

2.4.2 Reduction

The two important notions of reduction are α-reduction which identifies terms up to variable renaming and β-reduction which evaluates λ-terms.

[2] Notice that the v in λv is not an occurrence of v.

Definition 2.11 (Compatibility for the type free λ-calculus) We say that a binary relation $\rightarrow$ on the type free λ-calculus is compatible iff for all terms A, B of the λ-calculus and variable v, the following holds:

$$\frac{A \rightarrow B}{AC \rightarrow BC} \qquad \frac{A \rightarrow B}{CA \rightarrow CB} \qquad \frac{A \rightarrow B}{\lambda v.A \rightarrow \lambda v.B}$$

Definition 2.12 (Alpha reduction) $\rightarrow_\alpha$ is defined to be the least compatible relation closed under the axiom:

$$(\alpha) \qquad \lambda v.A \rightarrow_\alpha \lambda v'.A[v := v'] \qquad \text{where } v' \notin FV(A)$$

Example 2.13 $\lambda x.x \rightarrow_\alpha \lambda y.y$ but it is not the case that $\lambda x.xy \rightarrow_\alpha \lambda y.yy$.
Moreover, $\lambda z.(\lambda x.x)x \twoheadrightarrow_\alpha \lambda z.(\lambda y.y)x$.

Recall that $\lambda x.x \not\equiv \lambda y.y$ even though they represent the same function. They are actually identical modulo α-conversion. I.e. $\lambda x.x =_\alpha \lambda y.y$.

Definition 2.14 (Beta reduction) $\rightarrow_\beta$ is defined to be the least compatible relation closed under the axiom:

$$(\beta) \qquad (\lambda v.A)B \rightarrow_\beta A[v := B]$$

We use $\twoheadrightarrow_\beta$ to denote the reflexive transitive closure of $\rightarrow_\beta$. We say that a term A is a β-normal form if there is no B such that $A \rightarrow_\beta B$.

Example 2.15 Check that $(\lambda x.x)(\lambda z.z) \rightarrow_\beta \lambda z.z$, that $(\lambda y.(\lambda x.x)(\lambda z.z))xy \twoheadrightarrow_\beta y$, and that both $\lambda z.z$ and y are β-normal forms.

Here is a lemma about the interaction of β-reduction and substitution:

Lemma 2.16 Let $A, B, C, D \in \mathcal{M}$.

1. If $C \rightarrow_\beta D$ then $A[x := C] \twoheadrightarrow_\beta A[x := D]$.
2. If $A \rightarrow_\beta B$ then $A[x := C] \rightarrow_\beta B[x := C]$.

PROOF. By induction on the structure of A for 1, on the derivation $A \rightarrow_\beta B$ for 2. ∎

2.5 The syntax and rules of Pure Type Systems

Now we are ready to introduce the syntax and rules of Pure Type Systems (PTSs) which will be the basis of our theory of belief revision. There are two type disciplines: the implicit and the explicit. The implicit style, also known as typing à la Curry, does not annotate variables with types. For example, the identity function is written as in the type-free case, as $\lambda x.x$. The type of terms however is found using the typing rules of the system in use. The explicit style, also known as typing à la Church, does annotate variables and the identity function may be written as $\lambda x : \texttt{Bool}.x$ to represent identity over booleans. In this paper, we consider typing à la Church. We present what is known as *Pure Type Systems* or PTSs. Important type systems that are PTSs include Church's simply typed λ-calculus [11] and the calculus of constructions [12] which are also systems of the Barendregt cube [4]. Berardi [7] and

Terlouw [33] have independently generalised the method of generating type systems into the pure type systems framework. This generalisation has many advantages. First, it enables one to introduce eight logical systems that are in close correspondence with the systems of the Barendregt cube. Those eight logical systems can each be described as a PTS in such a way that the propositions-as-types interpretation obtains a canonical system form [4]. Second, the general setting of the PTSs makes it easier to write various proofs about the systems of the cube.

In PTSs, we have in addition to the usual λ-abstraction, a type forming operator Π. Briefly, if A is a type, and B is a type possibly containing the variable x, then $\Pi x{:}A.B$ is the type of functions that, given a term $a : A$, output a value of type $B[x := a]$. Here, again, $a : A$ expresses that a is of type A. If x does not occur in B, then $\Pi x{:}A.B$ is the type of functions from A to B, written $A \to B$. To the Π-abstraction at the level of types corresponds λ-abstraction at the level of objects. Roughly speaking, if M is a term of type B (M and B possibly containing x), then $\lambda x{:}A.M$ is a term of type $\Pi x{:}A.B$. All PTSs have the same typing rules but are distinguished from one another by the set $\mathcal{R}$ of triples of sorts (s_1, s_2, s_3) allowed in the so-called *type-formation* or Π-*formation* rule, (product). Each PTS has its own set $\mathcal{R}$. A Π-type can only be formed in a specific PTS if the (product) rule is satisfied for some (s_1, s_2, s_3) in the set $\mathcal{R}$ of that system. (see Figure 1).

Definition 2.17 The *set of pseudo-terms* $\mathcal{T}$, is generated by the grammar:
$\mathcal{T} ::= \mathcal{V} \mid \mathcal{C} \mid (\mathcal{T}\,\mathcal{T}) \mid (\lambda\mathcal{V} : \mathcal{T}.\mathcal{T}) \mid (\Pi\mathcal{V} : \mathcal{T}.\mathcal{T})$, where $\mathcal{V}$ is the infinite set of variables $\{x, y, z, \dots\}$ and $\mathcal{C}$ a set of constants over which, $c, c_1, \dots$ range. We use $A, B, \dots$ to range over $\mathcal{T}$ and $v, v', v'', \dots$ to range over $\mathcal{V}$. Throughout, we take $\pi \in \{\lambda, \Pi\}$.

Note that in the type free lambda calculus, there were only three possibilities for terms (given in Definition 2.1): variables, applications or abstractions, and that abstractions contained no typings for the variables abstracted over. The above Definition 2.17 on the other hand, gives the typing of the abstracted variable, and also defines types as well as terms. $\mathcal{C}$ is a set of constants which contains a subset $\mathcal{S}$ called the *sorts*. The set *sorts* contains amongst other things, four special elements: `set`, `prop`, $*$ and $\Box$, with the relations to be defined later that: `set`: $*$, `prop`: $*$ and $* : \Box$. If $A : *$ (resp. $A : \Box$) we say that A is a type (resp. a kind). If A : `set` (resp. A : `prop`), then we consider A as a set (resp. a proposition).

Definition 2.18 (Free and Bound variables) The free and bound variables in terms are defined similarly to those of Definition 2.7 with the exception that $FV(c) =_{def} BV(c) =_{def} \emptyset$ and in the case of abstraction, $FV(\pi v : A.B) =_{def} (FV(B) \setminus \{v\}) \cup FV(A)$ and $BV(\pi v : A.B) =_{def} BV(A) \cup BV(B) \cup \{v\}$.

We write $A[x := B]$ to denote the term where all the free occurrences of x in A have been replaced by B. Furthermore, we take terms to be equivalent up to variable renaming. We assume moreover, the Barendregt variable convention which is formally stated as follows:

Convention 2.19 (VC: Barendregt's Convention) Names of bound variables will always be chosen such that they differ from the free ones in a term. Moreover, different λ's have different variables as subscript. Hence, we will not have $(\lambda x : A.x)x$, but $(\lambda y : A.y)x$ instead.

The definition of compatibility of a reduction relation for PTSs is that of the type-free calculus (given in Definition 2.11) but where the case of abstraction is replaced by:

$$\frac{A_1 \to A_2}{\pi x : A_1.B \to \pi x : A_2.B} \qquad \frac{B_1 \to B_2}{\pi x : A.B_1 \to \pi x : A.B_2}$$

Definition 2.20 *β-reduction* is the least compatible relation on $\mathcal{T}$ generated by

$$(\beta) \qquad (\lambda x : A.B)C \to B[x := C]$$

Note that $(\lambda x : A.B)C$ is reduced and not $(\Pi x : A.B)C$. The latter needs special attention as is shown in [26, 27].

Now, we define some machinery needed for typing:

Definition 2.21

1. A *statement* is of the form $A : B$ with $A, B \in \mathcal{T}$. We call A the *subject* and B the *predicate* of $A : B$.
2. A *declaration* is of the form $x : A$ with $A \in \mathcal{T}$ and $x \in \mathcal{V}$. When d is $x : A$, we define $\mathtt{var}(d)$ and $\mathtt{type}(d)$ to be x and A respectively.
3. A *pseudo-context* is a finite ordered sequence of declarations, all with distinct subjects. We use $\Gamma, \Delta, \Gamma', \Gamma_1, \Gamma_2, \ldots$ to range over pseudo-contexts. The *empty* context is denoted by either $<>$ or nothing at all.
4. If $\Gamma = x_1 : A_1 . \ldots . x_n : A_n$ then $\Gamma, x : B = x_1 : A_1, \ldots, x_n : A_n, x : B$ and $\mathtt{dom}(\Gamma) = \{x_1, \ldots, x_n\}$.
5. We define substitutions on contexts by: $\emptyset[x := A] \equiv \emptyset$, and $(\Gamma, y : B)[x := A] \equiv \Gamma[x := A], y : B[x := A]$.

Definition 2.22 A *type assignment* relation is a relation between a pseudo-context and two pseudo-terms written as $\Gamma \vdash A : B$. The *rules of type assignment* establish which *judgments* $\Gamma \vdash A : B$ can be derived. A judgement $\Gamma \vdash A : B$ states that $A : B$ can be derived from the pseudo-context Γ.

Definition 2.23 Let Γ be a pseudo-context, A be a pseudo-term and $\vdash$ be a type assignment relation.

1. Γ is called legal if $\exists A, B \in \mathcal{T}$ such that $\Gamma \vdash A : B$.
2. $A \in \mathcal{T}$ is called a Γ-$\mathtt{term}$ if $\exists B \in \mathcal{T}$ such that $\Gamma \vdash A : B$ or $\Gamma \vdash B : A$.
   We take Γ-terms $= \{A \in \mathcal{T}$ such that $\exists B \in \mathcal{T}$ and $\Gamma \vdash A : B \vee \Gamma \vdash B : A\}$.
3. $A \in \mathcal{T}$ is called legal if $\exists \Gamma$ such that $A \in \Gamma$-terms.

Definition 2.24 The *specification* of a PTS is a triple $S = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{S}$ is a subset of $\mathcal{C}$, called the *sorts*. $\mathcal{A}$ is a set of *axioms* of the form $c : s$ with $c \in \mathcal{C}$ and $s \in \mathcal{S}$ and $\mathcal{R}$ is a set of *rules* of the form (s_1, s_2, s_3) with $s_1, s_2, s_3 \in \mathcal{S}$.

Definition 2.25 The notion of type derivation, denoted $\Gamma \vdash_{\lambda S} A : B$ (or simply $\Gamma \vdash A : B$), in a PTS whose specification is $S = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, is axiomatised by the axioms and rules of Figure 1.

Remark 2.26 Note that in Figure 1, we insist in the (start) and (weakening) rules that $x \notin \Gamma$, but we do not insist that $x \notin A$. The condition that $x \notin A$ can be derived from from the fact that $x \notin \Gamma$, that $\Gamma \vdash A : s$ and the properties of PTSs.

(axioms)	$\vdash c : s$	if $c : s \in \mathcal{A}$
(start)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	if $x \notin \Gamma$
(weakening)	$\dfrac{\Gamma \vdash B : C \quad \Gamma \vdash A : s}{\Gamma, x : A \vdash B : C}$	if $x \notin \Gamma$
(product)	$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.B) : s_3}$	if $(s_1, s_2, s_3) \in \mathcal{R}$
(application)	$\dfrac{\Gamma \vdash F : (\Pi x : A.B) \quad \Gamma \vdash C : A}{\Gamma \vdash F\,C : B[x := C]}$	
(abstraction)	$\dfrac{\Gamma, x : A \vdash C : B \quad \Gamma \vdash (\Pi x : A.B) : s}{\Gamma \vdash (\lambda x : A.C) : (\Pi x : A.B)}$	
(conversion)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$	

FIG. 1. PTSs with variables names

$\lambda_\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

FIG. 2. Different type formation condition

Each of the eight systems of the cube is obtained by taking $\mathcal{S} = \{*, \Box\}$, $\mathcal{A} = \{*, \Box\}$, and R to be a set of rules of the form (s_1, s_2, s_2) for $s_1, s_2 \in \{*, \Box\}$. We denote rules of the form (s_1, s_2, s_2) by (s_1, s_2). This means that the only possible (s_1, s_2) rules in the set R (in the case of the cube) are elements of the following set: $\{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\}$. The basic system is the one where $(s_1, s_2) = (*,*)$ is the only possible choice. All other systems have this version of the formation rules, plus one or more other combinations of $(*,\Box)$, $(\Box,*)$ and $(\Box,\Box)$ for (s_1, s_2). See Figures 2 and 3.

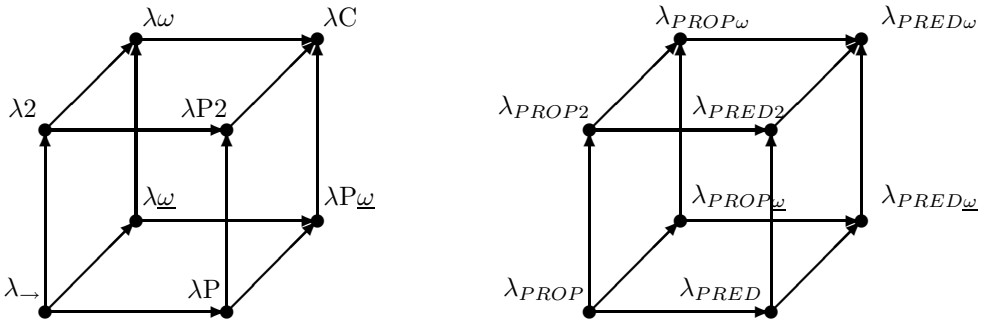


FIG. 3. The λ-cube and its corresponding logic cube

3 Type theory for knowledge representation

This section sets the stage for our account of belief revision with explicit justifications. We give our definition of knowledge and knowledge state, and explain how such knowledge states can be formalized in type theory.

3.1 Knowledge and type theory

PAT is suitable to express the proof as an object *embodying* its developmental history. As a consequence, type theory embodies an excellent machinery for storing (various kinds of) information, including knowledge. The connection between type theory and knowledge is the subject of this section.

We do not intend to present a philosophical or psychological theory of knowledge, but simply identify three characteristics of knowledge which we believe should be taken into account when formalizing knowledge:

- *Subjectivity:* Knowledge is formulated in terms of *concepts*. We assume these concepts are subjective in the sense that one person may judge something to be an instance of a certain concept, while another person would not recognize it as such. Another aspect of subjectivity is that a person's knowledge is *partial*: no one knows everything, and people differ in what they do and don't know.
- *Justification:* Knowledge is justified: persons not only *know* things, but they have *reasons* for knowing them. Generally, parts of knowledge are justified in terms of more basic parts; a person's body of knowledge is structured. And even atomic justifications are supports for the knowledge, since they point at an origin (an axiom, an observation, etc.).
- *Incrementality:* The knowledge of a person can be *extended* as new information becomes available. Whether this information can be incorporated by the person depends on the possibility to tie it to the knowledge already present. This may lead to simply adding the new information, to dismissing it (e.g., because it is incomprehensible) or even to a reorganization of the existing knowledge.

Under an account of knowledge satisfying these requirements, the traditionally made distinction between knowledge and belief disappears: there can be no knowledge which is true in any absolute sense, since an agent's knowledge depends on his

subjective conceptualisation of the world. At best some pieces of knowledge turn out to be more reliable than others and some things can be agreed upon by more agents than others. There is a natural way to capture these characteristics in type theory:

- *Subjectivity is captured by types:* Each concept is formalized as a type, each instance of the concept is a term inhabiting this type. A person's subjective ability to recognize something as an instance of a concept, is mirrored in the ability to judge that the corresponding term inhabits the corresponding type.
  Note that 'having a concept' is also subjective in the sense that different people may have formed different concepts in the course of time. This means that one person can have a concept, whereas another person has no comparable concept. And in case persons *do* have comparable concepts, they may differ in what they recognise as belonging to this concept. In case the type formalizing the concept is a 'set-type', this means that they may differ in what they regard as elements of the set (a rhododendron may be a tree for the one, but a shrub for the other). In case this type is a 'proposition-type', they may differ in what they accept as a justification for that proposition.
- *Justification is captured by terms:* By the PAT-principle, justifications are first-class citizens, formalized in the type-theoretical syntax as terms. The fact that term a justifies proposition T, is expressed as the statement $a : T$. The rules of type theory allow these terms to be combined into complex terms, which reflects that parts of knowledge may be a structured combination of more basic parts.
- *Incrementality is captured by contexts:* As we will explain below, a person's knowledge state can be formalized as a type-theoretical context. Addition of new information to the knowledge state can be formalized by adding statements to the context, dismissing information amounts to reducing the context. Information may only be added if it 'matches' a person's knowledge state. Type theory has an innate notion of 'matching': a statement can only extend a context if it obeys certain well-formedness restrictions.

3.2 Formalization of the knowledge state

The knowledge *state* of a person consists of 'everything he knows' at a certain instant. This knowledge state will be represented as a context Γ in our type system. Every statement in Γ represents a piece of knowledge the person has.

Given our characterization of knowledge, this means that everything in a knowledge state is formulated in terms of the person's concepts. This has several aspects:

- *Meaningfulness:* A person has formed his own, private concepts, and only things which are formulated by means of these concepts can be meaningful to him. Whether or not information coming from outside (by observation or communication) makes sense, depends on the concepts that are already available. (In this paper we will assume that the entirety of concepts of a person is fixed.)
- *Inhabitation:* Whatever a person knows about the world around him is recorded in a knowledge state in the form of meaningful expressions that he accepts. This includes expressions about which objects 'inhabit' the concepts in the world, and which propositions hold in the world, according to the person.

If we take the following (very simple) context as representing a person's knowledge states: $T_1 : \texttt{prop}, T_2 : \texttt{set}, x_1 : T_1, x_2 : T_2$, we can see:

- *Meaningfulness is captured by statements of the form* $T : \texttt{prop}$ *or* $T : \texttt{set}$. That is to say, in this example the person has two concepts, viz. T_1, which is a proposition to him, and T_2, which is a set. (Note that the statements $T_1 : \texttt{prop}$ by itself does not imply that the proposition T_1 holds according to the person, nor does $T_2 : \texttt{set}$ imply that the set T_2 is non-empty.) At this stage, there are no other concepts, i.e. all sets and propositions which are not constructed out of T_1 and/or T_2 are not meaningful to him.
- *Inhabitation is captured by statements of the form* $x : T$*, where* T *is meaningful.* In the example context, the inhabitant x_1 of T_1 represents the person's justification for the holding of T_1, and the inhabitant x_2 of T_2 is an element of the set T_2 which is recognized as such by the person[3].

'Everything a person knows' at a certain instant can be divided into two categories:

- *Explicit knowledge* is expressed by the statements in the context Γ . These are explicitly represented pieces of knowledge directly available to the person.
- *Implicit knowledge* is expressed by statements *derivable* on the context Γ. These are consequences of a person's explicit knowledge which he can get by inference.

Hence, in a judgement of the form $\Gamma \vdash a : T$, the explicit knowledge can be found to the left of the symbol $\vdash$, and the implicit knowledge to the right of $\vdash$.

Note that the knowledge state is not deductively closed, i.e. deriving consequences requires 'work', which is reflected in the construction of a compound justification a for T. Such a construction is a derivation using the rules of type theory; it consists of a *sequence* of judgements of which the just-mentioned compound justification is the final one. We come back to this in the next section.

Assumption 3.1 In order to derive *all* consequences of his explicit knowledge, a person would have to be able to perform possibly infinite derivations. Since this is not feasible, we assume a 'bound' on the derivation depth.

As the above discussion meant that statements of the form $A : B$ (where A may be complex) must be in the knowledge state (which is a context in type theory), and as formulations of type theory only allow statements of the form $x : B$ in the context, we will present here an extension of type theory where contexts not only contain statements of the form $x; B$, but also statements of the form $x := A : B$ (which also states that $A : B$), and are known as definitions.

3.3 PTSs with definitions

In this section we introduce the extension of PTSs given in section 2.5 with definitions.

Terms and types remain unchanged, but contexts are now a list of declarations of the form $x : A$ or of *definitions* of the form $x = B : A$. These latter definitions define x to be B and to have the type A. We extend Definition 2.21 to deal with definitions as

[3] Syntactically, x_1 and x_2 are *variables*. However, as we see later, each of these 'variables' may in fact be a *defined constant*, abbreviating a term which codes all details of the justification.

well as declarations, taking $\mathtt{var}(d)$, $\mathtt{type}(d)$ and $\mathtt{def}(d)$ to be x, A, and B respectively when d is $x = B : A$. We define $FV(x = B : A) \equiv FV(A) \cup FV(B)$. We extend $\mathtt{dom}$ to be $\mathtt{dom}(\Gamma) = \{x \mid x : A \in \Gamma \text{ or } x := B : A \in \Gamma\}$. Finally, we extend substitutions on contexts by $(\Gamma, y := B : C)[x := A] \equiv \Gamma[x := A], y := B[x := A] : C[x := A]$. Note that Definitions 2.22, 2.23, 2.24 and 2.25 are unchanged.

Definition 3.2 The new typing relation $\vdash$ is obtained by adding four new rules to the typing rules of Definition 2.25: (start-def), (weak-def), and (def) below, and by replacing the (conversion) by (new-conv) as follows:

$$\text{(start-def)}\quad \frac{\Gamma \vdash A : s \qquad \Gamma \vdash B : A}{\Gamma, x := B{:}A \vdash x : A} \qquad x \notin \mathtt{dom}(\Gamma)$$

$$\text{(weak-def)}\quad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x := D{:}C \vdash A : B} \qquad x \notin \mathtt{dom}(\Gamma)$$

$$\text{(def)}\quad \frac{\Gamma, x := B{:}A \vdash C : D}{\Gamma \vdash (\pi x : A.C)B : D[x := B]} \qquad \text{for } \pi \in \{\lambda, \Pi\}$$

$$\text{(new-conv)}\quad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad \Gamma \vdash B =_{def} B'}{\Gamma \vdash A : B'}$$

In (new-conv), $\Gamma \vdash B =_{def} B'$ is defined as the smallest equivalence relation closed under:

- If $B =_\beta B'$ then $\Gamma \vdash B =_{def} B'$
- If $x := D : C \in \Gamma$ and B' arises from B by substituting one particular free occurrence of x in B by D then $\Gamma \vdash B =_{def} B'$.

In Definition 3.2, (start-def) and (weak-def) are the start and weakening rules that deal with definitions in the context. The (def) rule types λ- and Π-redexes using definitions in the context.

Now, here are some lemmas that show that the above system is suitable for representing beliefs. The first lemma establishes that different beliefs have different justifications and that all justifications have their evidence in knowledge state Γ.

Lemma 3.3 (Free variable Lemma for $\vdash$) 1. If d and d' are two different elements in a legal context Γ, then $\mathtt{var}(d) \not\equiv \mathtt{var}(d')$.

2. If $\Gamma \equiv \Gamma_1, d, \Gamma_2$ and $\Gamma \vdash B : C$ then $FV(d) \subseteq \mathtt{dom}(\Gamma_1)$ and $FV(B), FV(C) \subseteq \mathtt{dom}(\Gamma)$.

PROOF. 1. If Γ is legal then for some B, C, $\Gamma \vdash B : C$. Now use induction on the derivation of $\Gamma \vdash B : C$. 2. is by induction on the derivation of $\Gamma \vdash B : C$. ∎

Lemma 3.4 (Substitution Lemma for $\vdash$) If $\Gamma, x := D : C, \Delta \vdash A : B$ or $(\Gamma, x : C, \Delta \vdash A : B$ and $\Gamma \vdash D : C)$ then $\Gamma, \Delta[x := D] \vdash A[x := D] : B[x := D]$.

PROOF. Induction on the derivation rules, using Lemma 3.3. ∎

The following corollary means that the person can track down those statements responsible for him entertaining a particular belief.

Corollary 3.5 (Strengthening Lemma for $\vdash$) For $\Gamma_1, y := E : T, \Gamma_2$ a legal context and M and B terms: if $\Gamma_1, y := E : T, \Gamma_2 \vdash M : B$ and $y \notin FV(\Gamma_2) \cup FV(M) \cup FV(B)$, then $\Gamma_1, \Gamma_2 \vdash M : B$.

The next lemma shows that all statements in a knowledge state are meaningful in the sense that if $\Gamma_1, x : A, \Gamma_2$ is legal then $\Gamma_1 \vdash x : A$; and if $\Gamma_1, x := B : A, \Gamma_2$ is legal then $\Gamma_1 \vdash x : A$ and $\Gamma_1 \vdash B : A$

Lemma 3.6 (Context Lemma for $\vdash$) Let Γ_1, d, Γ_2 be a legal context. Then we have: $\Gamma_1 \vdash \mathtt{type}(d) : s$ for some sort s,
$\Gamma_1, d \vdash \mathtt{var}(d) : \mathtt{type}(d)$ and if d is a definition then $\Gamma_1 \vdash \mathtt{def}(d) : \mathtt{type}(d)$.

PROOF. If Γ is legal then for some terms B, C: $\Gamma \vdash B : C$; now use induction on the derivation of $\Gamma \vdash B : C$. ∎

Lemma 3.7 (Thinning Lemma for $\vdash$) Let d be either a declaration or a definition and let Γ_1, d, Γ_2 be a legal context.

1. If $\Gamma_1, \Gamma_2 \vdash A : B$, then $\Gamma_1, d, \Gamma_2 \vdash A : B$.
2. If d is $x := D : C$ and $\Gamma_1, x : C, \Gamma_2 \vdash A : B$, then $\Gamma_1, d, \Gamma_2 \vdash A : B$.

Lemma 3.8 (Swap Lemma for $\vdash$) Assume each of d_1 and d_2 is either a declaration or a definition such that $\mathtt{var}(d_1) \notin FV(\mathtt{type}(d_2))$ and if d_2 is a definition then also $\mathtt{var}(d_1) \notin FV(\mathtt{def}(d_2))$.
If $\Gamma_1, d_1, d_2, \Gamma_2 \vdash A : B$, then $\Gamma_1, d_2, d_1, \Gamma_2 \vdash A : B$.

PROOF. By induction on the derivation $\Gamma_1, d_1, d_2, \Gamma_2 \vdash A : B$. ∎

4 Development of the knowledge state

The knowledge state of a person is not static. As time goes by, new information comes to the person's attention and has to be dealt with. With the conception of knowledge states as type-theoretical contexts in mind, as explained in the previous section, we distinguish several stages in the treatment of new information by a person, marked by decisions which the person has to make. We describe these stages below.

Meaningfulness In the first stage, the meaningfulness of the new information is at stake. New information may or may not be meaningful to a person depending on his current knowledge state. Type-theoretically, new information manifests itself in the form of a (sequence of) statement(s). Whether these statements are meaningful with respect to a knowledge state, can be syntactically decided. In section 3.2 we noted that type theory has an intrinsic notion of meaningfulness. Below we explain how this notion can be extended to statements of the form $x : T$, expressing the inhabitation of a proposition or set T.

We presuppose that a person only processes new information that is meaningful (makes sense) to him, i.e. meaningful with respect to his current knowledge state, and that he *decides* to dismiss this information otherwise. (In a communication setting, we expect the person to search for clarification, either by questioning his dialogue partner, or by (re-)inspecting his environment.)

Expanding the knowledge state If the information *is* meaningful, the person adds it provisionally to the knowledge state: Γ is extended to e.g. $\Gamma_1 \equiv \Gamma, y_1 : T_1, y_2 : T_2$.

The resulting knowledge state can turn out to be consistent, that is to say, the person cannot construct a term M such that $\Gamma_1 \vdash M : \bot$, where $\bot$ is falsum (the logical constant 'falsity'). Recall assumption 3.1 where we assume that the person has a limited deductive power, so he can only construct terms by derivations up to a certain length. Intuitively this means that the person has a 'horizon' behind which he cannot see the consequences of his knowledge state. Hence, the person's notion of 'consistency' is bound by his horizon. (Hence, a knowledge state can be inconsistent without the person being able to find this out at the current point in time.)

If the obtained knowledge state does not give contradictions *within the horizon*, then Γ_1 is accepted as the new context.

Revising the knowledge state There is, however, also the possibility that the person has found an inconsistency, i.e. he has constructed in his newly expanded knowledge state some term M such that $\Gamma_1 \vdash M : \bot$. In that case, he can decide to reject the new information and return to the previous knowledge state. But he can also decide to revise his new knowledge state in order to restore consistency. (The person may actually be able to construct more than one inhabitant of falsum; we assume that he concentrates on one of these.) The most natural thing to do, is to find one or more statements in the context representing his knowledge state, which enabled the construction of M. These statement can be located in the 'old' context, but also in the newly added piece of context, or in both. By removing one or more of these statements from his context, consistency may be regained, since this particular proof of falsum, M, cannot be constructed any more. Below we propose a syntactical iterative procedure which restores consistency. (In general, there is more than one way to regain consistency by removing statements from the knowledge state.)

The stages and decisions we distinguished above, are not intended to capture actual cognitive processes, but merely to state as clearly as possible which aspects of belief revision we do and do not consider in our formalization. For instance, the fact that the person decides which statements to remove, means that this is not decided by the formalism, in other words, we do not postulate so-called *epistemic entrenchement*. (For a comparison with standard theories of belief revision, see section 6.)

In sections 4.1 and 7.3 we discuss the various stages of dealing with information as explained just now, in more detail. We give special attention to the representation in type theory.

4.1 Adding information

The knowledge state of a person changes as new information becomes available to him. Since knowledge states are modeled by type-theoretical contexts, this means that contexts should change accordingly. In this subsection we demonstrate that type theory has the possibility to accommodate such a change in the knowledge state, viz. the addition of new information to the knowledge state.

Adding information to a type-theoretical context amounts to adding statements to this context. This does not mean that arbitrary information may be added, addition of information is subject to syntactical restrictions. We discuss this below, distinguishing between the addition of information originating from inside and from outside the knowledge state of the person.

Adding information from inside

A person is able to reason with his knowledge. For example, let us assume that the statements $A \to B : \texttt{prop}$ and $A : \texttt{prop}$ are meaningful to the person. I.e., from his knowledge state Γ, the person can derive $\Gamma \vdash A \to B : \texttt{prop}$ and $\Gamma \vdash A : \texttt{prop}$. Moreover, let us assume that the person has justifications for both propositions, since $A \to B$ and A are inhabited (e.g. $x : A \to B$ and $y : A$ occur in the context Γ representing his knowledge state). Then the person can infer that B holds, as well, expressed by the statement $xy : B$. This is the case since we have the following instance of the application rule (cf. Figure 1):

$$\frac{\Gamma \vdash x : A \to B \quad \Gamma \vdash y : A}{\Gamma \vdash xy : B}$$

This inference allows the person to combine his justification x for $A \to B$ with his justification y for A into a complex justification xy (pronounced as 'x applied to y') for the proposition B.

Note that there are no more than a small number of typing rules, which are all like the above rule in that they enable to derive a new judgement from one or more judgements which are given or derived earlier.

The judgement $\Gamma \vdash xy : B$ resulting from the person's inference as explained above, shows that the person is able to construct a justification for B on his knowledge state Γ. However, the statement $xy : B$ is not yet part of his knowledge state. To incorporate this statement, it would simply be sufficient to append it to Γ. However, for technical reasons only statements with variables as subject are allowed in the context. In order to circumvent this (technical) problem, in Section 3.3, we expanded our notion of 'context' given in Section 2.5, by allowing also a new kind of statements, called *definitions*, in the context. A definition is a statement of the form $z := E : T$, expressing that z is a name for the term E of type T. The new name z is the *subject* of the definition $z := E : T$. Formally, z is a *variable*. (This is in contrast with the good habit of calling such a defined name a *constant*.) By means of definitions, complex justifications can be abbreviated and recorded in the context. This definition mechanism is essential in the practical use of type theory for the formalization of 'bodies of knowledge', as has been shown e.g. in the Automath project [31].

A definition $z := E : T$ may be added to a context Δ whenever z is fresh with respect to Δ and $E : T$ is derivable on Δ. In the example above, this enables the person to record the inferred $xy : B$ in his knowledge state by adding the definition $u := xy : B$, using some fresh variable u. Hence, the context Γ has evolved into the context $\Gamma, u := xy : B$, reflecting the development of the person's knowledge state brought about by his reasoning. The proposition B (and its justification), which was implicit knowledge of the person (since it occurred at the right hand side of the $\vdash$), has now become explicit knowledge.

From a purely logical point of view, it may seem that adding a derived proposition to the knowledge state (making it *explicit*) does not contribute to the person's *implicit* knowledge. However, this is not the case since we assume a bound on the depth of derivations a person can perform. Under this assumption, the implicit knowledge is limited: it consists of everything a person can derive on his context *within a certain number of derivation steps*. As soon as the explicit knowledge has grown, in general there is more that can be derived by the person in the same number of steps, so the

implicit knowledge has grown as well: the person's 'deductive horizon' has broadened.

Adding information from outside

The knowledge state of a person can change by reasoning (which he does himself, from the *inside*), or by information originating from the *outside*. For the latter there are two important knowledge sources: observational and communicational.

- *Observation:* A person can *recognize an object* (visually, or by any other sensory perception) in his world as belonging to a certain *set*. For example, he sees an object which he characterizes as being a ball. But he can also *obtain evidence* for *propositions* by looking at the outside world. For example, he sees that the ball is yellow.

  In both cases, the new information can be added to the context of the person by the addition of a new statement with a fresh *atomic* subject, acting as the justification. The atomic character of this justification is caused by the impossibility to decompose the observation into smaller parts.

  The two observations in the example above could e.g. be combined into the context extension $b :$ *ball* , $o :$ *yellow* b.
- *Communication:* Another manner in which a person can change his knowledge state is by information passed to him by another person. Again, this information can involve (the existence of) objects as well as (the holding of) propositions.

  For this communication it is necessary that both persons share a language in which they communicate. We assume that each person speaking this language has a mapping between the words of the language and the subjective concepts present in his knowledge state, and vice versa. In [1] a type theoretical model of communication is developed based on this assumption. In this model, the types in a person's knowledge state are communicable via the (mappings to) the common language, but the inhabitants of these types (justifications) are not. Hence the contents of a communication take the form of a (sequence of) statement(s) of which the subjects are *atomic*, since the original justifications of the 'sender' are not communicable to the 'receiver'.

  Example: in a situation after the observation of the previous example, the utterance 'The yellow ball is hollow' can lead to the following extension of the person's context: $c :$ *hollow* b, provided that 'hollow' is a concept known to the person, and he is able to correctly match the definite description to the objects b and o in his context.

Hence, be it either observation or communication, the information to be added to a person's context has the form of a sequence of statements with atomic subjects, hence of the form $x : T$, where x is a variable; note that definitions do not play a role if we consider adding information from the outside.

However, as we said earlier, the types of the statements in the context give rise to a notion of meaningfulness. Only types 'constructable' from the statements already present in the context of a person are meaningful to him. This restricts the addition of statements originating from the outside.

Technically, this has the following form. Let Γ be the original context of the person and assume that the sequence $x_1 : T_1, \ldots, x_n : T_n$ is the information from the outside

(with fresh subjects $x_1, \ldots, x_n$). Then these statements are added one by one, thus changing the knowledge state incrementally. That is to say, for each $1 \leq i \leq n$, the statement $x_i : T_i$ may only be added if

$$\Gamma, x_1 : T_1, \ldots, x_{i-1} : T_{i-1} \vdash T_i : s$$

with $s \equiv \texttt{set}$ or $s \equiv \texttt{prop}$. In other words, a statement may only be added if its type is well-formed with respect to the current knowledge state. This shows, as we said before, that new information (a sequence of statements) can only be absorbed in a step-by-step fashion (statement by statement), where the possibility to append a new statement *depends on* the information available in the context at that stage, i.e. the original context *plus* the already appended statements.

This embodies precisely the notion of incrementality, discussed in subsection 3.1, which not only applies to the case of only one 'chunk' of information from the outside (i.e. one sequence of statements) as above, but also to subsequent additions of such chunks of information. For instance, if a person is in a dialogue with another person, each new utterance he receives will be added only if it is meaningful against the background of the utterances accepted before.

Remark 4.1 In treating observation and communication, we extended the use of type theory as it is traditionally described in the literature: one usually does not take into account that information can come from outside the context. When type theory is applied to knowledge representation, one usually models (the progress of) a solitary reasoning person, who can only extend his knowledge from the inside. However, since we adopted the same well-formedness criteria as usual to adding information from the outside, the resulting context in our extension will always be syntactically correct with respect to the original type-theoretical standards. Hence, this extension of the use of type theory does not lead to an extension of the formalism. (Even the complete process of adding information from the outside can be justified in type-theoretical sense. We will not go into that here.)

4.2 The problem of revision

As we saw in the previous section, a situation in which a person has to revise his knowledge state can be characterized as follows. The person is confronted with new information (which is meaningful to him), and decides to accept it. When it turns out that the incorporation of this new information leads to inconsistency of the resulting knowledge state, the person has to remove information from this new knowledge state to restore consistency. Below we describe how this can be done by means of type theory.

Revision from a type-theoretical perspective The need for revision can originate both from the inside and from the outside. We begin by describing the situation where new information is added from outside.

Suppose that the context Γ represents the person's current knowledge state (which is consistent within his horizon) and the sequence $x_1 : T_1, \ldots, x_n : T_n$ represents the new information from the outside resulting in the context $\Gamma_1 \equiv \Gamma, x_1 : T_1, \ldots, x_n : T_n$. The inconsistency of Γ_1 manifests itself in the existence of an inhabitant of falsity which the person can construct within his horizon: there is an M such that $\Gamma_1 \vdash M : \bot$.

There may be more than one such an inhabitant, but we assume that the person has chosen one of these. (We come back to this in section 5.)

The fact that all justifications are explicitly present enables the person to identify all 'suspects': the beliefs in Γ_1 that together cause the inconsistency. Since M embodies a derivation of falsity in the sense explained earlier, we find in M the justifications of all beliefs that are part of this derivation (M contains the full developmental history of the derivation). The suspect justifications occur as free variables in M, since these free variables point exactly at the premisses of the derivation of falsity: such a premiss $x : T$ gives rise to a free x in M. This is a property of the proposition as types interpretation of type theory. Moreover, the rules of type theory ensure that all free variables of M occur as subjects in Γ_1.

Example 4.2 Let $A : \texttt{prop}$ and $B : \texttt{prop}$ be statements belonging to the knowledge state (the context) and assume that the person has proofs of A, of $A \to B$ and of $\neg B$ (abbreviating $B \to \bot$, to be read as "B implies contradiction"). This is represented in the knowledge state by statements say $x : A$, $y : A \to B$ and $z : \neg B$. The rules of Type Theory enable the derivations of $\Gamma \vdash yx : B$ and $\Gamma \vdash z(yx) : \bot$. The free variables x, y and z in the 'proof object' $z(yx)$ point precisely at the propositions A, $A \to B$ and $\neg B$, which together enable the construction of the inconsistency.

Note that, given the consistency of Γ, there have to be free variables in M which occur as subjects in the *new* information $x_1 : T_1, \ldots, x_n : T_n$. (Otherwise, $M : \bot$ could already be constructed on Γ itself; this is a consequence of the Stengthening Corollary 3.5.)

New information can also originate from the inside, when a person adds a derived consequence to his knowledge state by means of a definition. This broadens his horizon and hence contradictions which were previously out of sight can now come into view (cf. section 4.1).

Example 4.3 Suppose Γ is consistent and $\Gamma \vdash N : P$ within the horizon. The result of adding $N : P$ to Γ by means of a definition is $\Gamma' \equiv \Gamma, u := N : P$. Now it is possible that there exists an M such that $\Gamma' \vdash M : \bot$ within the new horizon. As above, this M contains inhabitants of all 'suspects' as its free variables.

This shows that there is, technically speaking, no difference between revision due to information from outside and from inside. Intuitively it may seem strange that a person can be forced to revise his knowledge state by only adding a consequence of what he already knows to his knowledge state, without any external reason. However, if we take the idea of limited deductive power seriously, this is inevitable.

Restoring consistency by removing information In the above situation, when there is an M such that $\Gamma_1 \vdash M : \bot$, the person can try to regain consistency by removing one or more of the 'suspects' from Γ_1, being some of the statements $x_i : T_i$ occurring in Γ_1 where x_i occurs free in M. As we pointed out before, we assume that *the person decides* which statements he chooses to remove. Before making this choice, the person probably reconsiders the suspects, with the help of new observations or communications with others.

However, it is generally not sufficient to simply erase the chosen suspects from the knowledge state, since there may be beliefs depending on the 'suspect' beliefs. Such

a dependent belief should be removed as well, since it is no longer meaningful on the knowledge state from which the suspect(s) have been erased.

A belief can depend upon another belief in two ways:

1. A belief B may contain a free variable x which is the subject variable of a statement $x : A$ preceding $y : B$ in the context.
2. If $x : A$ precedes a definition statement $z := E : C$, both E and C may contain such a free variable x.

