



KATHOLIEKE UNIVERSITEIT
LEUVEN

FACULTEIT WETENSCHAPPEN



Deductive Reasoning in Guarded FO(ID)

door

Sander BECKERS

Promotor: prof. dr. M. Denecker

Proefschrift ingediend tot het
behalen van de graad van
Master in de Wiskunde

Academiejaar 2010-2011

Nous ne pouvons nous élever que par l'induction mathématique,
qui seule peut nous apprendre quelque chose de nouveau.

-Poincaré, La Science et l'Hypothèse.

© Copyright by K.U.Leuven

Without written permission of the promotor and the authors it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to K.U.Leuven, Faculteit Wetenschappen, Geel Huis, Kasteelpark Arenberg 11, 3001 Leuven (Heverlee), Telephone +32 16 32 14 01.

A written permission of the promotor is also required to use the methods, products, schematics and programs described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

Contents

1	Introduction	1
2	FO(ID)	2
2.1	Inductive definitions	2
2.2	Semantics	3
2.2.1	Informal Semantics	3
2.2.2	Syntax	4
2.2.3	Preliminaries: three-valued logic	4
2.2.4	Formal Semantics	5
2.3	Open Answer Set Programming	6
2.4	Comparison of Semantics	8
2.5	Tree model property and Guarding	12
3	The Decidability of Guarded FO(ID)	15
3.1	Introduction	15
3.2	Guarded FO(ID)	15
3.3	Tableaux	16
3.4	The Automaton \mathcal{A}_T	18
3.5	Guiding the Automata	19
3.6	Accepted Tableaux correspond to models and vice versa	20
4	A Reasoning Procedure for Simple Guarded FO(ID)	21
4.1	Introduction	21
4.2	Simple Guarded FO(ID)	22
4.3	Completion Structures	23
4.3.1	Trees	23
4.3.2	Expansion Rules	26
4.3.3	Simple Applicability Rules	28
4.4	Completeness	30
5	A Reasoning Procedure for Guarded FO(ID)	34
5.1	Completion Structures	34
5.1.1	Applicability Rules	34
5.1.2	An example of the reasoning procedure at work.	39

6	Satisfiability and Completion Structures	42
6.1	Introduction	42
6.2	Termination	42
6.3	Soundness	43
6.4	Completeness	47
7	Examples	54
7.1	Why one blocked node isn't enough	54
7.2	Proof of Even or Uneven	61
8	Conclusion	65
9	Appendix: Nederlandstalige Samenvatting	66

1 Introduction

The language of First Order Logic with Inductive Definitions (FO(ID)) was developed in [5] as an extension of First Order Logic (FO) in order to express inductively defined concepts. First order logic is already undecidable by itself, and adding inductive definitions makes automated deductive reasoning even more intractable. In the field of Knowledge Reasoning and Representation, as indeed in the discipline of logic on a whole, one is always looking for a balanced trade-off between expressivity and decidability. Therefore the guarded fragment of FO(ID) was introduced [20] - in analogy with the guarded fragment of FO [1] - in order to preserve the characteristics of FO(ID) and at the same time remain within reach of decidability. Although the decidability of guarded FO(ID) has been shown, no effective reasoning procedures have been developed yet. The goal of the present work is to amend this situation by creating such procedures, allowing one to verify the satisfiability of a guarded FO(ID) theory automatically. To achieve this we shall expand on existing techniques to be found in the related field of Open Answer Set Programming (OASP), an extension of the declarative programming language Answer Set Programming.

The structure of this work is as follows. In the next chapter we introduce FO(ID) and its semantics. The latter shall be compared to the semantics of OASP, to provide a connection between the two paradigms. Some comments are added on the notion of guardedness, and its relation to tree-shaped models. The third chapter presents the necessary theoretical results for guarded FO(ID). In chapter four a reasoning procedure for a restricted version of guarded FO(ID) is presented, that serves as a stepping stone to the full version and introduces the concepts to be used. The main result is then expounded in chapter five, containing a deductive reasoning procedure for guarded FO(ID). The correctness of our approach is proven in the following chapter, where we need to remark that we failed to construct a full proof of completeness. The use of the procedure is illustrated by giving two examples in chapter seven.

2 FO(ID)

2.1 Inductive definitions

The following is a familiar example of an inductive definition:

Definition 1. Let I be an interpretation for the set of propositions Σ . For a propositional formula φ in Σ , we define the relation I satisfies φ , denoted by $I \models \varphi$, by the following induction over the subformula order:

- For an atom $p \in \Sigma$, $I \models p$ iff $p \in I$;
- $I \models \psi \wedge \varphi$ if $I \models \psi$ and $I \models \varphi$;
- $I \models \psi \vee \varphi$ if $I \models \psi$ or $I \models \varphi$;
- $I \models \neg\varphi$ if $I \not\models \varphi$.

Induction is a concept which is of fundamental importance to mathematics. The idea of a proof by induction belongs to the standard techniques employed by mathematicians, and numerous mathematical concepts are defined inductively. Therefore it is a procedure with which all of us are very familiar. However, when this notion occurs in mathematical textbooks, it often does so in an informal fashion, as in the case of Definition 1. This is possible exactly because it is such a basic concept, of which everyone has an implicit grasp. Such a luxury is not present in the domain of logic, since there we are forced to state explicitly which rules are available for proving theorems, and which axioms or conjectures we have to start working with. All logically derived consequences are to be found solely using these two elements, there is no room for any informal argument which is clear to all but not of this required form.

So if we want to formalize a certain mathematical discipline, we need a way of expressing all its possibilities - regarding both proof procedures and definitions - in a logical language, i.e. we need to create a syntax and a semantics which captures them. In the 19th century Frege already set out to express the whole of arithmetics in predicate logic. By now we know this is impossible, exactly because of the inductive element that lies at the core of the concept of a number: something is a number if and only if it lies within the transitive closure of the successor relation, starting at 0.

Or, stated otherwise, it is impossible because it is impossible to express the Domain Closure Axiom in first order logic. FO(ID) is a language which

was created to overcome this limitation. It adds the possibility of expressing inductive definitions to FO. The former, for example, could be expressed by the following FO(ID) theory:

$$\begin{cases} \forall x : N(0) \leftarrow \\ \forall x : N(x) \leftarrow \exists y : Suc(x, y) \wedge N(y) \end{cases}$$

$$\forall x : N(x)$$

As with many inductive definitions, it is made up of a base case, and an inductive case. The first order formula states that the domain consists exactly of those elements that fall under the defined predicate.

Similarly, the satisfaction relation as defined in Definition 1, could be expressed in FO(ID) as:¹

$$\begin{cases} \forall i, p : Sat(i, p) \leftarrow p \in I \\ \forall i, f_1, f_2 : Sat(i, or(f_1, f_2)) \leftarrow Sat(i, f_1) \vee Sat(i, f_2) \\ \forall i, f : Sat(i, not(f)) \leftarrow \neg Sat(i, f) \end{cases}$$

Here, we define the relation *Sat* in terms of two formula building functors *or* and *not*, and a predicate '*∈*' representing the element relation of set theory. This FO(ID) definition is a correct logical representation of the natural language version in Definition 1. It has been shown that FO(ID) offers a uniform representation for all the usual kinds of inductive definitions found in mathematical texts: monotone definitions, definitions over a well-founded order and iterated definitions.

2.2 Semantics²

2.2.1 Informal Semantics

An FO(ID) theory consists of, on the one hand, a set of inductive definitions and, on the other hand, a set of regular first order logic formulas. Each definition is made up of rules, so that rules themselves are not FO(ID) formulas, but definitions, i.e., sets of rules, are. An FO(ID) theory contains two types of predicates, *defined* predicates and *open* predicates. The open predicates function

¹This example comes from [19].

²The definitions in the following section come from [19]

as predicates normally do in first order logic, namely any structure interpreting them that is consistent with the formulas in the theory is a possible partial model of the theory. To make the structure an actual model, it should interpret the defined predicates (given the interpretation of the open predicates) as the definitions dictate.

This means that the \leftarrow in the definitions is not to be confused with the normal implication, or equivalence, since these clearly aren't suited to express all forms of inductive definitions. Instead, the rules are interpreted according to the well-founded model semantics for logic programs (parametrized on the interpretation of the open predicate). This ensures that - also in the presence of negation - their meaning is indeed what one expects from an inductive definition.

2.2.2 Syntax

Syntactically, a definition in FO(ID) is a set of definitional rules, which are of the form:

$$\forall \mathbf{x} : P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$$

Here, $\varphi(\mathbf{x})$ is an FO formula whose free variables are x , and ' \leftarrow ' is a new symbol, the definitional implication. We refer to $P(\mathbf{x})$ as the *head* of the rule r , denoted $head(r)$, and to the formula φ as its *body*, denoted $body(r)$. An FO(ID) formula is any expression that can be formed by combining atoms and definitions, using the standard FO connectives and quantifiers. The meaning of the FO formulas and boolean connectives is standard; we therefore only need to define the semantics of a definition in order to define that of FO(ID).

2.2.3 Preliminaries: three-valued logic

Let us first introduce some semantical concepts. In this work, we shall only consider relational vocabularies. An *interpretation* \mathcal{S} for a vocabulary Σ consists of a non-empty domain D and a mapping from each predicate symbol P/n to a relation $R \subseteq D^n$. A three-valued interpretation ν is the same as a two-valued one, except that it maps each predicate symbol P/n to a function P^ν from D^n to the set of truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. Such a ν assigns a truth value to each logical atom $P(\mathbf{c})$, namely $P^\nu(c_1^\nu, \dots, c_n^\nu)$. This assignment can be extended to an assignment $\nu(\varphi)$ of a truth value to each formula φ , using the standard Kleene truth tables for the logical connectives:

ψ, φ	t, t	t, f	t, u	u, u	u, f	f, f
$\psi \vee \varphi$	t	t	t	u	u	f

ψ, φ	t, t	t, f	t, u	u, u	u, f	f, f
$\psi \wedge \varphi$	t	f	u	u	f	f

φ	t	u	f
$\neg\varphi$	f	u	t

We can order the truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ by a *truth order* \leq_t as follows: $t >_t u >_t f$. They can also be ordered by a *precision order* \leq_p : $\mathbf{t} >_p \mathbf{u}$ and $\mathbf{f} >_p \mathbf{u}$. This order induces also a precision order \leq_p on interpretations: $\nu \leq_p \nu'$ if for each predicate P/n and tuple $\mathbf{d} \in D^n$, $P^\nu(\mathbf{d}) \leq_p P^{\nu'}(\mathbf{d})$. Likewise for the truth order \leq_t .

For a predicate P/n and a tuple $\mathbf{d} \in D^n$ of domain elements, we denote by $\nu[P(\mathbf{d})/\mathbf{v}]$ the three-valued interpretation ν' that coincides with ν on all symbols apart from P/n , and for which $P^{\nu'}$ maps \mathbf{d} to \mathbf{v} and all other tuples \mathbf{d} to $P^\nu(\mathbf{d})$. We also extend this notation to sets $\{(P_1(d_1), \dots, P_n(d_n))\}$ of such pairs.

2.2.4 Formal Semantics

Our goal is to define when a (two-valued) interpretation \mathcal{S} is a model of a definition Δ . We call the predicates that appear in the head of a rule of Δ its *defined* predicates and we denote the set of all these by $Def(\Delta)$; all other symbols are called *open* and the set of open symbols is written $Op(\Delta)$. The purpose of Δ is now to define the predicates $Def(\Delta)$ in terms of the symbols $Op(\Delta)$, i.e., we should assume the interpretation of $Op(\Delta)$ as given and try to construct a corresponding interpretation for $Def(\Delta)$. Let O be the restriction $\mathcal{S}|_{Op(\Delta)}$ of \mathcal{S} to the open symbols. We are now going to construct a sequence of three-valued $(\nu_\alpha^O)_{0 \leq \alpha \leq \beta}$ interpretations, for some ordinal β , each of which extends O ; we will use the limit of such a sequence to interpret $Def(\Delta)$.

- ν_0^O assigns $O(P(\mathbf{d})) \in \{\mathbf{t}, \mathbf{f}\}$ to $P(\mathbf{d})$ if $P \in Op(\Delta)$ and \mathbf{u} if $P \in Def(\Delta)$;
- ν_{i+1}^O is related to ν_i^O in one of two ways:

- Either $\nu_{i+1}^O = \nu_i^O[P(\mathbf{d})/\mathbf{t}]$, such that Δ contains a rule $\forall \mathbf{x}P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$ with $\nu_i(\varphi[\mathbf{d}]) = \mathbf{t}$
- Or $\nu_{i+1}^O = \nu_i^O[U/\mathbf{f}]$, where U is any *unfounded set*, meaning that it consists of pairs of predicates P/n and tuple $\mathbf{d} \in D^n$ for which $P^{\nu_i^O}(\mathbf{d}) = \mathbf{u}$, and for each rule $\forall \mathbf{x}P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$, we have that $\nu_{i+1}(\varphi[\mathbf{d}]) = \mathbf{f}$.

- For each limit ordinal λ , ν_λ^O is the least upper bound w.r.t. \leq_p of all ν_δ^O for which $\delta < \lambda$.

We call such a sequence a *well-founded induction* of Δ in O . Each such sequence eventually reaches a limit ν_β^O . It has been shown that all sequences reach the same limit. It is now this ν_β^O that tells us how to interpret the defined predicates. To be more precise, we define that: $\mathcal{S} \models \Delta$ iff $\mathcal{S}|_{Def(\Delta)} = \nu_\beta^O$, with $O = \mathcal{S}|_{Op(\Delta)}$.

If there is some predicate P/n for which some tuple $\mathbf{d} \in D^n$ of domain elements is still assigned \mathbf{u} by ν_β^O , the definition has no models extending O . Intuitively, this means that, for this particular interpretation of its open symbols, Δ does not manage to unambiguously define the predicates $Def(\Delta)$. If no domain atom is assigned \mathbf{u} by ν_β^O , we say the definition is *total*. This corresponds to our normal understanding of a definition: there should be an unambiguous answer to the question whether some object falls under it or not, there should be no room for choices.

2.3 Open Answer Set Programming

The method we will develop in this work is inspired by the work done by Heymans et al. [6, 10, 11, 12, 13, 14] in the related field of Open Answer Set Programming, OASP for short. OASP arose as a generalization of Answer Set Programming, by introducing an open domain assumption. Lifschitz, one of the founders of Answer Set Programming, describes it as follows:

Answer Set Programming (ASP) is a form of declarative programming oriented towards difficult search problems. It has been applied, for instance, to plan generation and product configuration problems in artificial intelligence and in historical linguistics. [16]

Similarly to FO(ID), the origins of ASP lie in the field of logic programming. Syntactically, ASP programs look like Prolog programs, but the computational

mechanisms used in ASP are different: they are based on the ideas that have led to the development of fast satisfiability solvers for propositional logic. It has emerged from the interaction between two lines of research; the one on the semantics of negation in logic programming and the other on applications of satisfiability solvers to search problems.

An Answer Set Program consists of a finite set of rules of the form $F \leftarrow G$, where F is a disjunction of literals and G is a conjunction of literals. If the body of a rule is true, the head has to be true as well. Moreover, a positive atom p can only become true if it figures in the head of a rule with a true body, where the truth of the body doesn't depend on p . A negative atom, on the other hand, can be chosen to be true if this doesn't conflict with any of the rules.

We will give two simple examples to illustrate the semantics of ASP in an informal manner. Take the following program

$$p \leftarrow$$

$$r \leftarrow p, q$$

There are three sets of atoms that don't contradict these rules, namely

$$\{p\}, \{p, r\}, \{p, q, r\}$$

but only the set $\{p\}$ contains just the atoms which *have to be true* according to the rules, rather than simply agreeing with them. This set is called the answer set, it is the smallest set that doesn't conflict with the program.