In these cases, $y : B$ and $z := E : C$ depend on x for their well-formedness. Hence, removal of $x : A$ from the context has consequences for these statements as well. The most natural solution is to remove them.

There is a relatively simple, syntactical procedure for removing suspect beliefs *and* the beliefs depending on them, which we describe in section 5.1 The result of this procedure is a new knowledge state, Γ_2. It is, however, not necessarily the case that this Γ_2 is consistent within the person's horizon. Although the justification M of falsity is no longer constructable on Γ_2, there may have been more than one justification for falsity on Γ_1. Some of these justifications of falsity may still be constructible on Γ_2. In that case, the person chooses one of these justifications and selects a new set of suspects on which the procedure described above is repeated. Iteration leads to a sequence of knowledge states $\Gamma_1, \ldots$ which is finite, since in every iteration step at least one of the (finite number of) justifications of falsity is removed. So there is a final knowledge state Γ_n, on which no justifications of falsity are constructable. Hence, Γ_n is consistent within the person's horizon. This Γ_n is then the resulting *revised* knowledge state.

5 Belief revision

In this section we give a formal description of the process of belief revision in type theory, as described above. First we define the syntactical procedure for removing 'suspect' beliefs and the beliefs depending on them (section 5.1) stating some properties of this removal procedure. Finally, we discuss the full revision procedure, which may involve iterative removal of suspect beliefs, and we investigate the properties of the procedure.

5.1 The removal operation

We start with a knowledge state represented by a context Γ and new information represented by the sequence $x_1 : T_1, \ldots, x_n : T_n$. We add the new knowledge to the original knowledge state, obtaining $\Gamma_1 \equiv \Gamma, x_1 : T_1, \ldots, x_n : T_n$. We assume that this 'new' context Γ_1 turns out to be inconsistent and we assume that the person has chosen one or more suspect beliefs in Γ_1 which he wants to remove. Note the assumption that the suspect beliefs can be found in the *entire* Γ_1, so also among the new information: contrary to standard accounts of belief revision we do not award a special priority to the new information (cf. section 7.3).

The removal operation that we describe below results in the transformation of Γ_1 into a new context Γ_2. However, as we discuss below, regaining consistency may involve more than one such transformation, hence in our definition we define the

transformation as leading from Γ_i to Γ_{i+1}.

In order to give a general definition of removal, we write a context as if all statements in the context were definitions: $y_1 := E_1 : T_1, \ldots, y_m := E_m : T_m$, with the convention that $y_l := E_l : T_l$ must be read as $y_l : T_l$ if it is not a definition and we take $FV(E_l) = \emptyset$ in the last mentioned case. ($FV(M)$ is the set of all variables occurring free in M.)

We assume that V is the set of variables which are the subjects of suspect beliefs $y_k := E_k : T_k$ in Γ_i which the person has chosen to remove. As we explained at the end of section 4, also beliefs $y_l := E_l : T_l$ *depending* on the variables in V must be removed. Below we characterize the set $\mathtt{dep}_\Gamma(V)$ consisting of V plus all subject variables of statements depending on V.

We start with the definition of the notion 'subcontext'.

Definition 5.1 Let $\Gamma \equiv \Delta_1, y := E : T, \Delta_2$ and $\Gamma' \equiv \Delta_1, \Delta_2$ or $\Gamma' \equiv \Delta_1, y : T, \Delta_2$. Then $\Gamma' \subset \Gamma$. The relation $\subseteq$ is the reflexive and transitive closure of $\subset$. If $\Gamma_1 \subseteq \Gamma_2$ we say that Γ_1 is a *subcontext* of Γ_2.

Next we define the dependency relation $\leq$, a partial order between subject variables of a context Γ.

Definition 5.2 Let $\Gamma \equiv \Delta_1, y := E : T, \Delta_2$. Then $\mathtt{def}_\Gamma(y) = E$, $\mathtt{type}_\Gamma(y) = T$ and $\mathtt{stat}_\Gamma(y) = (y := E : T)$. For y and $z \in \mathtt{dom}(\Gamma)$ we say that $y < z$ if $y \in \mathtt{FV}(\mathtt{def}_\Gamma(z) \cup \mathtt{type}_\Gamma(z))$.(For convenience, we write '$<$' instead of $<_\Gamma$.) The relation $\leq$ is the reflexive and transitive closure of $<$. The set $\mathtt{dep}_\Gamma(y)$ is $\{z \in \mathtt{dom}(\Gamma) | y \leq z\}$. Moreover, $\mathtt{dep}_\Gamma(V)$ is $\bigcup_{y \in V} \mathtt{dep}_\Gamma(y)$, for $V \subseteq \mathtt{dom}(\Gamma)$.

Note that the set of variables depending on a set of variables V, *includes* V itself.

Next, we define a deletion operator $\mathtt{del}$, erasing statements from a context, and the removal operator '$\backslash$'.

Definition 5.3 For domain variable y of $\Gamma \equiv \Delta_1, y := E : T, \Delta_2$, we define $\Gamma - \mathtt{stat}_\Gamma(y)$ as Δ_1, Δ_2. For a set W of domain variables of Γ, we define $\mathtt{del}_\Gamma(W)$ as $\Gamma - \bigcup_{y \in W} \mathtt{stat}_\Gamma(y)$. For a context Γ and a set $V \subseteq \mathtt{dom}(\Gamma)$, the *removal operation* '$\backslash$' is defined by $\Gamma \backslash V = \mathtt{del}_\Gamma(\mathtt{dep}_\Gamma(V))$.

So, $\Gamma \backslash V$ is the context resulting from removing all statements depending on the set V of chosen subject variables, from Γ.

As explained in section 4.1, knowledge states are incremental, in the sense that the type of each statement should be meaningful given the statements preceding it. In type theory this is expressed by *legality* given in Definition 2.23 and which satisfies the important Context Lemmma 3.6. The removal operator applied to a legal context, results in a new, legal subcontext:

Lemma 5.4 Let Γ be a context and $V \subseteq \mathtt{dom}(\Gamma)$. Then $\Gamma \backslash V \subseteq \Gamma$. Moreover, if Γ is legal, then $\Gamma \backslash V$ is legal.

Proof. For the second part: Subsequently delete all $\mathtt{stat}(y)$ for $y \in \mathtt{dep}_\Gamma(V)$ from Γ, from right to left, using Strengthening Corollary 3.5. ∎

The removal operator has the nice property that the result of subsequent applications to V and W is the same as applying it in the reverse order, or by applying it to the union of V and W:

Lemma 5.5 If Γ is legal and V and W are subsets of $\mathtt{dom}(\Gamma)$, then $(\Gamma\backslash V)\backslash W = (\Gamma\backslash W)\backslash V = \Gamma\backslash(V \cup W)$.

PROOF. By the definition of $\backslash$ and basic set theory. ∎

5.2 The revision procedure

In this section we show how the removal operator can be used to regain consistency. We assume that a person has originally a legal and *consistent* knowledge state Γ. He extends his context Γ with new information $x_1 : T_1, \ldots, x_n : T_n$, obtaining $\Gamma_1 \equiv \Gamma, x_1 : T_1, \ldots, x_n : T_n$. Let's assume that Γ_1 is legal again, but that it has become inconsistent: he can now construct an M such that $\Gamma_1 \vdash M : \bot$. (Note: in this subsection we forget about the 'horizon' of a person, i.e. the limited deduction power of a human being; we consider this horizon in the following subsection.) We consider two cases in both of which the proof M of falsity is no longer derivable on the resulting context Γ_2:

- The person chooses to remove a *single* subject variable z occurring freely in M, plus all statements depending on this z. Hence, he obtains $\Gamma_2 \equiv \Gamma_1\backslash\{z\}$ as his new context. Note that the chosen variable z may be the inhabitant of a statement in the original context Γ or of a statement $x_i := E_i : T_i$ which is part of the extension. In the latter case, $\mathtt{dep}_{\Gamma_1}(z)$ contains only variables occurring as subjects in the extension. In the former case, however, $\mathtt{dep}_{\Gamma_1}(z)$ may contain subject variables of Γ *as well as* subject variables of the extension. Hence, the removal operation may change the new information in *both* cases.
- The person chooses a non-empty *set* V of variables occurring freely in M and obtains $\Gamma_2 \equiv \Gamma_1\backslash V$ as his new context. Note that by lemma 5.5, the removal of V has the same effect as removing the separate elements of V, one by one, in *any* order. (This also holds if V is the set of *all* free variables in M.)

The above does not guarantee that Γ_2 is consistent: it may be the case that the person can still construct a proof of falsity, say M', on Γ_2. Then the person can repeat the removal operation with one or more free variables occurring in M', and so on obtaining a sequence of contexts $\Gamma_1, \Gamma_2, \ldots$, where each Γ_{i+1} is a legal subcontext of Γ being properly 'smaller' (i.e. contains fewer statements) than Γ_i. It follows that the sequence $\Gamma_1, \Gamma_2, \ldots$ is finite, so that a context Γ_n which is consistent is finally obtained. (In the extreme case $\Gamma_n = \varepsilon$, but there is no proof of falsity on the empty context ε by the consistency of type theory.) This implies:

Lemma 5.6 Iterated application of the removal operation terminates resulting in a consistent knowledge state.

In other words, it is a *revision procedure*. It is interesting to note that this iteration can be summarized in a single application of the removal operation: Let's call the non-empty set of variables that the person chooses to remove in the transition from Γ_i to Γ_{i+1}, V_i (which can be a singleton set). Then $\Gamma_{i+1} = \Gamma_i\backslash V_i$. However:

Lemma 5.7 Successively removing V_i from Γ_i for $i = 1, \ldots, n-1$, leads to the same result as removing the union of all V_is from Γ_1: I.e. $\Gamma_n = \Gamma_1\backslash\bigcup_{i=1}^{n-1} V_i$.

PROOF. This is again a consequence of lemma 2. ∎

In this section we assumed that it is the *person* who makes the decision about which statements to remove, and not the formalism. We gave arguments for this point of view in section 4. However, in comparing our system with others in the literature we will in section 7.4 discuss formal *heuristics* for making these decisions.

5.3 Revision with horizon

In the previous subsection we assumed that the person is 'omniscient' in the sense that he is able to provide a proof of falsity at any time, if there *exists* one. This, of course, is not realistic. For this reason we introduced in the beginning of section 4 the notion of 'horizon' for the person. If we look at the revision procedure, the presence of a horizon has important consequences.

Firstly, a knowledge state Γ has only a limited number of consequences within a given horizon. We formulate this as a theorem, provable by combinatorial arguments:

Theorem 5.8 Given a context Γ and a number h limiting the derivation depth of derivations on Γ ('the distance to the horizon'), there is a *finite* number of statements derivable on Γ (modulo α-conversion).

Note that we do *not* consider the full deductive closure of Γ, which possibly corresponds with an 'infinite horizon', which is no horizon at all. For convenience, we denote the finite set of derivable statements from context Γ (the set of consequences of Γ) within horizon distance h by $Conseq_h(\Gamma)$.

Corollary 5.9 Given a context Γ that is inconsistent within horizon distance h, there is a finite number of inhabitants of falsity ('proofs of falsity') (modulo α-conversion). I.e., there are finitely many terms M such that $M : \bot \in Conseq_h(\Gamma)$.

By application of the revision procedure, statements are removed from the context Γ. This will eliminate a (number of) proof(s) of falsity, but the question arises whether there are *new* proofs of falsity on the revised (smaller) context. This is not the case:

Theorem 5.10 If $\Gamma\backslash V$ is the result of revising Γ with respect to V, then there is no statement derivable within horizon distance h on $\Gamma\backslash V$ which was not already derivable within horizon distance h on Γ. I.e., $Conseq_h(\Gamma\backslash V) \subseteq Conseq_h(\Gamma)$.

PROOF. *Note that $\Gamma\backslash V \subseteq \Gamma$ by lemma 5.4. For any two PTS-contexts Δ and Δ' the so-called Thinning Lemma holds: if $\Delta' \subseteq \Delta$ and $\Delta' \vdash A : B$, then $\Delta \vdash A : B$. Hence if $\Gamma\backslash V \vdash A : B$ then $\Gamma \vdash A : B$. However, if we regard the horizon distance, it might still be possible that there exists a statement $A : B$ which is derivable on $\Gamma\backslash V$ in at most h steps, and on Γ in more than h steps (due to extra steps needed to 'retrieve' the premisses on the larger context). We assume, however, that the axiomatization of Type Theory is such that the Start-rule allows any number of Weakenings. In that case, a derivation of $\Gamma\backslash V \vdash A : B$ can always be 'copied' into a derivation of $\Gamma \vdash A : B$ with the same number of derivation steps.* ∎

Corollary 5.11 The removal procedure does not allow the introduction of new proofs of falsity.

Corollaries 5.9 and 5.11 imply the following theorem, which says that we can always reach a consistent context *in one revision step*:

Theorem 5.12 Given an inconsistent context Γ and a horizon distance h, there exists a set of variables V such that $\Gamma\backslash V$ is consistent within the same horizon distance.

PROOF. *Take V to be the set of all free variables occurring in all proofs of falsity which can be derived on Γ within horizon distance h. By Corollary 5.9, this set is finite and by the definition of the revision procedure, none of these proofs of falsity are constructable on $\Gamma\backslash V$. By Corollary 5.11, there are no new proofs of falsity on $\Gamma\backslash V$, hence $\Gamma\backslash V$ is consistent within horizon distance h.* ∎

6 Situating our approach

In this paper, we presented an approach to belief revision based on type theory. As far as we know, this approach is new. In the setting of type theory, justifications of beliefs are 'first class citizens', which is not the case in current approaches to belief revisions. In this section we discuss the relations between our approach and well-known approaches from the literature. We take [18] as our guideline.

6.1 Belief bases with justifications

By the methodological taxonomy of [18], our approach has these characteristics:

- Beliefs are represented as statements in type theory, a person's belief state as a type-theoretical context (section 4). The result of a belief change operation is again a type-theoretical context (section 5.2).
- The statements that are elements of the context representing a person's belief state, represent his *explicit* beliefs. Beliefs derivable from these statements are his implicit beliefs (section 3.2). Contrary to standard practice, we assume the deductive powers of the person are limited by a deductive horizon and only statements that are derivable within this horizon count as his implicit beliefs.
- Our theory does not prescribe how choices are made concerning what beliefs to retract. It gives a set of candidates for retraction, but leaves the actual choice to the person (Section 5.2). One can give heuristics for this choice (Section 7.4).

Gärdenfors and Rott mention four *integrity constraints* guiding the construction of belief revision formalism:

- *The beliefs in the data base should be kept consistent whenever possible.* We adhere to this constraint taking 'consistent' to mean: 'consistent with respect to the limited deductive powers of the person'.
- *If the beliefs in the data base logically entail a sentence, then this sentence should be included in the data base ('deductive closure').* It will be clear from our earlier comments (sections 4 and 5.3) that we do not subscribe to this point of view. However, it is possible to explicitly include a derived belief (to be precise: derived within the person's horizon) in the knowledge state by means of a definition (section 4.1).

- *The amount of information lost in a belief change should be kept minimal.* In accordance with the fact that our theory says nothing about extra-logical factors governing the choice of beliefs-to-be-retracted, there is no notion of minimality inherent in our theory.
- *In so far as some beliefs are considered more important or entrenched than others one should retract the least important ones.* In line with our previous comment, a notion of extra-logical preference like entrenchment should in our opinion not be part of a theory as it belongs to the realm of heuristics.

The choices we made above imply that we work with the so-called *belief bases*: the knowledge state of a person is represented by a finite set of sentences, a context Γ. The belief set of the person consists of his explicit beliefs (statements in Γ) and his implicit beliefs (statements derivable on Γ within the horizon, i.e. $Conseq_h(\Gamma)$). Note that $\Gamma \subseteq Conseq_h(\Gamma)$: every explicit belief in the context Γ is derivable on Γ, and is hence *also* implicit. Therefore we can represent a person's belief set by $Conseq_h(\Gamma)$.

Since we choose to represent justifications for beliefs explicitly, as inhabitants, in the knowledge state, our approach is closely related to what is called *Foundations Theory* in the literature, see e.g. [17].

6.2 The relation with Foundations Theory

Foundations Theory is based on the principle that belief revision should consist in giving up all beliefs that no longer have a satisfactory justification, and in adding new beliefs that have become justified. This principle has a number of consequences:

- *Disbelief propagation* If in revising a knowledge state a certain belief is retracted, not only this belief should be given up, but also all beliefs depending on this belief for their justification. Since our theory has an explicit representation of justifications, this propagation can be captured syntactically, as was shown in definition 5.2, by means of the relation $\leq$. Hence, our approach does not have the drawbacks that are often associated with disbelief propagation, viz. 'chain reactions' and 'severe bookkeeping problems'.
- *Non-circularity.* Since beliefs can depend on other beliefs for their justification, we should be careful that the dependency graph is well-founded, i.e. does not contain circularities. In our approach such circularities cannot occur, since they are ruled out by the well-formedness requirements for the type-theoretical contexts (section 4).
- *Multiple justifications.* A belief may be supported by several independent beliefs. The removal of one of those justifications does not automatically lead to giving up the belief. This characteristic is reflected in our approach, where a belief may have more than one inhabitant. Suppose that a person has two justifications for the belief that A holds on his knowledge state Γ, for example: $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$. Since the free variable sets of M and N may be disjoint, it may be possible to retract the justification M of A, while retaining N and hence the belief that A (see section 5.2).

There is a well-known problem in Foundations Theory, following from the hypothesis that all beliefs must have a justification. This induces a distinction between beliefs:

some beliefs are justified by one or more other beliefs, but there must also exist beliefs which are justified 'by themselves'. These so-called *foundational beliefs* are considered to be 'self-evident', they need no further justification.

In Foundations Theory, justification is a relation on the level of the beliefs. In type theory, however, justifications are explicitly represented by terms *inhabiting* the beliefs they justify. The distinction between foundational and other beliefs is reflected in type theory in the structure of the term inhabiting the belief:

- *Atomic justifications.* If the term inhabiting the belief is a constant or a variable, the justification cannot be further analyzed. This corresponds to the foundational beliefs, but only to a certain extent: it does not imply that these beliefs are necessarily self-evident. The atomic justification simply reflects the person's decision to adopt the belief in its own right, e.g. on the basis of an observation, communication or an act of will. (See also section 2.)
- *Composite justifications.* If the term inhabiting the belief is a composite term, the justification can be analyzed according to the structure of the term. These terms occur in the context in *definitions*, e.g. in the statement $y := E : T$, where E is a composite justification for T. One can find the inhabitants of the other beliefs supporting the belief that T, as the free variables occurring in E.

Thus the justification relation from Foundations Theory becomes a relation between inhabitants of beliefs in type theory. This relation is captured by the dependency relation $\leq$ of definition 5.2.

7 Comparing operations for belief change

Before we can compare the formal properties of our revision procedure with those of the literature, we must formulate our equivalents of the three standard belief change operations: expansion, contraction and revision.

- *Expansion: Adding a new sentence A to the belief base K, regardless of the consistency of the resulting belief base.* The result is usually denoted by $K + A$.
  In our type-theoretical setting, expansion is just addition of either a statement or a definition to the context: Γ changes into $\Gamma, x : A$ (with x fresh), or into $\Gamma, x := M : A$. In the first case new information originating from outside is added, in the second case a consequence of the belief base is made explicit by adding it to the context.
  Note that, in both cases, the type A must already be well-formed with respect to Γ, i.e. $\Gamma \vdash A : s$ with s a sort in the set of sorts $\mathcal{S}$ of the type system (cf. sections 3.2 and 4.1). In the second case, $x := M : A$ may only be added when $\Gamma \vdash M : A$ is derivable. This again gives a well-formedness guarantee.
  NOTATION: The type-theoretical analogue of Expansion will be denoted by $Exp_{x:=M:A}(\Gamma; \Gamma')$ if the expansion of Γ with the statement or definition $x := M : A$ yields Γ'. Hence, $\Gamma' \equiv \Gamma, x := M : A$.
- *Contraction: Retracting some sentence A from the belief base K, as well as sentences depending on A (without adding new beliefs).* This is denoted by $K \stackrel{\circ}{-} A$.
  In type theory, retracting has to be done with statements instead of formulas. Moreover, given a context Γ and a horizon depth h, there can be several terms

inhabiting a belief A that is to be retracted. There is a *set* of terms t such that $t : A \in Conseq_h(\Gamma)$. If we take retraction to mean that *no* statement $M : A$ should be derivable any more, we need a retraction procedure similar to the one described in section 5.2. That is, the person iteratively chooses variables occurring free in such terms t inhabiting A and removes them from Γ, in order to eliminate evidence for A.

Formally, we can say that there is a set $V_A := FV\{t|t : A \in Conseq_h(\Gamma)\}$. The variables chosen by the person together constitute a subset V of V_A (cf. Lemma 4). Retraction of A with respect to Γ then amounts to a removal $\Gamma\backslash V$ with V chosen such that $\neg\exists_t(t : A \in Conseq_h(\Gamma\backslash V))$.

Note: In its generality, this procedure always gives the desired result. There is, however, a slight complication: there are sentences which we never want to be contracted, for example tautologies. How we can prevent in type theory that this kind of sentences can be retracted, is discussed in section 7.2.

NOTATION: The type-theoretical analogue of Contraction is denoted by $Ctr_A(\Gamma;\Gamma')$, if Γ' is the result of contracting Γ with respect to A. In case $A \notin Conseq_h(\Gamma)$, we take Γ' to be Γ.

- *Revision: Adding a new sentence A to the belief base K while maintaining consistency, by (possibly) deleting a number of sentences in K.* This is denoted by $K * A$.

  In the standard account, revision is related to contraction and expansion by means of the Levi-identity: $K * A = (K \dot{-} \neg A) + A$. This implies, that for belief bases, revision can be defined as a two step procedure:

  1. *Contract by* $\neg A$ 2. *Expand by* A

  We can match this so-called internal revision [21] via the two type-theoretical operations defined above:

  1. $Ctr_{\neg A}(\Gamma;\Gamma')$ 2. $Exp_{x:=M:A}(\Gamma';\Gamma'')$

  Note that this procedure will always lead to a context (Γ'') containing the new information ($x := M : A$), whereas the procedure described in sections 5.2 and 5.3 did not, since there it was possible that (parts of) the new information were removed as well, if this information contributed to the inconsistency. In literature, this alternative approach is known as 'semi-revision'. In section 7.3 we will show that the type-theoretical version of revision developed in this paper closely resembles the semi-revision operation *consolidation* of [21]. Anticipating on this, we introduce the following.

  NOTATION: The type-theoretical analogue of Revision (i.e., Contraction by $\neg A$ and Expansion by A) is denoted by $Rev_{x:=M:A}(\Gamma;\Gamma')$, if Γ' is the result of revising Γ with respect to $x := M : A$.

Finally we note that the operations of expansion and contraction, and hence revision, described above can also be executed with new information consisting of a *sequence* of statements $(x_1 := M_1 : A_1, \ldots, x_i := M_i : A_i)$, rather than a single statement $(x := M : A)$. From a type-theoretical point of view, this is a natural generalization. Moreover, experiences obtained in formalizing the addition of outside-information (as described in section 4.1) to type-theoretical knowledge states, suggests that such information generally takes the form of a sequence of statements.

Now we have given our equivalents of the standard belief change operations, expansion, contraction and revision, we give a more detailed comparison between the two approaches in order to position our approach with respect to the literature. We concentrate on the results of Gärdenfors [18] and Hansson [21].

7.1 Expansion

In the standard approach, expansion is the set-theoretical addition of a sentence to set of propostions representing a person's belief base. In the type theoretical approach it is the addition of a statement to the context representing a person's belief base. As explained above, the type theoretical addition requires that the new statement is well-formed with respect to the existing context, which ensures that the added information is meaningfull to the person. Assuming that this the case, as is usually done, expansion behaves the same in both approaches.

7.2 Contraction

We now look at the rationality postulates for contraction as they are reformulated for belief bases in [18]. As already remarked earlier, our approach is more fine-grained than that of Gärdenfors, because we deal with specific proofs of propositions, whereas the standard approach does only considers (sets of) propositions. Hence, when Gärdenfors contracts with respect to a proposition A, from our perspective, he implicitly quantifies over *all* proofs of A. This difference also plays a role in the formulation of the postulates themselves.

In some of the Gärdenfors postulates, conditions occur of the form $\vdash A$ and $\nvdash A$. Type-theoretically, we take these to state that there *exists* respectively *doesn't exist* a proof object for the type A within the horizon. Moreover, the fact that A is or isn't a tautology, suggests that this proof object can (or cannot) be constructed on the empty context ε. However, in type theory the type A itself must be well-defined before we can think about the construction of inhabitants of A. Hence, we need some *initial context* Γ_{init} which ensures the well-definedness of all propositions: $\vdash A$ is translated into $\exists_M(\Gamma_{init} \vdash M : A)$ and $\nvdash A$ into $\neg\exists_M(\Gamma_{init} \vdash M : A)$.

Of course, statements in the initial context should not be contracted in a revision process, since this initial context acts as a kind of 'axiom base' for the well-definedness of the propositions. The above contraction procedure $Ctr_A(\Gamma; \Gamma')$, will not consider variables inside Γ_{init}, since the statements of Γ_{init} are at the wrong level of typing to have their subjects appear in terms inhabiting propositions (cf. section 2).

Note that if A is a tautology, there exists a proof object in which no free variables occur: $\exists_M(\Gamma_{init} \vdash M : A)$ where $V_A = \emptyset$. Since M cannot be blocked by removing variables in V_A from the context, we cannot contract over tautologies. On the one hand this is a good thing: one does not want to lose tautologies. On the other hand, this has as a consequence that Contraction becomes a *partial* operation, which may be unsuccessful!

Below we present the Gärdenfors postulates for belief bases as given in [18], followed by their type-theoretical translation and a discussion of their validity. The original postulates quantify over all sentences A and belief sets H, their translations over all

types A and contexts Γ (where $\Gamma \supseteq \Gamma_{init}$). In addition, the postulates are stated using $Cn(H)$, the deductively closed set of consequences of H (i.e. with infinite horizon). In the translation of e.g. Gärdenfors's ($H \dot{-} 3$)-postulate we take $A \notin Cn(H)$ to mean that there exists no proof object of type A (within the horizon) on the person's context, $\neg\exists_N(N : A \in Conseq_h(\Gamma))$.

($H \dot{-} 1$) $H \dot{-} A$ is a belief set.

Its translation is:

If $Ctr_A(\Gamma;\Gamma')$, then Γ' is a well-formed context.

This holds: Assume $Ctr_A(\Gamma;\Gamma')$, then there exists some set $V \subseteq V_A$, possibly empty, such that $\Gamma \backslash V \equiv \Gamma'$. By Lemma 1, Γ' is a well-formed context.

($H \dot{-} 2$) $H \dot{-} A \subseteq H$.

Its translation is:

If $Ctr_A(\Gamma;\Gamma')$, then $\Gamma' \subseteq \Gamma$.

This follows from the definition of the removal-operation (definition 5.2).

($H \dot{-} 3$) If $A \notin Cn(H)$, then $H \dot{-} A = H$.

Its translation is:

If $\neg\exists_N(N : A \in Conseq_h(\Gamma))$ and $Ctr_A(\Gamma;\Gamma')$, then $\Gamma \equiv \Gamma'$.

This holds: Assume $\neg\exists_N(N : A \in Conseq_h(\Gamma))$ and $Ctr_A(\Gamma;\Gamma')$ and suppose $\Gamma \not\equiv \Gamma'$. Then (see $H \dot{-} 2$) Γ' is a proper subcontext of Γ. Hence there is some variable z occurring in Γ as a subject, such that $z \in V$, where V is the set of variables chosen to be removed and $z \in V$ not in Γ'. Hence z must have occurred free in some term N such that $\Gamma \vdash N : A$ within the horizon, but then $\exists_N(N : A \in Conseq_h(\Gamma))$. Contradiction.

($H \dot{-} 4$) If $\not\vdash A$, then $A \notin Cn(H \dot{-} A)$.

Its translation is:

If $Ctr_A(\Gamma;\Gamma')$, then $\neg\exists_M(M : A \in Conseq_h(\Gamma'))$.

This postulate holds by our definition of contraction.

Note that the condition $\not\vdash A$ ('A' is not a tautology) is implicitly present in our translation, because this is implied by the condition $Ctr_A(\Gamma;\Gamma')$. In fact, if A *is* a tautology, then A has a proof object, but this proof object has no free variables. Therefore the set V_A is empty and hence Contraction of A as described before is not possible (there is no Γ' such that $Ctr_A(\Gamma;\Gamma')$).

($H \dot{-} 5$) $H \subseteq (H \dot{-} A) + A$.

Its translation is:

If $Ctr_A(\Gamma;\Gamma')$, then $\Gamma \subseteq \Gamma', z : A$.

Note that we have to add a proof object z for A. We could not use a definition $z := M : A$, since this implies that $\Gamma' \vdash M : A$ for some M, which contradicts $Ctr_A(\Gamma;\Gamma')$.

This postulate, which has a controversial status in the literature (in fact: base contractions generally violate it), does not hold here. A simple counterexample is the following: Take $\Gamma \equiv \Gamma_{init}, x : B \to A, y : B \vdash xy : A$, then $Ctr_A(\Gamma,\Gamma')$, where $\Gamma' \equiv \Gamma_{init}$, but $\Gamma \not\subseteq \Gamma_{init}, z : A$.

($H \dot{-} 6$) If $\vdash A \Leftrightarrow B$, then $H \dot{-} A = H \dot{-} B$.

Its translation is:

If $\exists_N(\Gamma_{init} \vdash N : A \Leftrightarrow B)$ and $Ctr_A(\Gamma;\Gamma')$ and $Ctr_B(\Gamma;\Gamma'')$, then $\Gamma' \equiv \Gamma''$.

This postulate does not hold in general, but there is a case in which it holds, as we explain below.

First, observe that in type theory we have to do work to transform proofs of A into proofs of B (and vice versa) by means of the proof N of the equivalence of A and B which contains subproofs N_1 for $A \to B$ and N_2 for $B \to A$. Then for example: If $\Gamma \vdash M : A$ for some M, then $\Gamma \vdash N_1 M : B$ (and vice versa).

We call M a *direct* proof of A and $N_1 M$ an *indirect* proof of B. Note that transforming a direct proof of A into an indirect proof of B involves one extra proof step. Hence, this can lead to a situation in which the direct proof is within the horizon, whereas the indirect proof is not.

Disregarding this horizon problem, the postulate still does not hold in general: in order to block all proofs of B, all proofs of A also have to be blocked. Hence, a set V will have to be chosen which is a subset of the union of the variables occurring free in all proofs of A *and* all proofs of B, i.e., $V \subseteq (V_A \cup V_B)$. However, it might still be possible to find different subsets V_1 and V_2 which both block all proofs of A and B.

Example: $\Gamma \equiv \Gamma_{init}, x : C \to A, y : C, z : D \to B, u : D$, and $\Gamma \vdash N : A \Leftrightarrow B$. Then $V_A = V_B = \{x, y, z, u\}$. Now take $V_1 = \{x, z\}$ and $V_2 = \{y, u\}$. It is easy to check that both V_1 and V_2 block all proofs of A and B. If we take $\Gamma' \equiv \Gamma \backslash V_1$ and $\Gamma'' \equiv \Gamma \backslash V_2$, then $Ctr_A(\Gamma;\Gamma')$ and $Ctr_B(\Gamma;\Gamma'')$, but $\Gamma' \not\equiv \Gamma''$.

However, the postulate *does* hold if we use the 'safe contraction' described in section 7.4, i.e. take $V_1 = V_2 = V_A = V_B$, then $\Gamma' \equiv \Gamma''$.

Here we end our discussion of the basic postulates $H\dot{-}1$ to $H\dot{-}6$ for base contraction. There exist two more (non-basic) postulates, $H\dot{-}7$ and $H\dot{-}8$, concerning conjunctive formulas $A \wedge B$. We do not discuss those here for two reasons: as remarked above, the type-theoretical notion of contraction can easily be generalized to a sequence of statements, so that there is no need to give a special status to the $\wedge$-connective; moreover, it would require us to go into the technical details of coding conjunction in type theory, which does not serve the purpose of this paper.

Concluding, as in most approaches to base revision in the literature, postulates $H\dot{-}1$ through $H\dot{-}4$ are satisfied in the type-theoretical translation, but $H\dot{-}5$ does not hold. In addition, the type-theoretical equivalent of 'safe contraction' satisfies $H\dot{-}6$. This exactly reflects Theorem 5.4.1 of [18].

7.3 Revision

In the standard account of revising a belief base K with new information A, the new information is always accepted and beliefs in K are abandoned to maintain consistency. Objections have been raised to this account, on the grounds that too much priority is given to new information [21]: at each stage, new information is completely trusted. However, this complete trust is only temporary: once the new information is incorporated in the belief base, it is itself susceptible to abandonment when in the next stage even newer information becomes available. This seems awkward.

We agree with these objections. Moreover, this emphasis on 'new information' has a number of additional undesired consequences from our point of view. Firstly, the

new information always has to be accepted as a whole, whereas in our approach it is a possible outcome of revision that the person accepts only part of the new information. The standard account is also too absolute in another respect: because of the unlimited deductive power assumed in this approach, the person can detect beforehand whether a piece of new information is inconsistent with his current belief base, and hence whether revision should be carried out. Under the more realistic assumption of the deductive horizon, it is not possible to do this consistency check once and for all: inconsistencies, and hence the need for revision, may arise as proofs of falsity turn up inside the horizon. Finally, thinking of standard belief revision in the setting of communication, a person would be forced to accept every utterance by his dialogue partner(s), even if accommodating this information requires a major reconstruction of his own belief base. Therefore, new information and information in the belief base should be treated equally by the revision operation.

Revision procedures which do not necessarily accept the new information are known in literature as *non-prioritized* revision procedures. Hansson was one of the first to consider this kind of belief revision [20], and in recent years a number of different non-prioritized approaches have been developed, see [22]. For belief bases, a non-prioritized form of revision called *semi-revision* can be specified as a two-stage procedure [21]:

1. *Expand by A*
2. *Make the belief base consistent by deleting either A or some original belief(s)*

Compared to the revision procedure formulated at the beginning of section 7, the order of the steps is reversed[4] and the second step has been modified. The operation performed in the second stage is called *consolidation*, [21], and can be carried out by contracting over falsehood. In our approach, the procedure looks like:

1. $Exp_{x:=M:A}(\Gamma;\Gamma')$ 2. $Ctr_{\perp}(\Gamma';\Gamma'')$

In other words, revision and contraction are related by the following identity:

$$Rev_{x:=M:A}(\Gamma;\Gamma') = Ctr_{\perp}(\Gamma, x := M : A;\Gamma')$$

This is exactly the revision procedure described earlier in sections 5.2 and 5.3. First the new information, one or more statements, is added to the context Γ, then a number of statements from the expanded context is removed to block the construction of inhabitants of falsity.

There is a close resemblance between our revision procedure and that of [21], called *kernel consolidation*. This correspondence is given in Appendix A of this paper.

7.4 Heuristics

What we have done so far does not add up to a theory of belief revision in the traditional sense. We have shown how a person can find the suspect beliefs when his belief state has become inconsistent, and how he can remove a number of the suspects to regain consistency, but our revision procedure does not tell the person *which* suspects to remove. Standard approaches have a parametric selection mechanism which embodies some notion of "rational choice" between the various possibilities for revision in any given situation. Given a value for their parameters they select one "optimal"

[4] Reversing the order alone yields *external revision*, [21]

revision outcome. They usually introduce extra-logical structure in the belief state, and are computationally unwieldy. The underlying view is that of a solitary reasoner who has to solve the inconsistency in splendid isolation, using his infinite reasoning powers and looking only at the beliefs in his (infinite) belief state. Only recently, papers have started to appear that question some of these idealizations, and in which belief change operations are defined for resource-bounded agents, see e.g. [10]. Our concern is with agents who have finite belief states (including justifications), finite computational resources, and who have access to the world by means of observation and communication. Such agents have possibilities to (re)evaluate the various suspects, by performing observations/tests or by communicating with other agents, and a theory of belief revision cannot and should not prescribe how they make their choices. Strategies used by an agent to make these choices are not part of the theory, if they can be captured formally they could be used as heuristics on top of the theory. In this section, we briefly discuss how some selection mechanisms from standard approaches mentioned in [18] fit into our account as heuristic principles.

In so-called *(partial) meet contraction*, the idea is that the optimal contraction or revision is the one that requires the smallest number of insertions and/or deletions in the belief state. These contraction operations were originally defined for deductively closed belief sets, rather than belief bases. They start from the maximal subsets of a belief set that do not imply the proposition that is to be removed. In general, there can be quite a few of these. Picking an arbitrary maximal non-implying set as the result of the operation (choice contraction), will often yield a new belief set that is too large. In meet contraction, an intersection of maximal non-implying sets is taken, to obtain a new belief set based on the beliefs the non-implying sets have in common. Taking the intersection of *all* maximal non-implying sets (full meet contraction) can result in an empty set. Alchourrón, Gärdenfors, and Makinson introduced partial meet contraction [2] in which a selection function picks out a class of "best" or "most interesting" maximal non-implying subsets. These selected sets are then intersected to obtain the new belief set. (For a fresh look at (full) meet contraction for belief bases, see [14]).

The minimality criterion can be applied in the type theoretical approach. Given one particular proof of inconsistency, $\Gamma \vdash M : \bot$, removing any one of the statements of which the subjects occur free in M is sufficient to block this particular proof. However, these statements may have different numbers of statements depending on them in Γ, and so one could prefer to remove the statement with the least number of dependents to minimise the deletions from the belief state. In cases where more than one proof of falsity has to be blocked, a "blocking" subset has to be chosen from the set of all variables occurring free in these proofs. When there is more than one subset that does the job, one could again prefer the subset with the smallest number of statements (possibly taking the number of dependent statements into account).

As in the standard approach, this criterion will not always yield a single optimal solution. It is possible to end up with two or more minimal sets of statements whose removal will restore consistency. To overcome this indeterminism, additional structure is introduced in the belief state. The central idea in this construction is known as *epistemic entrenchement*: "not all sentences that are believed to be true are equal value for planning of problem-solving purposes, but certain pieces of knowledge and beliefs about world are more important than others when planning future actions

conducting scientific investigations or reasoning in general" [17]. In performing contraction or revision, the beliefs that are given up should be the ones with the lowest degree of epistemic entrenchement. Although in our opinion such an ordering of epistemic entrenchement of the beliefs in the belief state cannot be given once and forall independent of the current goals and activities of the agent performing the contraction or revision, such an ordering could in principle be added to the context representing the agent's belief state. Note that the imposed entrenchment ordering has to respect the dependency relations between the beliefs in the context: if a belief $y := N : B$ depends on a belief $x := M : A$, then $y := N : B$ should not be epistemically more entrenched than $x := M : A$ since removing $x := M : A$ without removing $y := N : B$ will result in a context which is not well-formed.

Another idea that can be applied, at least in spirit, in the type theoretical setting is that of *safe contraction*: a proposition B is safe with respect to a proposition A if it cannot be blamed for the derivability of A. To contract over A, all propositions that are not safe with respect to A have to be removed. This approach, introduced by Alchourrón and Makinson [3], starts from the so-called "entailment sets": minimal subsets of the belief state that entail the proposition to be removed. An element B of the belief state is said to be safe with respect to the proposition A if B is not a minimal element of any entailment set of A. The minimality is determined with respect to an acyclic ordering of the beliefs in the belief state, expressed by means of a relation '$<$'. This ordering can be seen as a form of the epistemic entrenchment described above, with $A < B$ meaning that A is "less secure, plausible or reliable" than B.

There is an obvious way to translate this idea to our approach to revision: a belief $x := M : A$ is safe if it cannot be blamed for the fact that a proof object for $\bot$ can be constructed on the belief state Γ. The simplest interpretation of "being to blame" for a statement in context would be "to have its subject appear as a free variable in a proof object for $\bot$". Hence the simplest form of safe contraction would be to remove all statements of which the subjects appear free in a proof object for $\bot$ and their dependents from the context. However, this does not suffice if all statements that are removed themselves depend upon earlier statements in context, since the proof object for $\bot$ could be rebuilt from these "ancestors". One way around this problem, is to use the construction of a so-called kernel set described in the Appendix. For a given derivation horizon and a given context, this construction inductively builds the set of minimal falsity implying subsets of statements in Γ. This kernel set can reasonably be said to contain all statements that are "to blame" for the inconsistency of the context (within the horizon), hence we can define safe contraction as the removal of all these statements and their dependents. Although this will yield a unique solution, it will usually not be minimal in terms of the number of statements that are removed.

8 Concluding remarks

Since its birth in 1903, type theory has proved to be a useful medium for the design and implementation of deductive systems, programming languages and theorem provers. This paper explored the use of type theory to provide a deductive approach to belief revision which can be easily implemented. The starting idea is that type theory enables explicit representations of justifications in belief revision. With the represen-

tation of beliefs as type theoretical statements and belief states as type theoretical contexts, we showed that the presence of justifications makes it easy to identify the beliefs that cause inconsistency of the belief state (section 4.2). Their presence also greatly simplifies the handling of dependencies between beliefs (section 5.1). With respect to literature, our initial assumptions put us in the area of foundations theory for belief bases. However, our account does not suffer from the drawbacks usually associated with foundations theory such as problems with disbelief propagation, circular justifications, and multiple justifications for the same belief (section 6.2). The operation of belief revision that naturally arises from our approach is one of non-prioritized revision: new information is not automatically completely trusted (section 7.3).

The fact that our approach is deductive, and that we do not require that our theory of belief revision itself selects which beliefs have to be removed, makes its applicable to agents with limited computational resources (see appendix). This holds independently of the strength of the logic in which the belief change operations are cast: the mechanisms that were used to represent justifications and dependency relations between beliefs are at the heart of type theory, making our approach applicable to a large family of type systems. Given the well established connections between type theory and logic, this means it is applicable in a wide range of logics. For instance, it can be applied in each of the Pure Type Systems from the well-known Logic Cube [4], which corresponds to logics ranging from minimal propositional logic to higher order predicate logic. Our immediate goal is to use the extensive research in implementations of logics based on type theory in order to provide a working automated system of belief revision based on the approach of this paper.

References

[1] Ahn, R., Borghuis, T., Communication Modelling and Context-Dependent Interpretation: an Integrated Approach. In: *Types for Proofs and Programs: International Workshop, TYPES'98.* Selected Papers, Altenkirch, T., Naraschewski, W., Reus, B. (eds.), LNCS 1657, Springer Verlag (1999), pp. 19 – 32.

[2] Alchourrón, C.E. Gärdenfors, P., and Makinson, D., On the logic of theory change: partial meet contraction and revision functions, *Journal of Symbolic Logic* 50 (1985), pp. 510-530.

[3] Alchourrón, and Makinson, D., On the logic of theory change: safe contraction, *Studia Logica* XLIV (1985), pp. 405-422.

[4] Barendregt, H., Lambda calculi with types. In *Handbook of Logic in Computer Science*, Abramsky, Gabbay and Maibaum (eds.), Oxford University Press, Oxford (1992), pp. 117 – 309.

[5] Barras, B., et al., The Coq Proof Assistant Reference Manual – Version V6.1, Technical report 0203, INRIA, 1997.

[6] J. van Benthem, Reflections on epistemic logic, *Logique & Analyse* 133-134 (1991), pp. 5-14.

[7] Berardi, S., Towards a mathematical analysis of the Coquand-Huet calculus of constructions and the other systems in Barendregt's cube. Technical report, Dept. of Computer Science, Carnegie-Mellon University and Dipartimento Matematica, Universita di Torino, 1988.

[8] Bruijn, N.G. de, AUTOMATH, a language for mathematics, Technical report 68-WSK-05, Technische Universiteit Eindhoven, 1968.

[9] Bunt, H., Ahn, R., Beun, R-J., Borghuis, T., and Van Overveld, K., Multimodal Cooperation with the DenK System. In: *Multimodal Human-Computer Interaction*, Bunt, H., Beun, R-J., Borghuis, T. (eds.), Lecture Notes in Artificial Intelligence 1374, Springer Verlag (1998), pp. 39 – 67.

[10] Chopra, S., Parikh, R. and Wasserman, R., Approximate belief revision, *Logic Journal of the IGPL*, Vol. 9 No. 6 (2001), pp. 755-768.

[11] Church, A., A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

[12] Coquand, T., and Huet, G., The calculus of constructions. *Information and Computation* 76, 95-120, 1988.

[13] Curry, H.B. and Feys, R., *Combinatory Logic I*, Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam, 1958.

[14] Freund, M., Full meet revision on stratified bases, *Theoria* 67 (2001) nr. 3, pp. 189-213.

[15] Gabbay, D., *Labelled Deductive Systems*. Oxford University Press.

[16] Gabbay, D., Hunter, A., Making inconsistency respectable 1: a logical framework for inconsistency in reasoning. In: *Foundations of AI Research*, Ph. Jarrand and J. Keteman (Eds.) LNCS 535, pp. 19-32 (1991).

[17] Gärdenfors, P., The dynamics of belief systems: Foundations versus coherence theories, *Revue Internationale de Philosophie*, 44 (1990), pp. 24 – 46.

[18] Gärdenfors, P., Roth, H., Belief Revision. In: *Handbook of Logic in Artificial Intelligence and Logic Programming*, Volume 4: Epistemic and Temporal Reasoning, Gabbay, D., Hogger, C., Robinson, J. (eds.), Clarendon Press (1995), pp. 36 – 119.

[19] Hansson, S., In defense of base contraction, *Synthese* 91 (1992), pp. 239 – 245.

[20] Hansson, S., Taking belief bases seriously. In: *Logic and the Philosophy of Science in Uppsala*, Prawitz, D. and Westerståhl, D. (eds.), Kluwer Academic Publishers (1994), pp. 13 – 28.

[21] Hansson, S., Semi-revision, *Journal of Applied Non-Classical Logics*, Volume 7, nr. 1-2 (1997), pp. 151 – 175.

[22] Hansson, S.O. (Ed.), *Special issue on non-prioritized belief revision*, *Theoria* 53 (1997) Part 1-2, pp. 1-134.

[23] Harper, R. and Honsell, F. and Plotkin, G., A framework for defining logics, IEEE Proceedings Second Symposium on Logic in Computer Science, pages 194–204, 1987.

[24] Heyting, A., *Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie*. Springer Verlag, Berlin, 1934.

[25] Howard, W.A., The formulas-as-types notion of construction, In To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, editors Seldin, J.P. and Hindley, J.R., Pages 479–490, 1980. Academic Press, New York.

[26] Kamareddine, F., Bloo, R., and Nederpelt, R.P., On π-conversion in the λ-cube and the combination with abbreviations. *Annals of Pure and Applied Logics*, 97:27–45, 1999.

[27] F. Kamareddine and R.P. Nederpelt. Canonical typing and Π-conversion in the Barendregt Cube. *Journal of Functional Programming*, 6(2):245–267, 1996.

[28] Kolmogorov, A.N., Zur Deutung der Intuitionistischen Logik, *Mathematisches Zeitschrift*, Volme 35, Pages 58–65, 1932.

[29] Laan, T., *The Evolution of Type Theory in Logic and Mathematics*, PhD thesis, Technische Universiteit Eindhoven, 1997.

[30] Milner, M., Tofte, M., and Harper, R., *The Definition of Standard ML*. MIT Press, 1991, Cambridge, MA.

[31] Nederpelt, R.P., Geuvers, J.H., De Vrijer, R.C. (Eds.), *Selected Papers on Automath*, Studies in Logic and the Foundations of Mathematics 133, North-Holland, Amsterdam (1994).

[32] Tait, W.W., Infinitely long terms of transfinite type, In *Formal Systems and Recursive Functions*, Crossley, J.N. and Dummett, M.A.E., (editors), North-Holland, Amsterdam, 1965.

[33] J. Terlouw. Een nadere bewijstheoretische analyse van GSTT's. Technical report, Department of Computer Science, University of Nijmegen, 1989.

Appendix

A Kernel consolidation

Our revision procedure is particularly close to what Hansson calls *kernel consolidation*. This form of consolidation is based on the idea that a subset of sentences in the knowledge base K implies falsity if and only if this subset contains some minimal falsity-implying subset of K. Hence the consistency

of K can be restored by removing at least one element of each minimal falsity-implying subset of K. Minimal falsity-implying subsets are called *kernels*, they are defined as follows.

Definition A.1 A subset X of sentences from a belief base K is a *kernel* if:
1. $X \subseteq K$
2. $\bot \in Cn(X)$, and
3. If $Y \subset X$, then $\bot \notin Cn(Y)$

The set of all kernels of K is called the *kernel set*, denoted by $K \coprod \bot$.

The sentences of K that have to be discarded to restore consistency, are selected by an *incision function*:

Definition A.2 An *incision function* σ for K is a function such that:
1. $\sigma(K \coprod \bot) \subseteq \cup(K \coprod \bot)$
2. If $X \in (K \coprod \bot)$, then $X \cap \sigma(K \coprod \bot) \neq \emptyset$

Definition A.3 Let σ be an incision function for K. The *kernel consolidation* $\approx_\sigma$ for K is defined as follows:

$$K \approx_\sigma \bot = K \backslash \sigma(K \coprod \bot)$$

In the typetheoretical approach, falsity-implying subsets of the context Γ are sets of statements of which the subjects occur free in a proof object inhabiting $\bot$, i.e. $\{\mathtt{stat}_\Gamma(y) | y \in FV(M)\}$, where M is a term such that $\Gamma \vdash M : \bot$. If we call this set of statements for a given proof object M 'S^M' ('suspects' in M), we can see that this set fulfils the first two criteria for kernels given in Definition A.1:
1. $S^M \subseteq \Gamma$
2. $\Gamma_{init}, S^M \vdash M : \bot$, that is: $\bot$ is a consequence of S^M (where Γ_{init} contains the well-typedness information needed for the derivation)

However, such a falsity-implying subset S^M is not necessarily minimal in the sense required for kernels (the third criterion): there may exist another proof object N such that $\Gamma \vdash N : \bot$ and $S^N \subset S^M$. This is due to the fact that proof objects code an entire proof for the proposition represented by their type, including proofs that contain 'detours', sequences of steps that could have been omitted in the proof. Such detours can invoke premises that are not really needed to prove the proposition, resulting in non-minimal subsets. A very simple example of this is the following: take $\Gamma \equiv \Gamma_{init}, x : A, z : A \to A, y : A \to \bot$, then there are at least two proof objects inhabiting falsity, $\Gamma \vdash y(zx) : \bot$ and $\Gamma \vdash yx : \bot$. Clearly, the falsity-implying subset for the first proof object is not minimal, the second proof object is constructed without using $z : A \to A$. Although in typed λ-calculus some detours can be eliminated by performing reductions on proof objects[5], we cannot in general prevent a person from having a belief state on which non-minimal proofs of falsity can be derived.

Moreover, in discussing the minimality of falsity-implying subsets, the limited deductive powers have to be taken into account. Since the person can only construct proofs of $\leq h$ steps, where h is the horizon distance, we can at best talk about falsity-implying subsets which are minimal with respect to these proofs. Given a subset S^M for some inhabitant M of falsity, there may exist a set S^N such that $S^N \subset S^M$ where the proof object N for falsity cannot be constructed within the horizon h. Hence, this smaller set S^N should not be considered by the selection procedure.

The assumption of horizon enables an inductive procedure for the constructing the kernel set $\Gamma \coprod^h \bot$, the set of all minimal falsity-implying subsets within the horizon. For a given context Γ, one systematically generates all derivations of length zero,then all derivations of length 1, then all derivations of length 2, ..., up to all derivations of length h. Among each layer of derivations, one picks out all derivations of an inhabitant of falsity. By comparing the sets of free variables of these inhabitants, the minimal falsity-implying subsets for that layer can be found, i.e. for the i-th layer ($1 \leq i \leq h$) all $FV(M)$ such that $\Gamma \vdash^i M : \bot$, and there is no N such that $\Gamma \vdash^i N : \bot$ and

[5] Sometimes a term representing a non-minimal proof can be β-reduced to a minimal one, since β- reduction corresponds to cut elimination: take $\Gamma \equiv \Gamma_{init}, x : A, y : B, z : A \to \bot$, and $M \equiv (((\lambda u : A.(\lambda v : B.u))x)y)z : \bot$, then the $\bot$-implying subset S^M is $\{x : A, y : B, z : A \to \bot\}$. After performing β-reduction twice, we find the normal form of M which is xz. Now $\{x : A, z : A \to \bot\}$ is a minimal $\bot$-implying subset.

$FV(N) \subset FV(M)$. The sets S^M that are minimal for a layer are then added to the kernel set $\Gamma \coprod^i \bot$ if there is no S^N already in $\Gamma \coprod^i \bot$ such that $S^N \subset S^M$. In other words, before adding the sets that are minimal in a layer it is a checked whether they are also minimal with respect to sets from previous layers.

Given the inductively constructed kernel set $\Gamma \coprod^h \bot$, the type theoretical analogons of *incision function* and *kernel consolidation* can be defined exctly as given in Definitions A.2 and A.3, but for the replacement of $K \coprod \bot$ by $\Gamma \coprod^h \bot$. Note that in the newly attained definition the slash in $\Gamma \backslash \sigma(\Gamma \coprod^h \bot)$ stands for the type theoretical removal operation defined in section 5.1, rather than the standard set theoretical operation in definition A.2, i.e. not only the statements selected by the incision function ($\sigma(\Gamma \coprod^h \bot)$) are removed from Γ but also all statements depending on them ($\mathtt{dep}_\Gamma(\sigma(\Gamma \coprod^h \bot))$). Since dependencies are not considered in the setting of Hansson, we need to be able to distinguish between those two kinds of statements. The notion of 'independence' can easily be defined as follows:

Definition A.4 A statement $x := M : A$ is an *independent member* of the set of statements Δ iff there is no statement $z := N : B \in \Delta$ such that $x \in \mathtt{dep}_\Delta(z)$.

In [21], kernel consolation is characterised by a theorem linking its construction to a number of postulates. We restate this theorem for type theoretical knowledge states:

Theorem A.5 An operation $>$ defined on type-theoretical knowledge states is an operation of kernel consolation iff for all contexts Γ:

1. $(\Gamma >)$ is consistent *(consistency)*
2. $(\Gamma >) \subseteq \Gamma$ *(inclusion)*
3. If $x := M : A$ is an independent member of $\Gamma - (\Gamma >)$, then there is Γ' such that $\Gamma' \subseteq \Gamma$, Γ' is consistent and $\Gamma', x := M : A$ is inconsistent *(core-retainment)*.

PROOF. As x is independent, the proof is analogous to that of Hansson. There are two cases in the proof where the independence is needed to ensure that a statement is an element of $\sigma(\Gamma \coprod^h \bot)$ rather than merely an element of $\mathtt{dep}_\Gamma(\sigma(\Gamma \coprod^h \bot))$: in proving core-retainment in the direction *from construction to postulates*, and in proving that σ is an incision function in the direction *from postulates to construction*. ∎

Received 10 February 2002

A Simple CPS Transformation of Control-Flow Information

Daniel Damian, *LION Bioscience Ltd. Compass House, 80-82 Newmarket Road, Cambridge CB5 8DZ, United Kingdom.*
E-mail: Daniel.Damian@uk.lionbioscience.com

Olivier Danvy, *BRICS*[1]*, Department of Computer Science, University of Aarhus, Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark.*
E-mail: danvy@brics.dk

Abstract

We build on Danvy and Nielsen's first-order program transformation into continuation-passing style (CPS) to design a new CPS transformation of flow information that is simpler and more efficient than what has been presented in previous work. The key to simplicity and efficiency is that our CPS transformation constructs the flow information in one go, instead of first computing an intermediate result and then exploiting it to construct the flow information.

More precisely, we show how to compute control-flow information for CPS-transformed programs from control-flow information for direct-style programs and vice-versa. As a corollary, we confirm that CPS transformation has no effect on the control-flow information obtained by constraint-based control-flow analysis. The transformation has immediate applications in assessing the effect of the CPS transformation over other analyses such as, for instance, binding-time analysis.

Keywords: Program analysis. Control-flow analysis. Constraints. Continuations. Continuation-passing style (CPS). CPS transformation. Administrative reductions. One-pass CPS transformation.

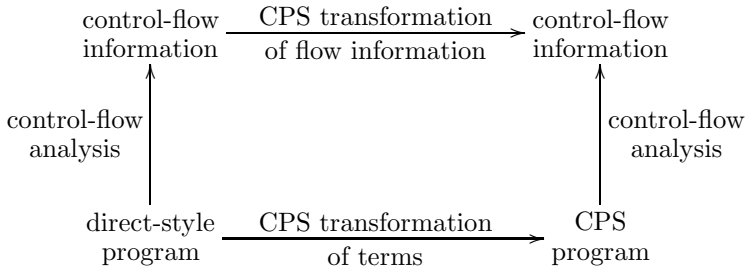
1 Introduction

The continuation-passing-style (CPS) transformation is a source-to-source program transformation of λ-terms that makes explicit the continuation of each λ-expression [36, 44]. Continuations have been discovered in many contexts [37] and form an active area of research [10, 39] with many applications, e.g., in compiler construction [1, 24, 43], program transformation [46], partial evaluation [19, 25], multi-processing [2, 15, 49], and, recently, goal-directed evaluation [12] and program security [50].

The call-by-value and call-by-name CPS transformations are due to Plotkin [36] and yield λ-terms that are independent on the order of evaluation. The CPS transformation has been extended to types [26, 47], which has led to the discovery of its logical content [17, 28]. Over the last two years, both Palsberg and Wand [35] and Damian and Danvy [6, 7, 9] have developed a CPS transformation of control-flow information. They have used it to show that a CPS transformation does not affect the control-flow information collected by a monovariant constraint-based control-flow analysis.

[1] Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

Graphically:



The canonical motivation for transferring the result of a program analysis across a program transformation is that the transfer is likely to be cheaper than analyzing the transformed program. In the present case, (1) the time complexity of control-flow analysis is cubic in the size of the analyzed program and (2) the time complexity of CPS-transforming control-flow information is linear in the size of the control-flow information, which is again linear in the size of the analyzed program.

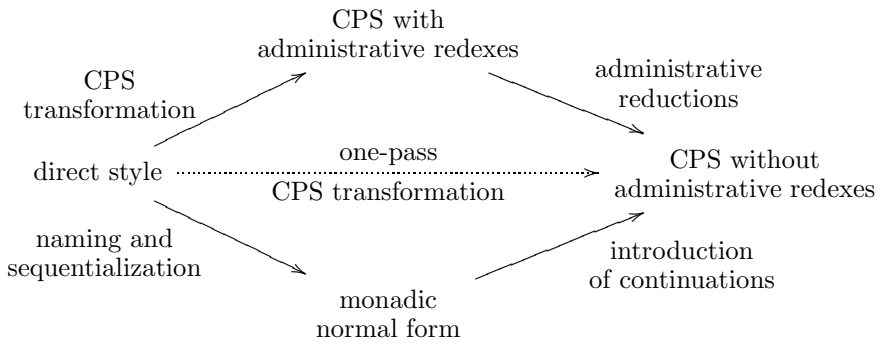
CPS transformations of flow information are based on CPS transformations of terms.

1.1 CPS transformation of terms

The CPS transformation has motivated a long line of research. Plotkin [36] and Steele [43] observed that it gives rise to large residual terms, due to so-called administrative redexes. Both theoretically and practically, these administrative redexes are in the way. For example, in his proof, Plotkin needs to interleave administrative and essential reductions. Yet a practically useful CPS-transformed program need not contain these redexes, and indeed, in his compiler, Steele performs all administrative reductions immediately after the CPS transformation. As an alternative to administrative post-reduction, compact CPS programs can also be obtained by first bringing the source program into monadic normal form and then introducing continuations [18].

Administrative redexes may be avoided altogether by using a one-pass CPS transformation. Existing one-pass CPS transformations use a higher-order accumulator [1, 11, 48] or are based on evaluation contexts [40, 42].

Graphically:



A one-pass CPS transformation makes it possible to reason directly over CPS-transformed terms. Unfortunately, existing one-pass CPS transformations are not immediate to use, either because they are higher-order or because they are not compositional. A higher-order accumulator requires a logical relation [13]. A non-compositional transformation requires well-founded induction rather than ordinary structural induction [40]. Fortunately, Danvy and Nielsen have recently discovered a one-pass CPS transformation that is both first-order and compositional [14, 30].

1.2 CPS transformation of flow information

In our initial work [7], we considered only one step of the CPS transformation, namely the introduction of continuations on terms in monadic normal form. We then turned to transforming source terms into monadic normal form [6, 9].

In a related work [35], Palsberg and Wand considered the first phase of the CPS transformation. In a followup work [6, 8], we addressed administrative reductions.

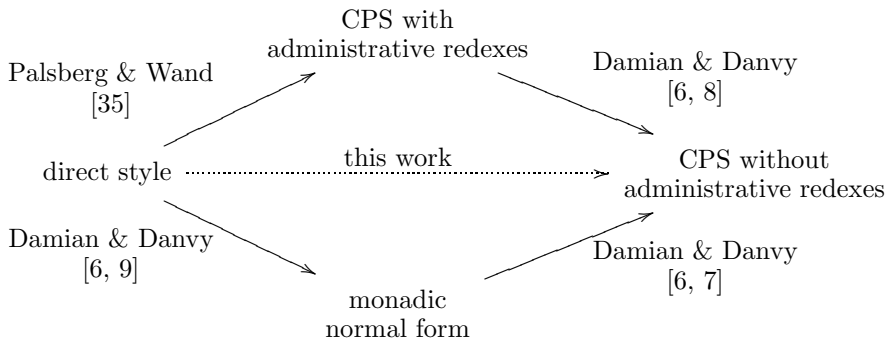
Therefore, the existing CPS transformations of flow information operate in two passes. The first pass computes an intermediate result and the second pass exploits it to construct the flow information.

In this article, we build on Danvy and Nielsen's new one-pass CPS transformation [14, 30] and we present a new and simpler CPS transformation of control-flow information that does not construct any intermediate result and thus is more efficient to use. It is also simpler to prove correct. Indeed, proving predicates defined by structural induction on a CPS-transformed program is simplest done with a first-order and compositionally-defined one-pass CPS transformation.

1.3 This work

We show how to directly construct control-flow information for a CPS program after administrative reductions, without the need for an intermediate form. Our construction confirms that the CPS transformation does not affect the result of a monovariant constraint-based control-flow analysis [6, 7, 9, 35]. It also opens the way to directly investigating the effect of the CPS transformation on other analyses, as for instance, binding-time analysis.

Graphically:



$e \in \mathit{Expr}$	(terms)	$e ::= s \mid t$
$s \in \mathit{Comp}$	(serious terms, i.e., computations)	$s ::= e_0^{\ell_0}\ e_1^{\ell_1}$
$t, K \in \mathit{Triv}$	(trivial terms, i.e., values)	$t ::= x \mid \lambda^{\pi} x.e^{\ell}$
$x \in \mathit{Ide}$	(identifiers)	
$\ell \in \mathit{Lab}$	(term labels)	
$\pi \in \mathit{Lam}$	(λ-abstraction labels)	

FIG. 1. The language of labeled λ-terms

Our CPS transformation of control flow is simpler than previous versions and addresses the λ-calculus without the need for an intermediate form or administrative reductions. The proofs of correctness are similar to the ones in our earlier work, but here source terms need not be in monadic normal form. They are also slightly simpler than Palsberg and Wand's since programs contain no administrative redexes.

2 Control-flow analysis for λ-terms

2.1 The language of λ-terms

We consider the language of labeled λ-terms defined in Figure 1. Following Reynolds [38] and Moggi [27], we distinguish among trivial terms t that denote values and serious terms s that may denote computations. Expressions are annotated with distinct labels ℓ from a countable set *Lab*. Each λ-abstraction has a unique associated label π. A program p is a closed labeled expression e^{ℓ}.

2.2 Control-flow analysis

We consider a standard constraint-based control-flow analysis (CFA) on λ-terms [5, 16, 20, 21, 22, 32, 33, 34].

Specifically, we consider the CFA specified in Nielson, Nielson, and Hankin's textbook [33]. Given an input program p, the functionality of the syntax-directed control-flow analysis relation $\models^p$ is defined in Figure 2. The analysis relation is defined inductively in Figure 3.

The relation is defined on a pair of a tuple $(\widehat{C}, \widehat{\rho})$ and a labeled expression e^{ℓ}. In the relation, $\widehat{C}$ is a cache mapping each expression label to a set of λ-abstractions that the expression might evaluate to, while $\widehat{\rho}$ is an environment mapping each program variable to a set of λ-abstractions that the variable might denote. It is known [33, Chapter 3] that a pair $(\widehat{C}, \widehat{\rho})$ satisfying the relation $(\widehat{C}, \widehat{\rho}) \models^p p$ is a safe analysis of the program p.

Given a source program p, solutions of the analysis of p always exist. The set of solutions of the analysis of p is closed under intersection: the pointwise intersection of two solutions always exists. Therefore, there exists a least solution of the analysis of p. The least solution can be computed with a standard work-list based algorithm [33, Chapter 3]. Through the rest of this article we use "the result of the analysis of p" to refer to the least result of the analysis.

$$
\begin{array}{rl}
Lam^p & \text{The set of } \lambda\text{-abstraction labels in } p \\
Var^p & \text{The set of identifiers in } p \\
Lab^p & \text{The set of term labels in } p
\end{array}
$$

$$
\begin{array}{rll}
Triv^p = \mathcal{P}(Lam^p) & & \text{Abstract values} \\
\widehat{C} \in Cache^p = Lab^p \to Triv^p & & \text{Abstract cache} \\
\widehat{\rho} \in \quad Env^p = Var^p \to Triv^p & & \text{Abstract environment}
\end{array}
$$

$$\vDash^p \subseteq (Cache^p \times Env^p) \times Lab^p$$

FIG. 2. Control-flow analysis relation for a program p

$$
\begin{array}{lcl}
(\widehat{C},\widehat{\rho}) \vDash^p x^\ell & \iff & \widehat{\rho}(x) \subseteq \widehat{C}(\ell) \\
(\widehat{C},\widehat{\rho}) \vDash^p (\lambda^\pi x.e^\ell)^{\ell_1} & \iff & (\widehat{C},\widehat{\rho}) \vDash^p e^\ell \wedge \pi \in \widehat{C}(\ell_1) \\
(\widehat{C},\widehat{\rho}) \vDash^p (e_0^{\ell_0}\ e_1^{\ell_1})^{\ell_2} & \iff & (\widehat{C},\widehat{\rho}) \vDash^p e_0^{\ell_0} \wedge (\widehat{C},\widehat{\rho}) \vDash^p e_1^{\ell_1} \wedge \\
 & & \forall \lambda^\pi x.e^\ell \in \widehat{C}(\ell_0).\widehat{C}(\ell_1) \subseteq \widehat{\rho}(x) \wedge \\
 & & \widehat{C}(\ell) \subseteq \widehat{C}(\ell_2)
\end{array}
$$

FIG. 3. Control-flow analysis

2.3 *Control-flow analysis: an example*

An example of CFA analysis is presented in Figure 4. The (labeled) λ-term T applies the identity function to itself. The control-flow analysis from Figure 3 on the term T results in the cache/environment pair also presented in Figure 4.

$$T = ((\lambda^{\pi_1} y.(y^{\ell_1}\ y^{\ell_2})^{\ell_3})^{\ell_4}\ (\lambda^{\pi_2} x.x^{\ell_5})^{\ell_6})^{\ell_7}$$

$\widehat{C}$

ℓ_1	$\to$	$\{\pi_2\}$	ℓ_5	$\to$	$\{\pi_2\}$
ℓ_2	$\to$	$\{\pi_2\}$	ℓ_6	$\to$	$\{\pi_2\}$
ℓ_3	$\to$	$\{\pi_2\}$	ℓ_7	$\to$	$\{\pi_2\}$
ℓ_4	$\to$	$\{\pi_1\}$			

$\widehat{\rho}$

y	$\to$	$\{\pi_2\}$
x	$\to$	$\{\pi_2\}$

FIG. 4. CFA example

We can see that the λ-abstraction π_2 is detected to flow into the variable y and from there into the variable x and as a result of the application. In the following section we illustrate the CPS transformation of the term T and how the flow information for the resulting CPS term can be computed from the flow information for T displayed in Figure 4.

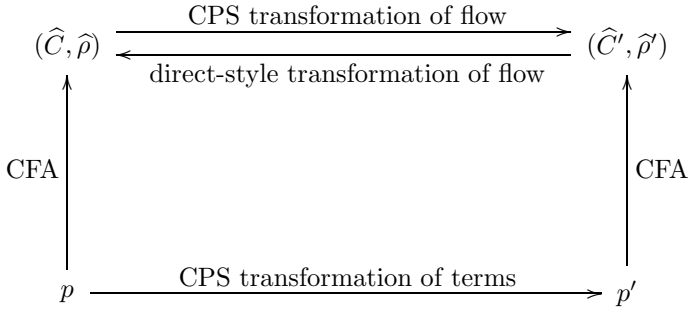
$$
\begin{array}{l}
\mathcal{E} : \mathit{Expr} \times \mathit{Ide} \to \mathit{Comp} \\
\qquad \begin{array}{rcl}
\mathcal{E}[\![t]\!]k & = & k\ \mathcal{T}[\![t]\!] \\
\mathcal{E}[\![s]\!]k & = & \mathcal{S}[\![s]\!](\lambda x.k\ x)
\end{array} \\
\\
\mathcal{S} : \mathit{Comp} \times \mathit{Triv} \to \mathit{Comp} \\
\qquad \begin{array}{rcl}
\mathcal{S}[\![t_0\ t_1]\!]K & = & \mathcal{T}[\![t_0]\!]\ \mathcal{T}[\![t_1]\!]\ K \\
\mathcal{S}[\![t_0\ s_1]\!]K & = & \mathcal{S}[\![s_1]\!](\lambda x_1.\mathcal{T}[\![t_0]\!]\ x_1\ K) \\
\mathcal{S}[\![s_0\ t_1]\!]K & = & \mathcal{S}[\![s_0]\!](\lambda x_0.x_0\ \mathcal{T}[\![t_1]\!]\ K) \\
\mathcal{S}[\![s_0\ s_1]\!]K & = & \mathcal{S}[\![s_0]\!](\lambda x_0.\mathcal{S}[\![s_1]\!](\lambda x_1.x_0\ x_1\ K))
\end{array} \\
\\
\mathcal{T} : \mathit{Triv} \to \mathit{Triv} \\
\qquad \begin{array}{rcl}
\mathcal{T}[\![x]\!] & = & x \\
\mathcal{T}[\![\lambda x.e]\!] & = & \lambda x.\lambda k.\mathcal{E}[\![e]\!]k
\end{array}
\end{array}
$$

FIG. 5. First-order one-pass CPS transformation (labels omitted)

3 CPS transformation and control-flow analysis

We show that the CPS transformation preserves the result of the control-flow analysis defined in Section 2.2. To this end, we define a transformation from control-flow information for a direct-style program into control-flow information for the CPS counterpart of this program. We also define a transformation of control-flow information for a CPS-transformed program into control-flow information for the direct-style counterpart of the program. Using the monotonicity of the two transformations, we show that the least analysis of a direct-style program is equivalent to the least analysis of its CPS counterpart and vice-versa.

Graphically:



3.1 CPS transformation of terms

In this article, CPS programs are obtained using Danvy and Nielsen's first-order CPS transformation [14, 30]. The CPS transformation for (unlabeled) λ-terms is defined in Figure 5. As in our earlier work [7, 9], we consider a transformation with η-expanded tail calls: the continuation passed at a function call is always a syntactic λ-abstraction.

The CPS transformation of a program preserves all the original variables of the

program. In turn, as in our earlier work [7, 9], we design the CPS transformation of labeled terms to preserve the labels of all trivial terms.

Danvy and Nielsen's one-pass CPS transformation yields CPS terms without administrative redexes. In Section 3.2, using this CPS transformation as a syntactic support, we define the CPS transformation of control-flow information for CPS programs without administrative redexes. In Section 3.3, we define the direct-style transformation of control-flow information from CPS programs without administrative redexes. In Section 3.4, with the same technique described in the first author's PhD thesis [6, Section 2.3], the variables and labels common to the original program and to its CPS counterpart are used to establish the preservation of flow information across CPS transformation.

3.2 CPS transformation of control flow

We define a CPS transformation of control-flow information following the CPS transformation of Figure 5. Let us show how control-flow information for a direct-style term can be used to compute control-flow information for the CPS transformed program.

To transformation relies on two auxiliary functions:

- γ extracts the labels of partially applied CPS λ-abstractions. Formally, considering A to be a set of CPS λ-abstractions $\{\lambda^{\pi_i} x_i.\lambda^{\pi_i'} k_i.e_i | 1 \leq i \leq n\}$, for some n, then $\gamma(A) = \{\pi_i' | 1 \leq i \leq n\}$.
- ξ assigns flow information to each continuation identifier k introduced by the CPS transformation of a λ-abstraction from p. This information can be obtained from the direct-style flow information, since we can syntactically identify the continuation of the CPS counterpart of any direct-style application.
  Given p, $\widehat{C}$, $\widehat{\rho}$, and a continuation identifier k introduced by the transformation of a λ-abstraction from p:
  $$\mathcal{T}[\![\lambda^{\pi} x.e^{\ell}]\!] = \lambda^{\pi} x.\lambda k.\mathcal{E}[\![e]\!]k$$
  we define $\xi(k)$ as the union of all sets $\widehat{C}'(\ell)$ such that in the CPS transformation of p into p' there exists a transformation step
  $$\mathcal{S}[\![e_0^{\ell_0}\ e_1^{\ell_1}]\!]K^{\ell} = \ldots$$
  such that $\pi \in \widehat{C}(\ell_0)$.

We construct the CPS control-flow information in two steps. First, in a recursive descent on the tree of the transformation, we compute $\widehat{C}'(\ell)$ for each label ℓ attached on the newly introduced λ-abstractions (continuations) and we construct the function ξ.

The second step consists of another recursive descent on the tree of the transformation. We assign control-flow information recursively, as defined for each step in Figure 6. At each transformation step, on the right-hand side, we construct the labeled CPS term corresponding to the left-hand side. We then assign flow information for each fresh label or variable. Trivial terms preserve their label and their flow information. Flow information for serious terms is transferred through calls to continuations. Fresh continuation identifiers are assigned flow information as computed by the ξ function.

$\mathcal{E} : \mathit{Expr} \times \mathit{Lab} \times \mathit{Ide} \to \mathit{Comp} \times \mathit{Lab}$

$$\begin{array}{rcll}
\mathcal{E}[\![t^\ell]\!]k & = & (k^{\ell_0}\ (\mathcal{T}[\![t]\!])^\ell)^{\ell_1} & \widehat{C}'(\ell_0) = \widehat{\rho}'(k) \\
 & & & \widehat{C}'(\ell) = \widehat{C}'(\ell) \quad \widehat{C}'(\ell_1) = \emptyset \\
\mathcal{E}[\![s^\ell]\!]k & = & (\mathcal{S}[\![s]\!](\lambda^\pi x.(k^{\ell_0}\ x^\ell)^{\ell_1})^{\ell_2})^{\ell_3} & \\
 & & & \widehat{C}'(\ell_0) = \widehat{\rho}'(k) \quad \widehat{C}'(\ell) = \widehat{\rho}'(x) = \widehat{C}(\ell) \\
 & & & \widehat{C}'(\ell_2) = \{\pi\} \quad \widehat{C}'(\ell_3) = \widehat{C}'(\ell_1) = \emptyset
\end{array}$$

$\mathcal{S} : \mathit{Comp} \times \mathit{Triv} \times \mathit{Lab} \to \mathit{Comp}$

$$\begin{array}{rcll}
\mathcal{S}[\![t_0^{\ell_0}\ t_1^{\ell_1}]\!]K^\ell & = & ((\mathcal{T}[\![t_0]\!])^{\ell_0}\ (\mathcal{T}[\![t_1]\!])^{\ell_1})^{\ell_2}\ K^\ell & \\
 & & & \widehat{C}'(\ell_0) = \widehat{C}(\ell_0) \quad \widehat{C}'(\ell_1) = \widehat{C}(\ell_1) \\
 & & & \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
\mathcal{S}[\![t_0^{\ell_0}\ s_1^{\ell_1}]\!]K^\ell & = & \mathcal{S}[\![s_1]\!](\lambda^\pi x_1.(((\mathcal{T}[\![t_0]\!])^{\ell_0}\ x_1^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4} & \\
 & & & \widehat{C}'(\ell_0) = \widehat{C}(\ell_0) \quad \widehat{C}'(\ell_1) = \widehat{\rho}'(x_1) = \widehat{C}(\ell_1) \\
 & & & \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
 & & & \widehat{C}'(\ell_3) = \emptyset \quad \widehat{C}'(\ell_4) = \{\pi\} \\
\mathcal{S}[\![s_0^{\ell_0}\ t_1^{\ell_1}]\!]K^\ell & = & \mathcal{S}[\![s_0]\!](\lambda^\pi x_0.((x_0^{\ell_0}\ (\mathcal{T}[\![t_1]\!])^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4} & \\
 & & & \widehat{C}'(\ell_0) = \widehat{\rho}'(x_0) = \widehat{C}(\ell_0) \quad \widehat{C}'(\ell_1) = \widehat{C}(\ell_1) \\
 & & & \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
 & & & \widehat{C}'(\ell_3) = \emptyset \quad \widehat{C}'(\ell_4) = \{\pi\} \\
\mathcal{S}[\![s_0^{\ell_0}\ s_1^{\ell_1}]\!]K^\ell & = & \mathcal{S}[\![s_0]\!](\lambda^\pi x_0.(\mathcal{S}[\![s_1]\!](\lambda^{\pi_1} x_1.((x_0^{\ell_0}\ x_1^{\ell_1})^{\ell_2}\ K^\ell)^{\ell_3})^{\ell_4})^{\ell_5})^{\ell_6} & \\
 & & & \widehat{C}'(\ell_0) = \widehat{\rho}'(x_0) = \widehat{C}(\ell_0) \\
 & & & \widehat{C}'(\ell_1) = \widehat{\rho}'(x_1) = \widehat{C}(\ell_1) \quad \widehat{C}'(\ell_2) = \gamma(\widehat{C}(\ell_0)) \\
 & & & \widehat{C}'(\ell_3) = \emptyset \quad \widehat{C}'(\ell_4) = \{\pi_1\} \\
 & & & \widehat{C}'(\ell_5) = \emptyset \quad \widehat{C}'(\ell_6) = \{\pi\}
\end{array}$$

$\mathcal{T} : \mathit{Triv} \to \mathit{Triv}$

$$\begin{array}{rcll}
\mathcal{T}[\![x]\!] & = & x & \\
\mathcal{T}[\![\lambda^\pi x.e^\ell]\!] & = & \lambda^\pi x.(\lambda^{\pi_1} k.\mathcal{E}[\![e^\ell]\!]k)^{\ell_0} & \widehat{C}'(\ell_0) = \{\pi_1\} \quad \widehat{\rho}'(k) = \xi(k)
\end{array}$$

FIG. 6. Transformation of control flow from direct style to CPS

Note that in contrast to the CPS transformation of unlabeled terms of Figure 5, the transformation of labeled serious terms takes an extra argument, namely the label of the syntactic continuation being passed as an argument. At each case in Figure 6, we do not make the label explicit: we rather place it directly over the constructed continuation. Similarly, the CPS transformation of a labeled expression returns a serious term and its enclosing label.

The CPS transformation of control flow is therefore defined as a monotone function:

$$\Phi_{\mathrm{cf}}^{\mathrm{CPS}} : (\mathit{Cache}^p \times \mathit{Env}^p) \to (\mathit{Cache}^{p'} \times \mathit{Env}^{p'}).$$

Theorem 3.1 Let $p = e^\ell$ be a uniquely labeled program. If $(\widehat{C}, \widehat{\rho}) \vDash^p e^\ell$ then $(\Phi_{\mathrm{cf}}^{\mathrm{CPS}}(\widehat{C}, \widehat{\rho})) \vDash^{p'} \lambda^\pi k.\mathcal{E}[\![e^\ell]\!]k$.

PROOF. By structural induction on the resulting CPS program. The proof is similar

with the proof of Theorem 6.1 of our previous work [9] which addressed terms in monadic normal form. In this proof, however, the main induction predicate states that a CPS-transformed serious term satisfies the relation when the term passed as a continuation is also satisfying the relation. The main induction predicate relies on:

a) an auxiliary predicate stating that the translation of a trivial term together with its associated label satisfies the flow constraints in CPS if the associated label is preserved;
b) an auxiliary predicate stating that the translation of an expression with its syntactic continuation satisfies the flow constraints in CPS.

At each iteration step we make use of an auxiliary Lemma (similar to Lemma 6.4 of the same previous work [9]) stating that the flow information extracted at an application point is passed into each possible continuation for the CPS equivalent of the application. ∎

The proof is also slightly simpler than Palsberg and Wand's proof [35] since programs contain no administrative redexes.

3.3 Direct-style transformation of control flow

The CPS transformation of flow from Figure 6 shows that the analysis of a CPS-transformed term can be at least as good as the analysis of the direct-style original term. The resulting CPS solution is the equivalent of the direct-style one, but may not be the best. We show that the direct-style and CPS analysis results are equivalent by exhibiting a direct-style transformation of flow.

We thus define a direct-style transformation of control-flow information. In other words, we transform control-flow information for the CPS-transformed term into control-flow information for the original direct-style term. The transformation is defined recursively in Figure 7. At each transformation step, on the right-hand side we construct flow information $(\widehat{C}, \widehat{\rho})$ for the direct-style program from the flow information $(\widehat{C}', \widehat{\rho}')$ for the CPS program.

Since at each function call the continuation is an explicit syntactic continuation, we are able to determine the control-flow information returned by each expression. In particular, at a transformation step

$$\mathcal{E}[\![s^{\ell}]\!]k = (\mathcal{S}[\![s]\!](\lambda^{\pi} x.(k^{\ell_0}\ x^{\ell})^{\ell_1})^{\ell_2})^{\ell_3}$$

we are able to assign control-flow information for the return label ℓ from the control-flow information collected by the continuation $\lambda^{\pi} x.(k^{\ell_0}\ x^{\ell})^{\ell_1}$.