The next example introduces negation into the body of the rules. One effect is that in this case there may be several answer sets.

$$p \leftarrow \text{not } q$$

$$q \leftarrow \text{not } p$$

There are four relevant sets of atoms for this program,

$$\emptyset, \{p\}, \{q\}, \{p, q\}$$

The first set is in conflict with the rules, for the absence of p and q implies their presence. The fourth set contains atoms for which there is no rule with a

true body and the respective atoms in the head, so it isn't an answer set either. The second and third sets are symmetrical, so what holds for the one will hold for the other. We have a look at the set containing only p . In this case, *not* q is true, therefore p has to be true as well. q only figures in one rule, of which the body is false, so it has to be false as well. Therefore $\{p\}$ is an answer set, and so is $\{q\}$.³ On page 39 this example is analyzed in terms of FO(ID).

Logic programs under the answer set programming paradigm are decidable, and answer set solvers exist, but ASP has a limited applicability due to its restriction to finite domains. OASP was developed to remedy this defect, but the expressivity gained by allowing infinite universes is countered by the loss of decidability. (For example, the undecidable domino problem can be reduced to it [11].) In order to regain decidability, the fragment of (Extended) Conceptual Logic Programs has been created [13]. The syntax of the programs is restricted, such that satisfiability can be decided. Besides theoretical decidability results for this fragment, effective reasoning procedures have been developed as well [6, 14]. Some of the ideas that underlie these procedures will prove equally fruitful within the domain of FO(ID).

2.4 Comparison of Semantics

The semantics of both FO(ID) and OASP can be seen as possible interpretations of Prolog programs: the well-founded semantics in the case of FO(ID), and the stable model semantics in the case of OASP. In FO(ID), the purpose of a definition Δ is to define the predicates $Def(\Delta)$ in terms of the open predicates $Op(\Delta)$, i.e. we should assume the interpretation of $Op(\Delta)$ as given and try to construct a corresponding interpretation for $Def(\Delta)$. For OASP there's a very similar interpretation. As in FO(ID), there are two categories of predicates: free and non-free predicates. A predicate p that occurs in a program P is free if it occurs in a free rule $p(\mathbf{x}) \vee notp(\mathbf{x}) \leftarrow$, and it is non-free if it doesn't. (A non-free predicate has to occur in the head of at least one rule, otherwise it's superfluous.) Free predicates correspond to open predicates, they can be chosen randomly; non-free predicates are determined by the rules in which they figure in the head.

Let O be $\mathcal{S}|_{Op(\Delta)}$ as before, and assume we have a sequence of interpre-

³Remark that such a situation can never occur for a total definition in FO(ID): if there are no open predicates then a definition will be unique, no choices will be possible. This corresponds to our informal expectations regarding a definition.

tations $(\nu_\alpha^O)_{0 \leq \alpha \leq \beta}$. The difference in semantics is determined by the different rules for determining ν_{i+1} out of ν_i .⁴ We consider three sets of rules for constructing a new interpretation out of a previous one, corresponding to three semantics for a “definition”. These sets are formed with the following rules:

1. **Basic:** $\nu_{i+1}^O = \nu_i^O[P(\mathbf{d})/\mathbf{t}]$, such that Δ contains a rule $\forall \mathbf{x}P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$ with $\nu_i^O(\varphi[\mathbf{d}]) = \mathbf{t}$.
2. **FO:** $\nu_{i+1}^O = \nu_i^O[U_1/\mathbf{f}, U_2/\mathbf{t}]$, where U_1 and U_2 are any (disjunct) unfounded sets, and for each rule $\forall \mathbf{x}P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$, $\nu_{i+1}^O(\varphi[\mathbf{d}]) = \mathbf{f}$, where $P(\mathbf{d}) \in U_1$, and such that Δ contains a rule $\forall \mathbf{x}Q(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$ with $\nu_{i+1}^O(\varphi[\mathbf{e}]) = \mathbf{t}$, where $Q(\mathbf{e}) \in U_2$.
3. **ID:** $\nu_{i+1}^O = \nu_i^O[U/\mathbf{f}]$, where U is any unfounded set, and for each rule $\forall \mathbf{x}P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$, $\nu_{i+1}^O(\varphi[\mathbf{d}]) = \mathbf{f}$, where $P(\mathbf{d}) \in U$.
4. **OASP:** $\nu_{i+1}^O = \nu_i^O[U_1/\mathbf{f}, U_2/\mathbf{t}]$, where U_1 and U_2 are any (disjunct) unfounded sets, and for each rule $\forall \mathbf{x}P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$, $\nu_{i+1}^O(\varphi[\mathbf{d}]) = \mathbf{f}$, where $P(\mathbf{d}) \in U_1$, and such that Δ contains a rule $\forall \mathbf{x}Q(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$ with $\nu_i^O[U_1/\mathbf{f}](\varphi[\mathbf{e}]) = \mathbf{t}$, where $Q(\mathbf{e}) \in U_2$.

The semantics for FOL, called the completion semantics, is such that it consists of rule 2 (which of course implies all others), which means that we’re interpreting Δ as a set of equivalences. The semantics for FO(ID) consists of rules 1 and 3, which means that we’re interpreting Δ as an inductive definition. Finally, the semantics for the fragment of OASP without negations in the head of non-free rules is given by rule 4 (which implies 1 and 3), which means that we’re interpreting Δ as a set of rules in a logic program under the stable model semantics.

To understand the difference between FO(ID) and this fragment of OASP, the following definition is helpful.

Definition 2. An FO(ID) theory T containing only one definition Δ , or T interpreted as an OASP program P^5 , is *stratified*, if there is an assignment S of numbers to the defined predicates in Δ such that the following holds:⁶

1. If Q positively occurs in $\varphi_P(\mathbf{x})$, then $S(P) \geq S(Q)$.

⁴We only consider programs of OASP such that there are no negations in the head of non-free rules. The fragment of Extended Conceptual Logic Programs falls within this category.

⁵i.e. Δ is interpreted as a set of OASP rules and the open predicates are represented by free rules.

⁶We assume that Δ is written in negation normal form.

2. If Q negatively occurs in $\varphi_P(\mathbf{x})$, then $S(P) > S(Q)$.

For a stratified theory, the semantics for OASP and FO(ID) with regard to Δ coincide, that is, every application of rule 4 can be broken down into several applications of rule 3 and an application of rule 1. We can explain this as follows.

The idea behind rule 1 is that a body has to be true first in order to justify a head, so that no defined predicate can ever be its own justification.⁷ Rule 3 serves to justify negated atoms. It contains an extra degree of freedom compared to rule 1, in that a set of falsehoods may be used to justify itself. This corresponds to the minimality of a definition: if it is consistent to assume that something doesn't fall under it, it doesn't. Rule 4, on the other hand, adds another degree of freedom: if a set of falsehoods justifies a set of truths, and in turn this set of truths together with the set of falsehoods justifies the set of falsehoods, both sets are justified. It is precisely at this point that an element of indeterminism creeps in - which is not to be found in case of a definition - since there may be different choices of the sets U_1 and U_2 that satisfy this condition. We will now prove our claim.

Theorem 3. *For a stratified FO(ID) theory T , any sequence of interpretations $(\nu_\alpha^O)_{0 \leq \alpha \leq \beta}$ that can be constructed from a given ν_0^O using rule 4, can be derived as well using only rules 1 and 3.*

Proof. Let's say we have a stratified theory T , and an application of rule 4 that cannot be split up in the said manner. So there are unfounded sets U_1 and U_2 at level ν_i^O , where we would like to obtain U_1/\mathbf{f} and U_2/\mathbf{t} at some level in the sequence, and this can only be achieved using rule 4.

This implies that there's an atom $P(\mathbf{d}) \in U_1$ and a rule $\forall \mathbf{x} P(\mathbf{x}) \leftarrow \varphi_P(\mathbf{x})$, with $\nu_i^O[U_1/\mathbf{f}](\varphi_P[\mathbf{d}]) = \mathbf{u}$, but $\nu_i^O[U_1/\mathbf{f}, U_2/\mathbf{t}](\varphi_P[\mathbf{d}]) = \mathbf{f}$. This means there is an atom $Q(\mathbf{e}) \in U_2$ that occurs negatively in a rule $\forall \mathbf{x} P(\mathbf{x}) \leftarrow \varphi_P(\mathbf{x})$, so $S(P) > S(Q)$.

If we can justify $Q(\mathbf{e})$ first by applying rule 1, and afterwards apply rule 3, the problem would be solved. So let's assume we cannot do this. Since rule 4 did work, there has to be a rule $\forall \mathbf{x} Q(\mathbf{x}) \leftarrow \varphi_Q(\mathbf{x})$ with $\nu_i^O[U_1/\mathbf{f}](\varphi_Q[\mathbf{e}]) = \mathbf{t}$. Now we consider two possibilities.

In the first case, assume $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}](\varphi_Q[\mathbf{e}]) = \mathbf{u}$. So adding $\neg P(\mathbf{d})$

⁷With justification we mean: causing it to be true, where of course the tautology $A \Rightarrow A$ isn't seen as a cause for truth.

changes $\varphi_Q[e]$ from \mathbf{u} to \mathbf{t} . This means that $P(\mathbf{d})$ occurs negatively in $\varphi_Q[e]$, and thus $S(Q) > S(P)$, which contradicts $S(P) > S(Q)$.

This leaves us with the second case, where $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}](\varphi_Q[e]) = \mathbf{t}$. We then take U_3 to be the smallest subset of U_1 such that $\nu_i^O[U_3/\mathbf{f}](\varphi_Q[e]) = \mathbf{t}$. So $X(\mathbf{x})$ occurs negatively in $\varphi_Q[e]$ for every $X(\mathbf{x}) \in U_3$, implying $S(Q) > S(X)$. Now we can try to apply rule 3 to obtain $\nu_{i+1}^O = \nu_i^O[U_3/\mathbf{f}]$. If this works, we could apply rule 1 to justify $Q(e)$. So assume it doesn't. This means there is another atom⁸ $R(\mathbf{g}) \in U_3$ such that there's a rule $\forall \mathbf{x} R(\mathbf{x}) \leftarrow \varphi_R(\mathbf{x})$, with $\nu_i^O[U_3/\mathbf{f}](\varphi_R[\mathbf{g}]) = \mathbf{u}$, and $\nu_i^O[U_1/\mathbf{f}, U_2/\mathbf{t}](\varphi_R[\mathbf{g}]) = \mathbf{f}$.

Again there are two cases to be distinguished.

First assume $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}, U_2/\mathbf{t}](\varphi_R[\mathbf{g}]) = \mathbf{u}$. This implies that $P(\mathbf{d})$ occurs positively in $\varphi_R[\mathbf{g}]$, and thus $S(R) \geq S(P)$. This leads to the contradiction $S(P) > S(Q) > S(R) \geq S(P)$.

So we may assume $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}, U_2/\mathbf{t}](\varphi_R[\mathbf{g}]) = \mathbf{f}$. Yet again we consider two cases.

1. $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}](\varphi_R[\mathbf{g}]) = \mathbf{u}$. We are then in a situation that is entirely similar to our starting point, except that $U_1 \setminus \{P(\mathbf{d})\}$ contains one element less than U_1 . We can continue this line of reasoning, until we end up with $U_n = \emptyset$. In that case rule 4 is equivalent to several applications of rule 1, so our hypothesis is falsified.
2. $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}](\varphi_R[\mathbf{g}]) = \mathbf{f}$. Take U_4 to be the smallest subset of $U_1 \setminus \{P(\mathbf{d})\}$ such that $\nu_i^O[U_4/\mathbf{f}](\varphi_R[\mathbf{g}]) = \mathbf{f}$. So $X(\mathbf{x})$ occurs positively in $\varphi_R[\mathbf{g}]$ for every $X(\mathbf{x}) \in U_4 \setminus \{R(\mathbf{g})\}$, implying $S(R) \geq S(X)$. We can then try to apply rule 3 to obtain $\nu_{i+1}^O = \nu_i^O[U_4/\mathbf{f}]$, and then this would allow us to obtain $\nu_{i+2}^O = \nu_{i+1}^O[U_3/\mathbf{f}]$. So assume rule 3 doesn't work. This means that there's a $T(\mathbf{i}) \in U_4 \setminus \{R(\mathbf{g})\}$ and a rule $\forall \mathbf{x} T(\mathbf{x}) \leftarrow \varphi_T(\mathbf{x})$, with $\nu_i^O[U_4/\mathbf{f}](\varphi_T[\mathbf{i}]) = \mathbf{u}$, and $\nu_i^O[U_1/\mathbf{f}, U_2/\mathbf{t}](\varphi_T[\mathbf{i}]) = \mathbf{f}$. Assume $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}, U_2/\mathbf{t}](\varphi_T[\mathbf{i}]) = \mathbf{u}$. As in the case of $R(\mathbf{g})$, $S(T) \geq S(P)$. This then leads to the contradiction $S(P) > S(Q) > S(R) \geq S(T) \geq S(P)$. So $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}, U_2/\mathbf{t}](\varphi_T[\mathbf{i}]) = \mathbf{f}$. There are two possibilities.

- (a) $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}](\varphi_T[\mathbf{i}]) = \mathbf{u}$. We are then in a situation that is entirely similar to our starting point, except that $U_1 \setminus \{P(\mathbf{d})\}$ contains

⁸For simplicity we assume there is only one such atom, the argument can be repeated in case there are several.

one element less than U_1 . We can continue this line of reasoning, until we end up with $U_n = \emptyset$. In that case rule 4 is equivalent to several applications of rule 1, so our hypothesis is falsified.

- (b) $\nu_i^O[(U_1 \setminus \{P(\mathbf{d})\})/\mathbf{f}](\varphi_T[\mathbf{i}]) = \mathbf{f}$. This situation is the same as for $R(\mathbf{g})$, except that $\max(|U_4|) = |U_1 \setminus \{P(\mathbf{d})\}| < \max(|U_3|) = |U_1|$. We can continue this line of reasoning, until we end up with $\max(|U_n|) = 0$, in which case rule 3 can surely be applied to give us $\nu_{i+1}^O = \nu_i^O[U_n/\mathbf{f}]$. This would then allow a sequence of applications of rule 3 to ν_{i+1}^O to obtain $\nu_{i+n-2}^O = \nu_{i+n-3}^O[U_3/\mathbf{f}]$.

□

One can easily use this result to construct a translation from Extended Conceptual Logic Programs - the most general fragment of OASP for which a reasoning procedure has been developed [14] - to the fragment of guarded FO(ID) in which all definitions are stratified.⁹ This implies that the reasoning procedure we will develop for guarded FO(ID) can handle a strictly more expressive language than the procedures developed so far by Heymans et al.

2.5 Tree model property and Guarding

Many decidability results depend significantly on something called the tree model property. There is, for example, the guarded fragment of first order logic, developed by Andr eka, van Benthem and N emeti [1]. The idea behind this fragment, which is decidable, is that all quantification is guarded by atomic formulas. Quantified formulas need to have the form:

$$\exists \mathbf{y}(G(\mathbf{x}, \mathbf{y}) \wedge \psi(\mathbf{x}, \mathbf{y}))$$

or

$$\forall \mathbf{y}(G(\mathbf{x}, \mathbf{y}) \rightarrow \psi(\mathbf{x}, \mathbf{y}))$$

Thus quantifiers may range over a tuple \mathbf{y} of variables, but are 'guarded' by an atom G that contains all the free variables of ψ . Guarded predicate logic theories have the tree model property, which comes down to stating that every

⁹The translation is not presented here since this would require a detailed study of Conceptual Logic Programs, which would lead us too far astray.

satisfiable theory has a model which has a tree-shape, i.e. it can be represented by a labeled tree. This property figures prominently in the decidability results.

We will see further on that the generalization of the guarded fragment to guarded FO(ID) will inherit the tree model property (or, at least, a variant of it) and with it the decidability. From this one may suspect that there is a connection between the property of guardedness and the tree model property. An informal suggestion to their connection can be hinted at as follows.