Control-flow information can therefore be constructed bottom-up. The direct-style transformation of control flow is thus defined as a monotone function:

$$\Psi_{\mathrm{cf}}^{\mathrm{CPS}} : (Cache^{p'} \times Env^{p'}) \to (Cache^{p} \times Env^{p})$$

Theorem 3.2 Let $p = e^{\ell}$ be a uniquely labeled program.
If $(\widehat{C}', \widehat{\rho}') \vDash^{p} \lambda^{\pi} k.\mathcal{E}[\![e^{\ell}]\!]k$ and $\widehat{\rho}'(k) = \emptyset$, then $\Psi_{\mathrm{cf}}^{\mathrm{CPS}}(\widehat{C}', \widehat{\rho}') \vDash^{p'} e^{\ell}$.

Proof. By structural induction on the direct-style source program. Again, the proof is similar to the proof of Theorem 6.5 of our earlier work [9]. In this proof the main

$$
\begin{array}{l}
\mathcal{E} : \mathit{Expr} \times \mathit{Ide} \to \mathit{Comp} \times \mathit{Lab} \\
\quad \mathcal{E}[\![t^{\ell}]\!]k = (k^{\ell_0}\ (\mathcal{T}[\![t]\!])^{\ell})^{\ell_1} \qquad \widehat{C}(\ell) = \widehat{C}'(\ell) \quad \widehat{C}(\ell_1) = \emptyset \\
\quad \mathcal{E}[\![s^{\ell}]\!]k = (\mathcal{S}[\![s]\!](\lambda^{\pi} x.(k^{\ell_0}\ x^{\ell})^{\ell_1})^{\ell_2})^{\ell_3} \qquad \widehat{C}(\ell) = \widehat{C}'(\ell) \\
\\
\mathcal{S} : \mathit{Comp} \times \mathit{Triv} \times \mathit{Lab} \to \mathit{Comp} \\
\quad \mathcal{S}[\![t_0^{\ell_0}\ t_1^{\ell_1}]\!]K^{\ell} = ((\mathcal{T}[\![t_0]\!])^{\ell_0}\ (\mathcal{T}[\![t_1]\!])^{\ell_1})^{\ell_2}\ K^{\ell} \\
\qquad\qquad \widehat{C}(\ell_0) = \widehat{C}'(\ell_0) \quad \widehat{C}(\ell_1) = \widehat{C}'(\ell_1) \\
\quad \mathcal{S}[\![t_0^{\ell_0}\ s_1^{\ell_1}]\!]K^{\ell} = \mathcal{S}[\![s_1]\!](\lambda^{\pi} x_1.(((\mathcal{T}[\![t_0]\!])^{\ell_0}\ x_1^{\ell_1})^{\ell_2}\ K^{\ell})^{\ell_3})^{\ell_4} \\
\qquad\qquad \widehat{C}(\ell_0) = \widehat{C}'(\ell_0) \quad \widehat{C}(\ell_1) = \widehat{\rho}'(x_1) \\
\quad \mathcal{S}[\![s_0^{\ell_0}\ t_1^{\ell_1}]\!]K^{\ell} = \mathcal{S}[\![s_0]\!](\lambda^{\pi} x_0.((x_0^{\ell_0}\ (\mathcal{T}[\![t_1]\!])^{\ell_1})^{\ell_2}\ K^{\ell})^{\ell_3})^{\ell_4} \\
\qquad\qquad \widehat{C}(\ell_0) = \widehat{\rho}'(x_0) \quad \widehat{C}(\ell_1) = \widehat{C}'(\ell_1) \\
\quad \mathcal{S}[\![s_0^{\ell_0}\ s_1^{\ell_1}]\!]K^{\ell} = \mathcal{S}[\![s_0]\!](\lambda^{\pi} x_0.(\mathcal{S}[\![s_1]\!](\lambda^{\pi_1} x_1.((x_0^{\ell_0}\ x_1^{\ell_1})^{\ell_2}\ K^{\ell})^{\ell_3})^{\ell_4})^{\ell_5})^{\ell_6} \\
\qquad\qquad \widehat{C}(\ell_0) = \widehat{\rho}'(x_0) \quad \widehat{C}(\ell_1) = \widehat{\rho}'(x_1) \\
\\
\mathcal{T} : \mathit{Triv} \to \mathit{Triv} \\
\quad \mathcal{T}[\![x]\!] = x \\
\quad \mathcal{T}[\![\lambda^{\pi} x.e^{\ell}]\!] = \lambda^{\pi} x.(\lambda^{\pi_1} k.\mathcal{E}[\![e^{\ell}]\!]k)^{\ell_0}
\end{array}
$$

FIG. 7. Transformation of control flow from CPS into direct style

induction predicate states that the constructed solution satisfies the flow constraints for any serious sub-term considered together with its enclosing label. The proof relies on:

a) an auxiliary predicate stating that a trivial term together with its associated label satisfies the flow constraints if it satisfies the constraints in CPS, considered together with its associated label;

b) an auxiliary predicate stating that an expression satisfies the flow constraints if the translation satisfies the flow constraints in CPS (considered together with its syntactic continuation).

At each iteration step we make use of an auxiliary Lemma (similar to Lemma 6.8 of the same previous work [9]) stating that the flow information extracted at an application point includes the flow information collected by each possible continuation for the CPS equivalent of the application. ∎

3.4 Preservation of flow

Following the construction of the CPS control-flow information in Figure 6, it is immediate to see that the flow information assigned to the program's original variables in CPS is identical to the one extracted from the direct-style original program. The same is valid for the reverse transformation of Figure 7: the control-flow information assigned to direct-style variables is identical to the one extracted from the CPS program.

$$
\begin{aligned}
T' \;=\;& (((\lambda^{\pi_1} y.(\lambda^{\pi_4} k_2.((y^{\ell_1}\ y^{\ell_2})^{\ell_{16}}\ (\lambda^{\pi_5} v_2.(k_2^{\ell_{12}}\ v_2^{\ell_{13}})^{\ell_{14}})^{\ell_{15}})^{\ell_{17}})^{\ell_{18}})^{\ell_4} \\
& (\lambda^{\pi_2} x.(\lambda^{\pi_6} k_3.(k_3^{\ell_{19}}\ x^{\ell_5})^{\ell_{20}})^{\ell_{21}})^{\ell_6})^{\ell_{22}} \\
& (\lambda^{\pi_3} v_1.(k_1^{\ell_8}\ v_1^{\ell_9})^{\ell_{10}})^{\ell_{11}})^{\ell_{23}}
\end{aligned}
$$

ξ

k_2	$\{\pi_3\}$
k_3	$\{\pi_5\}$

$\widehat{C}$

$\ell_1 \to \{\pi_2\}$	$\ell_{10} \to \{\}$	$\ell_{17} \to \{\}$
$\ell_2 \to \{\pi_2\}$	$\ell_{11} \to \{\pi_3\}$	$\ell_{18} \to \{\pi_4\}$
$\ell_4 \to \{\pi_1\}$	$\ell_{12} \to \{\pi_3\}$	$\ell_{19} \to \{\pi_5\}$
$\ell_5 \to \{\pi_2\}$	$\ell_{13} \to \{\pi_2\}$	$\ell_{20} \to \{\}$
$\ell_6 \to \{\pi_2\}$	$\ell_{14} \to \{\}$	$\ell_{21} \to \{\pi_6\}$
$\ell_8 \to \{\}$	$\ell_{15} \to \{\pi_5\}$	$\ell_{22} \to \{\pi_4\}$
$\ell_9 \to \{\pi_2\}$	$\ell_{16} \to \{\pi_6\}$	$\ell_{23} \to \{\}$

$\widehat{\rho}$

k_1	$\{\}$
k_2	$\{\pi_3\}$
k_3	$\{\pi_5\}$
x	$\{\pi_2\}$
y	$\{\pi_2\}$
v_1	$\{\pi_2\}$
v_2	$\{\pi_2\}$

FIG. 8. CPS transformation and analysis result

Theorem 3.3 follows from the monotonicity of the two transformations of control flow.

Theorem 3.3 Let p be a direct-style program and p' its CPS counterpart.

i) Let $(\widehat{C}, \widehat{\rho})$ be the solution of the control-flow analysis of p. Then $\Psi_{\text{cf}}^{\text{CPS}}(\Phi_{\text{cf}}^{\text{CPS}}(\widehat{C}, \widehat{\rho})) = (\widehat{C}, \widehat{\rho})$.

ii) Let $(\widehat{C}', \widehat{\rho}')$ be the solution of the control-flow analysis of p'. Then $\Phi_{\text{cf}}^{\text{CPS}}(\Psi_{\text{cf}}^{\text{CPS}}(\widehat{C}', \widehat{\rho}')) = (\widehat{C}', \widehat{\rho}')$.

3.5 CPS transformation of flow: an example

Let us now consider the CPS transformation of the term T in the example of Section 2.3. The CPS equivalent T' of the term T is illustrated in Figure 8. Even if the term T' is administratively reduced, the number of labels becomes difficult to manage without an automated calculation. The generated example illustrates the equivalence of flow information obtained by the CFA analysis of the original term T and of the CPS term T'.

As specified in Section 1.1, the CPS term T' maintains all the λ-abstraction labels and trivial-term labels of the original term T'. As specified by Theorem 3.3, the flow information associated to the labels of the trivial terms (i.e., $\ell_1, \ell_2, \ell_4, \ell_5$ and ℓ_6) are identical. Similarly, the variables of the original term (x and y) are preserved and their associated flow information is identical. We can observe that the labels ℓ_3 and ℓ_7 have disappeared, their associated flow information being transferred into the variables abstracted by continuations v_2 and v_1 respectively. The remainder of the labels are either final answer labels, and their associated flow information is empty, either labels surrounding a continuation in which case the associated flow information is a singleton containing the label of the continuation.

Therefore, given the flow information from Figure 4 we can avoid re-analyzing the CPS term T' by computing the flow information of Figure 8 according to the

transformation function $\Phi_{\mathrm{cf}}^{\mathrm{CPS}}$, with a provably lower complexity. Similarly, given the flow information of Figure 8, we can avoid re-analyzing the CPS term T' by computing the flow information of Figure 4 according to the transformation function $\Phi_{\mathrm{cf}}^{\mathrm{CPS}}$, again with a provably lower complexity.

4 Conclusions and future work

We have presented a one-pass CPS transformation of control-flow information. Our transformation improves both on our earlier CPS transformation and on Palsberg and Wand's which operate in two passes. This line of work aims at transferring the results of program analyses across program transformations as an alternative to analyzing transformed programs from scratch. The interaction between CPS and program analysis has been explored by a number of authors [3, 4, 19, 23, 25, 29, 31], sometimes leading to mixed results [41].

The complete CPS transformation of control flow can be used to assess the impact of the CPS transformation on the result of other program analyses, e.g., binding-times analysis. In a previous work [7, 9], we have shown that introducing continuations (1) does not worsen and (2) can improve the results of the standard binding-time analysis for traditional partial evaluation [23]. Transforming programs into monadic normal form can also lead to further binding-time improvements [6, 19]. Our initial investigations show that the current transformation of control flow can be used to characterize in one single theorem the binding-time improvements obtained by the CPS transformation [6].

Let us finish on the relation between tail-call optimization and control-flow analysis. In the CPS transformation of Figure 5, the η-expanded tail calls provide an explicit continuation for each function call for which we can extract control-flow information. More precisely, the CPS transformation of an expression introduces an explicit continuation:

$$\mathcal{E}[\![s]\!]k = \mathcal{S}[\![s]\!](\lambda x.k\ x)$$

The presence of such an explicit continuation facilitates the definition of the CPS transformation of control flow. We are currently investigating whether η-reducing these tail-calls (i.e., defining $\mathcal{E}[\![s]\!]k$ as $\mathcal{S}[\![s]\!]k$) also preserves control-flow information.

4.1 Acknowledgments:

This work was carried out while the first author was at BRICS.

References

[1] Andrew W. Appel. *Compiling with Continuations.* Cambridge University Press, New York, 1992.

[2] Edoardo Biagioni, Ken Cline, Peter Lee, Chris Okasaki, and Chris Stone. Safe-for-space threads in Standard ML. *Higher-Order and Symbolic Computation*, 11(2):209–225, 1998.

[3] Anders Bondorf. Improving binding times without explicit cps-conversion. In William Clinger, editor, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, LISP Pointers, Vol. V, No. 1, pages 1–10, San Francisco, California, June 1992. ACM Press.

[4] Charles Consel and Olivier Danvy. For a better support of static data flow. In John Hughes, editor, *Proceedings of the Fifth ACM Conference on Functional Programming and Computer*

Architecture, number 523 in Lecture Notes in Computer Science, pages 496–519, Cambridge, Massachusetts, August 1991. Springer-Verlag.

[5] Patrick Cousot and Radhia Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In Simon Peyton Jones, editor, *Proceedings of the Seventh ACM Conference on Functional Programming and Computer Architecture*, pages 170–181, La Jolla, California, June 1995. ACM Press.

[6] Daniel Damian. *On Static and Dynamic Control-Flow Information in Program Analysis and Transformation*. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark, July 2001. BRICS DS-01-5.

[7] Daniel Damian and Olivier Danvy. Syntactic accidents in program analysis: On the impact of the CPS transformation. In Philip Wadler, editor, *Proceedings of the 2000 ACM SIGPLAN International Conference on Functional Programming*, SIGPLAN Notices, Vol. 35, No. 9, pages 209–220, Montréal, Canada, September 2000. ACM Press. Extended version to appear in the Journal of Functional Programming.

[8] Daniel Damian and Olivier Danvy. CPS transformation of flow information, part II: Administrative reductions. Technical Report BRICS RS-01-40, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, October 2001. Accepted at the Journal of Functional Programming.

[9] Daniel Damian and Olivier Danvy. Syntactic accidents in program analysis: On the impact of the CPS transformation. *Journal of Functional Programming*, 2002. To appear. Extended version available as the technical report BRICS-RS-01-54.

[10] Olivier Danvy, editor. *Proceedings of the Second ACM SIGPLAN Workshop on Continuations*, Technical report BRICS-NS-96-13, University of Aarhus, Paris, France, January 1997.

[11] Olivier Danvy and Andrzej Filinski. Representing control, a study of the CPS transformation. *Mathematical Structures in Computer Science*, 2(4):361–391, 1992.

[12] Olivier Danvy, Bernd Grobauer, and Morten Rhiger. A unifying approach to goal-directed evaluation. *New Generation Computing*, 20(1):53–73, 2002. Preliminary version available in the proceedings of SAIG 2001 (LNCS 2196). Extended version available as the technical report BRICS RS-01-29.

[13] Olivier Danvy and Lasse R. Nielsen. A higher-order colon translation. In Herbert Kuchen and Kazunori Ueda, editors, *Fifth International Symposium on Functional and Logic Programming*, number 2024 in Lecture Notes in Computer Science, pages 78–91, Tokyo, Japan, March 2001. Springer-Verlag. Extended version available as the technical report BRICS RS-00-33.

[14] Olivier Danvy and Lasse R. Nielsen. A first-order one-pass CPS transformation. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002*, number 2303 in Lecture Notes in Computer Science, pages 98–113, Grenoble, France, April 2002. Springer-Verlag. Extended version available as the technical report BRICS RS-01-49.

[15] Richard P. Draves, Brian N. Bershad, Richard F. Rashid, and Randall W. Dean. Using continuations to implement thread management and communication in operating systems. Technical Report CMU-CS-91-115, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, October 1991. Also appears in the proceedings of the Thirteenth Symposium on Operating Systems Principles (SOSP), Asilomar, California, October 1991.

[16] Kirsten L. Solberg Gasser, Flemming Nielson, and Hanne Riis Nielson. Systematic realisation of control flow analyses for CML. In Mads Tofte, editor, *Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming*, pages 38–51, Amsterdam, The Netherlands, June 1997. ACM Press.

[17] Timothy G. Griffin. A formulae-as-types notion of control. In Paul Hudak, editor, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, San Francisco, California, January 1990. ACM Press.

[18] John Hatcliff and Olivier Danvy. A generic account of continuation-passing styles. In Hans-J. Boehm, editor, *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 458–471, Portland, Oregon, January 1994. ACM Press.

[19] John Hatcliff and Olivier Danvy. A computational formalization for partial evaluation. *Mathematical Structures in Computer Science*, pages 507–541, 1997. Extended version available as the technical report BRICS RS-96-34.

[20] Nevin Heintze. Set-based program analysis of ML programs. In Talcott [45], pages 306–317.

[21] Fritz Henglein. Simple closure analysis. Technical Report Semantics Report D-193, DIKU, Computer Science Department, University of Copenhagen, 1992.

[22] Suresh Jagannathan and Stephen Weeks. A unified treatment of flow analysis in higher-order languages. In Peter Lee, editor, *Proceedings of the Twenty-Second Annual ACM Symposium on Principles of Programming Languages*, pages 393–407, San Francisco, California, January 1995. ACM Press.

[23] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall International, London, UK, 1993. Available online at `http://www.dina.kvl.dk/~sestoft/pebook/`.

[24] David Kranz, Richard Kesley, Jonathan Rees, Paul Hudak, Jonathan Philbin, and Norman Adams. Orbit: An optimizing compiler for Scheme. In Stuart I. Feldman, editor, *Proceedings of the 1986 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 21, No 7, pages 219–233, Palo Alto, California, June 1986. ACM Press.

[25] Julia L. Lawall and Olivier Danvy. Continuation-based partial evaluation. In Talcott [45].

[26] Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda-calculi (summary). In Rohit Parikh, editor, *Logics of Programs – Proceedings*, number 193 in Lecture Notes in Computer Science, pages 219–224, Brooklyn, June 1985. Springer-Verlag.

[27] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[28] Chetan R. Murthy. *Extracting Constructive Content from Classical Proofs*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, 1990.

[29] Juarez A. Muylaert-Filho and Geoffrey L. Burn. Continuation passing transformation and abstract interpretation. In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*, Workshops in Computing Series, pages 247–259, Isle of Thorns, Sussex, 1993. Springer-Verlag.

[30] Lasse R. Nielsen. *A study of defunctionalization and continuation-passing style*. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark, July 2001. BRICS DS-01-7.

[31] Flemming Nielson. A denotational framework for data flow analysis. *Acta Informatica*, 18:265–287, 1982.

[32] Flemming Nielson and Hanne Riis Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In Neil D. Jones, editor, *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 332–345, Paris, France, January 1997. ACM Press.

[33] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer Verlag, 1999.

[34] Jens Palsberg and Michael I. Schwartzbach. Object-oriented type inference. In *Proceedings of OOPSLA'91, the ACM SIGPLAN Sixth Annual Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 146–161, Phoenix, Arizona, October 1991.

[35] Jens Palsberg and Mitchell Wand. CPS transformation of flow information. *Journal of Functional Programming*, 2002. To appear.

[36] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[37] John C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6(3/4):233–247, 1993.

[38] John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998. Reprinted from the proceedings of the 25th ACM National Conference (1972).

[39] Amr Sabry, editor. *Proceedings of the Third ACM SIGPLAN Workshop on Continuations*, Technical report 545, Computer Science Department, Indiana University, London, England, January 2001.

[40] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4):289–360, 1993.

[41] Amr Sabry and Matthias Felleisen. Is continuation-passing useful for data flow analysis? In Vivek Sarkar, editor, *Proceedings of the ACM SIGPLAN'94 Conference on Programming Languages Design and Implementation*, SIGPLAN Notices, Vol. 29, No 6, pages 1–12, Orlando, Florida, June 1994. ACM Press.

[42] Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems*, 19(6):916–941, 1997.

[43] Guy L. Steele Jr. Rabbit: A compiler for Scheme. Technical Report AI-TR-474, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1978.

[44] Christopher Strachey and Christopher P. Wadsworth. Continuations: A mathematical semantics for handling full jumps. *Higher-Order and Symbolic Computation*, 13(1/2):135–152, 2000. Reprint of the technical monograph PRG-11, Oxford University Computing Laboratory (1974).

[45] Carolyn L. Talcott, editor. *Proceedings of the 1994 ACM Conference on Lisp and Functional Programming*, LISP Pointers, Vol. VII, No. 3, Orlando, Florida, June 1994. ACM Press.

[46] Mitchell Wand. Continuation-based program transformation strategies. *Journal of the ACM*, 27(1):164–180, January 1980.

[47] Mitchell Wand. Embedding type structure in semantics. In Mary S. Van Deusen and Zvi Galil, editors, *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 1–6, New Orleans, Louisiana, January 1985. ACM Press.

[48] Mitchell Wand. Correctness of procedure representations in higher-order assembly language. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Proceedings of the 7th International Conference on Mathematical Foundations of Programming Semantics*, number 598 in Lecture Notes in Computer Science, pages 294–311, Pittsburgh, Pennsylvania, March 1991. Springer-Verlag.

[49] Mitchell Wand. Continuation-based multiprocessing. *Higher-Order and Symbolic Computation*, 12(3):285–299, 1999. Reprinted from the proceedings of the 1980 Lisp Conference.

[50] Steve Zdancewic and Andrew Myers. Secure information flow and CPS. In David Sands, editor, *Proceedings of the Tenth European Symposium on Programming*, number 2028 in Lecture Notes in Computer Science, pages 46–61, Genova, Italy, April 2001. Springer-Verlag.

Received March 17, 2002

Polymodal Logics of Commuting Functions

Aleksey G. Kravtsov, *Department of Mechanics and Mathematics, Moscow State University, Moscow, 117234.*
E-mail: kravtsov@lpcs.math.msn.ru

Abstract

Polymodal logics with functional modalities are natural generalisations of the well-known Segerberg's *Tomorrow* (or *Successor*) *Logic* **SL** [8] and *Tomorrow-Yesterday Logic* **SL.t** [7]. Extensions of these two logics were later studied by A. A. Muchnik [3]. But systematic investigation of logics with functional modalities was started in Segerberg's paper [9] and continued by F. Bellissima [1] and M. Kracht [6]. Logics of this kind can be interpreted as fragments of propositional dynamic logics of deterministic computations. They are also applied in mathematical linguistics [5]. This family is very large; one can easily construct undecidable modal logics of this type, moreover, there is no hope to obtain a reasonable classification here [6, Section 9.4].

Nevertheless, we can try to describe explicitly some of its subfamilies. A natural class are the logics, in which all modalities commute. The minimal logics with commuting modalities are the products $\mathbf{SL}^n$ described in [4, Section 14]. The semantics of $\mathbf{SL}^n$ is given by the set $\mathbb{N}^n$ with the relations represented by n correspondent coordinate shift functions.

We consider arbitrary extensions of $\mathbf{SL}^n$ and especially of $\mathbf{SL.t}^n$. We show that all extensions of $\mathbf{SL.t}^n$ have the finite model property and thus finitely axiomatisable of them are decidable. Moreover, we give a complete description of finitely axiomatisable extensions of $\mathbf{SL.t}^n$: they can be presented as finite intersections of logics of n-generated Abelian groups. By the same method we also give new proofs of Muchnik's theorems on pretabularity of the logics **SL** and **SL.t** [3]. [1]

Keywords: Functional Modalities, Commuting Modalities, $\mathbf{SL}^n$, $\mathbf{SL.t}^n$

1 Modal Logic Background.

We consider propositional formulas in a standard language containing propositional connectives ($\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\supset$, $\equiv$) and unary *modal operators*: $\Box_1, \ldots, \Box_n$. $\Diamond_i$ denotes $\neg\Box_i\neg$ as usual. Formulas in this language are called *n-modal*. If $n = 1$, we write $\Box$, $\Diamond$ rather than $\Box_1$, $\Diamond_1$.

Recall that a set Λ of n-modal formulas is a *(normal) n-modal logic* if it contains all classical tautologies, the axioms

$$\Box_i(p \supset q) \supset (\Box_i p \supset \Box_i q)$$

and is closed under Substitution, Modus Ponens, and the rules $\dfrac{A}{\Box_i A}$.

$\mathbf{K}_n$ denotes the minimal n-modal logic; $\mathbf{K}$ is $\mathbf{K}_1$.

$\Lambda + S$ denotes the smallest n-modal logic containing an n-modal logic Λ and a set of n-modal formulas S. We say that a set S *axiomatises* a logic Λ if $\Lambda = \mathbf{K}_n + S$.

[1] The work on this paper was partly supported by ESPRC project on many-dimensional modal logic.
The author is grateful to professor V. B. Shehtman for constant attention to this work and also to the anonymous referee for many useful comments.

$\bigotimes$ denotes the *fusion* of logics; it is obtained by unifying all axioms of these logics, with every $\Box$-operator endowed by a subscript, depending on the logic. The fusion of logics is the smallest logic containing the logics with the modalities subscripted.

$[\ldots]$ denotes the *commutator* of logics, i. e., the fusion of the logics plus the commutation axioms $\Box_i\Box_j p \equiv \Box_j\Box_i p$ and the axioms of the form $\Diamond_i\Box_j p \supset \Box_j\Diamond_i p$ (the Church-Rosser or confluence properties) for all the modalities.

Let us recall the definitions of some particular logics:

$$\begin{aligned}
\mathbf{Alt_1} &= \mathbf{K} + \Diamond p \supset \Box p,\\
(\mathbf{Alt_1})_n &= \textstyle\bigotimes_{i=1}^{n} \mathbf{Alt_1},\\
\mathbf{SL} &= \mathbf{K} + \Diamond p \equiv \Box p,\\
\mathbf{SL}^n &= [\underbrace{\mathbf{SL}, \ldots, \mathbf{SL}}_{n}],\\
\mathbf{SL.t} &= \mathbf{SL}^2 + \Box_1\Box_2 p \equiv p,\\
\mathbf{SL.t}^n &= [\underbrace{\mathbf{SL.t}, \ldots, \mathbf{SL.t}}_{n}].
\end{aligned}$$

SL is called *Tomorrow Logic* or *Successor Logic*, **SL.t** is called *Tomorrow-Yesterday Logic*.

For the case of $\mathbf{SL}^n$ and $\mathbf{SL.t}^n$, the Church-Rosser properties follows from the commutation axioms.

One can easily see that $(\mathbf{Alt_1})_n \subseteq \mathbf{SL}^n$ and $\mathbf{SL}^{2n} \subseteq \mathbf{SL.t}^n$.

Recall that an *(n-modal Kripke) frame* is an $(n+1)$-tuple

$$(W, R_1, \ldots, R_n),$$

where W is a non-empty set, R_i are binary relations on W.

A frame $F = (W, R_1, \ldots, R_n)$ is said to be a *rooted* (or *generated*) frame with *root* u if

$$W = \{u\} \cup R(u) \cup R^2(u) \cup \cdots,$$

where[2] $R \leftrightharpoons R_1 \cup \cdots \cup R_n$, $R^n \leftrightharpoons \underbrace{R \circ \cdots \circ R}_{n}$, $R(u) \leftrightharpoons \{v \mid (u, v) \in R\}$. In this case, the pair (F, u) is called a *cone* (with *root* u).

A *generated subframe* with root u of a Kripke frame F is a cone (F^u, u) where $F^u \leftrightharpoons (W', R'_1, \ldots, R'_n)$,

$$W' \leftrightharpoons \{u\} \cup R(u) \cup R^2(u) \cup \cdots$$

for the same R as in the definition of a generated frame, and $R'_i \leftrightharpoons R_i \cap W'^2$.

A *Kripke model* is a Kripke frame with a valuation of propositional variables. For a formula A, $(F, w, \theta) \vDash A$ denotes that the formula A is *true in a world* w in a Kripke model (F, θ); the corresponding definition is well-known.

A is *valid in* F (notation: $F \vDash A$) if $(F, w) \vDash A$ for any w. $\mathrm{L}(F)$ denotes the set of all (n-modal) formulas valid in an (n-modal) frame F; this is the *modal logic of* F. Logics of this kind are called *Kripke-complete*.

[2] $\leftrightharpoons$ denotes equality by definition; $\circ$ denotes the composition of relations

A frame F is said to be a Λ-*frame* (for an n-modal logic Λ) if $\Lambda \subseteq \mathrm{L}(F)$. A cone $C = (F, u)$ is a Λ-*cone* if $\Lambda \subseteq \mathrm{L}(F)$. Let us recall the characterisations of Λ-frames for $\Lambda = (\mathbf{Alt_1})_n, \mathbf{SL}^n, \mathbf{SL}.\mathbf{t}^n$:

Lemma 1.1

1. F is an $(\mathbf{Alt_1})_n$-frame iff all its accessibility relations are functional (i. e., every $R_i(x)$ is either empty or one-element);
2. F is an $\mathbf{SL}^n$-frame iff all its relations are functions (i. e., every $R_i(x)$ is a singleton) and they pairwise commute;
3. F is an $\mathbf{SL}.\mathbf{t}^n$-frame iff it is an $\mathbf{SL}^{2n}$-frame and $R_{2i-1} = R_{2i}^{-1}$ for $i = 1, \ldots, n$.

Let $\mathcal{N}^n$ be the monoid with the underlying set $\mathbb{N}^n$ and the operation of standard vector addition. Similarly, let $\mathcal{Z}^n$ be the group with the underlying set $\mathbb{Z}^n$ and the standard vector addition.

Let $\mathbf{e}_i$, $1 \leq i \leq n$, be the standard base vectors of $\mathcal{N}^n$, i.e. $\mathbf{e}_1 = (1, 0, 0, \ldots)$, $\mathbf{e}_2 = (0, 1, 0, \ldots)$, etc.

The following completeness result for $\mathbf{SL}^n$, $\mathbf{SL}.\mathbf{t}^n$ can be proved easily [4]:

Proposition 1.2

1. $\mathbf{SL}^n = \mathrm{L}(\mathbb{N}^n, R_1, \ldots, R_n)$, where R_i is the i-th coordinate shift function:
$$R_i(\mathbf{x}) \quad \leftrightharpoons \quad \{\mathbf{x} + \mathbf{e}_i\}.$$
2. $\mathbf{SL}.\mathbf{t}^n = \mathrm{L}(\mathbb{Z}^n, R_1, \ldots, R_{2n})$, where R_{2i-1}, R_{2i} are the coordinate shift functions:
$$\begin{array}{lcl} R_{2i-1}(\mathbf{x}) & \leftrightharpoons & \{\mathbf{x} + \mathbf{e}_i\}, \\ R_{2i}(\mathbf{x}) & \leftrightharpoons & \{\mathbf{x} - \mathbf{e}_i\}. \end{array}$$

Here $+$ and $-$ denote the operations in $\mathcal{N}^n$ and $\mathcal{Z}^n$.

The following theorem was proved in [6, Theorem 3.2.12].

Theorem 1.3 All extensions of $(\mathbf{Alt_1})_n$ are Kripke-complete.

Hence we readily obtain the following

Corollary 1.4 All extensions of $\mathbf{SL}^n$ and $\mathbf{SL}.\mathbf{t}^n$ are Kripke-complete.

Let f be a function defined on a set X. Then $f(X')$ denotes the image of a subset $X' \subseteq X$ under f.

We will use the following definition of a p-morphism:

Definition 1.5 A surjective map $f : W' \to W''$ such that
$$f(R_i'(x)) = R_i''(f(x)) \text{ for all } x \in W',$$
is said to be a *p-morphism from* F' *onto* F''; notation: $f : F' \twoheadrightarrow F''$.

Recall that a formula $\phi(p_1, \ldots, p_k)$ not containing modalities is said to be a *perfect disjunctive normal form* if
$$\phi(p_1, \ldots, p_k) = \bigvee_{\alpha \in S} p_1^{\alpha_1} \wedge \cdots \wedge p_k^{\alpha_k},$$
where S is a (probably, empty) set of boolean sequences of length k, $p^1 \leftrightharpoons p$, $p^0 \leftrightharpoons \neg p$. The well-known fact is that every formula without modalities is equivalent to some perfect disjunctive normal form.

2 Monoid Actions and Congruences.

This section contains some standard algebraic facts; they are very well-known for the case of groups and are easily extended for monoids. For the sake of generality, here we consider the non-commutative case, but later in this paper only commutative monoids will be used.

Definition 2.1 A *(left) action of a monoid G on a set W* is a map $\alpha : G \times W \to W$ such that for any $g, h \in G$, $x \in W$

$$\alpha(g * h, x) = \alpha(g, \alpha(h, x))$$

and

$$\alpha(1, x) = x,$$

where $*$ is the multiplication, 1 is the unit in G. The triple

$$\mathcal{W} = (G, W, \alpha)$$

is then called a *G-set*. We will also use the standard notation $g * x$ instead of $\alpha(g, x)$.

Definition 2.2 A G-set $\mathcal{W} = (G, W, \alpha)$ is said to be *rooted (with root u)* if

$$\forall x \in W \ \ \exists g \in G \ \ x = g * u.$$

In this case, the pair $(\mathcal{W}, u)$ is called a *conic G-set* (with *root* u).

Recall also the standard definitions of morphisms.

Definition 2.3 A *morphism* from a G-set $\mathcal{W}_1 = (G, W_1, \alpha_1)$ to a G-set $\mathcal{W}_2 = (G, W_2, \alpha_2)$ is a map $f : W_1 \to W_2$ such that for any g, x

$$f(\alpha_1(g, x)) = \alpha_2(g, f(x)).$$

A morphism of conic G-sets $f : (\mathcal{W}_1, u_1) \to (\mathcal{W}_2, u_2)$ is a morphism of the corresponding G-sets $f : \mathcal{W}_1 \to \mathcal{W}_2$ such that $f(u_1) = u_2$.

As usual, an *isomorphism* is a morphism having a converse, or equivalently, a bijective morphism.

Definition 2.4 Recall that a *left congruence on a semigroup* $(G, *)$ is an equivalence relation $\sim$ on G such that (for any a, b, c) $a \sim b$ implies $c * a \sim c * b$.

Definition 2.5 For a conic G-set $(\mathcal{W}, u)$ define a relation $\sim_{(\mathcal{W},u)}$ on G in the following way:

$$a \sim_{(\mathcal{W},u)} b \text{ iff } a * u = b * u,$$

where $*$ is the action in $\mathcal{W}$.

Lemma 2.6 For a conic G-set $(\mathcal{W}, u)$ the relation $\sim_{(\mathcal{W},u)}$ is a left congruence on G. This defines a functor from G-SETC to G-CON.

PROOF. One can easily check that $\sim_{(\mathcal{W},u)}$ is a left congruence. Now

$$f : (\mathcal{W}_1, u_1) \to (\mathcal{W}_2, u_2)$$

implies $\sim_{(\mathcal{W}_1,u_1)} \subseteq \sim_{(\mathcal{W}_2,u_2)}$. In fact, if $a * u_1 = b * u_1$ in $\mathcal{W}_1$, then

$$a * u_2 = a * f(u_1) = f(a * u_1) = f(b * u_1) = b * f(u_1) = b * u_2.$$

∎

Definition 2.7 For a left congruence $\sim$ on a semigroup G define a tuple

$$\mathcal{X}(\sim) \leftrightharpoons ((G, G/\sim, *), [1])^3,$$

where $G/\sim$ is the set of equivalence classes modulo $\sim$, $*$ is a function from $G \times G/\sim$ to $G/\sim$ defined as follows:

$$g * [a] \leftrightharpoons [g * a]\,.$$

Lemma 2.8 For every left congruence $\sim$ on G $\mathcal{X}(\sim)$ is a conic G-set. Furthermore, if $\sim_1 \subseteq \sim_2$, then the map $f : G/\sim_1 \to G/\sim_2$ such that $f([a]_1) = [a]_2$ (where $[b]_i$ is the class of b modulo $\sim_i$) is a surjective (conic) G-set morphism. This defines a functor from G-CON to G-SETC.

PROOF. It is easily checked that $*$ is an action with root [1]. We check that f is a morphism:

$$f(b * [a]_1) = f([b * a]_1) = [b * a]_2 = b * [a]_2\,.$$

∎

Lemmas 2.6 and 2.8 mean that the following two categories (for a discussion of categories see [2]) are equivalent:

- the category G-SETC of conic G-sets and surjective morphisms;
- the category G-CON of left congruences on G and inclusions.

3 $\mathbf{SL}^n$-cones and Congruences on $\mathcal{N}^n$.

Let us consider actions of the additive monoid $\mathcal{N}^n$.

Proposition 3.1 The following defines a correspondence between $\mathcal{N}^n$-actions and $\mathbf{SL}^n$-frames on a set W:

1. an $\mathcal{N}^n$-action $\alpha : \mathbb{N}^n \times W \to W$ corresponds to the frame $(W, R_1, \ldots, R_n)$, where for every i, x
$$R_i(x) \leftrightharpoons \{\alpha(\mathbf{e}_i, x)\}\,;$$
2. an $\mathbf{SL}^n$-frame $(W, R_1, \ldots, R_n)$ corresponds to the $\mathcal{N}^n$-action $\alpha : \mathbb{N}^n \times W \to W$ such that for every $x \in W$, $\mathbf{m} \in \mathbb{N}^n$
$$\{\alpha(\mathbf{m}, x)\} = R^{\mathbf{m}}(x),$$
where
$$R^{(m^1,\ldots,m^n)} \leftrightharpoons R_1^{m^1} \circ \cdots \circ R_n^{m^n}.$$

[3] Here $[a]$ is the equivalence class of a

PROOF. Almost obvious. In the frame defined by (1) we have

$$R_i \circ R_j = R_j \circ R_i$$

since $\mathcal{N}^n$ is commutative.

If α is defined by (2) then $\{\alpha(\mathbf{e}_i, x)\} = R_i(x)$ as required in (1).

Conversely, if $R_1, \ldots, R_n$ are defined by (1), then

$$\left\{\alpha(m^1\mathbf{e}_1 + \cdots + m^n\mathbf{e}_n, x)\right\} = (R_1^{m^1} \circ \cdots \circ R_n^{m^n})(x).$$

∎

So every $\mathbf{SL}^n$-frame corresponds to an $\mathcal{N}^n$-set. Also it follows that p-morphisms correspond to morphisms of $\mathcal{N}^n$-sets:

Proposition 3.2 Let F', F'' be $\mathbf{SL}^n$-frames. A surjection $f : F' \to F''$ is a p-morphism iff for all i, x

$$f(\mathbf{e}_i + x) = \mathbf{e}_i + f(x).$$

PROOF. Follows from the definition of a p-morphism, since

$$\begin{array}{lcl} R_i'(x) & = & \{\mathbf{e}_i + x\}, \\ R_i''(f(x)) & = & \{\mathbf{e}_i + f(x)\} \end{array}$$

by the definition of R_i in Proposition 3.1(1). ∎

Now we obtain that $\mathbf{SL}^n$-cones correspond to congruences on $\mathcal{N}^n$:

Proposition 3.3 The following defines a correspondence between $\mathbf{SL}^n$-cones and congruences on $\mathcal{N}^n$:

1. a cone C with root u corresponds to the congruence $\sim_C$ such that

$$\mathbf{a} \sim_C \mathbf{b} \text{ iff } R^{\mathbf{a}}(u) = R^{\mathbf{b}}(u) \text{ in } C;$$

2. a congruence $\sim$ corresponds to the cone C with the set of worlds $W \leftrightharpoons \mathbb{N}^n/\!\sim$, the relations

$$[\mathbf{x}]\, R_i\, [\mathbf{y}] \text{ iff } [\mathbf{y}] = [\mathbf{e}_i + \mathbf{x}],$$

and root $[\mathbf{0}]$.

PROOF. Follows from 2.6–3.2. ∎

4 $\mathbf{SL}^n$-normal Forms.

Notation 4.1 For $\mathbf{m} = (m^1, \ldots, m^n) \in \mathbb{N}^n$ let

$$\Box^{\mathbf{m}} \leftrightharpoons \Box_1^{m^1} \ldots \Box_n^{m^n}.$$

Definition 4.2 Let A be an n-modal formula. Let us define the *degrees of* A in the following way:

$$\begin{array}{lcl} d_i(\bot) & \leftrightharpoons & 0, \\ d_i(p) & \leftrightharpoons & 0, \\ d_i(A_1 \supset A_2) & \leftrightharpoons & \max(d_i(A_1), d_i(A_2)), \\ d_i(\Box_j A) & \leftrightharpoons & d_i(A), \text{ if } i \neq j, \\ d_i(\Box_i A) & \leftrightharpoons & d_i(A) + 1. \end{array}$$

Lemma 4.3 Let $A(p_1,\dots,p_k)$ be an n-modal formula, $\mathbf{d}=(d^1,\dots,d^n)\in\mathbb{N}^n$, and $d_i(A)\le d^i$. Then

$$\mathbf{SL}^n\vdash A\equiv\phi(p_1,\dots,p_k,\dots,\Box^{\mathbf{d}}p_1,\dots,\Box^{\mathbf{d}}p_k)$$

for some formula ϕ not containing modalities (here the formula ϕ contains $k(d^1+1)\cdots(d^n+1)$ variables; the $(i,a^1,\dots,a^n)$-th variable is substituted by $\Box^{(a^1,\dots,a^n)}p_i$).

Proof. By induction on the construction of A. Only the case $A=\Box_iB$ is not trivial.

By the inductive hypothesis

$$B=\psi(p_1,\dots,p_k,\dots,\Box^{\mathbf{c}}p_1,\dots,\Box^{\mathbf{c}}p_k),$$

in this case, $c^j\le d^j$ for $i\ne j$ and $c^i+1\le d^i$.

Then we apply induction on the construction of ψ. Obviously,

$$\begin{array}{lcl}\Box_i\bot & \equiv & \bot,\\ \Box_i(\psi_1\supset\psi_2) & \equiv & \Box_i\psi_1\supset\Box_i\psi_2.\end{array}$$

Hence

$$\Box_iB\equiv\psi(\Box_ip_1,\dots,\Box_ip_k,\dots,\Box^{\mathbf{c}+\mathbf{e}_i}p_1,\dots,\Box^{\mathbf{c}+\mathbf{e}_i}p_k),$$

what completes the proof. ∎

5 Properties of Finitely Axiomatisable Extensions of $\mathbf{SL}^n$.

Lemma 5.1 Let C be an $\mathbf{SL}^n$-cone with root u, A a formula. Then $C\vDash A$ iff $(C,u)\vDash A$.

Proof. The "only if" case is obvious. To show "if", assume $(C,u)\vDash A$.

We need to show that $(C,x)\vDash A$ for every point $x=\mathbf{m}+u$.

Let us construct the map f as follows: $f(y)\leftrightharpoons\mathbf{m}+y$. Obviously, f is a p-morphism of C onto C^x. Hence $(C^x,x)\vDash A$ and thus $(C,x)\vDash A$. ∎

Let us introduce the following notation: for $\mathbf{a},\mathbf{b}\in\mathbb{Z}^n$, $[\mathbf{a},\mathbf{b}]$ denotes the set of all $\mathbf{c}\in\mathbb{Z}^n$ such that $a^i\le c^i\le b^i$.

For a relation R on a set X, $Y\subseteq X$, let $R|_Y\leftrightharpoons R\cap Y^2$.

Proposition 5.2 Let $\sim_1$, $\sim_2$ be congruences on $\mathcal{N}^n$, A a formula, $\mathbf{d}=(d^1,\dots,d^n)\in\mathbb{N}^n$, $d_i(A)\le d^i$, $D\leftrightharpoons[\mathbf{0},\mathbf{d}]$, and

$$\sim_1|_D=\sim_2|_D.$$

Let C_i be the cone corresponding to $\sim_i$, u_i the root of C_i. Then $C_1\vDash A$ iff $C_2\vDash A$.

Proof. By Lemma 5.1, it is sufficient to construct for every valuation θ_1 in C_1 a valuation θ_2 in C_2 such that $u_1\in\theta_1(A)$ iff $u_2\in\theta_2(A)$.

Let us construct θ_2 as follows:

$$\mathbf{m}+u_2\in\theta_2(p)\text{ iff }\mathbf{m}+u_1\in\theta_1(p)\text{ whenever }m^i\le d_i(A),$$

in other points let us define θ_2 arbitrarily. θ_2 is well-defined due to the assumption.

By Lemma 4.3, $u_1\in\theta_1(A)$ iff $u_2\in\theta_2(A)$. ∎

Theorem 5.3 If the logic $\mathbf{SL}^n{+}A$ is consistent, then there exist $\mathbf{SL}^n$-cones $C_1, \ldots, C_k$ such that

$$\mathbf{SL}^n + A = \mathrm{L}(\{C_1, \ldots, C_k\}).$$

Proof. Let $d^i \leftrightharpoons d_i(A)$ be the corresponding degrees of the formula A, $\mathbf{d} \leftrightharpoons (d^1, \ldots, d^n)$, $D \leftrightharpoons [\mathbf{0}, \mathbf{d}] \subseteq \mathbb{N}^n$.

Let E be the set of all equivalence relations on D. E is finite thanks to the finiteness of D.

For $e \in E$ let

$$\mathcal{C}_e \leftrightharpoons \{C \mid C \vDash \mathbf{SL}^n + A \ \ \& \ \ {\sim_C}|_D = e\}.$$

Consider an equivalence e such that $\mathcal{C}_e \neq \varnothing$.

Define the relation $\sim_e$ as follows:

$$\mathbf{a} \sim_e \mathbf{b} \text{ iff } \forall C \in \mathcal{C}_e \ \ \mathbf{a} \sim_C \mathbf{b}.$$

Obviously, $\sim_e$ is a congruence. Let $C(e)$ be the cone corresponding to $\sim_e$.

Since ${\sim_C}|_D = e$ for all $C \in \mathcal{C}_e$, it follows that ${\sim_e}|_D = e$. Because $\mathcal{C}_e$ is not empty, there exists $C \in \mathcal{C}_e$ such that ${\sim_C}|_D = e = {\sim_e}|_D$, so $C(e) \vDash A$ by Proposition 5.2, and thus $C(e) \in \mathcal{C}_e$.

Since $\sim_e \subseteq \sim_C$, it follows that $\mathrm{L}(C(e)) \subseteq \mathrm{L}(C)$ for all $C \in \mathcal{C}_e$.

Therefore $\mathrm{L}(\mathcal{C}_e) = \mathrm{L}(C(e))$.

By completeness of $\mathbf{SL}^n + A$:

$$\mathbf{SL}^n + A = \mathrm{L}(\{C \mid C \vDash \mathbf{SL}^n + A\}) = \mathrm{L}(\bigcap \{\mathcal{C}_e \mid e \in E \ \ \& \ \ \mathcal{C}_e \neq \varnothing\}) =$$
$$\bigcap \{\mathrm{L}(\mathcal{C}_e) \mid e \in E \ \ \& \ \ \mathcal{C}_e \neq \varnothing\} = \bigcap \{\mathrm{L}(C_e) \mid e \in E \ \ \& \ \ \mathcal{C}_e \neq \varnothing\}$$

The theorem follows from the finiteness of the set $\{e \in E \mid \mathcal{C}_e \neq \varnothing\}$. ∎

Notation 5.4 Let $A(p_1, \ldots, p_k)$ and $B(p_1, \ldots, p_k)$ be n-modal formulas. Then let $A \sqcup B \leftrightharpoons A(q'_1, \ldots, q'_k) \vee B(q''_1, \ldots, q''_k)$ (q'_i and q''_i are new distinct propositional variables).

Proposition 5.5 Let C be an $\mathbf{SL}^n$-cone, $A_1, \ldots, A_k$ formulas. Then

$$C \vDash A_1 \sqcup \cdots \sqcup A_k \Leftrightarrow \exists i \ \ C \vDash A_i.$$

Proof. Due to Lemma 5.1, is sufficient to prove the proposition for validity of formulas at the root of C. Then the claim ($\Leftarrow$) is obvious, the claim ($\Rightarrow$) follows because we can assume that the variables in all A_i are distinct. ∎

Lemma 5.6 Let $\Lambda_i = \mathbf{SL}^n + A_i$, $1 \leq i \leq k$. Then

$$\bigcap_i \Lambda_i = \mathbf{SL}^n + A_1 \sqcup \cdots \sqcup A_k.$$

Proof. The inclusion ($\supseteq$) is obvious.

If $A \notin \mathbf{SL}^n + A_1 \sqcup \cdots \sqcup A_k$, then there exists an $\mathbf{SL}^n$-cone C such that $C \vDash A_1 \sqcup \cdots \sqcup A_k$ and $C \nvDash A$. And so there exists i such that $C \vDash A_i$, thus $\Lambda_i \nvDash A$. ∎

6 Extensions of the Logic SL.

Definition 6.1

1. Define the relation $\sim_{m,n}$ for $m \geq 0$, $n > 0$ in the following way:

$$a \sim_{m,n} b \text{ iff } a = b \vee (a \geq m \ \& \ b \geq m \ \& \ n \text{ divides } a - b).$$

2. Define the following **SL**-cones:
   - C_0 is the cone corresponding to the relation $=$,
   - $\mathrm{C}_{m,n}$ is the cone corresponding to the relation $\sim_{m,n}$.
3. For $m \geq 0$, $n > 0$ let $\mathrm{A}_{m,n} \leftrightharpoons \Box^m p \equiv \Box^{m+n} p$.

Proposition 6.2 $\mathrm{C}_{k,l} \vDash \mathrm{A}_{m,n}$ iff l divides n and $k \leq m$.

Proof.
($\Leftarrow$) Note that $\mathrm{C}_{m,n} \vDash \mathrm{A}_{m,n}$.

Consider the map $f : \mathrm{C}_{m,n} \to \mathrm{C}_{k,l}$ such that $f([x]_{\sim_{m,n}}) = [x]_{\sim_{k,l}}$. f is well-defined since l divides n and $k \leq m$. It is obvious that f is a p-morphism of $\mathrm{C}_{m,n}$ onto $\mathrm{C}_{k,l}$, hence $\mathrm{C}_{k,l} \vDash \mathrm{A}_{m,n}$.

($\Rightarrow$) Let us define the valuation θ in $\mathrm{C}_{k,l}$ as follows: $\theta(p) \leftrightharpoons \{\, [m]_{\sim_{k,l}} \}$.

Note that $(\mathrm{C}_{k,l}, [0]\,, \theta) \vDash \Box^m p$.

If $k > m$ or l does not divide n, then $[m+n] \neq [m]$, so $(\mathrm{C}_{k,l}, [0]\,, \theta) \nvDash \Box^{m+n} p$, therefore $\mathrm{C}_{k,l} \nvDash \mathrm{A}_{m,n}$.

∎

Lemma 6.3

1. All $\mathrm{C}_{m,n}$, C_0 are pairwise non-isomorphic.
2. If C is an **SL**-cone, then C is isomorphic either to C_0 or to $\mathrm{C}_{k,l}$ for some k, l.

Proof. It is obvious that C_0 is not isomorphic to any of $\mathrm{C}_{m,n}$.

Let C be an **SL**-cone with root u.

If all $n + u$, $n \in \mathbb{N}$, are different, then obviously C is isomorphic to C_0.

Otherwise let us choose the minimal m such that $m + u = m + n + u$ for some $n > 0$, and the minimal such n. Then C is isomorphic to $\mathrm{C}_{m,n}$ (since $\sim_C = \sim_{m,n}$). ∎

Lemma 6.4 The logics of the cones of **SL** are the following:

1. $\mathrm{L}(\mathrm{C}_0) = \mathbf{SL}$,
2. $\mathrm{L}(\mathrm{C}_{m,n}) = \mathbf{SL} + \mathrm{A}_{m,n}$.

Proof.
1. Follows from 1.2.

2. The inclusion ($\supseteq$) is obvious.

If $A \notin \mathbf{SL} + \mathrm{A}_{m,n}$, then there exists an **SL**-cone C such that $C \vDash \mathrm{A}_{m,n}$ and $C \nvDash A$. By Lemma 6.3, $C = \mathrm{C}_{k,l}$ for some k and l since $\mathrm{C}_0 \nvDash \mathrm{A}_{m,n}$. By Proposition 6.2, l divides n and $k \leq m$. Hence the map $f : \mathrm{C}_{m,n} \to \mathrm{C}_{k,l}$ defined as $f([a]_{\sim_{m,n}}) \leftrightharpoons [a]_{\sim_{k,l}}$ is a p-morphism, and so $\mathrm{C}_{m,n} \nvDash A$.

∎

Remark 6.5 If $A(p_1, \ldots, p_k)$ is a 1-modal formula, $d_1(A) \leq d$, then

$$\mathbf{SL} \vdash A \equiv \phi(p_1, \ldots, p_k, \ldots, \Box^d p_1, \ldots, \Box^d p_k)$$

for some formula ϕ not containing modalities.

The following theorem was proved first (using another method) by A. A. Muchnik [3, Theorem 1].

Theorem 6.6 Every consistent proper extension of **SL** is tabular.

Proof. Consider an extension $\Lambda \supseteq \mathbf{SL}$. Since every extension of a tabular logic is tabular, we can consider only the case when $\Lambda = \mathbf{SL} + A$, where $\mathbf{SL} \nvdash A$.

Let d be the degree of A, then

$$\mathbf{SL} \vdash A \equiv \phi(p_1, \ldots, p_k, \ldots, \Box^d p_1, \ldots, \Box^d p_k).$$

Let us write ϕ in the perfect disjunctive normal form:

$$\mathbf{SL} \vdash \phi(p_1, \ldots, p_k, \ldots, \Box^d p_1, \ldots, \Box^d p_k) \equiv \bigvee_{\alpha \in S} p_1^{\alpha_1^0} \wedge \cdots \wedge p_k^{\alpha_k^0} \wedge \cdots \wedge \Box^d p_1^{\alpha_1^d} \wedge \cdots \wedge \Box^d p_k^{\alpha_k^d}.$$

Here S is a set of binary matrices (α_j^i), where $0 \leq i \leq d$, $1 \leq j \leq k$.

Note that S does not contain all such matrices, for otherwise

$$\mathbf{SL} \vdash \phi(q_1^0, \ldots, q_k^0, \ldots, q_1^d, \ldots, q_k^d) \equiv \top,$$

what contradicts to $\mathbf{SL} \nvdash A$.

Then there exists a matrix $\tilde{\alpha}$ such that $\tilde{\alpha} \notin S$.

Let us prove that every cone validating A contains no more than d elements.

In fact, suppose there exists a cone C with root u such that $C \models A$ and C contains more than d elements. Let us evaluate p_i at $j + u$ as $\tilde{\alpha}_i^j$ for $0 \leq j \leq d$. The valuation of p_i is well-defined since all $j + u$, $0 \leq j \leq d$ are different. But under this valuation the disjunction turns out to be false, which contradicts to $C \models A$.

There exist finitely many non-isomorphic cones with no more than d elements. By completeness Λ is the logic of cones of this kind and so it is tabular. ∎

Theorem 6.7 If Λ is an extension of **SL**, then $\Lambda = \mathbf{SL} + \mathrm{A}_\Lambda$ where $\mathrm{A}_\Lambda = \top$ or $\mathrm{A}_\Lambda = \mathrm{A}_{m_1,n_1} \sqcup \cdots \sqcup \mathrm{A}_{m_k,n_k}$, $k \geq 0$.

Proof. This follows from Lemma 5.6 and the fact that every consistent proper extension of **SL** is a logic of a finite number of cones of the form $\mathrm{C}_{m,n}$. ∎

7 SL.t^n-cones and Congruences in $\mathcal{Z}^n$.

Let us consider actions of the additive group $\mathcal{Z}^n$.

Proposition 7.1 The following is a correspondence between $\mathcal{Z}^n$-actions and **SL**.t^n-frames on a set W:

1. a $\mathcal{Z}^n$-action $\alpha : \mathbb{Z}^n \times W \to W$ corresponds to the frame $(W, R_1, \ldots, R_{2n})$, where for every i, x
$$\begin{array}{lcl} R_{2i-1}(x) & \leftrightharpoons & \{\alpha(\mathbf{e}_i, x)\}, \\ R_{2i}(x) & \leftrightharpoons & \{\alpha(-\mathbf{e}_i, x)\}; \end{array}$$

2. an $\mathbf{SL}.\mathbf{t}^n$-frame $(W, R_1, \ldots, R_{2n})$ corresponds to the $\mathcal{Z}^n$-action $\alpha : \mathbb{Z}^n \times W \to W$ such that for every $x \in W$, $\mathbf{m} \in \mathbb{Z}^n$
$$\{\alpha(\mathbf{m}, x)\} = R^{\mathbf{m}}(x),$$
where
$$R^{(m^1,\ldots,m^n)} \leftrightharpoons \tilde{R}_1^{m^1} \circ \cdots \circ \tilde{R}_n^{m^n},$$
$\tilde{R}_i^k = R_{2i-1}^k$ if $k \geq 0$, $\tilde{R}_i^k = R_{2i}^{-k}$ if $k < 0$.

PROOF. Almost the same as for the case of $\mathbf{SL}^n$. ∎

So every $\mathbf{SL}.\mathbf{t}^n$-frame corresponds to a $\mathcal{Z}^n$-set. Also it follows that p-morphisms correspond to morphisms of $\mathcal{Z}^n$-sets:

Proposition 7.2 Let F' and F'' be $\mathbf{SL}.\mathbf{t}^n$-frames. A surjection $f : F' \to F''$ is a p-morphism iff for all i, x
$$f(\mathbf{e}_i + x) = \mathbf{e}_i + f(x).$$

PROOF. Almost the same as for the case of $\mathbf{SL}^n$. ∎

Now let us show that $\mathbf{SL}.\mathbf{t}^n$-cones correspond to congruences on $\mathcal{Z}^n$.

Proposition 7.3 The following is a correspondence between $\mathbf{SL}.\mathbf{t}^n$-cones and congruences on $\mathcal{Z}^n$:

1. a cone C with root u corresponds to the congruence $\sim_C^{\mathrm{t}}$ such that
$$\mathbf{a} \sim_C^{\mathrm{t}} \mathbf{b} \text{ iff } R^{\mathbf{a}}(u) = R^{\mathbf{b}}(u) \text{ in } C;$$

2. a congruence $\sim$ corresponds to the cone C with the set of worlds $W \leftrightharpoons \mathbb{Z}^n/\!\sim$, the relations
$$\begin{array}{lll} [\mathbf{x}]\, R_{2i-1}\, [\mathbf{y}] & \text{iff} & [\mathbf{y}] = [\mathbf{e}_i + \mathbf{x}], \\ [\mathbf{x}]\, R_{2i}\, [\mathbf{y}] & \text{iff} & [\mathbf{y}] = [-\mathbf{e}_i + \mathbf{x}], \end{array}$$
and root $[\mathbf{0}]$.

PROOF. Almost the same as for the case of $\mathbf{SL}^n$. ∎

8 $\mathbf{SL}.\mathbf{t}^n$-cones and Subgroups of $\mathcal{Z}^n$.

Proposition 8.1 The following is a correspondence between subgroups of $\mathcal{Z}^n$ and congruences on $\mathcal{Z}^n$:

1. a subgroup of $\mathcal{Z}^n$ M corresponds to the congruence $\sim_M^{\mathrm{t}}$, where for every $\mathbf{a}, \mathbf{b}$
$$\mathbf{a} \sim_M^{\mathrm{t}} \mathbf{b} \Leftrightarrow \mathbf{a} - \mathbf{b} \in M;$$

2. a congruence on $\mathcal{Z}^n$ $\sim$ corresponds to the subgroup of $\mathcal{Z}^n$ $M \leftrightharpoons [\mathbf{0}]_\sim$.

PROOF. This fact is well-known. ∎

Proposition 8.2 Let M_1, M_2 be subgroups of $\mathcal{Z}^n$, $\sim_i$ the congruence corresponding to M_i. Then

$$M_1 \subseteq M_2 \Leftrightarrow \sim_1 \subseteq \sim_2.$$

PROOF. Straightforward. ∎

Now let us show that $\mathbf{SL.t}^n$-cones correspond to subgroups of $\mathcal{Z}^n$.

Corollary 8.3 The following is a correspondence between $\mathbf{SL.t}^n$-cones and subgroups of $\mathcal{Z}^n$:

1. a cone C with root u corresponds to the group

$$M = \{\mathbf{x} \in \mathbb{Z}^n \mid \mathbf{x} + u = u\};$$

2. a group M corresponds to the cone C with the set of worlds $W \leftrightharpoons \mathbb{Z}^n/M$, the relations

$$\begin{array}{lcl} [\mathbf{x}]\, R_{2i-1}\, [\mathbf{y}] & \leftrightharpoons & [\mathbf{y}] = [\mathbf{e}_i + \mathbf{x}]\,, \\ [\mathbf{x}]\, R_{2i}\, [\mathbf{y}] & \leftrightharpoons & [\mathbf{y}] = [-\mathbf{e}_i + \mathbf{x}]\,, \end{array}$$

and root $[\mathbf{0}]$.

Corollary 8.4 Let M_1, M_2 be subgroups of $\mathcal{Z}^n$ such that $M_1 \subseteq M_2$, C_i the cone corresponding to M_i. Then the map $f : C_1 \to C_2$ sending $\mathbf{a} + M_1$ to $\mathbf{a} + M_2$ is a p-morphism.

9 Description of Finitely Axiomatisable Extensions of $\mathbf{SL.t}^n$.

Proposition 9.1 Every subgroup of $\mathcal{Z}^n$ is generated by some linearly independent vectors $\mathbf{l}_1, \ldots, \mathbf{l}_k$, where $0 \leq k \leq n$.

PROOF. This follows from the fact that $\mathcal{Z}^n$ is a finitely generated free Abelian group. ∎

Notation 9.2 For $\mathbf{m} = (m^1, \ldots, m^n) \in \mathbb{Z}^n$ let

$$\Box^{\mathbf{m}} \leftrightharpoons \boxdot_1^{m^1} \ldots \boxdot_n^{m^n},$$

where $\boxdot_i^k = \Box_{2i-1}^k$ if $k \geq 0$, $\boxdot_i^k = \Box_{2i}^{-k}$ if $k < 0$.

Definition 9.3 Let M be the subgroup generated by $\mathbf{l}_1, \ldots, \mathbf{l}_k$. Let us define the formula At_M:

$$\mathrm{At}_M \leftrightharpoons p \equiv \Box^{\mathbf{l}_1} p \wedge \cdots \wedge p \equiv \Box^{\mathbf{l}_k} p.$$

This definition produces equivalent formulas At_M for different bases of M due to the following lemma.

Lemma 9.4 Let M be generated by $\mathbf{l}_1, \ldots, \mathbf{l}_k$; $\mathbf{l} \in M$. Then

$$\mathbf{SL.t}^n \vdash \mathrm{At}_M \supset p \equiv \Box^{\mathbf{l}} p.$$

PROOF. Since $\mathbf{l} \in M$, it follows that

$$\mathbf{l} = \sum_{i=1}^{k} t_i \mathbf{l}_i.$$

for some $t_i \in \mathbb{Z}$. To complete the proof it is sufficient to note that

$$\mathbf{SL}.\mathbf{t}^n \vdash \mathrm{At}_M \supset p \equiv \Box^{t_i \mathbf{l}_i} p.$$

∎

Lemma 9.5 Let C be the $\mathbf{SL}.\mathbf{t}^n$-cone corresponding to a group $M \subseteq \mathbb{Z}^n$. Then:

$$\mathrm{L}(C) = \mathbf{SL}.\mathbf{t}^n + \mathrm{At}_M.$$

PROOF. The inclusion ($\supseteq$) is obvious.

Let M be the subgroup of $\mathcal{Z}^n$ generated by $\mathbf{l}_1, \ldots, \mathbf{l}_k$.

Consider $A \notin \mathbf{SL}.\mathbf{t}^n + \mathrm{At}_M$. Then by completeness there exists a cone C' such that $C' \vDash \mathrm{At}_M$ and $C' \nvDash A$. Let N be the group corresponding to C'.

Note that due to $C' \vDash \mathrm{At}_M$ we have $\mathbf{l}_i + N = N$ (otherwise we can evaluate p in $\mathbf{l}_i + N$ and in N in different ways). Thus $M \subseteq N$.

Let us construct the map $f : C \to C'$ as $f(\mathbf{a} + M) \leftrightharpoons \mathbf{a} + N$. f is well-defined since $M \subseteq N$.

Obviously, f is a p-morphism, hence $C \nvDash A$. ∎

So it is possible to axiomatise every logic of a finite number of cones:

Corollary 9.6 Let $M_1, \ldots, M_k$ be the groups corresponding to $\mathbf{SL}.\mathbf{t}^n$-cones $C_1, \ldots, C_k$. Then

$$\mathrm{L}(\{C_1, \ldots, C_k\}) = \mathbf{SL}.\mathbf{t}^n + \mathrm{At}_{M_1} \sqcup \cdots \sqcup \mathrm{At}_{M_k}.$$

PROOF. Follows from Lemma 5.6 and Lemma 9.5. ∎

Therefore it is possible to describe finitely axiomatisable extensions of $\mathbf{SL}.\mathbf{t}^n$:

Corollary 9.7 $\Lambda \supseteq \mathbf{SL}.\mathbf{t}^n$ is finitely axiomatisable iff

$$\Lambda = \mathrm{L}(\{C_1, \ldots, C_k\}).$$

Corollary 9.8 If $\Lambda \supseteq \mathbf{SL}.\mathbf{t}^n$ is finitely axiomatisable, then

$$\Lambda = \mathbf{SL}.\mathbf{t}^n + \mathrm{At}_\Lambda,$$

where $\mathrm{At}_\Lambda = \top$ or $\mathrm{At}_\Lambda = \mathrm{At}_{M_1} \sqcup \cdots \sqcup \mathrm{At}_{M_k}$, $k \geq 0$.

10 $\mathbf{SL}.\mathbf{t}^n$-normal Forms.

Definition 10.1 Let A be a $2n$-modal formula. Let us define the *degrees of A* in the following way:

$$\begin{array}{lcl}
dt_i(\bot) & \leftrightharpoons & 0, \\
dt_i(p) & \leftrightharpoons & 0, \\
dt_i(A_1 \supset A_2) & \leftrightharpoons & \max(dt_i(A_1), dt_i(A_2)), \\
dt_i(\Box_{2j-1} A) & \leftrightharpoons & dt_i(A), \text{ if } i \neq j, \\
dt_i(\Box_{2i-1} A) & \leftrightharpoons & dt_i(A) + 1, \\
dt_i(\Box_{2j} A) & \leftrightharpoons & dt_i(A), \text{ if } i \neq j, \\
dt_i(\Box_{2i} A) & \leftrightharpoons & dt_i(A) + 1.
\end{array}$$

Lemma 10.2 Let $A(p_1,\dots,p_k)$ be a $2n$-modal formula, $\mathbf{d}=(d^1,\dots,d^n)\in\mathbb{N}^n$, and $dt_i(A)\le d^i$. Then

$$\mathbf{SL}.\mathbf{t}^n\vdash A\equiv\phi(\Box^{-\mathbf{d}}p_1,\dots,\Box^{-\mathbf{d}}p_k,\dots,p_1,\dots,p_k,\dots,\Box^{\mathbf{d}}p_1,\dots,\Box^{\mathbf{d}}p_k)$$

for some formula ϕ not containing modalities (here the formula ϕ contains $k(2d^1+1)\cdots(2d^n+1)$ variables; the $(i,a^1,\dots,a^n)$-th variable is substituted by $\Box^{(a^1,\dots,a^n)}p_i$).

Proof. The claim is proved similarly to the case of $\mathbf{SL}^n$. ∎

Proposition 10.3 Let $\sim_1$, $\sim_2$ be congruences on $\mathcal{Z}^n$, A a formula, $\mathbf{d}=(d^1,\dots,d^n)\in\mathbb{Z}^n$, $dt_i(A)\le d^i$, $D\leftrightharpoons[-\mathbf{d},\mathbf{d}]$, and

$$\sim_1|_D=\sim_2|_D.$$

If C_i is an $\mathbf{SL}.\mathbf{t}^n$-cone corresponding to $\sim_i$, then $C_1\vDash A$ iff $C_2\vDash A$.

Proof. The claim is proved similarly to the case of $\mathbf{SL}^n$. ∎

11 The Finite Model Property for Extensions of $\mathbf{SL}.\mathbf{t}^n$.

Proposition 11.1 If M is a subgroup of $\mathcal{Z}^n$ of rank n, then $\mathbb{Z}^n/M$ is finite.

Proof. Let M be the subgroup of $\mathcal{Z}^n$ generated by $\mathbf{l}_1,\dots,\mathbf{l}_n$. Let us construct $D\subseteq\mathbb{Z}^n$:

$$D\leftrightharpoons\{\mathbf{a}\in\mathbb{Z}^n\mid\mathbf{a}=v_1\mathbf{l}_1+\dots+v_n\mathbf{l}_n,v_i\in\mathbb{R},0\le v_i<1\}.$$

Let us show that for all $\mathbf{b}\in\mathbb{Z}^n$ there exists $\mathbf{a}\in D$ such that $\mathbf{b}\sim^{\mathrm{t}}_M\mathbf{a}$:

$$\exists t_1,\dots,t_n\in\mathbb{R}\ \ \mathbf{b}=t_1\mathbf{l}_1+\dots+t_n\mathbf{l}_n\Rightarrow$$
$$\mathbf{b}\sim^{\mathrm{t}}_M t_1\mathbf{l}_1+\dots+t_n\mathbf{l}_n\sim^{\mathrm{t}}_M s_1\mathbf{l}_1+\dots+s_n\mathbf{l}_n$$

where s_i is the fractal part of t_i; such t_i exist since the rank of M is n. Therefore it is sufficient to take $a\leftrightharpoons s_1\mathbf{l}_1+\dots+s_n\mathbf{l}_n\in D$.

Then $|\mathbb{Z}^n/M|\le|D|<\infty$, D is finite by construction. ∎

Lemma 11.2 Given a formula A and an $\mathbf{SL}.\mathbf{t}^n$-cone C such that $C\nvDash A$, there exists a finite cone C' such that $C\twoheadrightarrow C'$ and $C'\nvDash A$.

Proof. Let $d^i\leftrightharpoons dt_i(A)$ be the degrees of the formula A, M the group corresponding to C. Let M be generated by linearly independent vectors $\mathbf{l}_1,\dots,\mathbf{l}_k$.

Let $\mathbf{d}\leftrightharpoons(d^1,\dots,d^n)$, $D\leftrightharpoons[-\mathbf{d},\mathbf{d}]\subseteq\mathbb{Z}^n$.

Then there exist $\mathbf{l}_{k+1},\dots,\mathbf{l}_n\in\mathbb{Z}^n$ such that $\mathbf{l}_1,\dots,\mathbf{l}_n$ are linearly independent and

$$D\subseteq\{\mathbf{a}\in\mathbb{Z}^n\mid\mathbf{a}=v_1\mathbf{l}_1+\dots+v_n\mathbf{l}_n,v_i\in\mathbb{R},\forall i>k\ \ |v_i|<1/2\}$$

(for the inclusion, we can choose the vectors $\mathbf{l}_{k+1},\dots,\mathbf{l}_n$ to be sufficiently long).

Let M' be the group generated by $\mathbf{l}_1,\dots,\mathbf{l}_n$, C' the $\mathbf{SL}.\mathbf{t}^n$-cone corresponding to M', u' the root of C'.

Let us check that $\forall\mathbf{a},\mathbf{b}\in D\ \mathbf{a}\sim^{\mathrm{t}}_{C'}\mathbf{b}\Leftrightarrow\mathbf{a}\sim^{\mathrm{t}}_{C}\mathbf{b}$:

($\Leftarrow$) $\mathbf{a} \sim^{\mathrm{t}}_{C} \mathbf{b} \Rightarrow \mathbf{a} \sim^{\mathrm{t}}_{C'} \mathbf{b}$ is obvious since $M \subseteq M'$.

($\Rightarrow$) $\mathbf{a} \sim^{\mathrm{t}}_{C'} \mathbf{b}$ implies $\exists t_1, \ldots, t_n \in \mathbb{Z}\ \ \mathbf{a} - \mathbf{b} = t_1\mathbf{l}_1 + \cdots + t_n\mathbf{l}_n$
$\exists v_1, \ldots, v_n \in \mathbb{R}\ \ \mathbf{a} - \mathbf{b} = v_1\mathbf{l}_1 + \cdots + v_n\mathbf{l}_n$ where $|v_i| < 1$ for $i > k$ since $\mathbf{a}, \mathbf{b} \in D$.
$t_i = v_i$ since $\mathbf{l}_1, \ldots, \mathbf{l}_n$ are linearly independent, hence $|t_i| < 1$ for $i > k$, so $t_i = 0$ for $i > k$.
Hence $\mathbf{a} - \mathbf{b} = t_1\mathbf{l}_1 + \cdots + t_k\mathbf{l}_k$, so $\mathbf{a} \sim_C \mathbf{b}$.

Therefore $C' \not\models A$.

Let u be the root of C, let us construct the map $f : C \to C'$ as $f(\mathbf{a} + u) \leftrightharpoons \mathbf{a} + M'$, f is well-defined:

$$\mathbf{a} + u = \mathbf{b} + u \Rightarrow \mathbf{a} \sim^{\mathrm{t}}_{C} \mathbf{b} \Rightarrow \mathbf{a} \sim^{\mathrm{t}}_{C'} \mathbf{b} \Leftrightarrow \mathbf{a} + u' = \mathbf{b} + u'.$$

Obviously, f is a surjection.

f is a p-morphism due to the inclusion $M \subseteq M'$.

It remains to notice that C' is finite since $\mathbf{l}_1, \ldots, \mathbf{l}_n$ are linearly independent. ∎

Theorem 11.3 Every extension of $\mathbf{SL.t}^n$ has the finite model property.

Proof. The theorem readily follows from Lemma 11.2 taking into account the completeness of every extension of $\mathbf{SL.t}^n$. ∎

This theorem gives us a way to obtain the decidability of finitely axiomatisable extensions of $\mathbf{SL.t}^n$:

Corollary 11.4 Every finitely axiomatisable extension of $\mathbf{SL.t}^n$ is decidable.

12 Extensions of the Logic SL.t.

Definition 12.1

1. Define the following **SL.t**-cones:
   - Ct_0 is the cone corresponding to the relation $=$,
   - Ct_n, $n > 0$, is the cone corresponding to the equivalence modulo n relation.
2. For $n > 0$ let $\mathrm{At}_n \leftrightharpoons p \equiv \Box^n p$.

The cone Ct_n can be pictured as a cycle containing the points $0, 1, \ldots, n-1$. Every point i is related to the next $(i+1)$ point (and $n-1$ is related to 0). We assume that the root of Ct_n is 0.

Proposition 12.2 $\mathrm{Ct}_l \vDash \mathrm{At}_n$ iff l divides n.

Proof.

($\Leftarrow$) Note that $\mathrm{Ct}_n \vDash \mathrm{At}_n$.
Consider the map $f : \mathrm{Ct}_n \to \mathrm{Ct}_l$ such that $f(x + n\mathbb{Z}) \leftrightharpoons x + l\mathbb{Z}$. f is well-defined since l divides n. Obviously, f is a p-morphism of Ct_n onto Ct_l, hence $\mathrm{Ct}_l \vDash \mathrm{At}_n$.

($\Rightarrow$) Let us define the valuation θ in Ct_l as $\theta(p) \leftrightharpoons \{n\mathbb{Z}\}$.
Note that $(\mathrm{Ct}_l, l\mathbb{Z}, \theta) \vDash p$.
If l does not divide n, then $n + l\mathbb{Z} \neq l\mathbb{Z}$, hence $(\mathrm{Ct}_l, l\mathbb{Z}, \theta) \not\models \Box^n p$, that is, $\mathrm{Ct}_l \not\models \mathrm{At}_n$.

∎

Lemma 12.3

1. All Ct_n, Ct_0 are pairwise non-isomorphic.
2. If C is an **SL.t**-cone, then C is isomorphic either to Ct_0 or to Ct_l for some l.

PROOF. Obviously, Ct_0 is not isomorphic to any of Ct_n.

Let C be an **SL.t**-cone with root u.

If all $n + u$, $n \in \mathbb{Z}$, are different, then obviously C is isomorphic to Ct_0.

Otherwise let us choose the minimal $n > 0$ such that $n + u = u$. Then C is isomorphic to Ct_n (since $\sim_C$ is the equivalence modulo n relation). ∎

Lemma 12.4 The logics of the cones of **SL.t** are the following:

1. $\mathrm{L}(\mathrm{Ct}_0) = \mathbf{SL.t}$,
2. $\mathrm{L}(\mathrm{Ct}_n) = \mathbf{SL.t} + \mathrm{At}_n$.

PROOF.

1. Follows from 1.2.

2. The inclusion ($\supseteq$) is obvious.

   If $A \notin \mathbf{SL.t} + \mathrm{At}_n$, then there exists an **SL.t**-cone C such that $C \vDash \mathrm{At}_n$ and $C \nvDash A$. By Lemma 12.3, $C = \mathrm{Ct}_l$ for some l since $\mathrm{Ct}_0 \nvDash \mathrm{At}_n$. By Proposition 12.2, l divides n. Hence the map $f : \mathrm{Ct}_n \to \mathrm{Ct}_l$ defined as $f(a + n\mathbb{Z}) \leftrightharpoons a + l\mathbb{Z}$ is a p-morphism, and so $\mathrm{Ct}_n \nvDash A$.

∎

Remark 12.5 If $A(p_1, \ldots, p_k)$ is a 2-modal formula, $dt_1(A) \leq d$, then

$$\mathbf{SL.t} \vdash A \equiv \phi(\Box^{-d}p_1, \ldots, \Box^{-d}p_k, \ldots, p_1, \ldots, p_k, \ldots, \Box^{d}p_1, \ldots, \Box^{d}p_k),$$

for some formula ϕ not containing modalities.

The following theorem was obtained first (using another method) by A. A. Muchnik [3].

Theorem 12.6 Every consistent proper extension of **SL.t** is tabular.

PROOF. Consider an extension $\Lambda \supseteq \mathbf{SL.t}$. Since every extension of a tabular logic is tabular, we can consider only the case when $\Lambda = \mathbf{SL.t} + A$, where $\mathbf{SL.t} \nvdash A$.

Let d be the degree of A, then

$$\mathbf{SL.t} \vdash A \equiv \phi(\Box^{-d}p_1, \ldots, \Box^{-d}p_k, \ldots, \Box^{d}p_1, \ldots, \Box^{d}p_k).$$

Let us write ϕ in the perfect disjunctive normal form:

$$\mathbf{SL.t} \vdash \phi(\Box^{-d}p_1, \ldots, \Box^{-d}p_k, \ldots, \Box^{d}p_1, \ldots, \Box^{d}p_k) \equiv \bigvee_{\alpha \in S} p_1^{\alpha_1^{-d}} \wedge \cdots \wedge p_k^{\alpha_k^{-d}} \wedge \cdots \wedge \Box^d p_1^{\alpha_1^d} \wedge \cdots \wedge \Box^d p_k^{\alpha_k^d}.$$

Here S is a set of binary matrices (α_j^i), where $1 \leq j \leq k$, $-d \leq i \leq d$.

Note that S does not contain all such matrices, for otherwise

$$\mathbf{SL.t} \vdash \phi(q_1^{-d}, \ldots, q_k^{-d}, \ldots, q_1^d, \ldots, q_k^d) \equiv \top,$$

what contradicts to $\mathbf{SL.t} \nvdash A$.

Hence there exists a matrix $\tilde{\alpha}$ such that $\tilde{\alpha} \notin S$.

Let us prove that every cone validating A contains no more than $2d$ elements.

In fact, suppose there exists a cone C with root u such that $C \vDash A$ and C contains more than $2d$ elements. We can evaluate p_i at $j + u$ as $\tilde{\alpha}_i^j$ for $-d \leq j \leq d$. The valuation of p_i is well-defined since all $j + u$, $-d \leq j \leq d$, are different. But under this valuation the disjunction turns out to be false, which contradicts to $C \vDash A$.

There exist finitely many non-isomorphic cones with no more than $2d$ elements. By completeness it follows that Λ is the logic of cones of this kind and thus it is tabular. ∎

Theorem 12.7 If Λ is an extension of **SL.t**, then $\Lambda = \mathbf{SL.t} + \mathrm{At}_\Lambda$ where $\mathrm{At}_\Lambda = \top$ or $\mathrm{At}_\Lambda = \mathrm{At}_{n_1} \sqcup \cdots \sqcup \mathrm{At}_{n_k}$, $k \geq 0$.

Proof. This follows from Lemma 5.6 and the fact that every consistent proper extension of **SL.t** is a logic of a finite number of cones of the form Ct_n. ∎

References

[1] F. Bellissima. On the lattice of extensions of the modal logics $\mathbf{K.Alt}_n$. *Arch. Math. Logic*, 27(2):107–114, 1988.

[2] P. M. Cohn. *Universal Algebra*. D. Reidel, 1981.

[3] N. M. Ermolaeva and A. A. Muchnik. Pretabular temporal logic (in russian). In *Investigations in non-classical logics*, pages 288–297. Moscow, Nauka, 1979.

[4] D. Gabbay and V. Shehtman. Products of modal logics, part 1. *Logic Journal of the IGPL*, 6(1):73–146, 1998.

[5] M. Kracht. Highway to the danger zone. *Journal of Logic and Computation*, 5:93–109, 1995.

[6] M. Kracht. *Tools and techniques in modal logic.* Studies in Logic and Found. Math., 142, 1999.

[7] A. Prior. *Past, present, and future*. Oxford, 1967.

[8] K. Segerberg. On the logic of tomorrow. *Theoria*, 33:45–52, 1967.

[9] K. Segerberg. Modal logics with functional alternative relations. *Notre Dame Journal of Formal Logic*, 27:504–522, 1986.

Received June 2002

PSPACE Reasoning with the Description Logic $\mathcal{ALCF}(\mathcal{D})$

CARSTEN LUTZ, *Institute for Theoretical Computer Science, Technical University Dresden, 01062 Dresden, Germany.*
E-mail: lutz@tcs.inf.tu-dresden.de

Abstract

Description Logics (DLs), a family of formalisms for reasoning about conceptual knowledge, can be extended with concrete domains to allow an adequate representation of "concrete qualities" of real-worlds entities such as their height, temperature, duration, and size. In this paper, we study the complexity of reasoning with the basic DL with concrete domains $\mathcal{ALC}(\mathcal{D})$ and its extension with so-called feature agreements and disagreements $\mathcal{ALCF}(\mathcal{D})$. We show that, for both logics, the standard reasoning tasks concept satisfiability, concept subsumption, and ABox consistency are PSPACE-complete if the concrete domain $\mathcal{D}$ satisfies some natural conditions.