The undecidability of predicate logic is due to the presence of quantifiers. Without the use of quantifiers, predicate logic poses no more challenges than propositional logic, which is decidable. The truth of a quantified formula in its most general sense, such as

$$\forall \mathbf{x} \psi(\mathbf{x})$$

can in the worst case only be decided by verifying the formula ψ for all possible combinations of domain elements \mathbf{x} . The same holds for existential quantifiers, in the sense that one might have to go through all possible combinations of domain elements to find out that it has been satisfied. The fact that in general a domain of quantification doesn't need to have any structure, which could guide the search process, causes the undecidability. Therefore a natural direction to obtain decidability is to limit the possibilities for \mathbf{x} , in such a way that all possibilities can be explored in a decidable fashion. Guardedness poses a restriction to the occurrence of quantifiers that does just this. The \mathbf{x} part in the guarded formulas above are either interpreted as specific domain elements, or they are themselves variables in a guarded formula of which these formulas are subformulas. In the end, the domain of every complex formula ψ will have been restricted to the domain of an atom, or to a subdomain of an atom.

If we use a tree to represent a model, what happens is that we group together domain elements which occur together in instantiations of atoms. In practice this means we limit the possibilities of satisfying an atom $G(a, x_1, \dots, x_n)$, where a is a domain element and the x_i are variables, to those elements which are represented by sets of nodes that have an element in common to the set of nodes representing a . In this way we are at the same time limiting the reach of quantified formulas to these elements, so that their truth or falsehood can be determined by checking properties of all nodes which represent them. The structure of the trees used is such that this comes down to checking properties

of nodes in a certain subtree. Even if this subtree is infinite, the elements out of which any such tree is built up are limited, so that at some point certain patterns will re-emerge and one can anticipate the continuation. It might be helpful to keep this explanation in mind in order to understand the approach used in this work.

3 The Decidability of Guarded FO(ID)

3.1 Introduction

Our goal is to give an effective reasoning procedure for the guarded fragment of FO(ID), but in order to achieve this we shall first introduce the decidability of guarded FO(ID) without giving such a procedure. This is the purpose of the current section. The results we shall present show a correspondence between satisfiable guarded FO(ID) theories and accepted tableaux constructed in function of such theories. We will use this correspondence as an intermediary for a further correspondence, namely between satisfiable guarded FO(ID) theories and so-called *completion structures* constructed in function of such theories. These completion structures are finite representations of tree-structured models of FO(ID) theories.

The guarded fragment is a decidable fragment of FO(ID), analogous to the guarded fragment of fixed point logic. The latter was introduced by Grädel in [8], by adding greatest and least fixpoints to the guarded fragment of first order logic, mentioned in section 2.5. We have already stressed the connections between the ideas behind the guarded fragment and the development of Conceptual Logic Programs, which form a decidable fragment of OASP. The decidability of the latter and of guarded FO(ID) is shown in a similar fashion, by reducing the question of satisfiability of a theory (or of a predicate, in the case of OASP) to the question of the non-emptiness of a two-way alternating tree automaton (2ATA) [8, 11]. The rest of this chapter presents the necessary theoretical results to be found in [20].

3.2 Guarded FO(ID)

We recall that a FO formula is called guarded if every one of its quantifiers is of the form

$$\exists \mathbf{y}(G(\mathbf{x}, \mathbf{y}) \wedge \varphi(\mathbf{x}, \mathbf{y}))$$

or

$$\forall \mathbf{y}(G(\mathbf{x}, \mathbf{y}) \rightarrow \varphi(\mathbf{x}, \mathbf{y}))$$

such that, $free(\varphi(x, y)) \subseteq free(G(x, y))$. We now define a guarded fragment of FO(ID). This fragment will allow only theories of the form $\{\Delta, \varphi\}$, where Δ is a definition and φ an FO formula. All FO(ID) theories can be brought into this form by a transformation which typically (though not always) preserves guardedness.

Definition 4. Let T be an FO(ID) theory, consisting of precisely one definition Δ and one FO formula φ . T is guarded if Δ is total, φ is a guarded formula and for each rule $\forall \mathbf{x}P(\mathbf{x}) \leftarrow \psi$, it holds that ψ is a guarded formula and $\mathbf{x} \subseteq free(\psi)$.

We introduce some simplifying assumptions. From now on, we will assume that all formulas are in a negation normal-form, in which negation only appears directly in front of atoms. Moreover, we also assume that, for each defined predicate D , there is precisely one rule $\forall \mathbf{x}D(\mathbf{x}) \leftarrow \psi$ in Δ . Every definition Δ can be brought into this form, by removing the set R_D of all such rules from Δ , and replacing it by the single rule $\forall \mathbf{x}D(\mathbf{x}) \leftarrow \bigvee_{r \in R_D} body(r)$; moreover, this transformation preserves guardedness. If r is the unique rule of Δ for which $head(r) = D(\mathbf{x})$, we will denote the formula $body(r)$ by $\varphi_D(\mathbf{x})$.

To ease notation, we fix Δ and φ from now on.

3.3 Tableaux

In this section, we define a tableau as a tree structure, whose nodes will be labeled with certain sets of formulas. These formulas are taken from the following set:

Definition 5. We inductively define the set of all φ -relevant formulas as consisting of those formulas that are:

- a subformula of φ ; or
- of the form $P(\mathbf{x})$ for some predicate P appearing in φ ; or
- φ_P -relevant, with P a defined predicate appearing in φ .

For a set of constants C containing $const(\varphi)$, the set of all (φ, C) -relevant formulas consists of all ground formulas $\psi(\mathbf{c})$ for which $\psi(\mathbf{x})$ is φ -relevant and $\mathbf{c} \subseteq C$.

To be more precise, we will consider sets of (φ, C) -relevant formulas that satisfy the following closure properties.

Definition 6. Let φ be a formula and C a set of constants. A (φ, C) -type is a set Γ of (φ, C) -relevant formulas, that satisfies the following conditions:

- For all (φ, C) -relevant atoms $P(c)$, Γ contains $P(c)$ or $\neg P(c)$, but not both;
- If $\varphi_1 \wedge \varphi_2$ belongs to Γ , then φ_1 and φ_2 also belong to Γ ;
- If $\varphi_1 \vee \varphi_2$ belongs to Γ , then φ_1 or φ_2 also belong to Γ ;
- If $\forall \mathbf{x} \gamma(\mathbf{x}) \Rightarrow \varphi(\mathbf{x})$ and $\gamma(\mathbf{c})$ belong to Γ , then $\varphi(\mathbf{c})$ belongs to Γ .
- For a defined predicate P , if $P(\mathbf{t})$ belongs to Γ , then $\varphi_{P(\mathbf{t})}$ belongs to Γ ; if $\neg P(\mathbf{t})$ belongs to Γ , then the negation normal form of $\neg \varphi_{P(\mathbf{t})}$ belongs to Γ .

When constructing a (φ, C) -type Γ , there are two kinds of choices to be made: for open atoms $P(\mathbf{c})$, we have to choose whether $P(\mathbf{c}) \in \Gamma$ or $\neg P(\mathbf{c}) \in \Gamma$; and for a disjunction $\psi_1 \vee \psi_2 \in \Gamma$, we have to choose whether ψ_1 , ψ_2 , or both belong to Γ . Given a structure \mathcal{S} and a valuation χ for C , we will say that a (φ, C) -type Γ is realized in \mathcal{S}, χ if $\mathcal{S}, \chi \models \Gamma$ and, for each $\psi_1 \vee \psi_2 \in \Gamma$, $\psi_i \in \Gamma$ if and only if $\mathcal{S}, \chi \models \psi_i$. It can easily be seen that the following property holds:

Lemma 7. *Let φ be a formula and C a set of constants. Let \mathcal{S} be a structure interpreting C and the alphabet of φ . For each subsentence ψ of φ such that $\mathcal{S} \models \psi$, there exists a (φ, C) -type that contains ψ and is realized in \mathcal{S} .*

Let C_0 be the set of all constants appearing in φ -relevant formulas. We now define tableaux as follows.

Definition 8. Let $K \supseteq C_0$ be a set of constants. A K -tableau for φ is a tree whose nodes are pairs (Γ, C) of a set of constants $C \subseteq K$ and a (φ, C) -type Γ , which can be constructed by the following non-deterministic induction:

- The root of the tree is a (φ, C_0) -type that contains φ ;
- Whenever there is a node (Γ, C) such that Γ contains $\exists \psi(\mathbf{x})$, but for all $\mathbf{c} \subseteq C$, $\psi(\mathbf{c}) \notin \Gamma$, we add a child (Γ', C') to this node as follows. We select a tuple of constants $c \subseteq K$ and let C' be $c \cup C_0 \cup \text{const}(\psi)$; Γ' is a (φ, C') -type that contains $\psi(\mathbf{c})$ and all sentences from Γ that contain only constants from C' .

3.4 The Automaton \mathcal{A}_T

The tableaux just presented will be used as input for a two way alternating tree automata. Such an automaton consists of the following components:

- A set of states Q , partitioned into a set of existential states, Q_{\exists} , and a set of universal states, Q_{\forall} ; one of these $q \in Q$ is designated as the initial state;
- A transition function t , mapping each pair (v, q) of a node v of the input tree and a state $q \in Q$ to a subset of $Q \cup \{\circ s \mid s \in Q\}$, where the meaning of the “ \circ ”-symbol is that if $\circ q' \in t(v, q)$, then the automaton can assume state q' and move to a neighbouring node v' of the input tree - this new v' can be either a child or a parent of v . If a state $q' \in t(v, q)$, on the other hand, then the automaton can assume state q' while remaining in node v .
- A score function Ω , mapping each state $q \in Q$ to a number $\Omega(q) \in \mathbb{N}$.

The accepting condition of such an automaton is defined by means of a parity game. This game is played between two players, 1 and 2. A *play* is defined as a sequence of state-node pairs in the obvious way, i.e., each play starts in (I, r) where I is the initial state of the automaton and r the root of the input tree, and a play reaching (v, q) can be extended by either a transition to (v, q') for which $q' \in t(v, q)$ or by a transition to (v', q') for which $\circ q' \in t(v, q)$ and v' is a child or parent of v . In these plays, player 1 chooses the move in all existential states $q \in Q_{\exists}$ and player 2 chooses the move in the other states.

The winner of a play is decided as follows. If the play is finite, the player who can no longer make a move loses. If the play is infinite, we consider the set Q_{∞} of states that are repeated infinitely often and look, in particular, at $\min_{q \in Q_{\infty}} \Omega(q)$: if this number is even, player 1 wins; otherwise, player 2 wins. A *strategy* for player 1 is a function f mapping each pair (v, q) to some other pair (v', q') such that (v', q') is a valid move in (v, q) . A play *follows* a strategy f if every state (v, q) for which $q \in Q_{\exists}$ is indeed followed by $f(v, q)$. A strategy is *winning* if and only if every play that follows it is indeed winning for player 1. An automaton *accepts* an input tree T if there is a winning strategy for T .

We now define an automaton \mathcal{A}_{φ} to check whether a tableau represents a model of $\{\Delta, \varphi\}$.

Definition 9. For a set of constants $C \supseteq C_0$, we define \mathcal{A}_{φ}^C as follows:

- Its states are (φ, C) -relevant formulas, where states of the form $\exists \mathbf{x}\psi$, $\psi_1 \vee \psi_2$ or f are existential and all others are universal; the initial state is φ ;
- The possible transitions $t(v, \psi)$ are as follows:
 - $t(v, \psi_1 \vee \psi_2) = t(v, \psi_1 \wedge \psi_2) = \{\psi_1, \psi_2\}$;
 - $t(v, \mathbf{f}) = t(v, \mathbf{t}) = \emptyset$;
 - For an open literal l , $t((\Gamma, C), l)$ is $\{\mathbf{t}\}$ if $l \in \Gamma$ and $\{\mathbf{f}\}$ otherwise;
 - For a defined predicate D , $t(v, D(\mathbf{t}))$ is $\{\varphi_{D(\mathbf{t})}\}$; $t(v, \neg D(\mathbf{t}))$ is $\{\psi\}$ with ψ the negation normal form of $\neg\varphi_{D(\mathbf{t})}$;
 - Let ψ be either $\exists \mathbf{x}\gamma(\mathbf{x}) \wedge \eta(\mathbf{x})$ or $\forall \mathbf{x}\gamma(\mathbf{x}) \Rightarrow \eta(\mathbf{x})$. If $\text{const}(\psi) \not\subseteq C$, then $t(v, \psi) = \emptyset$. Otherwise, $t(v, \psi) = \{\circ\psi\} \cup \{\eta(\mathbf{c}) \mid \mathbf{c} \subseteq C \text{ and } \gamma(\mathbf{c}) \in \Gamma\}$;
- The score function Ω assigns 4 to all states, except:
 - Ω assigns 1 to all states of the form $D(\mathbf{t})$ with D a defined predicate;
 - Ω assigns 2 to all states of the form $\neg D(\mathbf{t})$ with D a defined predicate;
 - Ω assigns 3 to all states of the form $\exists \mathbf{x}\psi(\mathbf{x})$;

3.5 Guiding the Automata

Our proofs will require us to construct winning strategies for our automaton. This means we need to guide it towards the right reason why some formula holds. For instance, let us consider the following definition:

$$\begin{cases} P \leftarrow P \vee Q \\ Q \leftarrow \mathbf{t} \end{cases}$$

This definition has one model, namely, both P and Q are true. Inspecting the formula $\varphi_P = P \vee Q$, we see that there are two possible motivations for why P holds: either it holds because P holds (which it does!), or it holds because Q holds (which it also does). The right explanation is of course the second one, since it should not be possible to derive P solely from itself. To make this distinction, we need some additional property of formulas.

We fix a well-founded induction $(\nu_\alpha)_{\alpha \leq \beta}$ for the definition Δ . For every formula ψ with $\nu_\beta(\psi) = \mathbf{t}$, there exists some ordinal $\eta \leq \beta$ such that for all

$\delta < \eta$, $\nu_\delta(\psi) = \mathbf{u}$, and for all $\delta \geq \eta$, $\nu_\delta(\psi) = \mathbf{t}$. We call this ordinal η the *level* of ψ and denote it as $|\psi|$. This level of course depends on the particular sequence $(\nu_\alpha)_{\alpha \leq \beta}$; in order to not overly complicate notation, however, we leave this implicit. For uniformity, we also define $|\psi|$ for formulas with $\nu_\beta(\psi) = \mathbf{f}$; here, we make $|\psi|$ some ordinal $\delta > \beta$. Analogously, for each ψ with $\nu_\beta(\psi) = \mathbf{f}$, there exists some ordinal $\eta \leq \beta$ such that for all $\delta < \eta$, $\nu_\delta(\psi) = \mathbf{u}$, and for all $\delta \geq \eta$, $\nu_\delta(\psi) = \mathbf{t}$. We call this η the *negation level* of ψ and denote it as $[\psi]$; for a formula ψ such that $\nu_\delta(\psi) = \mathbf{t}$, we define $[\psi]$ to be an ordinal $\delta > \beta$.

Lemma 10. *Let $(\nu_\alpha)_{\alpha \leq \beta}$ be a well-founded induction for Δ .*

- If $|\psi| \leq \beta$, then $\nu_\beta(\psi) = \mathbf{t}$; if $[\psi] \leq \beta$, then $\nu_\beta(\psi) = \mathbf{f}$;
- $|\psi_1 \vee \psi_2| = \min(|\psi_1|, |\psi_2|)$ and $|\exists \mathbf{x}\psi(\mathbf{x})| = \min_{\mathbf{d} \subseteq D} (|\psi[\mathbf{d}]|)$;
- $|\neg\psi| = [\psi]$ and $[\neg\psi] = |\psi|$.

This of course implies also $[\psi_1 \wedge \psi_2] = \min([\psi_1], [\psi_2])$ and $[\forall \mathbf{x}\varphi(\mathbf{x})] = \min_{\mathbf{d} \subseteq D} ([\varphi[\mathbf{d}]])$.

3.6 Accepted Tableaux correspond to models and vice versa

We now give the results which we will use later on, without proof.

Theorem 11. *If φ is satisfiable, there exists a tableau in a set $K \supseteq C_0$ of $2w(\varphi) + |C_0|$ constants that is accepted by \mathcal{A}_φ^K .*

In order to formulate the soundness result, we show how to construct a model \mathcal{S}_T from an accepted tableau \mathcal{T} . For a constant c , two nodes v, v' in \mathcal{T} are c -equivalent if each node on the path between them has c in its label. The domain of \mathcal{S}_T is the set of all c -equivalence classes $[v]_c$. \mathcal{S}_T interprets each constant $c \in C_0$ by $[v]_c = \mathcal{T}$ and each open predicate P as follows: $([v_1]_{c_1}, \dots, [v_n]_{c_n}) \in P^{\mathcal{S}_T}$ if there exists a $(\Gamma, C) \in \cap_i [v_i]_{c_i}$ such that $P(c_1, \dots, c_n) \in \Gamma$. Given this interpretation for the open predicates, we interpret the defined predicates as dictated by the definition Δ . For a node $v = (\Gamma, C) \in T$, we define the valuation χ_v as mapping each $c \in C$ to the domain element $[v]_c$.

Theorem 12. *If \mathcal{A}_φ^K accepts a tableau \mathcal{T} , then $\mathcal{S}_T \models \varphi$.*

4 A Reasoning Procedure for Simple Guarded FO(ID)

4.1 Introduction

The results given in the previous section depend on results for 2ATA's, and allow for satisfiability checking in an obscure and indirect way only. For conceptual logic programs this problem has been overcome, by developing effective reasoning procedures to check for satisfiability. At first, [6] this result was obtained for a restricted version of CoLP's, namely simple CoLP's, but recently these results have been generalized in [13, 14] to the area of extended conceptual logic programs. We will parallel their approach, by developing an effective reasoning procedure for a restricted part of the guarded fragment of FO(ID) in this section, and generalizing this result in the next section. We shall call the restricted fragment under consideration in this section the *simple* guarded fragment of FO(ID). The challenge will be to represent an infinite model in a finite way, which we shall resolve building on the notion of a completion structure as found in [6]. However, due to the differences between guarded FO(ID) and CoLP's, our completion structures and the associated reasoning procedure contain many elements not to be found in their counterpart for conceptual logic programs.

The working of the 2ATA's is such that in both cases they perform three checks on an input-tree, which is the representation of a possible model:

- check if the tree is well-behaved, in the sense that it has the right form to be a possible model of the theory/predicate
- check if it is consistent
- check if some acceptance condition is fulfilled, which is related to restrictions due to the semantics

The first two checks do not depend on the tree (and thus model) being finite or not, in the sense that you have to check for each node (or certain groups of nodes) that it fulfills certain conditions, and if the tree happens to be infinite you will have to perform these same checks an infinite amount of times. The third check, however, is a condition that has to be fulfilled by each infinite branch of the tree. Namely, that a certain kind of states (which correspond to certain characteristics of the node one is at) cannot occur an infinite amount of times (or, in the case of FO(ID), cannot occur if some other kind of state doesn't occur

an infinite amount of times). In both cases, the main condition is that there may not be an infinite amount of positive states. Informally, this can be explained as follows.

The tree represents a justification for either a formula (in FO(ID)) or an atom (in OASP). The first node states that the formula is true (or, the atom is part of a model), and the other nodes are formed to justify this statement. If the presence of a positive literal in a certain node requires an infinite justification, this will result in the tree having an infinite branch. If the 2ATA goes through this branch, it will enter a positive state whenever the node it is at contains a positive (defined, or non-free) literal. Thus, the acceptance condition is meant to express that it is not allowed to have a justification which requires an infinite amount of positive (defined, or non-free) literals to be part of the model. However, due to the way the 2ATA's are constructed in the CoLP case and the FO(ID) case respectively, there's a difference in interpretation of the previous statement. We now proceed to define the necessary concepts we need to built up our reasoning procedure.

4.2 Simple Guarded FO(ID)

Our first approach is to develop the procedure for theories that have a restricted syntax, due to two motivations. Firstly, it is easier to understand the full algorithm after a simpler version has been studied first. Secondly, unfortunately we failed to establish a full proof for the completeness of the general procedure, therefore the simpler version is at this moment the only one for which we have established a finished result.

There are two restrictions we will enforce, that serve to ensure that every positive defined atom can be justified by a finite number of different defined literals. This is not true in general for guarded FO(ID) theories, since although it is prohibited that any atom is justified by an infinite number of positive defined atoms, it may well be the case that one needs an infinite amount of negative defined atoms to justify it. Further, we also want to ensure that no positive defined atom $P(c)$ needs an atom $P(d)$ to justify it, where $c \neq d$.¹⁰

Let $D(T)$ be the marked predicate dependency graph of a guarded FO(ID) theory T , where $D(T)$ has as vertices the predicates from Δ and as arcs tuples (P, Q) where $Q(\mathbf{y})$ positively occurs in $\varphi_P(\mathbf{x})$, we call an arc (P, Q) marked if $\mathbf{x} \neq \mathbf{y}$.

¹⁰The case where $c = d$ is already prohibited by the semantics of FO(ID).

Definition 13. A guarded FO(ID) theory T is *simple* if it is stratified and its marked predicate dependency graph $D(T)$ doesn't contain any cycle with a marked edge.¹¹

We will now present the structures that will represent possible models of a theory T and contain the necessary information to justify φ . Our reasoning procedure will consist in building up such structures and checking if they fulfill the conditions for legitimately representing a model.

4.3 Completion Structures

4.3.1 Trees¹²

For an $x \in \mathbb{N}_0^*$, i.e. a finite sequence of natural numbers (excluding 0), we denote the concatenation of a number $c \in \mathbb{N}_0$ to x as xc . Formally, a (*finite*) *tree* is a (finite) subset of \mathbb{N}_0^* such that if $xc \in T$ for $x \in \mathbb{N}_0^*$ and $c \in \mathbb{N}_0$, then $x \in T$. Elements of T are called *nodes* and the empty word ε is the *root* of T . For a node $x \in T$ we call $\text{succ}_T(x) = \{xc \in T \mid c \in \mathbb{N}_0\}$, the *successors* of x . The *arity* of the tree is the maximum amount of successors any node has in the tree. The set $A_T = \{(x, y) \mid x, y \in T, \exists c \in \mathbb{N}_0 : y = xc\}$ denotes the edges of a tree T . We define a partial order \leq on a tree T such that for $x, y \in T$, $x \leq y$ iff x is a prefix of y . As usual, $x < y$ if $x \leq y$ and $y \not\leq x$. A (finite) *path* P in a tree T is a prefix-closed subset of T such that $\forall x \neq y \in P : |x| \neq |y|$. A *branch* B in a tree T is a maximal path (there is no path that contains it) which contains the root of T .

A labeled tree is a pair (T, t) where T is a tree and $t : T \rightarrow \Sigma$ is a labeling function; sometimes we will identify the tree (T, t) with t . We denote the subtree of T at x by $T[x]$, i.e., $T[x] = \{y \in T \mid x \leq y\}$. For labeled trees $t : T \rightarrow \Sigma$, the subtree of t at $x \in T$ is $t[x] : T[x] \rightarrow \Sigma$ such that $t[x](y) = t(y)$ for $y \in T[x]$. For a tree $t : T \rightarrow \Sigma$, a tree $s : S \rightarrow \Sigma$, and a symbol $a \in \Sigma$, we denote with $t_a s$, the tree t with the subtrees starting with the first node on every path with label a (in case such a node exists) replaced by s . Consider the trees s and t depicted in Figure 1. The tree resulted by the application of $t_a s$ is depicted in Figure 2.

The basic data structure for our algorithm will be a completion structure. (We continue to assume Δ and φ are fixed.)

¹¹Thus "simple" is nearly the same as stating that there may be no cycle in the dependency graph, where one also takes into account negative occurrences. The definition of stratified was given on page 9.

¹²We define trees as in [6].

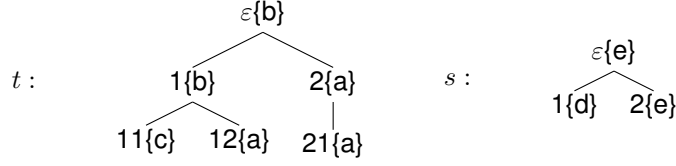


Figure 1: Two labeled trees: t and s .

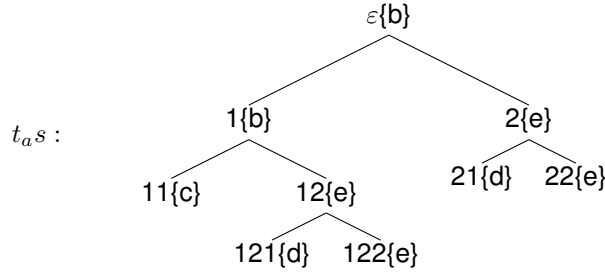


Figure 2: The new tree: $t_a s$.

Definition 14. A K -completion structure for φ is a tuple $\langle T, K, G, C, CT, ST, DJ, EX \rangle$.

K is a set of $2w(\varphi) + |C_0|$ constants, where C_0 are the constants appearing in φ and $C_0 \subseteq K$. We will assume K to be such a set throughout this work. T is a tree which together with the labeling functions C, CT, ST, DJ , and EX is used to represent/construct a tentative model. $G = \langle V, E \rangle$ is a directed graph with nodes $V \subseteq \{(x, \psi) | x \in T, \psi \in S_\varphi^K\}$, where S_φ^K is the set of all (φ, K) -relevant formulas. An edge $[(x, \psi_1); (y, \psi_2)]$ is to be interpreted as stating that formula ψ_2 in node y is needed to justify ψ_1 in x . Below the signature and the role for each labeling function is given.

- The *constant* function $C : T \rightarrow 2^K$ maps a node of the tree to a subset of constants from K . This function tells us which constants are used in formulas in the label of x .
- The *content* function $CT : T \rightarrow 2^{S_\varphi^K}$ maps a node $x \in T$ to a set of $(\varphi, C(x))$ -relevant formulas. The content function and the constant function are used to construct a (φ, C) -type, as occur in the tableaux we defined earlier.
- The *status* function $ST : \{(x, \psi) | x \in T, \psi \in CT(x)\} \rightarrow \{exp, unexp\}$ attaches to every formula in each node a status value which indicates whether the formula has already been expanded in that node. Formulas

need to be justified, which will happen through expanding them. Therefore it is important to know which formulas still need to be expanded.

- The *disjunct* function $DJ : \{(x, \psi) \mid x \in T, \psi \in CT(x), \psi = \psi_1 \vee \psi_2\} \rightarrow \{1, 2\}$ indicates which part of a disjunct motivates the truth of a disjunction. If two motivations are possible, one will be chosen nondeterministically.
- The *exist* function $EX : \{(x, \psi) \mid x \in T, \psi \in CT(x), \psi = \exists \mathbf{x} \gamma(\mathbf{x}) \wedge \eta(\mathbf{x})\} \rightarrow \{(y, \mathbf{c}) \mid y \in T, \mathbf{c} \subseteq K\}$ indicates in which node and by which constants an existential formula is instantiated. When there are several options, the algorithm will choose one nondeterministically.

An initial K -completion structure for φ is a completion structure with $T = \{\varepsilon\}$, $V = \{(\varepsilon, \varphi)\}$, $E = \emptyset$, $C(\varepsilon) = C_0$, $CT(\varepsilon) = \{\varphi\}$, $ST(\varepsilon, \varphi) = unexp$ and the other labeling functions undefined for every input.

To see how a completion structure can represent a possible model, one can use the same method as we defined for tableaux on page 20, whereby for a node $v = (\Gamma, C)$ of a tableau the equivalent notions in the language of completion structures are given by $CT(x) = \Gamma$ and $C(x) = C$ for a node x in T . This works because a K -completion structure is basically a finite K -tableau with extra labeling functions and an extra graph G . In order to deal with theories that have models with an infinite domain, we will show later on how to extend a completion structure into an infinite tableau.

We will now give rules for expanding a completion structure - the expansion rules - and conditions for determining when no more expansion is allowed. Also we will present some further conditions, in order for a completion structure to have served its purpose in proving that φ is satisfiable. In particular, the expansion rules will built up a completion structure starting from an initial structure by non-deterministically looking for a justification for every formula it contains. Because of the guarded nature of our formulas, this justification can take the form of a tree, or a tableau, as the results in chapter 3 have learned us. However, although one can check with the automaton \mathcal{A}_φ^K whether a tableau *contains* a correct justification for the initial formula φ , it by itself has only a very limited way of explicitly *showing* the dependency structure of formulas. Only for existential formulas one has an idea of how they are justified, namely by adding a successor node to the node which contains it and making sure it is reduced in that successor. It's precisely this lack of internal structure which made it

necessary to construct the automata in order to determine satisfiability of a formula. This explains the motivation behind the use of the labeling functions and the dependency graph with completion structures. Keeping this mind should facilitate the understanding of the following rules.

4.3.2 Expansion Rules

The expansion rules need to update the completion structure whenever we are using a ϕ in a node y in order to justify the presence of some ψ in a node x .¹³ We will have to insert ϕ into the content of y , mark it as unexpanded, and add an edge in the dependency graph between (x, ψ) and (y, ϕ) . More formally we define the update operation as follows

Definition 15. The operation $update((x, \psi); (y, \phi))$ is short for the following series of operations:

If $\phi \notin CT(y)$, then $CT(y) = CT(y) \cup \{\phi\}$ and $ST(y, \phi) = unexp$; also, $V = V \cup \{(y, \phi)\}$ and $E = E \cup \{(x, \psi); (y, \phi)\}$.

In general, $update((x, \psi); \beta)$ for a set β , means $update((x, \psi); (y, \phi))$ for each $(y, \phi) \in \beta$.

Another definition is required to determine the reach of universal formulas, to be able to interpret a completion structure as a possible model.

Definition 16. For a set of constants c , two nodes x, x' in T are c -equivalent, notated $x \sim_c x'$, if each node z on the path between them is such that $c \subseteq C(z)$.

Expansion Rules Assume we are looking at a pair (x, ψ) such that $ST(x, \psi) = unexp$. The expansion rules indicate for each type of formula how to expand it. They are:

- if $\psi = \psi_1 \wedge \psi_2$, then
 - $update((x, \psi); \{(x, \psi_1), (x, \psi_2)\})$;
 - set $ST(x, \psi) = exp$.
- if $\psi = l$ with l an open literal,
 - set $ST(x, \psi) = exp$.

¹³Of course only formulas ϕ can be considered such that $const(\phi) \subseteq C(y)$.

- if $\psi = \pm P(\mathbf{c})$ with $P \in Def(\Delta)$, (where $+P(\mathbf{c}) = P(\mathbf{c})$ and $-P(\mathbf{c}) = \neg P(\mathbf{c})$), then
 - *update* $((x, \psi); (x, \pm\varphi(\mathbf{c})_P))$;
 - **set** $ST(x, \psi) = exp$.
- if $\psi = \psi_1 \vee \psi_2$, then
 - **choose** an $i \in \{1, 2\}$;
 - **set** $DJ(x, \psi) = i$;
 - *update* $((x, \psi); (x, \psi_i))$;
 - **set** $ST(x, \psi) = exp$.
- if $\psi = \forall \mathbf{x} \gamma(\mathbf{x}) \Rightarrow \eta(\mathbf{x})$, then
 - *update* $((x, \psi); \{(x, \eta(\mathbf{c})) | \gamma(\mathbf{c}) \in CT(x)\})$;
 - *update* $((x, \psi); \{(y, \psi) | x \sim_{const(\psi)} y\})$;
 - **set** $ST(x, \psi) = exp$.
- if $\psi = \exists \mathbf{x} \gamma(\mathbf{x}) \wedge \eta(\mathbf{x})$, then
 - if there exists a \mathbf{c} such that $\gamma(\mathbf{c}) \wedge \eta(\mathbf{c}) \in CT(x)$, then
 - * **set** $EX(x, \psi) = (x, \mathbf{c})$;
 - * *update* $((x, \psi); (x, \gamma(\mathbf{c}) \wedge \eta(\mathbf{c})))$;
 - * *update* $((x, \psi); (x, \eta(\mathbf{c})))$;¹⁴
 - * **set** $ST(x, \psi) = exp$.
 - else, add a new node $y \in succ_T(x)$ to T ;
 - * **select** a tuple $\mathbf{c} \in K \setminus C_0$;
 - * **set** $EX(x, \psi) = (y, \mathbf{c})$;
 - * **set** $C(y) = \mathbf{c} \cup C_0 \cup const(\psi)$;
 - * **set** $CT(y) = \{\gamma(\mathbf{c}) \wedge \eta(\mathbf{c})\} \cup \{\phi | \phi \in CT(x), const(\phi) \subseteq C(y)\}$;
 - * **set** $ST(y, \phi) = exp$ for all non-universal formulas in $\{\phi | \phi \in CT(x), const(\phi) \subseteq C(y)\}$;¹⁵

¹⁴This update is added to conform to the possible transitions in the automaton \mathcal{A}_φ^K , besides that it is redundant.