Keywords: Description Logics, Concrete Domains, Feature (Dis)Agreements, Computational Complexity

1 Motivation

Description Logics (DLs) are a popular family of logical formalisms for the representation of and reasoning about conceptual knowledge [8]. The basic entity for knowledge representation with DLs are so-called concepts which can be understood as logical formulas and are constructed from concept names (unary predicates), role names (binary relations), and concept constructors. For example, the following concept is formulated in the basic propositionally closed DL $\mathcal{ALC}$ [39] and describes processes that are supervised by a human operator and involve only workpieces that are not radioactive:

$$\mathsf{Process} \sqcap \exists \mathsf{operator}.\mathsf{Human} \sqcap \forall \mathsf{workpiece}.\neg \mathsf{Radioactive}.$$

In this concept, Process, Human, and Radioactive are concept names while operator and workpiece are role names.

A major limitation of knowledge representation with Description Logics such as $\mathcal{ALC}$ is that "concrete qualities" of real world entities, such as their weight, temperature, and spatial extension, cannot be adequately represented. For example, $\mathcal{ALC}$ does not offer suitable means of expressivity for extending the above description of a process with information about its cost and duration, or about the relationship between the process' cost and the hourly wage of its operator. To allow an adequate representation of concrete qualities of real-world entities, Description Logics are frequently extended by so-called *concrete domains*, which have first been proposed by Baader and Hanschke in [4] and then further developed in several directions, c.f. the survey article [32]. A concrete domain consists of a set such as the natural numbers and a set of predicates such as the unary "$=_{60}$" and the binary ">" with the obvious,

fixed extension. The integration of concrete domains into the Description Logic $\mathcal{ALC}$ is achieved by adding

1. so-called *abstract features*, which are functional relations;
2. so-called *concrete features*, which are (partial) functions associating values from the concrete domain (e.g., natural numbers) to logical objects;
3. a concrete domain-based concept constructor.

The DL that is obtained by extending $\mathcal{ALC}$ in this way is called $\mathcal{ALC}(\mathcal{D})$, where $\mathcal{D}$ denotes a concrete domain that can be viewed as a parameter to the logic. For example, when using a suitable concrete domain $\mathcal{D}$, we can extend the above process description as desired: the $\mathcal{ALC}(\mathcal{D})$-concept

$$\mathsf{Process} \sqcap \exists \mathsf{duration}.=_{60} \sqcap \exists \mathsf{cost}, \mathsf{operator}\ \mathsf{wage}.>$$

describes a process whose duration is 60 minutes and which costs more than the (hourly) wage of its operator. Here, the second and third conjunct are instances of the concrete domain concept constructor, operator is an abstract feature, and duration, cost, and wage are concrete features.

The representation of concrete qualities has been identified as a crucial task for a vast number of applications such as mechanical engineering [6], temporal and spatial reasoning [16, 27], the semantic web [23, 24], and reasoning about entity relationship (ER) diagrams [31]. Consequently, apart from $\mathcal{ALC}(\mathcal{D})$ many other Description Logics with concrete domains have been proposed [16, 18, 20, 24, 27, 30, 29] and several implemented Description Logic reasoners such as CLASSIC [11] and RACER [17] provide for some kind of concrete domain. However, despite the considerable interest in DLs with concrete domains and the fact that complexity analysis plays an important role in the area of Description Logics, only very recently researchers have begun to investigate the computational complexity of reasoning with such logics [30]. The current paper is devoted to *establishing tight complexity bounds for reasoning with the fundamental Description Logic with concrete domains* $\mathcal{ALC}(\mathcal{D})$. More precisely, we do not only consider the DL $\mathcal{ALC}(\mathcal{D})$, but also its extension with so-called *feature agreements* and *feature disagreements*, two concept constructors that are quite closely related to concrete domains. Using feature (dis)agreements, one can for example describe processes that have two subprocesses, one of which works on the same workpiece as the mother process, and the other on a different one:

$$\mathsf{Process} \sqcap (\mathsf{workpiece} \downarrow \mathsf{subprocess1}\ \mathsf{workpiece}) \sqcap (\mathsf{workpiece} \uparrow \mathsf{subprocess2}\ \mathsf{workpiece}).$$

In this concept, the second conjunct uses the feature agreement constructor, the third conjunct uses the feature disagreement constructor, and all lowercase names denote abstract features.

There are several motivations for combining concrete domains and feature (dis)agreements in a single DL. First, there exists an obvious syntactic similarity between feature (dis)agreements and the concrete domain concept constructor: both take sequences of features as arguments. As we shall see in this paper, the similarity between concrete domains and feature (dis)agreements is not only syntactical: they are also amenable to similar algorithmic techniques. Second, the Description Logic $\mathcal{ALCF}(\mathcal{D})$ resulting

from the extension of $\mathcal{ALC}(\mathcal{D})$ with feature (dis)agreements has already found applications in knowledge representation [25]. And third, the PSPACE-completeness result for reasoning with $\mathcal{ALCF}(\mathcal{D})$ proved in Section 3 allows to show PSPACE-completeness of a well-known temporal Description Logic [3].

Let us now outline the organization of this paper and describe the obtained results in more detail.

In Section 2, we formally introduce concrete domains and the Description Logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}(\mathcal{D})$. Some example concrete domains are defined.

In Section 3, tight PSPACE complexity bounds for the satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts and $\mathcal{ALCF}(\mathcal{D})$-concepts are established. More precisely, we devise a tableau algorithm for deciding satisfiability of $\mathcal{ALCF}(\mathcal{D})$-concepts which uses the so-called *tracing* technique. This algorithm yields a PSPACE upper bound for $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability if the following conditions are satisfied:

- deciding the satisfiability of finite conjunctions of predicates from the concrete domain $\mathcal{D}$ (this task is called "$\mathcal{D}$-satisfiability" in what follows) is in PSPACE;
- the concrete domain is "admissible", i.e., it satisfies some weak closure conditions which, in this paper, we will generally assume to hold.

The corresponding PSPACE lower bound is easily obtained since $\mathcal{ALC}$-concept satisfiability is already PSPACE-hard [39]. Hence, both $\mathcal{ALC}(\mathcal{D})$-concept satisfiability and $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability are PSPACE-complete if $\mathcal{D}$-satisfiability is in PSPACE. Since concept subsumption, another important reasoning task for Description Logics, can easily be reduced to concept (un)satisfiability and vice versa, we also obtain that $\mathcal{ALC}(\mathcal{D})$-concept subsumption and $\mathcal{ALCF}(\mathcal{D})$-concept subsumption are PSPACE-complete if $\mathcal{D}$-satisfiability is in PSPACE. Note that adding concrete domains and feature (dis)agreements to $\mathcal{ALC}$ does thus not increase the complexity of reasoning. This is particularly interesting since there exist several seemingly "harmless" means of expressivity like acyclic TBoxes and inverse roles, whose addition to $\mathcal{ALC}(\mathcal{D})$ makes reasoning significantly more difficult—namely NEXPTIME-complete [28, 30, 1]. Thus, the logic $\mathcal{ALCF}(\mathcal{D})$ is situated on the boundary of polynomial space complexity.

Section 4 is devoted to extending the results from Section 3 to another standard reasoning task called ABox consistency. ABoxes are commonly used to describe snapshots of the real world [7, 12, 17, 38, 41]. For example, the following $\mathcal{ALC}(\mathcal{D})$-ABox describes a process a and its subprocess b:

$$a : \mathsf{Process} \quad b : \mathsf{Process} \quad (a, b) : \mathsf{subprocess} \quad (a, x) : \mathsf{duration} \quad x : =_{60}$$

We use the *precompletion* technique from [13, 21] to show that $\mathcal{ALCF}(\mathcal{D})$-ABox consistency is PSPACE-complete if $\mathcal{D}$-satisfiability is in PSPACE. As in the case of concept satisfiability, this implies that the same holds for $\mathcal{ALC}(\mathcal{D})$-ABox consistency.

In Section 5, we demonstrate the relevance of the results obtained in Sections 3 and 4 by considering two example concrete domains: the concrete domain A based on the rational numbers with predicates such as $<_{27}$, $\geq$, and $+$; and the concrete domain S based on the set of regions in two-dimensional space with a binary predicate for each of the well-known RCC8 topological relations [10]. We show that both A-satisfiability and S-satisfiability is in NP and thus obtain that, for $\mathcal{D} \in \{\mathsf{A}, \mathsf{S}\}$, $\mathcal{ALCF}(\mathcal{D})$-concept

satisfiability, $\mathcal{ALCF}(\mathcal{D})$-concept subsumption, and $\mathcal{ALCF}(\mathcal{D})$-ABox consistency are PSPACE-complete.

The paper ends with a conclusion in Section 6.

2 Preliminaries

We start this section with introducing concrete domains formally, then define some example concrete domains, and finally describe the Description Logic $\mathcal{ALCF}(\mathcal{D})$ in detail.

Definition 2.1 (Concrete Domain) A *concrete domain* $\mathcal{D}$ is a pair $(\Delta_\mathcal{D}, \Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is a set and $\Phi_\mathcal{D}$ a set of predicate names. Each predicate name $P \in \Phi_\mathcal{D}$ is associated with an arity n and an n-ary predicate $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$. Let V be a set of variables. A predicate conjunction of the form

$$c = \bigwedge_{i<k} (x_0^{(i)}, \ldots, x_{n_i}^{(i)}) : P_i,$$

where P_i is an n_i-ary predicate for $i < k$ and the $x_j^{(i)}$ are variables from V, is called *satisfiable* iff there exists a function δ mapping the variables in c to elements of $\Delta_\mathcal{D}$ such that $(\delta(x_0^{(i)}), \ldots, \delta(x_{n_i}^{(i)})) \in P_i^\mathcal{D}$ for each $i < k$. Such a function is called a *solution* for c. A concrete domain $\mathcal{D}$ is called *admissible* if the following conditions are satisfied:

1. $\Phi_\mathcal{D}$ contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$;
2. $\Phi_\mathcal{D}$ is closed under negation, i.e., for each n-ary predicate $P \in \Phi_\mathcal{D}$, we find another predicate $\overline{P} \in \Phi_\mathcal{D}$ of arity n such that $\overline{P}^\mathcal{D} = \Delta_\mathcal{D}^n \setminus P^\mathcal{D}$;
3. the satisfiability problem for finite conjunctions of predicates is decidable.

When devising algorithms for reasoning with Description Logics that are equipped with a concrete domain $\mathcal{D}$, one important subtask usually is to decide the satisfiability of finite conjunctions of predicates from $\Phi_\mathcal{D}$ as described in Definition 2.1 [4, 30]. For brevity, we refer to this task as $\mathcal{D}$-satisfiability. It is obvious that $\mathcal{D}$-satisfiability should be decidable if the concrete domain $\mathcal{D}$ is to be used in a DL reasoning algorithm. However, usually the slightly stronger requirement that $\mathcal{D}$ should be admissible is adopted. In this article, we follow this tradition and generally assume concrete domains to be admissible.

Before we proceed to defining the Description Logic $\mathcal{ALCF}(\mathcal{D})$ itself, let us introduce two example concrete domains, an arithmetic one and a spatial one. The arithmetic concrete domain A is defined by setting $\Delta_\mathsf{A} := \mathbb{Q}$ (i.e., the set of rational numbers), and defining Φ_A as the (smallest) set containing the following predicates:

- a unary predicate $\top_\mathsf{A}$ with $(\top_\mathsf{A})^\mathsf{A} = \mathbb{Q}$ and a unary predicate $\bot_\mathsf{A}$ with $(\bot_\mathsf{A})^\mathsf{A} = \emptyset$;
- unary predicates int and $\overline{\mathsf{int}}$ with $(\mathsf{int})^\mathsf{A} = \mathbb{Z}$ (where $\mathbb{Z}$ denotes the integers) and $(\overline{\mathsf{int}})^\mathsf{A} = \mathbb{Q} \setminus \mathbb{Z}$;
- unary predicates P_q for each $P \in \{<, \leq, =, \neq, \geq, >\}$ and each $q \in \mathbb{Q}$ with $(P_q)^\mathsf{A} = \{q' \in \mathbb{Q} \mid q' \; P \; q\}$;

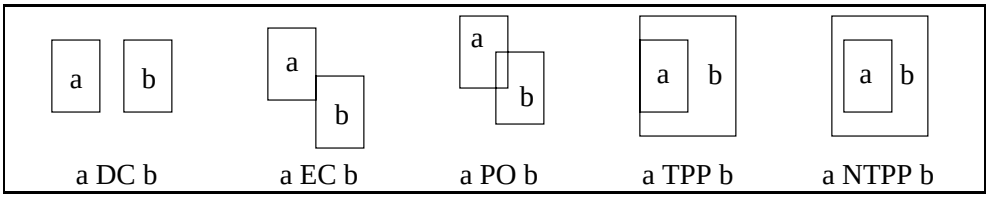


FIG. 1. The RCC8 relations in two-dimensional space.

- binary predicates $<, \leq, =, \neq, \geq, >$ with the obvious extension;
- ternary predicates $+$ and $\overline{+}$ with $(+)^{\mathsf{A}} = \{(q, q', q'') \in \mathbb{Q}^3 \mid q + q' = q''\}$ and $(\overline{+})^{\mathsf{A}} = \mathbb{Q}^3 \setminus (+)^{\mathsf{A}}$.

As an example for an (unsatisfiable) conjunction of A-predicates, consider the following one:

$$=_3(x) \wedge >_1(y) \wedge \mathsf{int}(y) \wedge +(x, y, z) \wedge *(x, y, z') \wedge \geq(z, z').$$

It is easily checked that the concrete domain A satisfies Conditions 1 and 2 of admissibility (Condition 3 will be treated in Section 5). The other concrete domain considered in this paper is related to the RCC-8 calculus and is called S. RCC-8 provides a set of eight jointly exhaustive and pairwise disjoint relations that describe the possible relationships between any two regular closed regions[1] in a topological space [34, 10, 36]. For 2D space, these relations are illustrated in Figure 1, where the equality relation EQ, the inverse TPPI of TPP, and the inverse NTPPI of NTPP have been omitted. The concrete domain S is defined by setting Δ_{S} to the set $\mathcal{RC}_{\mathbb{R}^2}$ of all regular closed subsets of $\mathbb{R}^2$ and defining Φ_{S} as the (smallest) set containing the following predicates:

- a unary predicate $\top_{\mathsf{S}}$ with $(\top_{\mathsf{S}})^{\mathsf{S}} = \mathcal{RC}_{\mathbb{R}^2}$ and a unary predicate $\bot_{\mathsf{S}}$ with $(\bot_{\mathsf{S}})^{\mathsf{S}} = \emptyset$;
- binary predicates rel and $\overline{\mathsf{rel}}$ for each of the topological relations rel such that $(\mathsf{rel})^{\mathsf{S}} = \{(r_1, r_2) \in \mathcal{RC}_{\mathbb{R}^2} \times \mathcal{RC}_{\mathbb{R}^2} \mid r_1 \ \mathsf{rel} \ r_2\}$.

An example (unsatisfiable) S-conjunction is

$$\top_{\mathsf{S}}(x) \wedge DC(x, y) \wedge EC(y, z) \wedge NTPP(z, x) \wedge \overline{PO}(y, y).$$

It is easily checked that S satisfies Conditions 1 and 2 of admissibility. For Property 3, we again refer to Section 5.

Based on concrete domains, we can now define $\mathcal{ALCF}(\mathcal{D})$-concepts.

Definition 2.2 ($\mathcal{ALCF}(\mathcal{D})$ syntax) Let $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_{cF}}$ be pairwise disjoint and countably infinite sets of *concept names*, *role names*, and *concrete features*. Furthermore, let $\mathsf{N_{aF}}$ be a countably infinite subset of $\mathsf{N_R}$. The elements of $\mathsf{N_{aF}}$ are called *abstract features*. An *abstract path* p is a composition $f_1 \cdots f_n$ of n abstract features ($n \geq 1$). A *concrete path* u is a composition $f_1 \cdots f_n g$ of n abstract features $f_1, \ldots, f_n$ ($n \geq 0$) and a concrete feature g. Let $\mathcal{D}$ be a concrete domain. The set of $\mathcal{ALCF}(\mathcal{D})$-concepts is the smallest set such that

[1] A region r is regular closed if it satisfies $ICr = r$, where C is the topological closure operator and I is the topological interior operator.

1. every concept name is a concept
2. if C and D are concepts, R is a role name, g is a concrete feature, p_1 and p_2 are abstract paths, $u_1, \ldots, u_n$ are concrete paths, and $P \in \Phi_{\mathcal{D}}$ is a predicate of arity n, then the following expressions are also concepts:

$$\neg C,\ C \sqcap D,\ C \sqcup D,\ \exists R.C,\ \forall R.C,\ p_1{\uparrow}p_2,\ p_1{\downarrow}p_2,\ \exists u_1, \ldots, u_n.P,\ \text{ and } g{\uparrow}.$$

We use $\top$ to abbreviate $A \sqcup \neg A$, where A is an arbitrary concept name, and $\bot$ to abbreviate $\neg\top$. Moreover, we write $\forall p.C$ for $\forall f_1.\cdots\forall f_k.C$ if $p = f_1 \cdots f_k$ and $u{\uparrow}$ for $\forall f_1.\cdots\forall f_k.g{\uparrow}$ if $u = f_1 \cdots f_k g$. An $\mathcal{ALCF}(\mathcal{D})$-concept that does not contain subconcepts $p_1{\uparrow}p_2$ and $p_1{\downarrow}p_2$ is called $\mathcal{ALC}(\mathcal{D})$-concept. An $\mathcal{ALC}(\mathcal{D})$-concept that does not use any abstract or concrete features is called $\mathcal{ALC}$-concept.

Throughout this paper, we use the letter A to denote concept names, C, D, and E to denote (possibly complex) concepts, R to denote role names, f to denote abstract features, g to denote concrete features, p to denote abstract paths, u to denote concrete paths, and P to denote predicate names from the concrete domain.

The Description Logic $\mathcal{ALCF}(\mathcal{D})$ is equipped with a Tarski-style set-theoretic semantics that incorporates the concrete domain $\mathcal{D}$.

Definition 2.3 ($\mathcal{ALCF}(\mathcal{D})$ semantics) An *interpretation* $\mathcal{I}$ is a pair $(\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta_{\mathcal{I}}$ is a set called the *domain* and $\cdot^{\mathcal{I}}$ the *interpretation function*. The interpretation function maps

- each concept name C to a subset $C^{\mathcal{I}}$ of $\Delta_{\mathcal{I}}$,
- each role name R to a subset $R^{\mathcal{I}}$ of $\Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}}$,
- each abstract feature f to a partial function $f^{\mathcal{I}}$ from $\Delta_{\mathcal{I}}$ to $\Delta_{\mathcal{I}}$, and
- each concrete feature g to a partial function $g^{\mathcal{I}}$ from $\Delta_{\mathcal{I}}$ to $\Delta_{\mathcal{D}}$.

If $u = f_1 \cdots f_n g$ is a concrete path, then $u^{\mathcal{I}}(d)$ is defined as $g^{\mathcal{I}}(f_n^{\mathcal{I}} \cdots (f_1^{\mathcal{I}}(d)) \cdots)$, and similarly for abstract paths. The interpretation function is extended to arbitrary concepts as follows:

$$\begin{aligned}
(\neg C)^{\mathcal{I}} &:= \Delta_{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \{e \mid (d,e) \in R^{\mathcal{I}}\} \cap C^{\mathcal{I}} \neq \emptyset\} \\
(\forall R.C)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \{e \mid (d,e) \in R^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}\} \\
(p_1{\uparrow}p_2)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \exists e_1, e_2 \in \Delta_{\mathcal{I}} : p_1^{\mathcal{I}}(d) = e_1,\ p_2^{\mathcal{I}}(d) = e_2, \text{ and } e_1 \neq e_2\} \\
(p_1{\downarrow}p_2)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \exists e \in \Delta_{\mathcal{I}} : p_1^{\mathcal{I}}(d) = p_2^{\mathcal{I}}(d) = e\} \\
(\exists u_1, \ldots, u_n.P)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \exists x_1, \ldots, x_n \in \Delta_{\mathcal{D}} : u_i^{\mathcal{I}}(d) = x_i \text{ for } 1 \leq i \leq n \\
&\qquad\qquad \text{and } (x_1, \ldots, x_n) \in P^{\mathcal{D}}\} \\
(g{\uparrow})^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid g^{\mathcal{I}}(d) \text{ undefined}\}
\end{aligned}$$

An interpretation $\mathcal{I}$ is a *model* of a concept C iff $C^{\mathcal{I}} \neq \emptyset$. A concept C is *satisfiable* iff it has a model. C is *subsumed by* a concept D (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$.

It is well-known that, in Description Logics providing for full negation such as $\mathcal{ALCF}(\mathcal{D})$, subsumption can be reduced to (un)satisfiability and vice versa: $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable and C is satisfiable iff $C \not\sqsubseteq \bot$. This allows us to concentrate on concept satisfiability in the remainder of this paper.

Note that feature (dis)agreements $p_1{\uparrow}p_2$ and $p_1{\downarrow}p_2$ take abstract paths as arguments and are thus not concerned with elements from the concrete domain. However, if the concrete domain provides for equality and inequality predicates (as both A and S do), it is obvious that we can express (dis)agreement of concrete paths using the concrete domain constructor. Also note that $a \in (p_1{\uparrow}p_2)^{\mathcal{I}}$ implies that $p_1^{\mathcal{I}}(a)$ and $p_1^{\mathcal{I}}(a)$ are defined. Thus, $p_1{\uparrow}p_2$ is *not* the negation of $p_1{\downarrow}p_2$ (also see Section 3.2 and Figure 3).

We should like to comment on a minor difference between our variant of $\mathcal{ALCF}(\mathcal{D})$ and the original version of $\mathcal{ALC}(\mathcal{D})$ as defined by Baader and Hanschke [4]: instead of separating concrete and abstract features, Baader and Hanschke define only one type of feature which is interpreted as a partial function from $\Delta_{\mathcal{I}}$ to $\Delta_{\mathcal{I}} \cup \Delta_{\mathcal{D}}$. We prefer the "typed" approach since, in our opinion, it improves the readability of concepts. Moreover, it is not hard to see that the combined features can be "simulated" using pairs of concrete and abstract features.

3 Concept Satisfiability

In the following, we devise a tableau algorithm for deciding satisfiability of $\mathcal{ALCF}(\mathcal{D})$-concepts that needs at most polynomial space if $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in PSpace. The algorithm also yields tight complexity bounds if $\mathcal{D}$-satisfiability is NExpTime-complete or ExpSpace-complete.

3.1 Overview

Since there exist rather different variants of tableau algorithms in Modal Logic and First Order Logic, we call the family of tableau algorithms commonly used for Description Logics *completion algorithms*. The reader is referred to [9] for an overview over such algorithms. Completion algorithms are characterized by an underlying data structure, a set of *completion rules* operating on this data structure, and a (possibly trivial) strategy for applying the rules. In principle, a completion algorithm starts with an initial data structure induced by the concept D whose satisfiability is to be decided and repeatedly applies completion rules according to the strategy. Repeated rule application can be thought of as making implicit knowledge explicit or as constructing a canonical model for the input concept (represented in terms of the underlying data structure). The algorithm stops if it encounters a contradiction or if no more completion rules are applicable. It returns satisfiable iff the latter is the case *and* no obvious contradiction was found, i.e., if the algorithm succeeds in constructing a (witness for a) model of the input concept. Otherwise, it returns unsatisfiable.

If a PSpace upper bound is to be proved using a completion algorithm, some additional efforts have to be made. To simplify discussion, let us consider the logic $\mathcal{ALC}$ for the moment [39]. A naive completion algorithm for $\mathcal{ALC}$ does not yield a PSpace upper bound since there exist satisfiable $\mathcal{ALC}$-concepts all of whose models are of size exponential in the concept length [19, 39]. Thus, an algorithm keeping

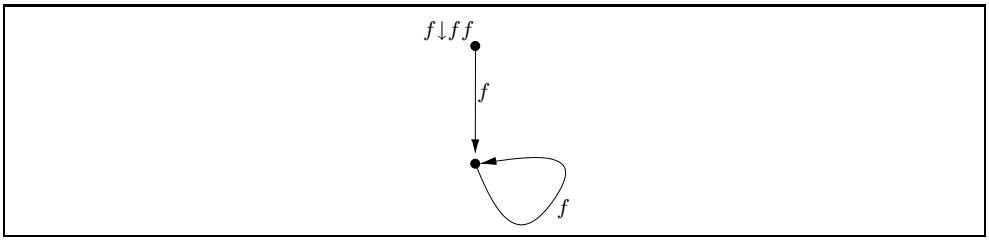


FIG. 2. A model of the $\mathcal{ALCF}(\mathcal{D})$-concept $f{\downarrow}ff$.

the entire (representation of a) model in memory needs exponential space in the worst case. However, there exists a well-known way to overcome this problem: the key observation is that canonical models $\mathcal{I}$ constructed by completion algorithms are *tree models*, i.e., they have the form of a tree if viewed as a graph with $\Delta_\mathcal{I}$ the set of vertexes and $\bigcup_{R\in\mathsf{N_R}} R^\mathcal{I}$ the set of edges. It is sufficient to consider only such tree models since $\mathcal{ALC}$ has the *tree model property*, which means that each satisfiable concept has a tree model [19]. To check for the existence of tree models for a given concept, we may try to construct one by performing depth-first search over role successors keeping only paths of the tree model in memory. Since, in the case of $\mathcal{ALC}$, the length of paths is at most polynomial in the length of the input concept [19], this technique—which is known as *tracing* [39]—yields an algorithm that needs at most polynomial space in the worst case. Completion algorithms for $\mathcal{ALC}$-concept satisfiability that use tracing are very similar to the well-known K-world algorithm from Modal Logic [26].

The tracing technique has to be modified to deal with $\mathcal{ALCF}(\mathcal{D})$-concepts for two reasons:

(1) Due to the presence of feature (dis)agreements, $\mathcal{ALCF}(\mathcal{D})$ does not enjoy the tree model property. For example, the concept $f{\downarrow}ff$ is satisfiable but, due to the functionality of the abstract feature f, has only non-tree models such as the one depicted in Figure 2.

(2) Due to the presence of the concrete domain constructor, even in tree models the paths of the tree cannot be considered in isolation. For example, the canonical tree model for the concept $\exists(f_1f_2g),(f_1'f_2'g').P$ is comprised of two paths with edge labels f_1, f_2, g and f_1', f_2', g', respectively. However, since the final node of the first path and the final node of the second path are elements of the concrete domain that must be related via the predicate P, we have to consider both paths together.

Since only abstract features (but no role names from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$) are admitted in feature (dis)agreements and the concrete domain constructor, it is not hard to see that the described problems are due to substructures of models whose elements are connected by abstract features, only. Based on this observation, we define *generalized tree models*.

Definition 3.1 (Generalized Tree Model) Let $\mathcal{I}$ be a model of an $\mathcal{ALCF}(\mathcal{D})$-

concept C and define a relation $\sim$ on $\Delta_\mathcal{I}$ as follows:

$$d \sim e \quad \text{iff} \quad \begin{array}{l} d = e \text{ or there exists an abstract path } f_1 \cdots f_k \text{ and domain elements} \\ d_0, \ldots, d_k \in \Delta_\mathcal{I} \text{ such that } d_0 = d,\ d_k = e, \text{ and } d_{i+1} = f_{i+1}^\mathcal{I}(d_i) \text{ or} \\ d_i = f_{i+1}^\mathcal{I}(d_{i+1}) \text{ for } i < k. \end{array}$$

It is easy to see that $\sim$ is an equivalence relation. By $[d]_\sim$, we denote the equivalence class of $d \in \Delta_\mathcal{I}$ w.r.t. $\sim$. The model $\mathcal{I}$ is a *generalized tree model* of C iff $\mathcal{I}$ is a model of C and the graph $(V_\mathcal{I}, E_\mathcal{I})$ defined as

$$\begin{array}{rcl} V_\mathcal{I} & := & \{[d]_\sim \mid d \in \Delta_\mathcal{I}\} \\ E_\mathcal{I} & := & \{([d]_\sim, [e]_\sim) \mid \exists d' \in [d]_\sim, e' \in [e]_\sim \text{ such that} \\ & & \qquad (d', e') \in R^\mathcal{I} \text{ for some } R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}\} \end{array}$$

is a tree.

It will be a byproduct of the results obtained in this section that $\mathcal{ALCF}(\mathcal{D})$ has the *generalized tree model property*, i.e., that every satisfiable $\mathcal{ALCF}(\mathcal{D})$-concept C has a generalized tree model. Note that the identification of some kind of tree model property is usually very helpful for devising decision procedures [42, 15]. Our completion algorithm for $\mathcal{ALCF}(\mathcal{D})$ uses tracing on generalized tree models: it keeps only fragments of models $\mathcal{I}$ in memory that induce paths in the abstraction $(V_\mathcal{I}, E_\mathcal{I})$. Intuitively, such a fragment consists of a sequence of "clusters" of domain elements, where each cluster is an equivalence class w.r.t. the relation $\sim$, i.e., a set of elements connected by abstract features. Succeeding clusters in the sequence are connected by roles from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$. Fortunately, as we shall see later, there always exists a generalized tree model $\mathcal{I}$ in which the cardinality of clusters and the depth of the tree $(V_\mathcal{I}, E_\mathcal{I})$ is at most polynomial in the length of the input concept. We use these facts to devise a completion algorithm for $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability running in polynomial space.

The polynomial size of object clusters is also exploited for dealing with the concrete domain. Along with constructing the "logical part" of the model for the input concept, our completion algorithm will build up a predicate conjunction describing its "concrete part". This predicate conjunction is required to be satisfiable in order for the constructed data structure to represent a model (see the general description of completion algorithms above). However, if this is done in a straightforward way, the number of conjuncts in the predicate conjunction may become exponential in the length of the input concept—see e.g. the algorithm for $\mathcal{ALC}(\mathcal{D})$ concept satisfiability presented in [4]. In our algorithm, we address this problem as follows: domain elements that are in different clusters of the generalized tree model are not connected through abstract paths. Therefore, it cannot be enforced that concrete successors of domain elements from different clusters are related by a concrete predicate. This, in turn, means that it is sufficient to *separately* check the satisfiability of predicate conjunctions associated with clusters. Since the size of predicate conjunctions associated with a cluster is at most polynomial in the length of the input concept, this separate checking allows to devise a PSPACE algorithm (if $\mathcal{D}$-satisfiability is in PSPACE).

$$
\begin{array}{llll}
\neg(C \sqcap D) & \rightsquigarrow & \neg C \sqcup \neg D & \neg(C \sqcup D) \rightsquigarrow \neg C \sqcap \neg D \\
\neg(\exists R.C) & \rightsquigarrow & \forall R.\neg C & \neg(\forall R.C) \rightsquigarrow \exists R.\neg C \\
\neg(p_1{\uparrow}p_2) & \rightsquigarrow & p_1{\downarrow}p_2 \sqcup \forall p_1.\bot \sqcup \forall p_2.\bot & \neg(p_1{\downarrow}p_2) \rightsquigarrow p_1{\uparrow}p_2 \sqcup \forall p_1.\bot \sqcup \forall p_2.\bot \\
\neg\neg C & \rightsquigarrow & C & \\
\neg(\exists u_1, \ldots, u_n.P) & \rightsquigarrow & \exists u_1, \ldots, u_n.\overline{P} \sqcup u_1{\uparrow} \sqcup \cdots \sqcup u_n{\uparrow} & \\
\neg(g{\uparrow}) & \rightsquigarrow & \exists g.\top_{\mathcal{D}} &
\end{array}
$$

FIG. 3. The NNF rewrite rules.

3.2 The Completion Algorithm

In the following, we assume that concepts are in negation normal form (NNF), i.e., that negation occurs only in front of concept names. Every $\mathcal{ALCF}(\mathcal{D})$-concept C can be transformed into an equivalent one in NNF by exhaustively applying the rewrite rules displayed in Figure 3 (recall that $\overline{P}$ denotes the negation of the predicate P). Let us start the presentation of the completion algorithm by introducing ABoxes as the underlying data structure.

Definition 3.2 (ABox Syntax) Let $\mathsf{O_a}$ and $\mathsf{O_c}$ be countably infinite and mutually disjoint sets of *abstract objects* and *concrete objects*. If C is an $\mathcal{ALCF}(\mathcal{D})$-concept, $R \in \mathsf{N_R}$ a role name, g a concrete feature, $a, b \in \mathsf{O_a}$, $x, x_1, \ldots, x_n \in \mathsf{O_c}$, and $P \in \Phi_{\mathcal{D}}$ with arity n, then

$$a : C, \quad (a, b) : R, \quad (a, x) : g, \quad (x_1, \ldots, x_n) : P, \quad \text{and} \quad a \not\approx b$$

are *ABox assertions*. An *ABox* is a finite set of such assertions.

Let $\mathcal{A}$ be an ABox, $a, b \in \mathsf{O_a}$ and $x \in \mathsf{O_c}$. We write $\mathcal{A}(a)$ to denote the set of concepts $\{C \mid a : C \in \mathcal{A}\}$. The abstract object b is called *R-successor of a in $\mathcal{A}$* iff $(a, b) : R$ is in $\mathcal{A}$. The notions *g-successor* (for concrete features g), *p-successor* (for abstract paths p), and *u-successor* (for concrete paths u) are defined analogously. In what follows, we used a and b to denote abstract objects and x to denote concrete objects.

For proving the soundness and completeness of the completion algorithm to be devised, it is convenient to equip ABoxes with a semantics:

Definition 3.3 (ABox Semantics) In interpretations $\mathcal{I}$, the interpretation function $\cdot^{\mathcal{I}}$ maps, additionally, abstract objects a to elements $a^{\mathcal{I}} \in \Delta_{\mathcal{I}}$ and concrete objects x to elements $x^{\mathcal{I}} \in \Delta_{\mathcal{D}}$. An interpretation $\mathcal{I}$ satisfies an assertion

$$
\begin{array}{rll}
a : C & \text{iff} & a^{\mathcal{I}} \in C^{\mathcal{I}}; \\
(a, b) : R & \text{iff} & (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}; \\
(a, x) : g & \text{iff} & g^{\mathcal{I}}(a^{\mathcal{I}}) = x^{\mathcal{I}}; \\
(x_1, \ldots, x_n) : P & \text{iff} & (x_1^{\mathcal{I}}, \ldots, x_n^{\mathcal{I}}) \in P^{\mathcal{D}}; \\
a \not\approx b & \text{iff} & a^{\mathcal{I}} \neq b^{\mathcal{I}}.
\end{array}
$$

An interpretation $\mathcal{I}$ is called a *model* of an ABox $\mathcal{A}$ iff it satisfies every assertion in $\mathcal{A}$. An ABox is called *consistent* iff it has a model.

It should be obvious how ABoxes can be used to represent models. If the satisfiability of a concept D is to be decided, the completion algorithm is started with the *initial ABox* for D defined as $\mathcal{A}_D = \{a : D\}$. To keep the presentation of the completion rules succinct, we introduce an operation that allows to introduce new objects on paths and concrete paths.

Definition 3.4 ("+" operation) An abstract or concrete object is called *fresh* w.r.t. an ABox $\mathcal{A}$ if it does not appear in $\mathcal{A}$. Let $p = f_1 \cdots f_n$ be an abstract path (resp. $u = f_1 \cdots f_n g$ be a concrete path). By $\mathcal{A} + apb$ (resp. $\mathcal{A} + aux$), where $a \in \mathsf{O_a}$ is used in $\mathcal{A}$ and $b \in \mathsf{O_a}$ (resp. $x \in \mathsf{O_c}$), we denote the ABox $\mathcal{A}'$ which can be obtained from $\mathcal{A}$ by choosing distinct objects $b_1, \ldots, b_n \in \mathsf{O_a}$ which are fresh in $\mathcal{A}$ and setting

$$\begin{aligned} \mathcal{A}' &:= \mathcal{A} \cup \{(a, b_1) : f_1,\ \ldots,\ (b_{n-1}, b) : f_n\} \\ (\text{resp. } \mathcal{A}' &:= \mathcal{A} \cup \{(a, b_1) : f_1,\ \ldots,\ (b_{n-1}, b_n) : f_n,\ (b_n, x) : g\}. \end{aligned}$$

When nesting the + operation, we omit brackets writing, e.g., $\mathcal{A} + ap_1b + bp_2c$ for $(\mathcal{A} + ap_1b) + bp_2c$.

The completion rules can be found in Figure 4. Note that the $\mathsf{R}\sqcup$ rule is nondeterministic, i.e., it has more than one possible outcome. Thus, the described completion algorithm is a nondeterministic decision procedure. Such an algorithm accepts its input (i.e. returns satisfiable) iff there is *some* way to make the nondeterministic decisions such that a positive result is obtained. A convenient way to think of nondeterministic rules is that they "guess" the correct outcome, i.e., if there is an outcome which, if chosen, leads to a positive result, then this outcome is in fact considered.

Most completion rules are standard and known from, e.g., [5] and [22]. The $\mathsf{R}\exists\mathsf{f}$ and $\mathsf{R}\forall\mathsf{f}$ rules are special in that they only deal with concepts $\exists f.C$ and $\forall f.C$ where f is an abstract feature. As we will see later, concepts $\exists R.C$ and $\forall R.C$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ are not treated by completion rules but through recursion calls of the algorithm. The Rfe rule also deserves some attention: it ensures that, for any object $a \in \mathsf{O_a}$, there exists at most a single f-successor for each $f \in \mathsf{N_{aF}}$ and at most a single g-successor for each $g \in \mathsf{N_{cF}}$. Redundant successors are eliminated by identification. This process is often referred to as *fork elimination* (hence the name of the rule). In many cases, fork elimination is not explicitly formulated as a completion rule but viewed as an integral part of the other completion rules. In the presence of feature (dis)agreements, this latter approach seems to be less transparent. Consider for example the ABox

$$\{a : \exists f_1.\top,\ a : \exists f_2.\top,\ a : f_1{\downarrow}f_2\}.$$

Assume the $\mathsf{R}\exists\mathsf{f}$ rule is applied twice adding the assertions $(a, b) : f_1$ and $(a, c) : f_2$. Now, the $\mathsf{R}\downarrow$ rule is applied adding $(a, b') : f_1$ and $(a, b') : f_2$. Clearly, we may now apply the Rfe rule to the assertions $(a, b) : f_1$ and $(a, b') : f_1$. Say the rule application replaces b' by b, and we obtain the ABox

$$\{a : \exists f_1.\top,\ a : \exists f_2.\top,\ a : f_1{\downarrow}f_2,\ (a, b) : f_1,\ (a, c) : f_2,\ (a, b) : f_2\}.$$

Obviously, we may now apply Rfe to $(a, c) : f_2$ and $(a, b) : f_2$ replacing b by c. Observe that this latter fork elimination does not involve any objects generated by

R⊓	if $C_1 \sqcap C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \not\subseteq \mathcal{A}(a)$ then $\mathcal{A} := \mathcal{A} \cup \{a : C_1, a : C_2\}$
R⊔	if $C_1 \sqcup C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \cap \mathcal{A}(a) = \emptyset$ then $\mathcal{A} := \mathcal{A} \cup \{a : C\}$ for some $C \in \{C_1, C_2\}$
R∃f	if $\exists f.C \in \mathcal{A}(a)$ and there is no f-successor b of a with $C \in \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{(a, b) : f,\ b : C\}$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R∀f	if $\forall f.C \in \mathcal{A}(a)$, b is an f-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$
Rc	if $\exists u_1, \ldots, u_n.P \in \mathcal{A}(a)$ and there exist no $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is u_i-successor of a for $1 \leq i \leq n$ and $(x_1, \ldots, x_n) : P \in \mathcal{A}$ then set $\mathcal{A} := (\mathcal{A} + au_1x_1 + \cdots + au_nx_n) \cup \{(x_1, \ldots, x_n) : P\}$ with $x_1, \ldots, x_n \in \mathsf{O_c}$ fresh in $\mathcal{A}$
R↓	if $p_1 \downarrow p_2 \in \mathcal{A}(a)$ and there is no b that is both a p_1-successor of a and a p_2-successor of a then set $\mathcal{A} := \mathcal{A} + ap_1b + ap_2b$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R↑	if $p_1 \uparrow p_2 \in \mathcal{A}(a)$ and there are no b_1, b_2 with b_1 p_1-successor of a, b_2 p_2-successor of a, and $(b_1 \not\sim b_2) \in \mathcal{A}$ then set $\mathcal{A} := (\mathcal{A} + ap_1b_1 + ap_2b_2) \cup \{(b_1 \not\sim b_2)\}$ for $b_1, b_2 \in \mathsf{O_a}$ fresh in $\mathcal{A}$
Rfe	if $\{(a, b) : f, (a, c) : f\} \subseteq \mathcal{A}$ and $b \neq c$ (resp. $\{(a, x) : g, (a, y) : g\} \subseteq \mathcal{A}$ and $x \neq y$) then replace b by c in $\mathcal{A}$ (resp. x by y)

FIG. 4. Completion rules for $\mathcal{ALCF}(\mathcal{D})$.

the last "non-Rfe" rule application. To make such effects more transparent, we chose to formulate fork elimination as a separate rule.

Let us now formalize what it means for an ABox to be contradictory.