¹⁵These formulas represent the same formulas as their copies in $CT(x)$, so we don't need to expand them again. Only universal formulas can impose new constraints.

- * set $ST(y, \psi) = unexp$;
- * set $ST(y, \gamma(\mathbf{c}) \wedge \eta(\mathbf{c})) = unexp$;
- * update((x, ψ) ; (y, ψ));
- * set $ST(x, \psi) = exp$.

Once we have expanded a pair (x, ψ) , it can be ignored for the further construction of the completion structure.

Choice Rule Suppose we have an $x \in T$ for which none of the $\psi \in CT(x)$ can be expanded anymore, and there's an atom (open or defined) $p(\mathbf{c}) \in S_\varphi^{C(x)}$, such that $p(\mathbf{c}) \notin CT(x)$ and $\neg p(\mathbf{c}) \notin CT(x)$. Then, either add $p(\mathbf{c})$ to $CT(x)$ with $ST(x, p(\mathbf{c})) = unexp$, or add $\neg p(\mathbf{c})$ to $CT(x)$ with $ST(x, \neg p(\mathbf{c})) = unexp$.

4.3.3 Simple Applicability Rules

Besides rules for expanding the completion structure, there are rules which restrict the use of the expansion rules.

Definition 17. We will call a node $x \in T$ *saturated* if for all atoms $p(\mathbf{c}) \in S_\varphi^{C(x)}$, either $p(\mathbf{c}) \in CT(x)$, or $\neg p(\mathbf{c}) \in CT(x)$ and none of the $\psi \in CT(x)$ can be expanded anymore.

We impose that no expansions can be performed on a node from T until its predecessor is saturated.

Definition 18. We call a node $x \in T$ *blocked* if

- it is saturated, and;
- there is an ancestor y of x , $y < x$, such that $CT(x) = CT(y)$ and $C(x) = C(y)$;
- for each $\psi = \psi_1 \vee \psi_2$, it holds that $DJ(x, \psi) = DJ(y, \psi)$.

We call (y, x) a *blocking pair* and say that y blocks x ; we will also refer to x as a blocked node and to y as the blocking node for a blocking pair (y, x) . Also, we say that x and y are copies of each other. $blocked(T)$ is the set of all blocking pairs of the completion structure T . We impose that no expansions may occur on a blocked node, and we delete all nodes from T that are younger than blocked nodes. Likewise we delete all nodes and edges containing these nodes in G .

The goal of a completion structure is to give a finite representation of a possibly infinite model. We shall end up with a finite tree T , which will serve as a concise representation of an infinite tree T_{ext} in the following sense: T_{ext} is entirely made up of subtrees of T , some of which will re-occur an infinite amount of times. The idea is that although a model of φ might require an infinite justification, this justification will be built up of finite patterns that keep coming back. Intuitively this assumption is motivated by the fact that the number of φ -relevant formulas and the connections that hold between them is finite, so that any infinite justification must be a repetition of the same dependencies between a set of formulas for different sets of domain elements. Formally we shall vindicate it in our proofs in chapter 6.

Definition 19. For a graph $F = \langle V_F, E_F \rangle$ and a set S of pairs of nodes from T , we define the *closure* of F by S , denoted $F_S = \langle V_{F_S}, E_{F_S} \rangle$, as the result of the following operations:

- $E_{F_S} = E_F \setminus \{[(z, \psi_1); (x, \psi_2)] \mid \exists v \in T : (v, x) \in S \vee (v, z) \in S\}$;
- $E_{F_S} = E_{F_S} \cup \{[(z, \psi_1); (v, \psi_2)] \mid \exists x \in T : (v, x) \in S, [(z, \psi_1); (x, \psi_2)] \in E_F\}$;
- $E_{F_S} = E_{F_S} \cup \{[(v, \psi_1); (z, \psi_2)] \mid \exists x \in T : (v, x) \in S, [(x, \psi_1); (z, \psi_2)] \in E_F\}$;
- $V_{F_S} = V_F \setminus \{(x, \psi) \mid (x, \psi) \in V_F, \exists v : (v, x) \in S\}$.

Informally the closure of F by S is formed by removing reference to nodes x which figure on the right side of a couple in S , and replace it everywhere with the nodes v that figure on the left side of each such couple.

We defined this concept to introduce the next one:

Definition 20. For a node $y \in T$, we define its y -graph as the graph G_y which can be constructed as follows:

- If there exists no node x so that $(y, x) \in blocked(T)$, $G_y = \emptyset$.
- We denote the set of all nodes in $T[y]$ that occur before the blocked nodes or on a branch without a blocked node as TL_y , i.e. $TL_y = \{x \mid x \in T[y], \forall w \in T[y] : (y, w) \in blocked(T) \Rightarrow w \not\prec x\}$.
- Take from G only those nodes and edges containing the nodes in TL_y , we name the resulting graph G'_y .¹⁶

¹⁶So G'_y is the dependency graph for the subtree $T[y]$. This is because we want to simulate the dependencies that hold if we were to replace every blocked node with the subtree $T[y]$, and then in turn replace all new copies of blocked nodes again with $T[y]$, and so on.

- Define G_y as the closure of G'_y by $\{(y, x) \mid (y, x) \in \text{blocked}(T)\}$.

To find out whether an y -graph is suitable, we introduce the following condition.

Definition 21. A graph G is *cycle-free* if either $G = \emptyset$, or if there are no cycles in G containing a positive defined atom. For a node $y \in T$, if its y -graph G_y is cycle-free, we will call y and $T[y]$ cycle-free as well.

At a certain point no further expansion will be possible, since eventually each branch will either require no further expansion or it will contain a blocked node. We then transform the directed graph G by creating the closure of G by $\text{blocked}(T)$. We will refer to this operation as the *closing rule*, and after it has been applied we will refer to the closure of G by $\text{blocked}(T)$ simply as G .

The effect of this transformation will be that all references to the blocked nodes are deleted, and the edges to nodes (nodes from V , which are couples made up of a node and a formula)¹⁷ containing it are replaced by edges to the associated nodes containing its blocking node. It's important to note that the closing rule makes sure that all edges which occur in a cycle-free G_y also end up in G , and that vice versa all edges in G added by the closing rule appear as well in some G_y . Therefore G is cycle-free iff all nodes y are cycle-free.

Definition 22. A *simply complete* K -completion structure for φ is a completion structure that results from applying the expansion rules to the initial K -completion structure for φ , taking into account the simple applicability rules, the delete rule and the closing rule such that no rules can be further applied.

Further, a node x could never correspond to a set of true formulas in a model if there is an atom $p(\mathbf{c})$ such that $p(\mathbf{c}) \in CT(x)$ and $\neg p(\mathbf{c}) \in CT(x)$. A complete completion structure for which there is such a node and atom is called *contradictory*. Putting all this together, we say a completion structure T is *simple clash-free* if (1) T is simply complete, (2) T is not contradictory, and (3) G is cycle-free.

4.4 Completeness

In the next chapter we will develop a more general algorithm, which will have more complex applicability rules. It will become clear that every *simple* clash-free completion structure also is a clash-free structure. Therefore termination

¹⁷Nodes can be elements of T , but the graph G also contains nodes, which are of the form (x, ψ) . The context will make clear which kind of nodes are meant.

and soundness of our approach for simple clash-free structures will be implied by the corresponding results for clash-free completion structures. So granted that these results will be established, we only need to prove the completeness of our procedure.

Theorem 23. *Completeness: If T is a simple guarded FO(ID) theory, and φ is satisfiable, then there is a simple clash-free K -completion structure for φ .*

Proof. Let $\mathcal{S} \models \varphi$. We fix a well-founded induction $(\nu_\alpha)_{\alpha < \beta}$ in $\mathcal{S} \upharpoonright_{Op(\Delta)}$. We construct a simple clash-free K -completion structure T and, for each node x in T , a valuation χ_x for $C(x)$, while preserving the invariant that $\mathcal{S}, \chi_x \models CT(x)$:

- The root is the initial completion structure for φ .
- We apply the algorithm for expanding a completion structure, where we take into account the following:
 - if $\psi = \psi_1 \vee \psi_2$, then choose an $i \in \{1, 2\}$ such that $|\psi_{\chi_x, i}| = \min\{|\psi_{\chi_x, 1}|, |\psi_{\chi_x, 2}|\}$;
 - if $\psi = \exists \mathbf{x} \gamma(\mathbf{x}) \wedge \eta(\mathbf{x})$, then consider the non-empty set of tuples $\mathbf{d} \subseteq \text{dom}(\mathcal{S})$ for which $\mathcal{S}, \chi_x \models \gamma[\mathbf{d}] \wedge \eta[\mathbf{d}]$, and select from this a tuple \mathbf{d} with minimal $|\gamma_{\chi_x}(\mathbf{d}) \wedge \eta_{\chi_x}(\mathbf{d})|$.
 - * add a new node $y \in \text{succ}_T(x)$ to T ;
 - * select a tuple $\mathbf{c} \subset K \setminus C_0$ of fresh constants;
 - * the valuation χ_y coincides with χ_x on $\text{const}(\psi) \cup C_0$ and maps \mathbf{c} to \mathbf{d} . This clearly preserves the invariant.
 - when applying the choice rule, always add literals $l(\mathbf{c})$ so that $\mathcal{S}, \chi_x \models l(\mathbf{c})$.

It is clear that our structure can be simply completed, as any structure can. It is also clear that it is non-contradictory. So what remains to be shown is that T is cycle-free.

We will start by proving that a certain kind of cycle cannot occur. Assume there is a cycle in G containing a positive defined atom $P(\mathbf{c})$ such that the cycle does not contain an edge that was created by an application of the closing rule. Thus there is a sequence of connected nodes $(m_i, \psi_i)_{i \in [a, b]}$ with $\psi_a = \psi_b = P(\mathbf{c})$ and $m_a = m_b$. Assume $\chi_{m_a}(\mathbf{c}) = \mathbf{d}$. Observe that T is so constructed that, up until the nodes which were predecessors of blocked

nodes (the nodes that were cut off by applying the closing rule), the dependency graph G represents the actual dependencies of formulas in S . More formally, this can be expressed by the following property:

Proposition 24. *Given that S and T are as above, then it holds that for any $[(x, \alpha); (y, \beta)] \in E$ which was not the result of applying the closing rule, $|\alpha_{\chi_x}| \geq |\beta_{\chi_y}|$.*

Proof. This is a direct result of our choice of expanding the completion structure, and lemma 10. \square

By considering the expansion rule for defined atoms, we know that $m_{a+1} = m_a$ and $\psi_{a+1} = \varphi_P(\mathbf{c})$. On the other hand, the semantics of FO(ID) dictate that $|P[\mathbf{d}]| < \beta \Rightarrow |P[\mathbf{d}]| > |\varphi_P[\mathbf{d}]|$, i.e., a defined atom can only become true if the truth of its defining body is already known to be true. We know that $|P[\mathbf{d}]| < \beta$, since $\mathcal{S}, \chi_{m_a} \models P(\mathbf{c})$. But together with the above proposition this leads to the contradiction $|\psi_{\chi_{m_a}, a}| > |\psi_{\chi_{m_a}, a+1}| \geq |\psi_{\chi_{m_a}, a}|$. Therefore we conclude that there is no cycle in G containing a positive defined atom which doesn't contain an edge that exists due to the closing rule.

Now assume there is a cycle in G containing a positive defined atom $P(\mathbf{c})$ which contains an edge that exists due to the closing rule. We will go through the cycle, starting from the node $(x, (P(\mathbf{c})))$ for some node x . The first edge needs to be $[(x, (P(\mathbf{c}))); (x, \varphi_P(\mathbf{c}))]$. The node $(x, \varphi_P(\mathbf{c}))$ is then further connected to a node with a subformula of $\varphi_P(\mathbf{c})$, that can take three different forms:

1. either it is of the form $\pm Q(\mathbf{d})$ with $Q \in Op(\Delta)$,
2. or it is a quantified formula,
3. or it is of the form $\pm Q(\mathbf{c})$ with $Q \in Def(\Delta)$.

In the first case the path has to stop, since open atoms aren't expanded. So this is impossible.

In the second case, remark that a quantified formula that occurs in the body of a defined predicate cannot contain a positive defined predicate, due to the restriction on the marked predicate dependency graph. The reduction of a quantified formula always has to end in a node $(y, \pm R(\mathbf{d}))$. If $R \notin Def(\Delta)$ we end up in the first case, so $R \in Def(\Delta)$. Therefore the occurrence of R is negative: the node is of the form $(y, \neg R(\mathbf{d}))$. Because T is stratified, $S(P) > S(R)$. It is clear from the expansion rules that for any path $(a, \pm A(\mathbf{v})) \mapsto (b, \pm B(\mathbf{w}))$

with $A, B \in Def(\Delta)$, it holds that $S(A) \geq S(B)$. Combining this with the fact that G contains the path $(y, \neg R(\mathbf{d})) \mapsto (x, P(\mathbf{c}))$, we obtain the contradiction $S(P) > S(R) \geq S(P)$.

This leaves us with the third possibility, namely that we reach a node $(x, \pm Q(\mathbf{c}))$, with $Q \in Def(\Delta)$. If the node is $(x, \neg Q(\mathbf{c}))$, the previous argument applies, so it must be of the form $(x, Q(\mathbf{c}))$. This situation is entirely similar to our starting point, so by the same reasoning the cycle must connect this node to a node $(x, R(\mathbf{c}))$, with $R \in Def(\Delta)$, without there occurring any quantified formulas in between. This continues until finally we end up in $(x, (P(\mathbf{c}))$ again. The only expansion rules that allow one to move from one node of T to another, are those relating to quantified formulas. Therefore the cycle remains in the node x throughout. But any edge that was added due to the closing rule is an edge connecting different nodes from T , so again we obtain a contradiction.

From this we conclude that G is cycle-free, and thus that our completion structure is simple clash-free. \square

5 A Reasoning Procedure for Guarded FO(ID)

5.1 Completion Structures

To develop the more general reasoning procedure for the whole fragment of guarded FO(ID), we need to adapt the applicability rules for deciding when a completion structure is completed. Thus the structures themselves are built up with the same tools as was the case for simple guarded FO(ID), but in general it will require more expansions to obtain a structure that can represent a tree-shaped model of a theory.

5.1.1 Applicability Rules

The first requirement we make is that all expansions occur breadth-first. This is important to avoid infinitely expanding on a single branch because of a problem that occurs in another branch which we never get to expanding. This shall become clearer when we consider cycles.

The notions of a saturated node and a blocked node remain as before, and likewise we impose that no expansions can be performed on a node from T until its predecessor is saturated. The next definition presents a new concept that was of no use in the simple case.

Definition 25. If for a node x it holds that $(y, x) \in blocked(T) \wedge [\neg \exists z \in T : x < z \wedge (y, z) \in blocked(T)]$, we shall say that $(y, x) \in lastblocked(T)$ and x is *last-blocked* by y . Of course a pair may go from being lastblocked to just blocked if new copies of y are added later on.

To predict which justifications will exist in the extended tree T_{ext} , we look at suitable subtrees in T which are possible candidates for re-occurring an infinite amount of times. In those trees we connect certain end-nodes to nodes which in T_{ext} will be the roots of a next copy of a subtree, by connecting the corresponding node-formula pairs in G . In this manner cycles are created, which represent infinite justificatory/dependence paths in T_{ext} . If such a cycle contains a positive defined atom we know that this attempt at justification will never succeed, since the semantics of FO(ID) does not allow for a positive defined atom to depend on an infinite number of positive defined atoms. Hence the following definitions.