Definition 3.5 (Clash) With each ABox $\mathcal{A}$, we associate a predicate conjunction

$$\zeta_{\mathcal{A}} = \bigwedge_{(x_1,\ldots,x_n):P \in \mathcal{A}} P(x_1, \ldots, x_n).$$

The ABox $\mathcal{A}$ is called *concrete domain satisfiable* iff $\zeta_{\mathcal{A}}$ is satisfiable. It is said to contain a *clash* iff one of the following conditions applies:

1. $\{A, \neg A\} \subseteq \mathcal{A}(a)$ for a concept name A and object $a \in \mathsf{O_a}$,
2. $(a \not\sim a) \in \mathcal{A}$ for some object $a \in \mathsf{O_a}$,
3. $g\uparrow \in \mathcal{A}(a)$ for some $a \in \mathsf{O_a}$ such that there exists a g-successor of a, or
4. $\mathcal{A}$ is not concrete domain satisfiable.

If $\mathcal{A}$ does not contain a clash, then $\mathcal{A}$ is called *clash-free*.

define procedure $\mathsf{sat}(\mathcal{A})$
  $\mathcal{A} := \mathsf{fcompl}(\mathcal{A})$
  if $\mathcal{A}$ contains a clash **then**
    return unsatisfiable
  forall assertions $\exists R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ **do**
    Fix $b \in \mathsf{O_a}$
    if $\mathsf{sat}(\{b : C\} \cup \{b : E \mid \forall R.E \in \mathcal{A}(a)\}) = \mathsf{unsatisfiable}$ **then**
      return unsatisfiable
  return satisfiable

define procedure $\mathsf{fcompl}(\mathcal{A})$
  while a rule from Figure 4 is applicable to $\mathcal{A}$ **do**
    Choose an applicable rule R s.t. $\mathsf{R} = \mathsf{Rfe}$ if Rfe is applicable
    Apply R to $\mathcal{A}$
  return $\mathcal{A}$

FIG. 5. The $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability algorithm.

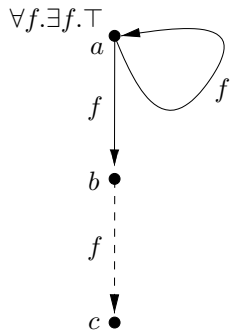


FIG. 6. The "yo-yo" effect.

The completion algorithm itself can be found in Figure 5. We briefly summarize the strategy followed by the algorithm. The argument to sat is an ABox containing exactly one object $a \in \mathsf{O_a}$ and only assertions of the form $a : C$. The algorithm uses the fcompl function to create all feature successors of a, all feature successors of these feature successors and so on. However, fcompl does not generate any R-successors for role names $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. In other words, fcompl generates a cluster of objects as described in Section 3.1. After the call to the fcompl function, the algorithm makes a recursion call for each role successor enforced via an $\exists R.C$ assertion (with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$). A single such recursion call corresponds to moving along a path in a generalized tree model, i.e, to moving to a successor cluster of the cluster under consideration. Each cluster of objects is checked separately for contradictions. Note that, due to Definition 3.5, checking for a clash involves checking whether the predicate conjunction $\zeta_{\mathcal{A}}$ is satisfiable. This, in turn, is a decidable problem since we assume $\mathcal{D}$ to be admissible.

R∃r	if $\exists R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ and there is no R-successor b of a with $C \in \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{(a,b) : R\ b : C\}$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R∀r	if $\forall R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, b is a R-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$

FIG. 7. Virtual completion rules for $\mathcal{ALCF}(\mathcal{D})$.

Observe that fcompl applies the Rfe rule with highest priority. Without this strategy, the algorithm would not terminate: consider the ABox

$$\mathcal{A} = \{a : \forall f.\exists f.\top,\ (a,a) : f,\ (a,b) : f\}.$$

This ABox, which is depicted in the upper part of Figure 6, is encountered if, for example, the algorithm is started on the input concept $f'{\downarrow}f'f \sqcap \exists f'.(\forall f.\exists f.\top \sqcap \exists f.\top)$. Now assume that the completion rules are applied to $\mathcal{A}$ without giving Rfe the highest priority. This means that we can apply the R∀f rule and obtain $b : \exists f.\top$. We can then apply R∃f generating $(b,c) : f,\ c : \top$. Fork elimination may now identify a and b and thus we are back at the initial situation (up to renaming). Clearly, this sequence of rule applications may be repeated indefinitely—the algorithm does not terminate. This "yo-yo" effect was also described, e.g., in [9].

3.3 Correctness and Complexity

In this section, we prove that the completion algorithm is sound, complete, and terminating and can be executed using only polynomial space provided that $\mathcal{D}$-satisfiability is in PSPACE. With D, we denote the input concept to the completion algorithm whose satisfiability is to be decided.

We first prove termination of the algorithm. It is convenient to start with establishing an upper bound for the number of rule applications performed by the fcompl function and, closely related, an upper bound for the size of ABoxes generated by the fcompl function. Before we do this, let us introduce the two additional completion rules displayed in Figure 7, which will play an important role in the termination and correctness proofs. These rules are not applied explicitly by the algorithm, but rather can the recursion calls of the sat function be viewed as a single application of the R∃r rule together with multiple applications of the R∀r rule. Let us now return to the upper bounds for the fcompl function. With foresight to the ABox consistency algorithm to be devised in the next section, we consider the precompl function instead of the fcompl function, where precompl is defined exactly as fcompl except that it also applies the R∀r rule. A formal definition of the precompl function can be found in Figure 9. It is not hard to see that upper bounds for the number of rule applications performed by precompl or the size of ABoxes generated by precompl also apply to the fcompl function: if the fcompl functions perform a computation on an input ABox $\mathcal{A}$, then precompl can perform precisely the same computation on the input ABox $\mathcal{A}'$ obtained from $\mathcal{A}$ by replacing all subconcept $\forall R.C$ appearing in $\mathcal{A}$ with concept names.

In what follows, we use $\mathsf{sub}(C)$ to denote the set of subconcepts of the concept C and $\mathsf{sub}(\mathcal{A})$ to denote the union of the sets of subconcepts of all those concepts C that appear in assertions $a : C$ in the ABox $\mathcal{A}$. Moreover, we use $|C|$ to denote the *length* of a concept C, i.e., the number of symbols used to write it down. The *size* $|\alpha|$ of an ABox assertion α is defined as $|C|$ if $\alpha = a : C$ and 1 otherwise. The size $|\mathcal{A}|$ of an ABox $\mathcal{A}$ is defined as the sum of the sizes of its assertions.

Lemma 3.6 For any input $\mathcal{A}$, the function $\mathsf{precompl}$ terminates after at most $|\mathcal{A}|^4$ rule applications and constructs an ABox $\mathcal{A}'$ with $|\mathcal{A}'| \leq |\mathcal{A}|^6$.

PROOF. In the following, we call assertions of the form $a : C$ *concept assertions*, assertions of the form $(a, b) : f$ or $(a, x) : g$ *feature assertions*, and assertions of the form $(a, b) : R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ *role assertions*.

The main task is to show that

$$\mathsf{precompl} \text{ terminates after at most } |\mathcal{A}|^4 \text{ rule applications.} \qquad (*)$$

For suppose that $(*)$ has been shown. We can then prove the lemma by making the following two observations, which clearly imply that the size of the ABox $\mathcal{A}'$ generated by $\mathsf{precompl}$ is bounded by $|\mathcal{A}|^6$.

(i) We have $|\alpha| < |\mathcal{A}|$ for each new assertion α added by rule application: concept assertions are the only kind of assertions that may have a size greater than one and, if a concept assertion $a : C$ is added by rule application, then $C \in \mathsf{sub}(\mathcal{A})$;

(ii) Each rule application adds at most $|\mathcal{A}|$ new assertions: each application adds either no new assertions (the Rfe rule) or at most $|C|$ new assertions, where $a : C$ is the concept assertion appearing in the (instantiated) rule premise. In the latter case, we have $|C| \leq |\mathcal{A}|$ since C is in $\mathsf{sub}(\mathcal{A})$.

Hence, let us prove $(*)$. Let $\mathcal{A}_0, \mathcal{A}_1, \ldots$ be the sequence of ABoxes computed by $\mathsf{precompl}$. More precisely, $\mathcal{A}_0 = \mathcal{A}$ and $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by the i-th rule application performed by $\mathsf{precompl}$.

We first introduce some notions. For $i \geq 0$ and $a \in \mathsf{O_a} \cup \mathsf{O_c}$, we use $\mathsf{nm}_i(a)$ to denote the set of names that a had "until $\mathcal{A}_i$". More precisely, $\mathsf{nm}_0(a) = \{a\}$ for all $a \in \mathsf{O_a} \cup \mathsf{O_c}$. If the Rfe rule is applied to an ABox $\mathcal{A}_i$ renaming an object a to b, then $\mathsf{nm}_{i+1}(b) = \mathsf{nm}_i(a) \cup \mathsf{nm}_i(b)$ and $\mathsf{nm}_{i+1}(c) = \mathsf{nm}_i(c)$ for all $c \neq b$. For all other rule applications, we simply have $\mathsf{nm}_{i+1}(a) = \mathsf{nm}_i(a)$ for all $a \in \mathsf{O_a} \cup \mathsf{O_c}$. The following properties, which we summarize under the notion *persistence*, are easily proved using the fact that assertions are never deleted:

- If $a : C \in \mathcal{A}_i$ and $a \in \mathsf{nm}_j(a')$ for some $j > i$ and $a' \in \mathsf{O_a}$, then $a' : C \in \mathcal{A}_j$.
- if $(a, b) : R \in \mathcal{A}_i$, $a \in \mathsf{nm}_j(a')$, and $b \in \mathsf{nm}_j(b')$ for some $j > i$ and $a', b' \in \mathsf{O_a}$, then $(a', b') : R \in \mathcal{A}_j$.
- If $(a, x) : g \in \mathcal{A}_i$, $a \in \mathsf{nm}_j(a')$, and $x' \in \mathsf{nm}_j(x)$ for some $j > i$, $a' \in \mathsf{O_a}$, and $x' \in \mathsf{O_c}$, then $(a', x') : g \in \mathcal{A}_j$.
- If $(x_1, \ldots, x_n) : P \in \mathcal{A}_i$ and $x'_i \in \mathsf{nm}_j(x_i)$ for $1 \leq i \leq n$, then $(x'_1, \ldots, x'_n) : P \in \mathcal{A}_j$.

A concept assertion $a : C$ is called *touched* in $\mathcal{A}_i$ if there exists an $a' \in \mathsf{nm}_i(a)$ such that one of the first i rule applications involved $a' : C$ in the (instantiated)

rule premise and *untouched* otherwise. By $\sharp_{\mathsf{feat}}(\mathcal{A})$, we denote the number of feature assertions in $\mathcal{A}$. For role assertions $(a,b):R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, we use $\lambda_{\mathcal{A}_i}(a,b:R)$ to denote the number of concepts $\forall R.C$ in $\mathsf{sub}(\mathcal{A})$ for which there exist no $a' \in \mathsf{nm}_i(a)$ and $b' \in \mathsf{nm}_i(b)$ such that one of the first i rule applications involved both $a' : \forall R.C$ and $(a',b') : R$ in the (instantiated) rule premise.

For $i \geq 0$, define

$$w(\mathcal{A}_i) := \sum_{a:C \text{ is untouched in } \mathcal{A}_i} |a:C| \quad + \quad \sharp_{\mathsf{feat}}(\mathcal{A}_i) \quad + \quad |\mathcal{A}| \cdot \sum_{(a,b):R \in \mathcal{A}_i} \lambda_{\mathcal{A}_i}(a,b:R).$$

We show that $w(\mathcal{A}_{i+1}) < w(\mathcal{A}_i)$ for $i \geq 0$, which implies that the length of the sequence $\mathcal{A}_0, \mathcal{A}_1, \ldots$ is bounded by $|\mathcal{A}|^4$ since it is readily checked that $w(\mathcal{A}_0) \leq |\mathcal{A}|^4$. A case distinction is made according to the completion rule applied.

- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R}\sqcap$ rule. By definition of this rule and due to persistence, it is applied to an untouched assertion $a : C_1 \sqcap C_2$ in $\mathcal{A}_i$: for suppose that $a : C_1 \sqcap C_2$ is touched in $\mathcal{A}_i$. By definition of "touched", this implies that there exists an $a' \in \mathsf{nm}_i(a)$ such that $\mathsf{R}\sqcap$ has been applied to $a' : C_1 \sqcap C_2$ in the j-th rule application for some $j < i$. By definition of $\mathsf{R}\sqcap$, this implies $\{a' : C_1, a' : C_2\} \subseteq \mathcal{A}_j$. By persistence, we have $\{a : C_1, a : C_2\} \subseteq \mathcal{A}_i$ and, thus, the $\mathsf{R}\sqcap$ rule is not applicable to $a : C_1 \sqcap C_2$ in $\mathcal{A}_i$ which is a contradiction. Hence, we have shown that $a : C_1 \sqcap C_2$ is untouched in $\mathcal{A}_i$. Moreover, this assertion is clearly touched in $\mathcal{A}_{i+1}$. The rule application generates new concept assertions $a : C_1$ and $a : C_2$ which may both be untouched in $\mathcal{A}_{i+1}$. Moreover, it generates no new feature and role assertions. By definition of the size of assertions and the length of concepts, we have $|a : C_1 \sqcap C_2| > |a : C_1| + |a : C_2|$. Thus $w(\mathcal{A}_{i+1}) < w(\mathcal{A}_i)$.
- The $\mathsf{R}\sqcup$ case is analogous to the previous case.
- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R}\forall\mathsf{f}$ rule. The rule is applied to assertions $a : \forall f.C$ and $(a,b) : f$. Suppose that $a : \forall f.C$ is touched in $\mathcal{A}_i$, i.e., that the $\mathsf{R}\forall\mathsf{f}$ rule has been applied in a previous step to an assertion $a' : \forall f.C$ with $a' \in \mathsf{nm}_i(a)$. It then added $c : C$ for an f-successor c of a'. The facts that (i) Rfe is applied with highest priority, (ii) b is an f-successor of a in $\mathcal{A}_{i+1}$, and (iii) the $\mathsf{R}\forall\mathsf{f}$ rule is applicable imply that we have $c \in \mathsf{nm}_i(b)$. This, in turn, implies $b : C \in \mathcal{A}_i$ by persistence and we have obtained a contradiction to the assumption that $\mathsf{R}\forall\mathsf{f}$ is applicable. Hence, we have shown that $a : \forall f.C$ is untouched in $\mathcal{A}_i$. The assertion is touched in $\mathcal{A}_{i+1}$. Rule application generates a new assertion $b : C$ that is untouched in $\mathcal{A}_{i+1}$. However, $|a : \forall f.C| > |b : C|$. No new feature or role assertions are generated.
- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R}\forall\mathsf{r}$ rule. The rule is applied to assertions $a : \forall R.C$ and $(a,b) : R$ in $\mathcal{A}_i$. Due to persistence, there do not exist $a' \in \mathsf{nm}_i(a)$ and $b' \in \mathsf{nm}_i(b)$ such that the $\mathsf{R}\forall\mathsf{r}$ rule has previously been applied to $a' : \forall R.C$ and $(a',b') : R$. Hence, $\lambda_{\mathcal{A}_{i+1}}(a,b:R) = \lambda_{\mathcal{A}_i}(a,b:R) - 1$ and the third summand of $w(\mathcal{A}_i)$ exceeds the third summand of $w(\mathcal{A}_{i+1})$ by $|\mathcal{A}|$. The rule application adds no feature or role assertions and a single concept assertion $b : C$. Since $\forall R.C \in \mathsf{sub}(\mathcal{A})$, we have $|b : C| < |\mathcal{A}|$ and hence $w(\mathcal{A}_{i+1}) < w(\mathcal{A}_i)$.
- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R}\exists\mathsf{f}$ rule. As in the $\mathsf{R}\sqcap$ case, it is easy to show that the rule is applied to an untouched assertion

$a : \exists f.C$. It generates new assertions $(a, b) : f$ and $b : C$ (and no new role assertions). The assertion $b : C$ is untouched in $\mathcal{A}_{i+1}$ and $a : \exists f.C$ is touched in $\mathcal{A}_{i+1}$. The new feature assertion $(a, b) : f$ yields $\sharp_{\mathsf{feat}}(\mathcal{A}_{i+1}) = \sharp_{\mathsf{feat}}(\mathcal{A}_i)+1$. On the other hand, no role assertion is added and we clearly have $|a : \exists f.C| > |b : C| + 1$.

- The Rc, $\mathsf{R}\!\downarrow$, and $\mathsf{R}\!\uparrow$ rules touch a (due to persistence) previously untouched concept assertion $a : C$ appearing in the instantiated premise and do not add new concept or role assertions. It is readily checked that the number of feature assertions added by rule application is smaller than $|a : C|$.
- Assume that the Rfe rule is applied to an ABox $\mathcal{A}_i$. This obviously implies $\sharp_{\mathsf{feat}}(\mathcal{A}_{i+1}) < \sharp_{\mathsf{feat}}(\mathcal{A}_i)$, i.e., the second summand of $w(\mathcal{A}_{i+1})$ is strictly smaller than the second summand of $w(\mathcal{A}_i)$. If the rule application renames a concrete object, these are the only changes and we are done. If an abstract object is renamed, some work is necessary to show that the first and third summand of $w(\mathcal{A}_{i+1})$ are not greater than the corresponding summands of $w(\mathcal{A}_i)$. Assume that $a \in \mathsf{O_a}$ is renamed to b. We then have $\mathsf{nm}_{i+1}(b) = \mathsf{nm}_i(a) \cup \mathsf{nm}_i(b)$.
  - First summand. Let us first consider concept assertions $c : C \in \mathcal{A}_{i+1} \cap \mathcal{A}_i$. Such an assertion is untouched in $\mathcal{A}_{i+1}$ only if it is untouched in $\mathcal{A}_i$ since (i) $\mathsf{nm}_{i+1}(c) = \mathsf{nm}_i(c)$ if $c \neq b$ and (ii) $\mathsf{nm}_i(b) \subseteq \mathsf{nm}_{i+1}(b)$ if $c = b$. Moreover, if there exists an assertion $b : C \in \mathcal{A}_{i+1} \setminus \mathcal{A}_i$ due to variable renaming, then $a : C \in \mathcal{A}_i \setminus \mathcal{A}_{i+1}$, and $b : C$ being untouched in $\mathcal{A}_{i+1}$ implies $a : C$ being untouched in $\mathcal{A}_i$ since $\mathsf{nm}_i(a) \subseteq \mathsf{nm}_{i+1}(b)$. Hence, the first summand does not increase.
  - Third summand. Let $(c, d) : R \in \mathcal{A}_{i+1} \cap \mathcal{A}_i$ (implying $c \neq a$ and $d \neq a$). We distinguish several subcases:
    1. $c \neq b$ and $d \neq b$. Then, clearly, $\lambda_i(c, d : R) = \lambda_{i+1}(c, d : R)$.
    2. $c = b$ and $d \neq b$. By definition of λ_i, $\mathsf{nm}_i(b) \subseteq \mathsf{nm}_{i+1}(b)$ implies $\lambda_i(b, d : R) \geq \lambda_{i+1}(b, d : R)$.
    3. $c \neq b$ and $d = b$. As previous case.
    4. $c = d = b$. As previous case.

    Now let $(c, d) : R \in \mathcal{A}_{i+1} \setminus \mathcal{A}_i$ (implying $c = b$ or $d = b$). We can distinguish the cases (i) $c = b$, $d \neq b$, (ii) $d = b$, $c \neq b$, and (iii) $c = d = b$. Since all cases are similar, we concentrate on (i). In this case, $(a, d) : R \in \mathcal{A}_i \setminus \mathcal{A}_{i+1}$. Moreover, $\mathsf{nm}_i(a) \subseteq \mathsf{nm}_{i+1}(b)$ implies $\lambda_{\mathcal{A}_{i+1}}(b, d : R) \leq \lambda_{\mathcal{A}_i}(a, d : R)$.
    Summing up, the third summand may only decrease but not increase.

∎

The role depth of concepts is defined inductively as follows, where $|p|$ denotes the length of the abstract path p and $|u|$ denotes the length of the concrete path u (including the trailing concrete feature):

- $\mathsf{rd}(A) = \mathsf{rd}(g\!\uparrow) = 0$;
- $\mathsf{rd}(\exists u_1, \ldots, u_n.P) = \max(|u_1|, \ldots, |u_n|)$;
- $\mathsf{rd}(p_1\!\downarrow p_2) = \mathsf{rd}(p_1\!\uparrow p_2) = \max(|p_1|, |p_2|)$;
- $\mathsf{rd}(\neg C) = \mathsf{rd}(C)$;
- $\mathsf{rd}(C \sqcap D) = \mathsf{rd}(C \sqcup D) = \max(\mathsf{rd}(C), \mathsf{rd}(D))$;
- $\mathsf{rd}(\exists R.C) = \mathsf{rd}(\forall R.C) = \mathsf{rd}(C) + 1$;

We now prove a technical lemma that, together with Lemma 3.6, immediately yields termination.

Lemma 3.7 Assume that the completion algorithm was started with input D. Then

1. in each recursion call, the size $|\mathcal{A}|$ of the argument $\mathcal{A}$ passed to sat is bounded by $|D|^2$;
2. in each recursion step of sat, at most $p(|D|)$ recursion calls are made, where p is a polynomial; and
3. the recursion depth of sat is bounded by $|D|$.

PROOF. Let us first prove Point 1. ABoxes passed to sat contain assertions of the form $a : C$ for a single object a. Since only concepts from $\mathsf{sub}(D)$ are generated during rule application, the number of distinct assertions of this form is bounded by $|\mathsf{sub}(D)| \leq |D|$. Obviously, the size of each such assertion is also bounded by $|D|$ which yields an upper bound of $|D|^2$ for the size of arguments to sat.

For Point 2, note that in each recursion step, the number of recursion calls made is bounded by the number of assertions $a : \exists R.C$ in the ABox $\mathcal{A}$ obtained by application of fcompl. By Point 1, the size of argument ABoxes to sat is bounded by $|D|^2$. Hence, by Lemma 3.6, the size of $\mathcal{A}$ is bounded by $p(|D|)$ where p is a polynomial and the same bound applies to the number of recursion calls made in each recursion step.

We now turn to Point 3. As a consequence of (i) the fact that rule application performed by fcompl may not introduce concepts with a role depth greater than the role depth of concepts that have already been in the ABox and (ii) the way in which the argument ABoxes for recursion calls to sat are constructed, we have that the role depth of concepts in the argument ABoxes passed to sat strictly decreases with recursion depth. It follows that the role depth of D is an upper bound for the recursion depth, i.e., the recursion depth is bounded by $|D|$. ∎

Proposition 3.8 The completion algorithm terminates on any input $\mathcal{A}_D$.

PROOF. Immediate consequence of Lemma 3.6 and Points 2 and 3 from Lemma 3.7. ∎

We now come to proving soundness and completeness of the completion algorithm. Recall that, intuitively, the completion algorithm traverses a generalized tree model in a depth-first manner without keeping the entire model in memory. For the proofs, it is convenient to make the model traversed by the algorithm explicit—or more precisely the ABox representing it. To do this, we define an extended version of the completion algorithm. This extended algorithm is identical to the original one but additionally constructs a sequence of ABoxes $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \ldots$ collecting all assertions that the algorithm generates. Hence, it returns satisfiable if and only if the original algorithm does. We will show that, if the extended algorithm is started on an initial ABox $\mathcal{A}_D$ and terminates after n steps returning satisfiable, then the ABox $\mathcal{A}^n_\cup$ defines a canonical model for $\mathcal{A}_D$. Since the extended algorithm returns satisfiable if the original one does, this yields soundness. Completeness can also be shown using the correspondence between the two algorithms. Note that the extended version of the algorithm is defined just to prove soundness and completeness of the original version and we do not claim that the extended version itself can be executed in polynomial space.

```
*  Initialization:
*  rc := sc := 0
*  A^0_∪ := {a_0 : D} if A_D = {a : D}

   define procedure sat(A)
      A := fcompl(A)
      if A contains a clash then
         return unsatisfiable
      forall assertions ∃R.C ∈ A(a) with R ∈ N_R \ N_aF do
*        sc := sc + 1
*        rc := rc + 1
         Fix b ∈ O_a
*        A^rc_∪ := A^{rc-1}_∪ ∪ {(a_{sc-1}, b_sc) : R} ∪ {b_sc : C} ∪
*                                  {b_sc : E | a : ∀R.E ∈ A(a)}
         if sat({b : C} ∪ {b : E | ∀R.E ∈ A(a)}) = unsatisfiable then
            return unsatisfiable
      return satisfiable

   define procedure fcompl(A)
*     A_0 := A
      while a rule R from Figure 4 is applicable to A do
         Choose an applicable rule R s.t. R = Rfe if Rfe is applicable
         Apply R to A
*     rc := rc + 1
*     N := A \ A_0
*     Replace each a ∈ O_a (resp. x ∈ O_c) in N with a_sc (resp. x_sc)
*     A^rc_∪ := A^{rc-1}_∪ ∪ N
      return A
```

FIG. 8. The extended satisfiability algorithm.

The extended algorithm can be found in Figure 8. The extensions are marked with asterisks. If the algorithm is started on the initial ABox $\mathcal{A}_D = \{a : D\}$, we set $\mathcal{A}^0_\cup := \{a_0 : D\}$. The algorithm uses two *global* variables sc and rc, which are both initialized with the value 0. The first one is a counter for the number of calls to the sat function. The second one counts the number of ABoxes $\mathcal{A}^i_\cup$ that have already been generated. The introduction of the global variable sc is necessary due to the following technical problem: the object names created by the algorithm are unique only within the ABox considered in a single recursion step. For the accumulating ABoxes $\mathcal{A}^i_\cup$ that collect assertions from many recursion steps, we have to ensure that an object a from one recursion step can be distinguished from a in a different step since these two objects do clearly not represent the same domain element in the constructed model. To achieve this, objects are renamed before new assertions are added to an ABox $\mathcal{A}^i_\cup$ by indexing with the value of the counter sc.

Observe that, for $i > 0$, the ABox $\mathcal{A}^i_\cup$ is obtained either

1. by multiple applications of completion rules from Figure 4 to the ABox $\mathcal{A}^{i-1}_\cup$ or

2. by a recursion call made while the counter rc has value $i-1$.

Let us be a little bit more precise about the second point. W.r.t. the sequence of ABoxes $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \ldots$, recursion calls can be viewed as applications of the completion rules displayed in Figure 7: if $\mathcal{A}^i_\cup$ is obtained from $\mathcal{A}^{i-1}_\cup$ by a recursion call, then this is equivalent to a single application of the R$\exists$r rule together with exhaustive application of the R$\forall$r rule.

Non-applicability of all completion rules to an ABox will be an important property in what follows.

Definition 3.9 (Complete ABox) An ABox $\mathcal{A}$ is *complete* iff no completion rule from Figures 4 and 7 is applicable to $\mathcal{A}$.

The following two lemmas are central for proving soundness and completeness.

Lemma 3.10 Let $\mathcal{A}$ be an ABox and R a completion rule from Figure 4 or Figure 7 such that R is applicable to $\mathcal{A}$. Then $\mathcal{A}$ is consistent iff R can be applied such that the resulting ABox $\mathcal{A}'$ is consistent.

PROOF. Let us first deal with the "if" direction. This is trivial if R $\neq$ Rfe since this implies $\mathcal{A} \subseteq \mathcal{A}'$ and, hence, every model of $\mathcal{A}'$ is also a model of $\mathcal{A}$. Assume that the Rfe rule is applied to assertions $\{(a,b) : f, (a,c) : f\} \in \mathcal{A}$ and replaces c with b. Let $\mathcal{I}$ be a model of $\mathcal{A}'$. Construct an interpretation $\mathcal{I}'$ from $\mathcal{I}$ by setting $c^{\mathcal{I}'} := b^{\mathcal{I}}$. It is straightforward to check that $\mathcal{I}'$ is a model of $\mathcal{A}$. The case that Rfe is applied to assertions $\{(a,x) : g, (a,y) : g\} \in \mathcal{A}$ is analogous.

Now for the "only if" direction. We make a case distinction according to the completion rule R.

- The R$\sqcap$ rule is applied to an assertion $a : C_1 \sqcap C_2$ and $\mathcal{A}' = \mathcal{A} \cup \{a : C_1, a : C_2\}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (C_1 \sqcap C_2)^{\mathcal{I}}$, we have $a^{\mathcal{I}} \in C_1^{\mathcal{I}}$ and $a^{\mathcal{I}} \in C_2^{\mathcal{I}}$ by the semantics of $\mathcal{ALCF}(\mathcal{D})$, which implies that $\mathcal{I}$ is also a model of $\mathcal{A}'$.
- The R$\sqcup$ rule is applied to an assertion $a : C_1 \sqcup C_2$. The rule can be applied such that either $\mathcal{A}' = \mathcal{A} \cup \{a : C_1\}$ or $\mathcal{A}' = \mathcal{A} \cup \{a : C_2\}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (C_1 \sqcup C_2)^{\mathcal{I}}$, we have either $a^{\mathcal{I}} \in C_1^{\mathcal{I}}$ or $a^{\mathcal{I}} \in C_2^{\mathcal{I}}$ by the semantics of $\mathcal{ALCF}(\mathcal{D})$. Hence, we can apply the rule such that $\mathcal{I}$ is a model of $\mathcal{A}'$.
- The R$\exists$f rule is applied to an assertion $a : \exists f.C$ yielding the ABox $\mathcal{A}'$. Then $\mathcal{A}' = \mathcal{A} \cup \{(a,b) : f, b : C\}$ where b is fresh in $\mathcal{A}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\exists f.C)^{\mathcal{I}}$, there exists a $d \in \Delta_{\mathcal{I}}$ such that $f^{\mathcal{I}}(a^{\mathcal{I}}) = d$ and $d \in C^{\mathcal{I}}$. Let $\mathcal{I}'$ be the interpretation obtained from $\mathcal{I}$ by setting $a^{\mathcal{I}'} := d$. It is easily checked that $\mathcal{I}'$ is a model of $\mathcal{A}'$.
- The R$\exists$r rule is treated analogously to the previous case.
- The R$\forall$f rule is applied to an assertion $a : \forall f.C$ and $\mathcal{A}' = \mathcal{A} \cup \{b : C\}$ where b is an f-successor of a in $\mathcal{A}$ and $\mathcal{A}'$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\forall f.C)^{\mathcal{I}}$ and $f^{\mathcal{I}}(a^{\mathcal{I}}) = b^{\mathcal{I}}$, we have $b \in C^{\mathcal{I}}$. Hence, $\mathcal{I}$ is also a model of $\mathcal{A}'$.
- The R$\forall$r rule is treated analogously to the previous case.
- The Rc rule is applied to an assertion $a : \exists u_1, \ldots, u_n.P$ with $u_i = f_1^{(i)} \cdots f_{k_i}^{(i)} g_i$ yielding the ABox $\mathcal{A}'$. Then there exist abstract objects $a_j^{(i)}$ with $1 \leq i \leq n$ and $1 \leq j \leq k_i$ which are fresh in $\mathcal{A}$ and concrete objects $x_1, \ldots, x_n$ which are fresh in $\mathcal{A}$ such that, for $1 \leq i \leq n$,

– $a_1^{(i)}$ is $f_1^{(i)}$-successor of a,
– $a_j^{(i)}$ is $f_j^{(i)}$-successor of $a_{j-1}^{(i)}$ for $1 < j \leq k_i$,
– x_i is g_i-successor of $a_{k_i}^{(i)}$, and
– $(x_1, \ldots, x_n) : P \in \mathcal{A}'$.

Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\exists u_1, \ldots, u_n.P)^{\mathcal{I}}$, there exist domain elements $d_j^{(i)} \in \Delta_{\mathcal{I}}$ with $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and $z_1, \ldots, z_n \in \Delta_{\mathcal{D}}$ such that, for $1 \leq i \leq n$, we have
– $(a^{\mathcal{I}}, d_1^{(i)}) \in (f_1^{(i)})^{\mathcal{I}}$,
– $(d_{j-1}^{(i)}, d_j^{(i)}) \in (f_j^{(i)})^{\mathcal{I}}$ for $1 < j \leq k_i$,
– $g_i^{\mathcal{I}}(d_{k_i}^{(i)}) = z_i$, and
– $(z_1, \ldots, z_n) \in P^{\mathcal{D}}$.

Define $\mathcal{I}'$ as the interpretation obtained from $\mathcal{I}$ by setting

$$(a_j^{(i)})^{\mathcal{I}'} := d_j^{(i)} \text{ for } 1 \leq i \leq n \text{ and } 1 < j \leq k_i$$

and

$$x_i^{\mathcal{I}'} := z_i \text{ for all } i \text{ with } 1 \leq i \leq n.$$

It is straightforward to check that $\mathcal{I}'$ is a model of $\mathcal{A}'$.

- Applications of the $\mathsf{R}\downarrow$ rule are treated similar to the previous case.
- Applications of the $\mathsf{R}\uparrow$ rule are also treated similar to the Rc case.
- The Rfe rule is applied to assertions $\{(a, b) : f, (a, c) : f\} \in \mathcal{A}$ and replaces c with b. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Due to the presence of the above two assertions and since features are interpreted as partial functions, we have $b^{\mathcal{I}} = c^{\mathcal{I}}$. It is readily checked that this implies that $\mathcal{I}$ is a model of $\mathcal{A}'$. The case that two concrete objects are identified can be treated in the same way.

∎

Lemma 3.11 Let $\mathcal{A}$ be an ABox. If $\mathcal{A}$ is complete and clash-free, then it is consistent.

PROOF. Based on $\mathcal{A}$, a canonical interpretation $\mathcal{I}$ can be defined as follows. Fix a solution δ for $\zeta_{\mathcal{A}}$ which exists since $\mathcal{A}$ is clash-free.

1. $\Delta_{\mathcal{I}}$ consists of all abstract objects used in $\mathcal{A}$,
2. $A^{\mathcal{I}} := \{a \in \mathsf{O_a} \mid a : A \in \mathcal{A}\}$ for all $A \in \mathsf{N_C}$,
3. $R^{\mathcal{I}} := \{(a, b) \in \mathsf{O_a} \times \mathsf{O_a} \mid (a, b) : R \in \mathcal{A}\}$ for all $R \in \mathsf{N_R}$,
4. $g^{\mathcal{I}} := \{(a, \delta(x)) \in \mathsf{O_a} \times \Delta_{\mathcal{D}} \mid (a, x) : g \in \mathcal{A}\}$ for all $g \in \mathsf{N_{cF}}$,
5. $a^{\mathcal{I}} := a$ for all $a \in \mathsf{O_a}$, and
6. $x^{\mathcal{I}} := \delta(x)$ for all $x \in \mathsf{O_c}$.

Note that $\mathcal{I}$ is well-defined: Since the Rfe rule is not applicable, $f^{\mathcal{I}}$ and $g^{\mathcal{I}}$ are functional for all $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. We prove that $\mathcal{I}$ is a model of $\mathcal{A}$, i.e., that all assertions in $\mathcal{A}$ are satisfied by $\mathcal{I}$. It is an immediate consequence of the definition of $\mathcal{I}$ that $(a, b) : R \in \mathcal{A}$ implies $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $(a, x) : g \in \mathcal{A}$ implies $g^{\mathcal{I}}(a^{\mathcal{I}}) = x^{\mathcal{I}}$. Moreover, if $(a\neg \sim b) \in \mathcal{A}$, then $a \neq b$ since $\mathcal{A}$ is clash-free. Hence, $(a\neg \sim b) \in \mathcal{A}$ implies $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Since δ is a solution for $\zeta_{\mathcal{A}}$, $(x_1, \ldots, x_n) : P \in \mathcal{A}$

implies $(x_1^{\mathcal{I}}, \ldots, x_n^{\mathcal{I}}) \in P^{\mathcal{D}}$. It thus remains to show that $a : C \in \mathcal{A}$ implies $a \in C^{\mathcal{I}}$. This is done by induction on the structure of C. For the induction start, we make a case distinction according to the form of C:

- If $C \in \mathsf{N_C}$, then the above claim is an immediate consequence of the definition of C.
- $C = \neg E$. Since we assume all concepts to be in negation normal form, E is a concept name. Since $\mathcal{A}$ is clash-free, $a : E \notin \mathcal{A}$ and, by definition of $\mathcal{I}$, $a \notin E^{\mathcal{I}}$. Hence, $a \in (\neg E)^{\mathcal{I}}$.
- $C = \exists u_1, \ldots, u_n.P$. Since the Rc rule is not applicable to $\mathcal{A}$, there exist $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is u_i-successor of a in $\mathcal{A}$ for $1 < i \leq n$. By definition of $\mathcal{I}$, we have $u_i^{\mathcal{I}}(a) = \delta(x_i)$ for $1 < i \leq n$. Furthermore, we have $(x_1, \ldots, x_n) : P \in \mathcal{A}$ and, since δ is a solution for $\zeta_{\mathcal{P}}$, $(\delta(x_1), \ldots, \delta(x_n)) \in P^{\mathcal{D}}$. Summing up, $a \in (\exists u_1, \ldots, u_n.P)^{\mathcal{I}}$.
- $C = p_1{\downarrow}p_2$. Since the $\mathsf{R}{\downarrow}$ rule is not applicable to $\mathcal{A}$, there exists an object $b \in \mathsf{O_a}$ which is both a p_1-successor and a p_2-successor of a in $\mathcal{A}$. By definition of $\mathcal{I}$, we have $p_1^{\mathcal{I}}(a) = p_2^{\mathcal{I}}(a) = b$ and, hence, $a \in (p_1{\downarrow}p_2)^{\mathcal{I}}$.
- $C = p_1{\uparrow}p_2$. Since the $\mathsf{R}{\uparrow}$ rule is not applicable to $\mathcal{A}$, there exist $b_1, b_2 \in \mathsf{O_a}$ such that b_1 is a p_1-successor of a in $\mathcal{A}$, b_2 is a p_2-successor of a in $\mathcal{A}$, and $b_1 \neg \sim b_2 \in \mathcal{A}$. Since $\mathcal{A}$ is clash-free, we have $b_1 \neq b_2$. By definition of $\mathcal{I}$, we have $p_1^{\mathcal{I}}(a) = b_1$ and $p_2^{\mathcal{I}}(a) = b_2$ and, hence, $a \in (p_1{\uparrow}p_2)^{\mathcal{I}}$.
- $C = g{\uparrow}$. Since $\mathcal{A}$ is clash-free, a has no g-successor x in $\mathcal{A}$. By definition of $\mathcal{I}$, $g^{\mathcal{I}}(a)$ is undefined and hence $a \in (g{\uparrow})^{\mathcal{I}}$.

For the induction step, we make a case analysis according to the topmost constructor in C.

- $C = C_1 \sqcap C_2$. Since the $\mathsf{R}{\sqcap}$ rule is not applicable to $\mathcal{A}$, we have $\{C_1, C_2\} \subseteq \mathcal{A}(a)$. By induction, $a \in C_1^{\mathcal{I}}$ and $a \in C_2^{\mathcal{I}}$, which implies $a \in (C_1 \sqcap C_2)^{\mathcal{I}}$.
- $C = C_1 \sqcup C_2$. Similar to the previous case.
- $C = \exists R.E$. Since neither the $\mathsf{R}{\exists}\mathsf{f}$ nor the $\mathsf{R}{\exists}\mathsf{r}$ rule is applicable to $\mathcal{A}$, there exists an object $b \in \mathsf{O_a}$ such that b is an R-successor of a in $\mathcal{A}$ and $E \in \mathcal{A}(b)$. By definition of $\mathcal{I}$, b being an R-successor of a implies $(a, b) \in R^{\mathcal{I}}$. By induction, we have $b \in E^{\mathcal{I}}$ and may hence conclude $a \in (\exists R.E)^{\mathcal{I}}$.
- $C = \forall R.E$. Let $b \in \Delta_{\mathcal{I}}$ such that $(a, b) \in R^{\mathcal{I}}$. By definition of $\mathcal{I}$, b is an R-successor of a in $\mathcal{A}$. Since neither the $\mathsf{R}{\forall}\mathsf{f}$ not the $\mathsf{R}{\forall}\mathsf{r}$ rule is applicable to $\mathcal{A}$, we have $E \in \mathcal{A}(b)$. By induction, it follows that $b \in E^{\mathcal{I}}$. Since this holds for all b, we can conclude $a \in (\forall R.E)^{\mathcal{I}}$.

∎

In the following, the *i-th recursion step* denotes the recursion step of the extended completion algorithm in which the counter *sc* has value i.

Proposition 3.12 (Soundness) If the completion algorithm returns satisfiable, then the input concept is satisfiable.

PROOF. Assume that the completion algorithm is started on an input concept D and there exists a way to make the non-deterministic decisions such that the algorithm returns satisfiable. Moreover assume that the extended algorithm constructs the ABox

$\mathcal{A}^n_\cup$ if the non-deterministic decisions are made in precisely the same way, i.e., the counter rc has value n upon termination. We first establish the following claim:

Claim: $\mathcal{A}^n_\cup$ is complete and clash-free.