Definition 26. For a node $y \in T$, we define an y -tree as a tree T_y which can

be constructed by cutting of each branch of $T[y]$ at some node z in such a way that either $(y, z) \in \text{lastblocked}(T_y)$, or $\neg \exists x \in T : (y, x) \in \text{blocked}(T) \wedge z < x$.

We now define a more general version of an y -graph.

Definition 27. For a node $y \in T$ and an y -tree T_y , we define an y -graph as a graph G_y which can be constructed as follows:

- If there exists no node x so that $(y, x) \in \text{blocked}(T)$, $G_y = \emptyset$.
- For every node x such that $(y, x) \in \text{lastblocked}(T_y)$, choose a $v \in T_y$ so that $(y, v) \in \text{blocked}(T_y)$, $(y, v) \notin \text{lastblocked}(T_y)$ or $v = y$; we denote the set of all such pairs as S_y . Further, we denote the set of all nodes in T_y that occur before the lastblocked nodes or on a branch without a lastblocked node as TL_y , i.e. $TL_y = \{x \mid x \in T_y, \forall w \in T_y : (y, w) \in \text{lastblocked}(T_y) \Rightarrow w \not< x\}$.
- Take from G only those nodes and edges containing the nodes in TL_y , we name the resulting graph G'_y .¹⁸
- Define G_y as the closure of G'_y by S_y .

Some cycles in a y -graph G_y containing positive defined predicates can cause problems, because they might represent an infinite justification for a defined atom which keeps decreasing in truth-level. To separate those that are problematic from those that aren't, we have the following definition.

Definition 28. A cycle $(x_i, \psi_i)_{i \in [k, l]}$ in a graph F_S that is the closure of F by S containing a positive defined atom is *harmless* if all of the following holds:

- It contains an edge $[(x_j, \psi_j); (x_{j+1}, \psi_{j+1})]$ which isn't part of F , i.e. there is a couple $(x_{j+1}, y) \in S$ and x_j is the predecessor of y ;
- It contains an edge $[(x_m, \psi_m); (x_{m+1}, \psi_{m+1})]$ so that $x_m = x_{j+1}$ and $x_{m+1} < x_m$;
- The part of the cycle in between these two edges doesn't contain another edge which isn't part of F .

¹⁸So G'_y is the dependency graph for the subtree T_y with all nodes younger than a lastblocked node cut off. This is because we want to simulate the dependencies that hold if we were to replace every lastblocked node with the subtree starting at its corresponding node on the left of a couple in S_y , and then in turn replace all new copies of lastblocked nodes again with the corresponding subtree, and so on. Therefore all nodes younger than lastblocked nodes are irrelevant for this test.

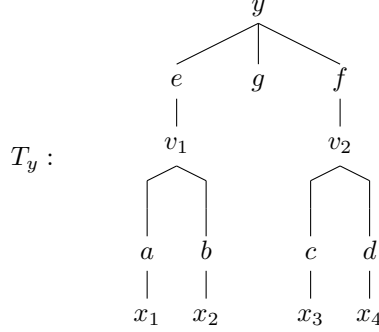


Figure 3: The y -tree T_y .

A harmless cycle is harmless because it doesn't represent any dependencies which will occur if the completion structure is extended into an infinite tableau, as will happen later on.

To illustrate the notions we just introduced, consider the example in Figures 3 and 4.

We are trying to find out whether T_y can be extended into an infinite tree which consists of subtrees $T_y[v]$, with $(y, v) \in \text{blocked}(T)$ or $v = y$. Take it that $\{y, v_1, v_2, x_1, x_2, x_3, x_4\}$ are all copies of each other. So we look for a suitable y -graph, which means that for every lastblocked node in $\{x_1, x_2, x_3, x_4\}$ we have to choose a node in $\{y, v_1, v_2\}$. Let's say we choose

$$S_y = \{(v_1, x_1); (v_1, x_3); (v_2, x_2); (v_2, x_4)\}$$

This choice is a simulation of the extended tree $T_{y,ext}$ that results from replacing every x_i with $T_y[v_i]$, for which the first replacement is shown in Figure 4.

The newly created nodes $\{x'_1, x'_2, x'_3, x'_4, x''_1, x''_2, x''_3, x''_4\}$ will be the subject of the next replacement, and so on. Rather than actually applying this infinite sequence of extension operations, we simulate which dependencies would hold by creating G_y . It is with this in mind that the harmless criterion should be understood: a cycle which contains both an edge, say $[(a, \alpha); (v_1, \nu_1)]$ and $[(v_1, \nu_1); (e, \eta)]$, does not represent actual dependencies in the extended tree as can be seen in Figure 4, since only v'_1 is connected to a and only v_1 is connected to e . The third criterion serves to deal with more complex examples, where copies of the edge $[(v_1, \nu_1); (e, \eta)]$ would occur in the extended tree as well and could be connected to a copy of $[(a, \alpha); (v_1, \nu_1)]$ by going through

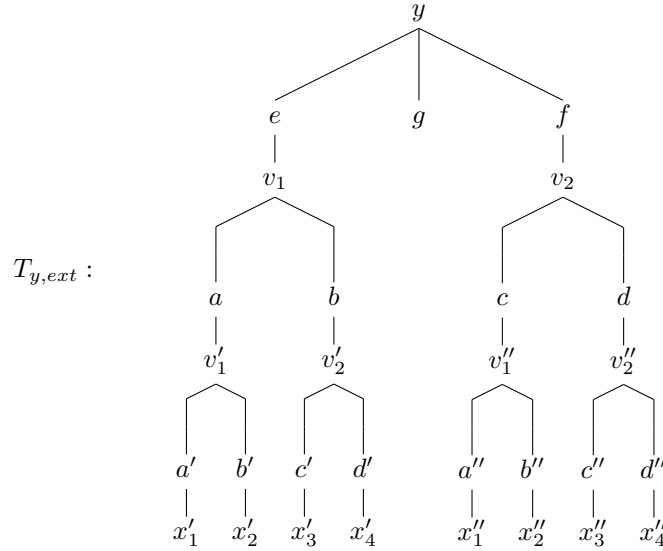


Figure 4: The situation we want to simulate.

another edge that was created by the closure operation.

To find out which choices of y -trees and y -graphs are suitable, we introduce a modified version of cycle-freeness, which will be important to establish when we can stop expanding on copies of a node y .

Definition 29. A graph G is *cycle-free* if either $G = \emptyset$, or if the only cycles in G containing a positive defined atom are harmless. For a node $y \in T$, if there exists an y -tree T_y for which there exists an y -graph G_y that is cycle-free, we will call y and T_y cycle-free as well.

We define a set $safe(T)$, which initially is empty. Whenever we encounter a cycle-free node y , and a corresponding cycle-free y -tree and y -graph G_y , we add S_y to $safe(T)$. We call every pair $(v, x) \in safe(T)$ *safe*, x is *safely blocked* and v is *safely blocking* x . The following conditions decide when the expansion of a node is terminated:

- Whenever we encounter a new blocking pair (y, x) , i.e. there is no other branch yet containing two copies of y , we choose a number n_y .¹⁹

¹⁹This number is limited by a maximum value $n_{y,MAX}$, which can be calculated using characteristics of the theory T . As we noted earlier, at this point we have failed to prove the completeness of our procedure. The obstacle in our proof is precisely that we have not yet found a method for effectively determining this value. However, in practice, small values of n_y will be sufficient for most theories.

- If a node y is blocking n_y nodes, then we impose that no more expansions can be performed on T . We will say that T is *failed*, and the algorithm terminates.²⁰
- If a node x occurs as a last-blocked node in a cycle-free y -tree T_y , we impose that no more expansions can be performed on x as long as y remains cycle-free. Note that this ensures the uniqueness of the cycle-free y -tree and y -graph G_y .²¹

At a certain point no further expansion will be possible,²² we then adapt the structure for each safely blocked node x by the following *delete rule*:

- $T = T \setminus \{z \mid z \in T, x < z\}$;
- $V = V \setminus \{(z, \psi) \mid z \in T, \psi \in S_\varphi^K, x < z\}$;
- $E = E \setminus \{[(z, \psi_1); (z', \psi_2)] \mid z, z' \in T, \psi_1, \psi_2 \in S_\varphi^K, x < z \vee x < z'\}$.

In other words, we delete all references to nodes younger than the safely blocked nodes.

In addition to this, we transform the directed graph G by creating the closure of G by $safe(T)$. We will refer to this operation as the *closing rule*, and after it has been applied we will refer to the closure of G by $safe(T)$ simply as G . Also, since for every $y \in T$ there is exactly one cycle-free y -graph G_y and corresponding T_y , from now on when we speak of the G_y for a cycle-free node y , we are talking about the unique cycle-free y -graph.

The effect of this transformation will be that all references to the safely blocked nodes are deleted, and the edges to nodes (nodes from V , which are couples made up of a node and a formula)²³ containing it are replaced by edges to the associated nodes containing its safely blocking node. It's important to note that, because $safe(T) = \bigcup_{y \in T} G_y$, the closing rule makes sure that all edges which occur in a cycle-free G_y also end up in G , and that vice versa all edges in G added by the closing rule appear as well in some G_y .

²⁰This is why the breadth-first requirement is important. If we went depth-first, it could happen that no n_y is large enough, since the cycle preventing cycle-freeness occurs due to another branch which we never get to.

²¹There may be multiple choices within T_y to create a cycle-free G_y , but for only one of them the set S_y is added to $safe(T)$.

²²Even if there remain last-blocked nodes that aren't blocked by a cycle-free node, eventually there will be a node y blocking n_y nodes, since there are only a finite amount of different nodes.

²³Nodes can be elements of T , but the graph G also contains nodes, which are of the form (x, ψ) . The context will make clear which kind of nodes are meant.

Definition 30. A *complete* K -completion structure for φ is a non-failed completion structure that results from applying the expansion rules to the initial K -completion structure for φ , taking into account the applicability rules, the delete rule and the closing rule such that no rules can be further applied.

The graph G is meant to show the dependency relations between formulas. An edge between pairs (x, ψ) and (y, ϕ) , can be interpreted as saying that the truth of the occurrence of ψ in x depends on the truth of the occurrence of ϕ in y . A set $C(x) \subset K$ occurring in T can correspond to an infinite number of “similar” domain elements in a model for which G represents the dependency graph, so any cycle in G from a node $(x, P(c))$ - with $P \in Def(\Delta)$ - to itself can be interpreted as stating that the truth of every $P(d)$ depends on the truth of a $P(e)$, where d and e are domain elements corresponding to c in the manner explained on page 20. Such dependencies pose a problem for the satisfiability of φ , since they are to be interpreted as stating that there is a positive defined atom that depends on an infinite amount of other positive defined atoms for which the same holds. Just as the automaton \mathcal{A}_φ^K doesn't allow an infinite play to pass a state for a positive defined atom an infinite amount of times, the graph G of a completion structure needs to be cycle-free.

Putting all this together, we say a completion structure T is *clash-free* if (1) T is complete, (2) T is not contradictory, and (3) G is cycle-free.

5.1.2 An example of the reasoning procedure at work.

To illustrate the working of the algorithm, we now give a simple example. The example has an additional value, in that it also illustrates the difference in semantics between FO(ID) and OASP. The corresponding theory for OASP was discussed on page 7.

Consider the theory given by:

$$\begin{cases} \forall x : P(x) \leftarrow \neg Q(x) \\ \forall x : Q(x) \leftarrow \neg P(x) \end{cases}$$

$$\varphi = \exists x : P(x) \wedge \top$$

In FO(ID) this theory is unsatisfiable, from which we conclude that $\forall x : \neg P(x)$ is entailed by every model, i.e. the extension of P has to be empty in



Figure 5: The tree T .

$$(\varepsilon, \exists x : P(x)) \Longrightarrow (1, \exists x : P(x))$$

Figure 6: The dependency graph $G, 1$.

every model.²⁴ An intuitive argument for this is that every $P(c)$ requires the negation of another defined predicate, $Q(c)$, which in turn requires $P(c)$ to be true. Therefore the truth of $P(c)$ depends on itself. But the semantics of FO(ID) dictate that the body $\varphi_P(c)$ becomes true before $P(c)$, or in the terminology of page 19, $|P(c)| \leq \beta \Rightarrow |P(c)| > |\varphi_P|$, where β stands for the limit of a well-founded induction $(\nu_\alpha)_{\alpha \leq \beta}$ for Δ . This leads to the conditional $|P(c)| \leq \beta \Rightarrow |P(c)| > |\neg Q(c)| \geq |P(c)|$, from which it follows that $|P(c)| > \beta$, i.e. $|\neg P(c)| < \beta$.

Now let's have a look at how our reasoning procedure tackles this example. We start out with the initial completion structure $T = \{\varepsilon\}$, $V = \{(\varepsilon, \varphi)\}$, $E = \emptyset$, $C(\varepsilon) = \emptyset$, $CT(\varepsilon) = \{\varphi\}$, $ST(\varepsilon, \varphi) = \text{unexp}$ and the other labeling functions undefined. The first step is expanding (ε, φ) . This creates a successor of $\varepsilon, 1$. The tree T is depicted in Figure 5.

Further, we choose a constant c , we change the status of (ε, φ) to *exp*, and update the exist function to $EX(\varepsilon, \exists x : P(x)) = (1, c)$. At the same time we fill in the labels for 1: $C(1) = \{c\}$, $CT(1) = \{\exists x : P(x); P(c)\}$, $ST(1, \exists x : P(x)) = \text{unexp}$ and $ST(1, P(c)) = \text{unexp}$. The update operation changes G to Figure 6.

Now ε is saturated, so we may expand $(1, \exists x : P(x))$. This means we set $EX(1, \exists x : P(x)) = (1, c)$, change the status to *expand*, $ST(1, \exists x : P(x)) = \text{exp}$, and update G , which gives Figure 7.

This leaves $(1, P(c))$ to expand. We add $\varphi_{P(c)}$ to the content, $CT(1) = \{\exists x : P(x); P(c); \neg Q(c)\}$, set $ST(1, P(c)) = \text{exp}$ and $ST(1, \neg Q(c)) = \text{unexp}$. Also, we

²⁴We have to insert a tautology (\top) in φ because of the syntactical rules for Guarded FO(ID), but it obviously has no effect.

$$(\varepsilon, \exists x : P(x)) \Longrightarrow (1, \exists x : P(x)) \longrightarrow (1, P(c))$$

Figure 7: The dependency graph G , 2.

$$(\varepsilon, \exists P(x)) \Longrightarrow (1, \exists P(x)) \longrightarrow (1, P(c)) \longrightarrow (1, \neg Q(c))$$

Figure 8: The dependency graph G , 3.

update G , leading to Figure 8.

Finally, we expand $(1, \neg Q(c))$. There's no need to add $\neg\varphi_{Q(c)}$ to $CT(1)$, because it's already present as $P(c)$. So the only action to take is to update the graph and set $ST(1, \neg Q(c)) = exp$. This results in the dependency graph in Figure 9.

There are no more unexpanded formulas left, and the choice rule cannot be applied either, so 1 is saturated as well. This means that the algorithm terminates. Remark that we didn't make any choices, so this application of the algorithm was completely deterministic. However, we see that G isn't cycle-free, so we conclude that the theory is unsatisfiable.


$$(\varepsilon, \exists P(x)) \Longrightarrow (1, \exists P(x)) \longrightarrow (1, P(c)) \longrightarrow (1, \neg Q(c))$$


Figure 9: The dependency graph G , 4.

6 Satisfiability and Completion Structures

6.1 Introduction

We will now establish a connection between clash-free completion structures for φ and models of φ . If a complete completion structure is not contradictory, each pair $(CT(x), C(x))$ is a $(\varphi, C(x))$ -type. Therefore, when we ignore the other labeling functions and the graph G , a non-contradictory complete K -completion structure for φ can be seen as a finite K -tableau for φ , as defined on page 17. This will allow us to create a link between completion structures and tableaux.

We proceed to show three results, of which the third still remains unfinished, which together constitute a correspondence between models of φ and clash-free completion structures for φ :

1. *Termination*: an initial K -completion structure for φ can always be expanded to a complete completion structure;
2. *Soundness*: if there is a clash-free K -completion structure for φ , φ is satisfiable;
3. *Completeness*: if φ is satisfiable there is a clash-free K -completion structure for φ .

6.2 Termination

Theorem 31. *Termination: One can always construct a finite complete K -completion structure for φ by a finite number of applications of the expansion rules to the initial K -completion structure for φ , taking into account the applicability rules, the delete rule and the closing rule.*

Proof. Assume one has a finite completion structure which is not yet complete. If no more successors are introduced, each node can be saturated by applying the expansion rules a finite amount of times. However, the process of introducing successors will stop after a finite number of applications, since for every node y we have the restriction that if it is blocking n_y nodes we can no longer expand on it. There are only a finite amount of different nodes, and every n_y is finite. Further, the arity of the tree is limited by the maximum number of different (φ, K) -relevant existential formulas, thus there will be only a finite number of (finite) paths. \square

6.3 Soundness

In section 3 we constructed a 2ATA \mathcal{A}_φ^K for φ such that an accepted tableau of \mathcal{A}_φ^K can be transformed into a model which satisfies φ , and it was shown that every model of $\{\Delta, \varphi\}$ can be transformed into an accepted tableau of \mathcal{A}_φ^K . Our claim is that if there exists a model of $\{\Delta, \varphi\}$ we can construct a clash-free completion structure, and that, vice versa, a clash-free completion structure can be extended into an accepted tableau and thus into a model. So the existing results for guarded FO(ID) together with the proofs presented here, suffice to derive the soundness and completeness of the reasoning procedure.²⁵

First we need to transform the completion structure into its extended tableau.

Definition 32. For a complete completion structure T we define its extended tableau T_{ext} , as a K -tableau for φ as introduced on page 17 which is the result of the following series of operations.

Given a set of n symbols Σ , where n is the number of nodes in T , define a one-to-one labeling function on T (and with it a labeled tree): $t : T \rightarrow \Sigma$ which assigns to every node of T a symbol from Σ .

In case $blocked(T) = \emptyset$, $T_{ext} = T$. Otherwise, repeat the following an infinite number times: for every $(y, x) \in safe(T)$ do $t_{t(x)}t[y]$ (every safely blocked node is replaced with the subtree starting at the safely blocking node). After each iteration new nodes (or only one) are created which will be the subject of the next transformations. The resulting tree is T_{ext} .

We observe that $\forall x \in T_{ext}, \exists! y \in T : t(x) = t(y)$, i.e., for every node in the constructed tree exactly one node in the original tree exists that has the same label; we denote such y for x as \bar{x} . Given the way T_{ext} was constructed (by concatenation of subtrees from T), either $x = \bar{x}$ or $x \in T'$, with $T' \subset T_{ext}$ being a version of a subtree of T in T_{ext} , $T[z]$, where z is a safely blocking node in T , and $\bar{x} \in T[z]$. Being a version of each other, implies that T' and $T[z]$ have the same tree structure. For $x \in T'$, $\bar{x} \in T[z]$, and $t(x) = t(\bar{x})$, one could say that x is the counterpart of \bar{x} in T' .²⁶ Considering this, we can extend the labeling functions as well, in the following manner:

- $C(x) = C(\bar{x})$,
- $CT(x) = CT(\bar{x})$,

²⁵Given of course that the completeness proof can be finished.

²⁶Note that a safely blocked node cannot have a counterpart.

- for every disjunct $\psi = \psi_1 \vee \psi_2 \in CT(x) : DJ(x, \psi) = DJ(\bar{x}, \psi)$,
- for every existential formula $\psi = \exists \mathbf{x} \gamma(\mathbf{x}) \wedge \eta(\mathbf{x}) \in CT(x) : EX(x, \psi) = (y, \mathbf{c})$, such that $EX(\bar{x}, \psi) = (\bar{y}, \mathbf{c})$.

Next we prove a theorem that allows us to move from clash-free completion structures to accepted tableaux. In this way we can rely on already established results to prove soundness.

Theorem 33. *If T is a clash-free K -completion structure for φ , its extended tableau T_{ext} will be accepted by \mathcal{A}_φ^K .*

Proof. We need to show that there exists a winning strategy for T_{ext} . We pick a strategy f as follows. (Recall that nodes v in tableaux are in general not named, and therefore denoted as a pair of a type and a set of constants. Since in T_{ext} we do have names, and labeling functions, we will abuse notation and refer to v as if we were referring to its name x . Thus, for each $v = (\Gamma, C)$, $\Gamma = CT(x)$ and $C = C(x)$.)

- For every $\psi = (v, \psi_1 \vee \psi_2)$, f selects $\psi_{DJ(v, \psi)}$.
- For every $\psi = (v, \exists \mathbf{x} \gamma(\mathbf{x}) \wedge \eta(\mathbf{x}))$, with $v = (\Gamma, C)$ and $\exists \mathbf{x} \gamma(\mathbf{x}) \wedge \eta(\mathbf{x}) \in \Gamma$,
 - if $EX(v, \psi) = (v, \mathbf{c})$ for some \mathbf{c} , f selects the move such that $f(v, \psi) = (v, \eta(\mathbf{c}))$;
 - if, on the other hand, $EX(v, \psi) = (w, \mathbf{c})$ for some \mathbf{c} , it selects the move such that $f(v, \psi) = (v, \psi)$. (By construction of T , it's always true that one of these will be the case.)

The proof consists of several stages. First we shall provide a connection between the states of a losing play p that follows our strategy f , and the labels of the nodes in T_{ext} . Secondly we reason from such a losing play towards the existence of a certain kind of sequence. Thirdly we prove a lemma which provides a connection between the transitions between states of a losing play following f , and edges in the graph G . Finally, we will use this lemma to prove that the said sequence cannot exist, which in turn shows that any play following f has to be winning.

Lemma 34. *If a losing play p that follows f reaches a state (v, ψ) , with $v = (\Gamma, C)$, then $const(\psi) \subseteq C$ and $\psi \in \Gamma$.*

Proof. We will prove this by induction. The root provides the base case. For the induction step, assume that the property holds for (v, ψ) and let $p(v, \psi) = (v', \psi')$ with $v' = (\Gamma', C')$.

- If ψ is a conjunction or an open literal, the result is trivial.
- Suppose $\psi = \psi_1 \vee \psi_2$. By the induction hypothesis, there is an $i \in \{1, 2\}$ for which $\psi_i \in \Gamma$. By the choice of our strategy, we therefore have that $\psi' = \psi_i$, (unless of course $\{\psi_1, \psi_2\} \subseteq \Gamma$, and f chooses the other disjunct), $\Gamma' = \Gamma$ and thus $\psi' \in \Gamma'$.
- Suppose $\psi = \exists x \gamma(x)$. Our choice of strategy preserves the invariant.
- Suppose ψ is a defined literal $P(\mathbf{t})$ or $\neg P(\mathbf{t})$. Then $v' = v$ and $\psi' = \varphi_{P(\mathbf{t})}$ or, respectively, the negation normal form of $\neg \varphi_{P(\mathbf{t})}$. Because Γ is a type, it follows directly from the induction hypothesis that $\psi \in \Gamma'$.
- Suppose $\psi = \forall x \gamma(x) \Rightarrow \eta(x)$. We distinguish two cases:
 - Assume $\psi' = \psi$, $v' \neq v$. If $\text{const}(\psi) \not\subseteq C'$, then the play is winning. We can therefore assume that $\text{const}(\psi) \subseteq C'$. The result then follows from the induction hypothesis and the expansion rules.
 - Assume $v = v'$ and $\psi' = \eta(\mathbf{c})$ for some \mathbf{c} such that $\gamma(\mathbf{c}) \in \Gamma$, it then follows from the induction hypothesis that $\eta(\mathbf{c}) \in \Gamma'$.

□

We now show that the play p is winning. Assume that it is losing.

If it is finite, it must either end in \mathbf{f} , or \mathbf{t} , or end with a universal formula whose constants are not part of its label. In the first case, the previous state was $(v, L(\mathbf{c}))$ for some open literal $L(\mathbf{c})$. Since the play ends in \mathbf{f} , it is losing. Thus $L(\mathbf{c}) \notin \Gamma$, where $v = (\Gamma, C)$. This contradicts the property stated above, so p cannot end in \mathbf{f} . The latter two cases are winning, thus the play is infinite.

We distinguish two cases. First, assume that some quantifier is not reduced. By definition of our strategy, each existential quantifier is reduced, so it must be a universal one, in which case the play is winning. Second, assume that some defined literals are regenerated infinitely often. If these are all negative literals, the play is winning. Assume therefore that there is some positive atom $P(\mathbf{t})$ among them. Let $(v_i, \psi_i)_{i \geq 0}$ be the infinite sequence of states reached by the play, such that (v_0, ψ_0) is some state in which $\psi_0 = P(\mathbf{t})$

and each (v_{i+1}, ψ_{i+1}) is $p(v_i, \psi_i)$. Since $P(\mathbf{t})$ is regenerated infinitely often, and there are only a finite amount of nodes in T , there exists a subsequence $(v_j, \psi_j)_{j \in [a, b]}$ of $(v_i, \psi_i)_{i \geq 0}$ such that $\psi_a = \psi_b = P(\mathbf{t})$ and $\bar{v}_a = \bar{v}_b$.

We will show that the existence of such a sequence is impossible, by using the following lemma.

Lemma 35. *For any infinite losing play p that follows f , it holds that for any two couples (v, ψ) and (v', ψ') , if $p(v, \psi) = (v', \psi')$ then $[(\bar{v}, \psi); (\bar{v}', \psi')] \in E$.*

Proof. Assume that $p(v, \psi) = (v', \psi')$, where p is a losing play that follows f . We know from the above that, abusing notation, $\text{const}(\psi) \subseteq C(v)$, $\text{const}(\psi') \subseteq C(v')$, $\psi \in CT(v)$ and $\psi' \in CT(v')$. Therefore $\text{const}(\psi) \subseteq C(\bar{v})$, $\text{const}(\psi') \subseteq C(\bar{v}')$, $\psi \in CT(\bar{v})$ and $\psi' \in CT(\bar{v}')$.

- If $\psi = \psi_1 \wedge \psi_2$, then $\psi' = \psi_1$ or $\psi' = \psi_2$. The result is trivial by looking at the expansion rule for a conjunction.
- Suppose $\psi = \psi_1 \vee \psi_2$. Because f selects $\psi_{DJ(v, \psi)}$, the result follows from the expansion rules.
- Suppose $\psi = \exists \mathbf{x} \gamma(\mathbf{x}) \wedge \eta(\mathbf{x})$. Then there are two cases to be considered:
 - either $\psi' = \eta(\mathbf{c})$ for some \mathbf{c} , and $v' = v$,
 - or $\psi' = \psi$ and $v' \neq v$.

In both cases the corresponding choices made by f and the expansion rules imply the result.

- Suppose ψ is a defined literal $P(\mathbf{t})$ or $\neg P(\mathbf{t})$. Then $v' = v$ and $\psi' = \varphi_{P(\mathbf{t})}$ or, respectively, the negation normal form of $\neg \varphi_{P(\mathbf{t})}$. The result follows directly by looking at the expansion rules.
- Suppose that $\psi = \forall \mathbf{x} \gamma(\mathbf{x}) \Rightarrow \eta(\mathbf{x})$ for some γ and η . The result follows directly by looking at the expansion rules.

□

As a result of this lemma, we deduce that the sequence of states $(\bar{v}_j, \psi_j)_{j \in [a, b]}$ form a cycle in G containing the defined atom $P(\mathbf{t})$. Since we assumed G to be cycle-free, this has to be a harmless cycle. We show that this is impossible.

Consider the edge $[(\bar{c}, \gamma); (\bar{d}, \delta_1)]$ in this cycle which was created by an application of the closing rule, i.e. the closure of G by $\text{safe}(T)$. This means \bar{d} is

safely blocking the successor of \bar{c} in T , which implies that $\bar{c} \not\prec \bar{d}$. To this corresponds a move $p(c, \gamma) = (d, \delta_1)$ of the play, where d is the successor of c in T_{ext} which was created by adding a copy of $t[\bar{d}]$ in place of the original successor of c . (Thus $d \neq \bar{d}$.)

Now consider the edge mentioned in the second criterion of harmlessness, $[(\bar{d}, \delta_2); (\bar{e}, \eta)]$ where $\bar{e} < \bar{d}$. To this corresponds a move $p(d', \delta_2) = (e, \eta)$, where e is the predecessor of d' . By construction, the play can only leave the current copy of $t[\bar{d}]$ by following an edge that was the result of the closing rule. (This is precisely why the closing rule was introduced: to simulate edges between different subtrees that are copies of subtrees of T .)²⁷ Thus the third criterion for harmlessness dictates that d' is part of the copy of $t[\bar{d}]$ that took the place of the original successor of c . Since the labeling function t is one-to-one, $\bar{d}' = \bar{d}$ implies that $d' = d$. The predecessor of d is c , so $c = e$. But $\bar{e} < \bar{d}$ and $\bar{c} \not\prec \bar{d}$, so we obtain a contradiction.

We therefore conclude that such a sequence cannot occur, and each defined atom $P(\mathbf{t})$ can occur at most a finite number of times in p , thus establishing that p is winning. □

Corollary 36. *Soundness: If there exists a clash-free K -completion structure for φ , then φ is satisfiable.*

Proof. Theorem 33, together with theorem 12 implies the result. □

6.4 Completeness

On the other hand we can transform a model for φ into a clash-free K -completion structure for φ .

Conjecture 37. *Completeness: If φ is satisfiable there is a clash-free K -completion structure for φ .*

Proof. Let $\mathcal{S} \models \varphi$. We fix a well-founded induction $(\nu_\alpha)_{\alpha < \beta}$ in $\mathcal{S} \upharpoonright_{Op(\Delta)}$. We construct a clash-free K -completion structure T and, for each node x in T , a valuation χ_x for $C(x)$, while preserving the invariant that $\mathcal{S}, \chi_x \models CT(x)$:

- The root is the initial completion structure for φ .

²⁷To be entirely accurate: at least one of the subtrees is a copy of a subtree of T .

- We apply the algorithm for expanding a completion structure, where we take into account the following:
 - if $\psi = \psi_1 \vee \psi_2$, then choose an $i \in \{1, 2\}$ such that $|\psi_{\chi_x, i}| = \min\{|\psi_{\chi_x, 1}|, |\psi_{\chi_x, 2}|\}$;
 - if $\psi = \exists \mathbf{x} \gamma(\mathbf{x}) \wedge \eta(\mathbf{x})$, then consider the non-empty set of tuples $\mathbf{d} \subseteq \text{dom}(\mathcal{S})$ for which $\mathcal{S}, \chi_x \models \gamma[\mathbf{d}] \wedge \eta[\mathbf{d}]$, and select from this a tuple \mathbf{d} with minimal $|\gamma_{\chi_x}(\mathbf{d}) \wedge \eta_{\chi_x}(\mathbf{d})|$.
 - * add a new node $y \in \text{succ}_T(x)$ to T ;
 - * select a tuple $\mathbf{c} \subset K \setminus C_0$ of fresh constants;
 - * the valuation χ_y coincides with χ_x on $\text{const}(\psi) \cup C_0$ and maps \mathbf{c} to \mathbf{d} . This clearly preserves the invariant.
 - when applying the choice rule, always add literals $l(\mathbf{c})$ so that $\mathcal{S}, \chi_x \models l(\mathbf{c})$.
 - whenever we encounter a new blocking pair (y, x) , we provisionally choose $n_y = 2$.
 - whenever we run into an unsafe pair (y, x) , we provisionally set $n_y = n_y + 1$.

It is clear that our structure is not failed by construction. It is also clear that it is non-contradictory. The structure can be completed if every n_y is finite, so what remains to be shown is that n_y is finite for every y ²⁸ and T is cycle-free. We have as yet not found a way to determine a maximum value for n_y , so this part of the proof will remain unsettled. We do present a method that shows how one could go about finding such a maximum. More specifically, we consider the case where T consists of one branch and suggest how this method could be generalized. Lastly, we will prove that G is cycle-free.