First for completeness. We distinguish several cases. First assume that a rule

$$\mathsf{R} \in \{\mathsf{R}\sqcap, \mathsf{R}\sqcup, \mathsf{R}\exists\mathsf{f}, \mathsf{Rc}, \mathsf{R}\downarrow, \mathsf{R}\uparrow, \mathsf{R}\exists\mathsf{r}\}$$

is applicable to $\mathcal{A}^n_\cup$. This is due to the presence of an assertion $a_i : C$ in $\mathcal{A}^n_\cup$. If, e.g., $\mathsf{R} = \mathsf{R}\sqcap$, then C has the form $C_1 \sqcap C_2$. By construction of $\mathcal{A}^n_\cup$, this implies that $a : C$ is either part of the argument $\mathcal{A}$ to sat in the i-th recursion call or has been added to $\mathcal{A}$ by the fcompl function during the i-th recursion step. In either case, if $\mathsf{R} \neq \mathsf{R}\exists\mathsf{r}$, the rule R has been applied to $a : C$ by the fcompl function during the i-th recursion step, which, again by construction of $\mathcal{A}^n_\cup$, implies that R is not applicable to $a_i : C$ in $\mathcal{A}^n_\cup$: contradiction. If $\mathsf{R} = \mathsf{R}\exists\mathsf{r}$, then $C = \exists R.E$. Clearly, $(a_i, b_j) : R$ and $b_j : C$ (for some $j > i$) is added to $\mathcal{A}^n_\cup$ due to a subsequent recursion call and we obtain a contradiction to the applicability of $\mathsf{R}\exists\mathsf{r}$ to $a_i : C$ in $\mathcal{A}^n_\cup$.

Now assume that the $\mathsf{R}\forall\mathsf{f}$ rule is applicable to $\mathcal{A}^n_\cup$. This is due to the presence of assertions $a_i : \forall f.C$ and $(a_i, b_j) : f$ in $\mathcal{A}^n_\cup$. Since assertions $(a_i, b_j) : f$ are only added to $\mathcal{A}^n_\cup$ because of applications of the rules $\mathsf{R}\exists\mathsf{f}$, Rc, $\mathsf{R}\downarrow$, and $\mathsf{R}\uparrow$ performed by the fcompl function, we have $i = j$. It follows that $a : \forall f.C$ and $(a, b) : f$ are in $\mathcal{A}$ in the i-th recursion step. Hence, the $\mathsf{R}\forall\mathsf{f}$ rule is applied by fcompl to these assertions. This implies that $b : C$ is in $\mathcal{A}$ in the i-th recursion step which allows us to conclude $b_i : C \in \mathcal{A}^n_\cup$, a contradiction.

Assume that $\mathsf{R}\forall\mathsf{r}$ is applicable to $\mathcal{A}^n_\cup$ due to the presence of assertions $a_i : \forall R.C$ and $(a_i, b_j) : R$. By construction of $\mathcal{A}^n_\cup$, $a_i : \forall R.C$ is in $\mathcal{A}$ in the i-th recursion step and $(a_i, b_j) : R$ has been added to $\mathcal{A}^n_\cup$ due to a recursion call made during the i-th recursion step. By definition of the annotated algorithm, these two facts imply that $b_j : C$ has also been added to $\mathcal{A}^n_\cup$ in the i-th recursion step. Again a contradiction.

To finish the proof that $\mathcal{A}^n_\cup$ is complete, assume that Rfe is applicable to $\mathcal{A}^n_\cup$ due to the presence of assertions $(a_i, b_j) : f$ and $(a_i, c_\ell) : f$. Since assertions $(a_i, b_j) : f$ are only added to $\mathcal{A}^n_\cup$ because of applications of the rules $\mathsf{R}\exists\mathsf{f}$, Rc, $\mathsf{R}\downarrow$, and $\mathsf{R}\uparrow$ performed by the fcompl function, we have $i = j = \ell$. It follows that $(a, b) : f$ and $(a, c) : f$ are in $\mathcal{A}$ in the i-th recursion step. Hence, the Rfe rule is applied by fcompl. This, however, implies that either $(a_i, b_j) : f$ or $(a_i, c_\ell) : f$ is not in $\mathcal{A}^n_\cup$.

We now prove that $\mathcal{A}^n_\cup(a_i)$ is clash-free. Assume $\{A, \neg A\} \subseteq \mathcal{A}^n_\cup(a_i)$. Then $\{A, \neg A\} \subseteq \mathcal{A}(a)$ in the i-th recursion step. Since $\mathcal{A}$ is clash-free in every recursion step (the algorithm returned satisfiable), we obtain a contradiction. Clashes of the form $a_i \neg \sim a_i \in \mathcal{A}^n_\cup$ are treated analogously. Now assume $a_i : g\uparrow$ and $(a_i, x_j) : g$ are in $\mathcal{A}^n_\cup$. Since assertions $(a_i, x_j) : g$ are only added due to applications of the Rc rule by fcompl, we have $i = j$. It is again straightforward to derive a contradiction.

It remains to show that $\mathcal{A}^n_\cup$ is concrete domain satisfiable. For every $i \leq n$, let $\mathcal{A}_i$ be the ABox $\mathcal{A}$ in the i-th recursion step after the application of fcompl and let δ_i be a solution for $\zeta_{\mathcal{A}_i}$, which exists since $\mathcal{A}_i$ is clash-free. Define $\delta(x_i) := \delta_i(x_i)$ for all x_i occurring in $\mathcal{A}^n_\cup$. It is readily checked that δ is a solution for $\zeta_{\mathcal{A}^n_\cup}$: fix an assertion $((x_1)_{h_1}, \ldots, (x_k)_{h_k}) : P \in \mathcal{A}^n_\cup$. Since such assertions are only added due to applications of the Rc rule by fcompl, there exists an $i \leq n$ such that $h_j = i$ for all j with $1 \leq j \leq k$. Hence, $(x_1, \ldots, x_k) : P \in \mathcal{A}_i$ and $(\delta_i(x_1), \ldots, \delta_i(x_k)) \in P^{\mathcal{D}}$. By

definition of δ, it follows that $(\delta((x_1)_{i_1}), \ldots, \delta((x_k)_{i_k})) \in P^{\mathcal{D}}$, as was to be shown.

The proof of the claim is now finished and we return to the proof of soundness. By Lemma 3.11, the claim implies that $\mathcal{A}^n_{\cup}$ is consistent. By construction, we have $a_0 : D \in \mathcal{A}^n_{\cup}$. It immediately follows that D is satisfiable. ∎

Proposition 3.13 (Completeness) If the completion algorithm is started on a satisfiable input concept, then it returns satisfiable.

PROOF. Since the completion algorithm returns satisfiable iff the extended algorithm does, it suffices to concentrate on the extended algorithm. Let the extended completion algorithm be started on an input concept D that is satisfiable. Then, the initial ABox $\mathcal{A}_D = \{a : D\}$ is obviously consistent. By Lemma 3.10 and due to the fact that performing a recursion step corresponds to the application of rules from Figure 7, we can make the non-deterministic decisions of the extended algorithm such that every ABox in the sequence $\mathcal{A}^0_{\cup}, \mathcal{A}^1_{\cup}, \ldots$ is consistent. By Proposition 3.8 and since the extended algorithm terminates iff the original one does, this sequence is comprised of a finite number n of ABoxes. Moreover, the extended algorithm does not detect a clash: if a clash is detected in an ABox $\mathcal{A}$, then we have $\mathcal{A} \subseteq \mathcal{A}^n_{\cup}$ up to variable renaming which clearly contradicts the consistency of $\mathcal{A}^n_{\cup}$. Because of this and again due to Proposition 3.8, the algorithm terminates returning satisfiable. ∎

It may be viewed as a byproduct of the soundness and completeness proof that $\mathcal{ALCF}(\mathcal{D})$ has the generalized tree model property defined in Section 3.1: assume that the extended algorithm is started with initial ABox $\mathcal{A}_D = \{a : D\}$ and that D is satisfiable. By Proposition 3.13 and the correspondence of the original and the extended algorithm, the extended algorithm returns satisfiable. From the proof of Proposition 3.12, we learn that in this case the ABox $\mathcal{A}^n_{\cup}$ (where n is the value of the counter *sc* upon termination) is complete and clash-free. In the proof of Lemma 3.11, a canonical model $\mathcal{I}$ of $\mathcal{A}^n_{\cup}$ is constructed where $\Delta_{\mathcal{I}}$ is the set of abstract objects used in $\mathcal{A}^n_{\cup}$. It is straightforward to check that this model is a generalized tree model for D since

1. $a_0 : D$ is in $\mathcal{A}^n_{\cup}$,

2. the sets $X_i := \{a_i \mid a_i \in \Delta_{\mathcal{I}}\}$ for $0 \leq i \leq n$ are equivalence classes w.r.t. $\mathcal{I}$ and $\sim$ as in Definition 3.1, and

3. due to the recursive nature of the completion algorithm, the graph $(V_{\mathcal{I}}, E_{\mathcal{I}})$ (see Definition 3.1) is a tree.

We now analyze the time and space requirements of our algorithm.

Proposition 3.14 1. If $\mathcal{D}$-satisfiability is in PSPACE, then the completion algorithm can be executed in polynomial space.

2. If $\mathcal{D}$-satisfiability is in NEXPTIME, then the completion algorithm can be executed in nondeterministic exponential time.
3. If $\mathcal{D}$-satisfiability is in EXPSPACE, then the completion algorithm can be executed in exponential space.

PROOF. By Point 1 of Lemma 3.7 and Lemma 3.6, the maximum size of ABoxes $\mathcal{A}$ encountered in recursion steps is bounded by $p(|D|)$, where p is a polynomial. Since, by Point 3 of Lemma 3.7, the recursion depth is bounded by $|D|$, sat can be executed in polynomial space if the check for concrete domain satisfiability is not taken into account.

Assume that $\mathcal{D}$-satisfiability is in PSPACE. Since the maximum size of ABoxes $\mathcal{A}$ encountered in recursion steps is bounded by $p(|D|)$, the maximum number of conjuncts in predicate conjunctions $\zeta_{\mathcal{A}}$ checked for concrete domain satisfiability is also bounded by $p(|D|)$. Together with the fact that the complexity class PSPACE is oblivious for polynomial blowups of the input, it follows that the completion algorithm can be executed in polynomial space. Along the same lines, it can be shown that the algorithm can be executed in exponential space if $\mathcal{D}$-satisfiability is in EXPSPACE.

Now assume that $\mathcal{D}$-satisfiability is in NEXPTIME. From Lemma 3.6, we know that fcompl terminates after at most $|\mathcal{A}|^4$ rule applications if started on input $\mathcal{A}$. Since, by Point 1 of Lemma 3.7, the size of its input is bounded by $|D|^2$, it terminates after at most $|D|^8$ rule applications. Since the recursion depth is bounded by $|D|$, and, by Point 2 of Lemma 3.7, at most $q(|D|)$ recursion calls are made per recursion step for some polynomial q, sat can be executed in nondeterministic exponential time if the check for concrete domain satisfiability is not taken into account. By the bounds on the recursion depth and the number of recursion calls per recursion steps, the number of concrete domain satisfiability checks performed is at most exponential in $|D|$. Since the size of predicate conjunctions passed in each step is bounded by $p(D)$ and $\mathcal{D}$-satisfiability is in NEXPTIME, we can perform each check in (non-deterministic) time exponential in $|D|$. Summing up, the sat algorithm an be executed in nondeterministic exponential time. ∎

Combining this result with the PSPACE lower bound of $\mathcal{ALC}$-concept satisfiability [39] and using Savitch's Theorem which implies that PSPACE = NPSPACE and EXPSPACE = NEXPSPACE [37], we obtain the following theorem.

Theorem 3.15 Let $\mathcal{D}$ be an admissible concrete domain.

1. If $\mathcal{D}$-satisfiability is in PSPACE, then $\mathcal{ALC}(\mathcal{D})$-concept satisfiability and $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability are PSPACE-complete.
2. If $\mathcal{D}$-satisfiability is in $\mathrm{C} \in \{\text{NEXPTIME}, \text{EXPSPACE}\}$, then $\mathcal{ALC}(\mathcal{D})$-concept satisfiability and $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability are also in C.

Since lower complexity bounds obviously transfer from $\mathcal{D}$-satisfiability to $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability, Point 2 of this theorem yields tight complexity bounds if $\mathcal{D}$-satisfiability is NEXPTIME-complete or EXPSPACE-complete (instead of just *in* the respective class). Moreover, since subsumption can be reduced to (un)satisfiability and vice versa, we obtain corresponding complexity bounds for subsumption:

Corollary 3.16 Let $\mathcal{D}$ be an admissible concrete domain.

1. If $\mathcal{D}$-satisfiability is in PSPACE, then $\mathcal{ALC}(\mathcal{D})$-concept subsumption and $\mathcal{ALCF}(\mathcal{D})$-concept subsumption are PSPACE-complete.
2. If $\mathcal{D}$-satisfiability is in NEXPTIME, then $\mathcal{ALC}(\mathcal{D})$-concept subsumption and $\mathcal{ALCF}(\mathcal{D})$-concept subsumption are in CO-NEXPTIME.
3. If $\mathcal{D}$-satisfiability is in EXPSPACE then $\mathcal{ALC}(\mathcal{D})$-concept subsumption and $\mathcal{ALCF}(\mathcal{D})$-concept subsumption are in EXPSPACE.

4 ABox Consistency

In the preceding section, we used ABoxes merely as a data structure. However, ABoxes are interesting in their own right since they are frequently used to represent assertional knowledge about the state of affairs in a particular "world". In this section, we extend the complexity results obtained in the previous section from concept satisfiability to ABox consistency by devising a *precompletion algorithm* in the style of [13, 21]. Most importantly, the extended algorithm yields a tight PSPACE complexity bound for $\mathcal{ALCF}(\mathcal{D})$-ABox consistency if $\mathcal{D}$-satisfiability is in PSPACE.

4.1 The Algorithm

The algorithm works by reducing ABox consistency to concept satisfiability. First, a set of precompletion rules is exhaustively applied to the input ABox $\mathcal{A}$ yielding a *precompletion* of $\mathcal{A}$. Intuitively, rule application makes all implicit knowledge in the ABox explicit except that it does *not* generate new R-successors for roles $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. Then, several *reduction concepts* are generated from the precompletion and passed to the concept satisfiability algorithm devised in the previous section. The input ABox is satisfiable iff the precompletion contains no obvious contradiction and all reduction concepts are satisfiable.

The precise formulation of the algorithm can be found in Figure 9. We assume all concepts in the input ABox to be in NNF. As already mentioned in Section 3.3, the precompl function is identical to the fcompl function in Figure 5 except that it additionally applies the R$\forall$r rule. This is necessary since, in contrast to ABoxes processed by the sat algorithm, the input ABox to cons may contain assertions of the form $(a, b) : R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. Although not generating new R-successors for roles $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, the precompletion algorithm *does* generate new f-successors and new g-successors for features $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. Intuitively, the input ABox induces a set of clusters of objects as discussed in Section 3.1 and these clusters are constructed by the precompl function.

Note that the construction of a reduction concept corresponds to a single application of the R$\exists$r rule together with exhaustive application of the R$\forall$r rule very similar to recursion calls of the sat functions in Figure 5.

4.2 Correctness and Complexity

Termination of the precompletion algorithm is easily obtained.

Proposition 4.1 The precompletion algorithm terminates on any input.

define procedure $\mathsf{cons}(\mathcal{A})$
  $\mathcal{A} := \mathsf{precompl}(\mathcal{A})$
  if $\mathcal{A}$ contains a clash **then**
    return inconsistent
  forall assertions $\exists R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ **do**
    Fix $b \in \mathsf{O_a}$
    if $\mathsf{sat}(\{b : C \sqcap \bigsqcap_{\forall R.E \in \mathcal{A}(a)} b : E) = \mathsf{unsatisfiable}$ **then**
      return inconsistent
  return consistent

define procedure $\mathsf{precompl}(\mathcal{A})$
  while a rule from $\{\mathsf{R}\sqcap, \mathsf{R}\sqcup, \mathsf{R}\forall\mathsf{r}, \mathsf{R}\forall\mathsf{f}, \mathsf{R}\exists\mathsf{f}, \mathsf{Rc}, \mathsf{R}\downarrow, \mathsf{R}\uparrow, \mathsf{Rfe}\}$
      is applicable to $\mathcal{A}$ **do**
    Choose an applicable rule R s.t. $\mathsf{R} = \mathsf{Rfe}$ if Rfe is applicable
    Apply R to $\mathcal{A}$
  return $\mathcal{A}$

FIG. 9. The $\mathcal{ALCF}(\mathcal{D})$-ABox consistency algorithm.

PROOF. By Lemma 3.6, the precompl function terminates, and, by Proposition 3.8, the sat function also terminates. ∎

We now prove soundness and completeness. In the following, an ABox $\mathcal{A}'$ is called a *precompletion* of an ABox $\mathcal{A}$ iff $\mathcal{A}'$ can be obtained by applying the precompl function to $\mathcal{A}$. Note that precompl is non-deterministic (due to the use of the R⊔ rule) and hence there may exist more than a single precompletion for a given ABox $\mathcal{A}$.

Proposition 4.2 (Soundness) If the precompletion algorithm returns consistent, then the input ABox is consistent.

PROOF. If the algorithm is started on input ABox $\mathcal{A}$ returning consistent, then there exists a precompletion $\mathcal{A}_p$ for $\mathcal{A}$ that does not contain a clash and all reduction concepts $C_1, \ldots, C_n$ of $\mathcal{A}_p$ that are passed as arguments to the sat algorithm are satisfiable. We show that this implies that $\mathcal{A}_p$ has a model, which, by Lemma 3.10 and the definition of precompletion, proves the proposition.

Let $\mathcal{I}_1, \ldots, \mathcal{I}_n$ be the models of the reduction concepts $C_1, \ldots, C_n$ and $a_i : \exists R_i.E_i$ be the assertion in $\mathcal{A}_p$ that triggered the construction of the reduction concept C_i. W.l.o.g., we assume that

- $\Delta_{\mathcal{I}_i} \cap \Delta_{\mathcal{I}_j} = \emptyset$ for $1 \leq i < j \leq n$ and
- $\Delta_{\mathcal{I}_i} \cap \mathsf{O_a} = \emptyset$ for $1 \leq i \leq n$.

For each i with $1 \leq i \leq n$, we fix an element $d_i \in \Delta_{\mathcal{I}_i}$ with $d_i \in C_i^{\mathcal{I}_i}$. Moreover, we fix a solution δ for $\zeta_{\mathcal{A}_p}$, which exists since $\mathcal{A}_p$ is clash-free. Define an interpretation $\mathcal{I}$ as follows:

1. $\Delta_{\mathcal{I}} := \mathsf{O_a} \uplus \Delta_{\mathcal{I}_1} \uplus \cdots \uplus \Delta_{\mathcal{I}_n}$,
2. $A^{\mathcal{I}} := \{a \in \mathsf{O_a} \mid a : A \in \mathcal{A}_p\} \cup \bigcup_{1 \leq i \leq n} A^{\mathcal{I}_i}$ for all $A \in \mathsf{N_C}$,

3. $R^{\mathcal{I}} := \{(a,b) \in \mathsf{O_a} \times \mathsf{O_a} \mid (a,b) : R \in \mathcal{A}\} \cup \{(a_i, d_i) \mid 1 \leq i \leq n \text{ and } R = R_i\}$
   $\cup \bigcup_{1 \leq i \leq n} R^{\mathcal{I}_i}$ for all $R \in \mathsf{N_R}$,
4. $g^{\mathcal{I}} := \{(a, \delta(x)) \in \mathsf{O_a} \times \Delta_{\mathcal{D}} \mid (a,x) : g \in \mathcal{A}\} \cup \bigcup_{1 \leq i \leq n} g^{\mathcal{I}_i}$ for all $g \in \mathsf{N_{cF}}$,
5. $a^{\mathcal{I}} := a$ for all $a \in \mathsf{O_a}$, and
6. $x^{\mathcal{I}} := \delta(x)$ for all $x \in \mathsf{O_c}$.

$\mathcal{I}$ is well-defined: due to the non-applicability of the Rfe rule to $\mathcal{A}_p$, $f^{\mathcal{I}}$ and $g^{\mathcal{I}}$ are functional for all $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. The following claim is an easy consequence of the construction of $\mathcal{I}$:

Claim: Let $1 \leq i \leq n$. For all $d \in \Delta_{\mathcal{I}_i}$ and $C \in \mathsf{sub}(\mathcal{A}_p)$, $d \in C^{\mathcal{I}_i}$ implies $d \in C^{\mathcal{I}}$.

It remains to show that $\mathcal{I}$ is a model of $\mathcal{A}_p$, i.e., that all assertions in $\mathcal{A}_p$ are satisfied by $\mathcal{I}$. For assertions of the form $(a,b) : R$ and $(a,x) : g$, this is an immediate consequence of the definition of $\mathcal{I}$. Assertions $a\neg \sim b$ are satisfied since $\mathcal{A}_p$ is clash-free and assertions $(x_1, \ldots, x_n) : P$ are satisfied since δ is a solution for $\zeta_{\mathcal{A}_p}$. It thus remains to show that $a : C \in \mathcal{A}_p$ implies $a \in C^{\mathcal{I}}$. This is done by induction over the structure of C as in the proof of Lemma 3.11. The only differences are in the following cases of the induction step:

- $a : \exists R.E \in \mathcal{A}_p$. Then there is an i with $1 \leq i \leq n$ such that $a = a_i$, $R = R_i$, and $E = E_i$ appears as a conjunct in the reduction concept C_i. By definition of $\mathcal{I}$, we have $(a, d_i) \in R^{\mathcal{I}}$. By the above claim together with $d_i \in C_i^{\mathcal{I}_i}$, we have $d_i \in C_i^{\mathcal{I}}$. Since E is a conjunct in C_i, this clearly implies $d_i \in E^{\mathcal{I}}$ and thus $a \in (\exists R.E)^{\mathcal{I}}$.
- $a : \forall R.E \in \mathcal{A}_p$. Fix a $b \in \Delta_{\mathcal{I}}$ such that $(a,b) \in R^{\mathcal{I}}$. Then either b is an R-successor of a in $\mathcal{A}_p$ or $a = a_i$, $R = R_i$, and $b = d_i$ for some $1 \leq i \leq n$. The first case was already treated in the proof of Lemma 3.11. Hence, let us stick to the second case. By construction of C_i, E appears as a conjunct in C_i. By the claim, we have $d_i \in C_i^{\mathcal{I}}$ and hence $d_i \in E^{\mathcal{I}}$.

∎

Proposition 4.3 (Completeness) If the precompletion algorithm is started on a consistent input ABox, then it returns $\mathsf{consistent}$.

PROOF. Suppose that the algorithm is started on a consistent ABox $\mathcal{A}$. By Lemma 3.10, the $\mathsf{precompl}$ function can apply the completion rules such that only consistent ABoxes are obtained. Hence, by Lemma 3.6, the $\mathsf{precompl}$ function generates a consistent precompletion $\mathcal{A}_p$ of $\mathcal{A}$. Consistency of $\mathcal{A}_p$ clearly implies that the reduction concepts constructed from $\mathcal{A}_p$ are satisfiable. Since, by Proposition 3.8, the sat function terminates, the precompletion algorithm also terminates and returns $\mathsf{consistent}$. ∎

It remains to analyze the time and space requirements of our algorithm.

Proposition 4.4 1. If $\mathcal{D}$-satisfiability is in PSPACE, then the precompletion algorithm can be executed in polynomial space.
2. If $\mathcal{D}$-satisfiability is in NEXPTIME, then the precompletion algorithm can be executed in nondeterministic exponential time.
3. If $\mathcal{D}$-satisfiability is in EXPSPACE, then the precompletion algorithm can be executed in exponential space.

PROOF. Let $\mathcal{A}$ be the input ABox to the precompletion algorithm. By Lemma 3.6, the precompl function terminates after at most $|\mathcal{A}|^4$ steps generating an ABox $\mathcal{A}'$ of size at most $|\mathcal{A}|^6$. Since all complexity classes mentioned in the proposition are oblivious for polynomial blowups of the input, the concrete domain satisfiability check does not spoil the upper bound on the time/space requirements. Concerning the calls to the sat function, it suffices to refer to Proposition 3.14. ∎

As in the previous section, we use the PSPACE lower bound of $\mathcal{ALC}$-concept satisfiability and the fact that PSPACE =NPSPACE and EXPSPACE =NEXPSPACE to obtain the following theorem.

Theorem 4.5 Let $\mathcal{D}$ be an admissible concrete domain.

1. If $\mathcal{D}$-satisfiability is in PSPACE, then $\mathcal{ALC}(\mathcal{D})$-ABox consistency and $\mathcal{ALCF}(\mathcal{D})$-ABox consistency are PSPACE-complete.
2. If $\mathcal{D}$-satisfiability is in C $\in \{\text{NEXPTIME}, \text{EXPSPACE}\}$, then $\mathcal{ALC}(\mathcal{D})$-ABox consistency and $\mathcal{ALCF}(\mathcal{D})$-ABox consistency are also in C.

5 Applying the Results

We give some example applications of the results just obtained by reconsidering the concrete domains A and S introduced in Section 2. In order to apply Theorems 3.15 and 4.5, we need to determine the complexity of A-satisfiability and S-satisfiability. More precisely, we show that both problems are in NP.

Let us start with the concrete domain A. The proof is by a reduction to mixed integer programming (MIP), i.e., to linear programming where some of the variables must take integer values. More precisely, a *mixed integer programming problem* has the form $Ax = b$, where A is an $m \times n$-matrix of rational numbers, x is an n-vector of variables, each of them being either an integer variable or a rational variable, and b is an m-vector of rational numbers (see, e.g. [40]). A *solution* of $Ax = b$ is a mapping δ that assigns an integer to each integer variable in x and a rational number to each rational variable in x such that the equality $Ax = b$ holds. Deciding the *satisfiability* of a MIP problem means to decide whether such a problem has a solution.

Proposition 5.1 A-satisfiability is in NP.

PROOF. We sketch a non-deterministic polynomial time algorithm for A-satisfiability. The algorithm is based on several normalization steps, simple inconsistency checks, and a final call to an algorithm which is capable of deciding the satisfiability of MIP problems.

Let c be a finite conjunction of A predicates. The following steps are executed sequentially to decide the satisfiability of c:

1. Return unsatisfiable if c contains the $\bot_{\mathsf{A}}$ predicate.
2. Eliminate all occurrences of the $\top_{\mathsf{A}}$ predicate from c and call the result c_1.
3. Eliminate each occurrence of predicates $\overline{\mathsf{int}}$, P_q, and $+$:
   • replace each conjunct $\overline{\mathsf{int}}(x)$ with the conjuncts

$$>(x, f),\ \mathsf{int}(f),\ =_1(o),\ +(f, o, f'),\ <(x, f'),$$

   where f, f', o are fresh (i.e. previously unused) variables.

- replace each conjunct $P_q(x)$ (where $P \in \{<, \leq, \neq, \geq, >\}$ and $q \in \mathbb{Q}$) with the two conjuncts $=_q(f)$ and $P(x, f)$, where f is a fresh variable.
- replace each conjunct $\overline{+}(x, y, z)$ with $+(x, y, f)$ and $\neq(f, z)$, where f is a fresh variable.

Call the result c_2

4. Eliminate each occurrence of the predicates $\leq$, $\neq$, $\geq$, and $>$ in c_2: conjuncts $\leq(x, y)$ are non deterministically replaced with either $<(x, y)$ or $=(x, y)$. The other predicates can be treated similarly. Call the result c_3. Note that c_3 does only contain the predicates int, $=_q$, $<$, $=$, and $+$.
5. Transform c_3 into a MIP problem in the obvious way:
   - every variable x used in c_3 such that $\mathsf{int}(x)$ is a conjunct of c_3 becomes an integer variable in the MIP problem. All other variables appearing in c_3 become rational variables;
   - every conjunct $=_q(x)$ is translated into an equation $x = q$;
   - every conjunct $=(x, y)$ is translated into an equation $x - y = 0$;
   - every conjunct $<(x, y)$ is translated into an equation $x + s - y = 0$, where s is a fresh rational variable (also known as slack variable);
   - every conjunct $+(x, y, z)$ is translated into an equation $x + y - z = 0$.

   Use a standard NP algorithm to decide the satisfiability of this problem and return the result.

It is straightforward to prove the correctness of the sketched algorithm by showing that (i) each of the normalization steps preserves (un)satisfiability, and (ii) the reduction to MIP is correct. Moreover, it is not hard to see that the algorithm can be executed in nondeterministic polynomial time: each of the normalization steps leads to at most a polynomial blowup of the size of the predicate conjunction. Finally, deciding the satisfiability of MIP problems can be done in NP [14]. ∎

An application of Theorems 3.15 and 4.5 immediately yields the complexity of reasoning with the Description Logic $\mathcal{ALCF}(\mathsf{A})$.

Corollary 5.2 $\mathcal{ALCF}(\mathsf{A})$-concept satisfiability and $\mathcal{ALCF}(\mathsf{A})$-ABox consistency are PSPACE-complete.

Now for the concrete domain S. It is straightforward to reduce S-satisfiability to the satisfiability problem of so-called RCC8 networks [10, 36]. Such a network is simply a finite set of assertions $\mathsf{rd}(X, Y)$, where rd is a disjunction $\mathsf{rel}_0 \vee \cdots \vee \mathsf{rel}_k$ of RCC8 relations and X and Y are region variables from some fixed set of variables $\mathcal{V}$. A triple $\langle U, T, \delta \rangle$, where (U, T) is a topology and δ maps each region variable from $\mathcal{V}$ to an element of T, is a *model* of an RCC8 network N iff, for each $\mathsf{rel}_0 \vee \cdots \vee \mathsf{rel}_k(X, Y) \in N$, there exists an $i \leq k$ such that $\delta(X)\ \mathsf{rel}_i\ \delta(Y)$. N is satisfiable iff it has a model.

Proposition 5.3 S-satisfiability is in NP.

PROOF. It is easy to reduce S-satisfiability to the satisfiability of RCC8 networks: given a finite conjunction c of predicates from Φ_{S}, first eliminate any occurrences of the $\top_{\mathsf{s}}$ predicate and return $\mathsf{unsatisfiable}$ if c contains the $\bot_{\mathsf{s}}$ predicate; then replace all predicates $\overline{\mathsf{rel}}$ by the disjunction of all elements of $\mathsf{RCC8} \setminus \{\mathsf{rel}\}$, where RCC8 denotes the set of all eight RCC8 relations; finally, translate each conjunct in c into an

RCC8 assertion $\mathsf{rd}(X, Y)$ in the obvious way. As shown by Renz and Nebel in [36], the satisfiability of the resulting RCC8 network can be decided in nondeterministic polynomial time. Moreover, every satisfiable RCC8 network has a model in the topological space $\mathcal{RC}_{\mathbb{R}^2}$ [35]. ∎

Again, we obtain the desired corollary by applying Theorems 3.15 and 4.5.

Corollary 5.4 $\mathcal{ALCF}(\mathsf{S})$-concept satisfiability and $\mathcal{ALCF}(\mathsf{S})$-ABox consistency are PSPACE-complete.

6 Discussion and Related Work

In this paper, we have established tight complexity bounds for concept- and ABox-reasoning with the basic Description Logic with concrete domains $\mathcal{ALC}(\mathcal{D})$ and its extensions with feature (dis)agreements $\mathcal{ALCF}(\mathcal{D})$. The upper bound for concept satisfiability has been obtained by a completion algorithm that uses the tracing technique while the upper bound for ABox consistency has been established by a precompletion-style reduction to concept satisfiability. We have strictly separated the algorithms for these two reasoning problems since this makes more explicit the additional means necessary for dealing with ABoxes instead of with concepts. However, for the implementation of DL reasoners that can decide ABox consistency, it may be more appropriate to use a "direct" ABox consistency algorithm instead of reducing this reasoning task to concept satisfiability. Considering the two algorithms developed in this paper, it should be straightforward to devise such a direct algorithm.

Using an arithmetic concrete domain A and a spatial concrete domain S, we have demonstrated the relevance of the obtained complexity results: since A-satisfiability and S-satisfiability are in NP, it follows from the established complexity bounds that concept- and ABox-reasoning with both $\mathcal{ALCF}(\mathsf{A})$ and $\mathcal{ALCF}(\mathsf{S})$ is PSPACE-complete. We have also established upper bounds for the case that $\mathcal{D}$-satisfiability is in NEXPTIME or EXPSPACE. A rather expressive concrete domain R based on Tarski algebra (also known as real closed fields), for which R-satisfiability is EXPSPACE-complete, can be found in [30, 5]. Using the results from this paper and the obvious fact that $\mathcal{D}$-satisfiability can be polynomially reduced to $\mathcal{ALC}(\mathcal{D})$-concept satisfiability, we immediately obtain EXPSPACE-completeness of concept- and ABox-reasoning with the Description Logic $\mathcal{ALC}(\mathsf{R})$. Other important concrete domains that are captured by the presented results are the temporal ones that can be found in [33, 30, 27].

The results presented in this paper have stimulated interesting further research. For example, in [3] the PSPACE upper bound for $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability has been used to obtain a PSPACE upper bound for reasoning with the interval-based temporal Description Logic $\mathcal{TL}$-$\mathcal{ALCF}$, which was first described in [2]. Perhaps most interesting, it has been found that the PSPACE upper bounds established in this paper are fragile in the following sense: there exist several standard means of expressivity whose addition to $\mathcal{ALC}(\mathcal{D})$ leads to the complexity of reasoning leaping from PSPACE-completeness to NEXPTIME-completeness—at least for so-called arithmetic concrete domains [28, 30, 1]. Examples for such means of expressivity include acyclic TBoxes, inverse roles, nominals, and role conjunction. This is particularly surprising since (i) the mentioned means of expressivity are usually considered "harmless" w.r.t. the

complexity of reasoning, i.e., for most standard DLs, their addition does *not* change the complexity of reasoning; (ii) many concrete domains suggested in the literature (such as the concrete domain A described in this paper) are arithmetic; and (iii) there exist rather simple arithmetic concrete domains $\mathcal{D}$—in particular some for which $\mathcal{D}$-satisfiability is in PTIME.

References

[1] Carlos Areces and Carsten Lutz. Concrete domains and nominals united. In Carlos Areces, Patrick Blackburn, Maarten Marx, and Ulrike Sattler, editors, *Proceedings of the fourth Workshop on Hybrid Logics (HyLo'02)*, 2002.

[2] Alessandro Artale and Enrico Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research (JAIR)*, 9:463–506, 1998.

[3] Alessandro Artale and Carsten Lutz. A correspondence between temporal description logics. In Patrick Lambrix, Alex Borgida, Maurizio Lenzerini, Ralf Möller, and Peter Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, number 22 in CEUR-WS (http://ceur-ws.org/), pages 145–149, 1999.

[4] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, Australia, 1991.

[5] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. DFKI Research Report RR-91-10, German Research Center for Artificial Intelligence (DFKI), 1991.

[6] Franz Baader and Philipp Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proceedings of the 16th German AI-Conference (GWAI-92)*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143. Springer-Verlag, 1992.

[7] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proceedings of the Workshop on Processing Declarative Knowledge (PDK-91)*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 67–86. Springer-Verlag, 1991.

[8] Franz Baader, Deborah L. McGuiness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2002. To appear.

[9] Franz Baader and Ulrike Sattler. Tableau algorithms for description logics. In R. Dyckhoff, editor, *Proceedings of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 1–18. Springer-Verlag, 2000.

[10] Brandon Bennett. Modal logics for qualitative spatial reasoning. *Journal of the Interest Group in Pure and Applied Logic*, 4(1), 1997.

[11] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with classic: When and how to use a KL-ONE-like language. In John F. Sowa, editor, *Principles of Semantic Networks – Explorations in the Representation of Knowledge*, chapter 14, pages 401–456. Morgan Kaufmann, 1991.

[12] Giuseppe De Giacomo and Maurizio Lenzerini. TBox and ABox reasoning in expressive description logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 316–327. Morgan Kaufmann Publishers, 1996.

[13] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in concept languages: from subsumption to instance checking. *Journal of Logic and Computation*, 4(4):423–452, 1994.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, USA, 1979.

[15] Erich Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.

[16] Volker Haarslev, Carsten Lutz, and Ralf Möller. A description logic with concrete domains and role-forming predicates. *Journal of Logic and Computation*, 9(3):351–384, 1999.

[17] Volker Haarslev and Ralf Möller. RACER system description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 701–705. Springer-Verlag, 2001.

[18] Volker Haarslev, Ralf Möller, and Michael Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning IJCAR'01*, number 2083 in Lecture Notes in Artifical Intelligence, pages 29–44. Springer-Verlag, 2001.

[19] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–380, 1992.

[20] Philipp Hanschke. Specifying role interaction in concept languages. In William Nebel, Bernhard; Rich, Charles; Swartout, editor, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 318–329. Morgan Kaufmann, 1992.

[21] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, 18:133–157, 1996.

[22] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. DFKI Research Report RR-90-04, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany, 1990.

[23] Ian Horrocks and Peter Patel-Schneider. The generation of DAML+OIL. In Carole Goble, Deborah L. McGuinness, Ralf Möller, and Peter F. Patel-Schneider, editors, *Proceedings of the International Workshop in Description Logics 2001 (DL2001)*, number 49 in CEUR-WS (http://ceur-ws.org/), pages 30–35, 2001.

[24] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 199–204. Morgan-Kaufmann, 2001.

[25] Martina Kullmann, François de Bertrand de Beuvron, and François Rousselot. A description logic model for reacting in a dynamic environment. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, number 33 in CEUR-WS (http://ceur-ws.org/), pages 203–212, 2000.

[26] Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977.

[27] Carsten Lutz. Interval-based temporal reasoning with general TBoxes. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 89–94. Morgan-Kaufmann, 2001.

[28] Carsten Lutz. NExpTime-complete description logics with concrete domains. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 45–60. Springer-Verlag, 2001.

[29] Carsten Lutz. Adding numbers to the $\mathcal{SHIQ}$ description logic—First results. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*. Morgan Kaufman, 2002.

[30] Carsten Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2002.

[31] Carsten Lutz. Reasoning about entity relationship diagrams with complex attribute dependencies. In Ian Horrocks and Sergio Tessaris, editors, *Proceedings of the International Workshop in Description Logics 2002 (DL2002)*, number 53 in CEUR-WS (http://ceur-ws.org/), pages 185–194, 2002.

[32] Carsten Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logics (AiML) 2002*, To appear.

[33] Carsten Lutz, Volker Haarslev, and Ralf Möller. A concept language with role-forming predicate restrictions. Technical Report FBI-HH-M-276/97, University of Hamburg, Computer Science Department, Hamburg, 1997.

[34] David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 165–176. Morgan Kaufman, 1992.

[35] Jochen Renz. A canonical model of the region connection calculus. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 330–341. Morgan Kaufmann, San Francisco, California, 1998.

[36] Jochen Renz and Bernhard Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, 108(1–2):69–123, 1999.

[37] Walter J. Savitch. Relationsship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[38] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278, 1993.

[39] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[40] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, UK, 1986.

[41] Sergio Tessaris, Ian Horrocks, and Graham Gough. Evaluating a modular abox algorithm. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pages 227–239. Morgan Kaufman, 2002.

[42] Moshe Y. Vardi. Why is modal logic so robustly decidable? In Neil Immerman and Phokion G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.

Received 4 June 2002. Revised 27 September 2002

Acknowledgements

The Editor-in-Chief would like to thank the following colleagues who have helped maintain the standards set for a scientific journal, through their refereeing of the papers that have been submitted.[1]

Francesco Donini
Chris Hankin
David Makinson
Amr Sabry
Sergio Tessaris

[1] The list includes the referees for the papers in this issue, plus the referees of papers rejected meanwhile.

Interest Group in Pure and Applied Logics (IGPL)

The Interest Group in Pure and Applied Logics (IGPL) is sponsored by The European Association for Logic, Language and Information (FoLLI), and currently has a membership of over a thousand researchers in various aspects of logic (symbolic, mathematical, computational, philosophical, etc.) from all over the world (currently, more than 50 countries). Our main activity is that of a research and information clearing house.

Our activities include:

- Exchanging information about research problems, references and common interest among group members, and among different communities in pure and applied logic.
- Helping to obtain photocopies of papers to colleagues (under the appropriate copyright restrictions), especially where there may be difficulties of access.
- Supplying review copies of books through the journals on which some of us are editors.
- Helping to organise exchange visits and workshops among members.
- Advising on papers for publication.
- Editing and distributing a Newsletter and a Journal (the first scientific journal on logic which is FULLY electronic: submission, refereeing, revising, typesetting, publishing, distribution; first issue: July 1993): the Logic Journal of the Interest Group on Pure and Applied Logics. (For more information on the Logic Journal of the IGPL, see the Web homepage: http://www.oup.co.uk/igpl)
- Keeping a public archive of papers, abstracts, etc., accessible via ftp.
- Wherever possible, obtaining reductions on group (6 or more) purchases of logic books from publishers.

If you are interested, please send your details (name, postal address, phone, fax, e-mail address, research interests) to:

IGPL Headquarters
c/o Prof. Dov Gabbay
King's College, Dept of Computer Science
Strand
London WC2R 2LS
United Kingdom
e-mail: dg@dcs.kcl.ac.uk

For the organisation, Dov Gabbay, Ruy de Queiroz and Hans Jürgen Ohlbach