Recall from page 32 that there is no cycle in G containing a positive defined atom which doesn't contain an edge that exists due to the closing rule.²⁹ This result can easily be extended to a v -graph G_v for some node v : there is no cycle in G_v containing a positive defined atom which doesn't contain an edge that exists due to the closure of G'_v by S_v . This holds because every such cycle appearing in G_v would consist entirely of edges in E that weren't the result of

²⁸for which a n_y is defined, of course.

²⁹This result didn't depend on the applicability rules being simple or not, so it holds in this context as well.

the closing rule. From this we learn that, if a node v is not cycle-free, there has to be a non-harmless cycle containing a positive defined atom in every v -graph G_v for every v -tree T_v , that contains an edge which was added by taking the closure of G'_v by S_v .

From the expansion rules it's clear that only universal and existential formulas can occur in edges between different nodes of T , and it's a consequence of the previous argument that a cycle preventing cycle-freeness must visit more than one node. Therefore such a cycle contains a pair (x, D) where D is a universal or existential formula for each node x that it passes. We will focus on these pairs.

Assume there is a node $x \in T$ for which n_x is infinite.³⁰ We attempt to prove that this is impossible.

For reasons of simplicity, let's assume that there are only two existential or universal formulas in $CT(x)$, A and B . The argument can be generalized to any number of such formulas.

We consider a very restricted case only, namely when T consists of only one branch. The node x is not cycle-free if for every x -tree T_x there exists no x -graph which is cycle-free, i.e. which is either empty or for which all cycles containing a positive defined atom are harmless. All x -trees are given by all finite branches starting at x and ending in some copy z of x , so $lastblocked(T_x) = \{(x, z)\}$. For a given x -tree T_x , the possible sets S_x for an x -graph G_x are all of the form $S_x = \{(y, z)\}$, where $y = x$ or $(x, y) \in blocked(T)$ and $y < z$. Every such x -graph must contain a non-harmless cycle containing a positive defined predicate that contains an edge which was added by taking the closure of G'_x by S_x , i.e. an edge $[(v, \alpha); (y, \beta)]$ such that v is the predecessor of z (or an edge $[(y, \beta); (v, \alpha)]$), where β is either A or B . The rest of the cycle thus consists of a path $(y, \beta) \mapsto (v, \alpha)$. There are only six possible forms this path can take, as can be seen from Figure 10.

Now because this has to hold for every x -tree and x -graph, we know that in between every two copies y and z of x where $y < z$, one of the following six patterns must occur:

1. There is a path containing a positive defined predicate $(y, A) \mapsto (z, A)$.
2. There is a path containing a positive defined predicate $(y, B) \mapsto (z, B)$.
3. There is a path containing a positive defined predicate $(y, A) \mapsto (z, B)$,

³⁰Without loss of generality we may take x to be first copy of x in T .

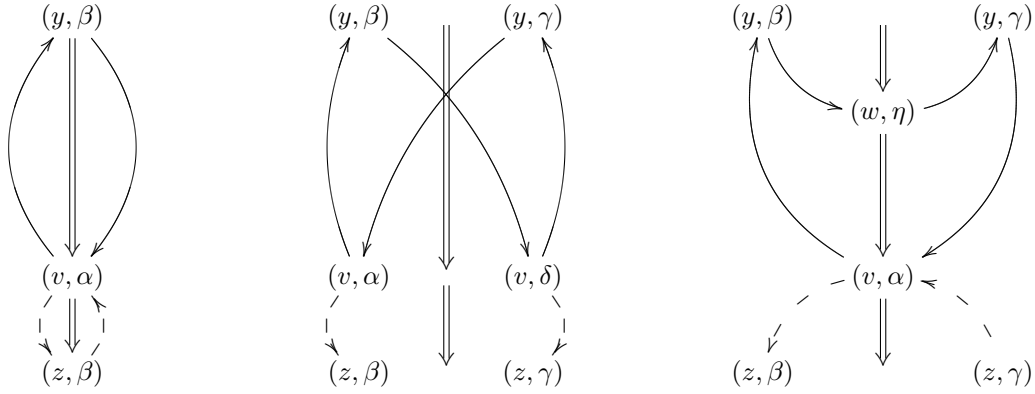


Figure 10: Possible cycles. The dashed lines are edges before the closing rule, the full lines are edges after the closing rule. β and γ are either A or B. The left tree represents patterns 1 and 2, the middle tree represents 3 and 4, and 5 and 6 are shown in the right tree.

and a path $(y, B) \mapsto (z, A)$.

4. There is a path containing a positive defined predicate $(y, B) \mapsto (z, A)$, and a path $(y, A) \mapsto (z, B)$.
5. There is a path containing a positive defined predicate $(y, A) \mapsto (y, B)$, and a path $(z, B) \mapsto (z, A)$.
6. There is a path $(y, A) \mapsto (z, B)$, and a path containing a positive defined predicate $(z, B) \mapsto (z, A)$.

We will attempt to prove that an infinite branch T containing an infinite amount of copies of x that satisfies this characterisation must contain a path that contains an infinite amount of positive defined predicates. By Proposition 24 on page 32 and the argument following it, this implies the existence of an infinite strictly descending sequence of ordinals, which is impossible. Here we have made yet another simplification, namely that we consider only those paths just mentioned, going from the couple on the left to the couple on the right, and not their symmetric counterparts. For example, besides the first pattern, theoretically it could also be the case that there is a path containing a positive defined predicate $(z, A) \mapsto (y, A)$, although it's difficult to imagine any actual scenario where a model would have this property.

We will represent the situation by building up a typed first order theory which describes the most important aspects. If this theory turns out unsatisfiable, we

have reached our goal. The branch T can be seen as the set of integers, where each number represents the next copy of x , so 3 represents x_3 , the fourth copy of x in T . These integers each have two elements A and B . To represent paths between elements of nodes we use a five place predicate P , which also indicates the number of positively defined predicates that appear in this path. For example, $P(1, A, 2, B, 1)$ states that there is a path $(x_1, A) \mapsto (x_2, B)$ that contains one positive defined predicate.

The first constraint restricts the search space a little.

$$\forall x, y, e, f, m, n : P(x, e, y, f, m) \wedge n \neq m \Rightarrow \neg P(x, e, y, f, n)$$

The second expresses that there is a maximum number of positively defined predicates that any path may contain. It is this proposition which we want to falsify.

$$\exists m \in \mathbb{N} : \forall x, y, e, f, n : P(x, e, y, f, n) \Rightarrow n \leq m$$

Next we express that P is “additively” transitive.

$$\forall x, y, z, e, f, g, m, n : P(x, e, y, f, m) \wedge P(y, f, z, g, n) \Rightarrow P(x, e, z, g, n + m)$$

Likewise, every path must be the union of two shorter paths, i.e. every path must enter each node it passes.³¹

$$\forall x, y, e, f, m : P(x, e, y, f, m) \Rightarrow (\exists g, n : n \leq m \wedge P(x, e, y-1, g, n) \wedge P(y-1, g, y, f, m-n))$$

Then a formula that prohibits cycles with positive defined predicates.

$$\forall x, e, n : n > 0 \Rightarrow \neg P(x, e, x, e, n)$$

And lastly we add the constrain that all couples must be connected in one of the six ways described.

$$\forall x, y : x < y \Rightarrow (\exists e, f : (P(x, e, y, f, 1) \wedge P(x, f, y, e, 0)) \vee$$

³¹The given formula implies this.

$$(P(x, A, x, B, 1) \wedge P(y, B, y, A, 0)) \vee (P(x, A, x, B, 0) \wedge P(y, B, y, A, 1))$$

We used the IDP-system³² to try and prove for finite trees that the maximum number of positive defined predicates in a path increases with the length of the tree, but this only proved succesful up until a tree containing 16 copies of x , for which it was proven that every model will have a path with at least 4 positive defined predicates. For larger trees, the propositional SAT-solver that lies behind the system ran out of memory. To prove the theory unsatisfiable in the infinite case would require a theorem-prover capable of inductive proofs.

We will now prove that G is cycle-free. (Assuming that the former proof is completed, of course.) We have shown earlier already that there is no cycle in G containing a positive defined atom which doesn't contain an edge that exists due to the closing rule, therefore assume there is a cycle which isn't harmless in G containing a positive defined atom $P(\mathbf{c})$ such that the cycle contains an edge that was created by an application of the closing rule. Thus there is a sequence of connected nodes $(m_i, \psi_i)_{i \in [a, b]}$ with $\psi_a = \psi_b = P(\mathbf{c})$ and $m_a = m_b$. For some $[(x, \phi); (y, \psi)]$ occurring in $(m_i, \psi_i)_{i \in [a, b]}$, it holds that $\exists z : (y, z) \in \text{safe}(T)$ and $[(x, \phi); (y, \psi)]$ came into existence - replacing $[(x, \phi); (z, \psi)]$ - by an application of the closing rule. If y is not an ancestor node of x , then by construction there must be a cycle-free node v , a v -tree T_v , a v -graph G_v and an ancestor w of x such that $(v, y), (v, w) \in \text{blocked}(T)$,³³ and $[(x, \phi); (y, \psi)] \in G_v$. The same holds for any other edge $[(p, \phi); (r, \psi)]$ of the cycle that was created by the closing rule and for which $(v, r) \in \text{blocked}(T)$ or $v = r$, since all safe pairs with copies of v come from the same S_v .

First assume that all edges that resulted from the closing rule in the cycle are of this form. Since the cycle is not harmless, and the first condition for harmlessness is already fulfilled, it doesn't contain any node a for which $a < v$. If it did, then for the edge which resulted from the closing rule that occurs nearest in the cycle before the edge going to a , the second and third criterion would also have to be fulfilled.³⁴ But this means that the cycle occurs in G_v as

³²This was developed by the Knowledge Representation and Reasoning research group at the Catholic University of Leuven. It is a finite model-generator for FO(ID), see <http://dtai.cs.kuleuven.be/krr/software/idp> for details.

³³Or $v = w$, or $w = x$, but not both.

³⁴Although the harmless criterion does allow there to be edges $[(a, \alpha); (v, \psi)]$ with $a < v$, such an edge implies the existence of an edge which is explicitly forbidden.

well, contradicting that v is cycle-free.

Now assume there is another edge $[(s, \phi); (t, \psi)]$ in the cycle that resulted from the closing rule, but $(v, t) \notin \text{blocked}(T)$ and $v \neq r$. If $(t, v) \in \text{blocked}(T)$, then we can repeat the previous argument by replacing v with t . Therefore assume that $(t, u) \in \text{safe}(T)$ for some u which isn't a copy of v , i.e. $(u, v) \notin \text{blocked}(T) \wedge (v, u) \notin \text{blocked}(T) \wedge v \neq u$. As in the previous situation, there is a cycle-free node d such that $(d, u) \in \text{blocked}(T)$ (or $d = u$) and $[(s, \phi); (t, \psi)] \in G_d$. Similarly, since the cycle is not harmless, it doesn't contain any node e for which $e < d$. Since there must a cycle-free node f such that the whole cycle occurs in a f -tree T_f , it follows that $f = d = v$. This contradicts that $(d, u) \in \text{blocked}(T)$. Thus we have established that G is cycle-free, and hence that T is clash-free. \square

7 Examples

7.1 Why one blocked node isn't enough

Our reasoning procedure has quite complex conditions with regard to terminating the expansion of nodes, requiring for each node x of which copies keep showing up in T that we check every possible x -tree and every possible x -graph for cycle-freeness. A superficial argument suggests that the simple applicability rules should be sufficient, it goes as follows.

Assume there is a branch in $T[x]$ which prevents x from being cycle-free because there is a copy of x on it and in between those there is a problematic path containing a positive defined atom. Either the same holds for every copy of x and the corresponding branch, so that no matter how many copies we add x will never be cycle-free; or there is a copy x' for which the same branch can be expanded without this problem arising. In the first case it's obviously useless to keep expanding T and hoping for the best, so the complex applicability rules are of no avail. In the second case one could just as well have expanded the original branch starting in x in the same way as one did in x' , since they are copies of one another. Therefore we could simplify the applicability rules to stating that no expansions may occur on a blocked node.

To prove that this approach is too simplistic, and that it is very well possible that one may need several copies of x in order to see how a problematic branch may become unproblematic in the light of a larger tree, we present an example of a satisfiable theory which would be dismissed as unsatisfiable by the simple approach. The definition is given by:

$$\left\{ \begin{array}{l} \forall x, y : P(x, y) \leftarrow M(x, y) \wedge [\forall v : T(y, v) \Rightarrow [\forall w : V(v, w) \Rightarrow (\forall l : M(w, l) \Rightarrow P(w, l))] \\ \forall x, y : P'(x, y) \leftarrow M'(x, y) \wedge [\forall v : T(y, v) \Rightarrow [\forall w : V(v, w) \Rightarrow (\forall l : M'(w, l) \Rightarrow P'(w, l))] \\ \forall x, y : Q(x, y) \leftarrow M(x, y) \vee [\exists v : T(y, v) \wedge [\exists w : V(v, w) \wedge (\exists l : M(w, l) \wedge \neg P(w, l))] \\ \forall x, y : Q'(x, y) \leftarrow M'(x, y) \vee [\exists v : T(y, v) \wedge [\exists w : V(v, w) \wedge (\exists l : M'(w, l) \wedge \neg P'(w, l))] \end{array} \right.$$

There are four defined predicates, which can be grouped into pairs in two ways: P and P' (and Q and Q') have an identical structure, and P and $\neg Q$ (and P' and $\neg Q'$) have the same universal formula in their body. Their meaning is as follows: we will create an infinite branch T with an infinite number of copies of a node x , x_0, x_1, x_2, \dots . In between each path $x_i \mapsto x_{i+1}$, there will

be either a copy of a node y or a copy of a node z . These are constructed so that $\{P(1, 2); M(1, 2); Q(1, 2); \neg Q'(1, 2); \neg M'(1, 2); \neg P'(1, 2)\} \subset CT(y)$ and $\{P'(1, 2); M'(1, 2); Q'(1, 2); \neg Q(1, 2); \neg M(1, 2); \neg P(1, 2)\} \subset CT(z)$.

We will look more closely at $CT(y)$, the situation is symmetric for z . The predicate M is open, so it doesn't depend on anything. $P(1, 2) \in CT(y)$ because of $M(1, 2)$ and the presence of the universal formula in $\varphi_P(1, 2)$ in $CT(y)$. $Q(1, 2) \in CT(y)$ because of $M(1, 2)$, the presence of $\neg P'(1, 2)$ in $CT(y)$ is justified by the absence of $M'(1, 2)$, and $\neg Q'(1, 2) \in CT(y)$ because of the absence of $M'(1, 2)$ and the presence of the universal formula in $\neg\varphi_{Q'}(1, 2)$, which is the same formula as in the body of $P'(1, 2)$. This last fact is what allows this universal formula to be present in both $CT(y)$ and in $CT(z)$, the one time because of the absence of Q' , the other because of the presence of P' . It's precisely this formula which will be responsible for there being an infinite amount of copies of x . Note that only the presence of $P(1, 2)$ is relevant for the cycle-freeness of G .

The formula φ to be satisfied is a conjunction of formulas that have different purposes. The first three are given by

$$\varphi_1 = \forall x, y : S(x, y) \Rightarrow \exists z : T(y, z) \wedge \top$$

$$\varphi_2 = \forall x, y : T(x, y) \Rightarrow \exists z : V(y, z) \wedge \top$$

$$\varphi_3 = \forall x, y : V(x, y) \Rightarrow \exists z : S(y, z) \wedge \top$$

These universal formulas are there simply to create an infinite domain and an infinite branch in T . This could be done with only two formulas of this kind, (and thus only two predicates), but we added a third to make sure that we end up in a node x_{i+1} with the exact same labels as an older node x_i , including the constants $C(x_i)$. If we only used two formulas, $x_i \sim_c x_{i+1}$, for some $c \in C(x_i)$, so that it would refer to the same domain element in both nodes, which isn't the intention.

$$\varphi_4 = \forall x, y : S(x, y) \Rightarrow (M(x, y) \wedge \neg Q'(x, y)) \vee (M'(x, y) \wedge \neg Q(x, y))$$

This formula makes sure that one ends up either in a copy of y or in a

copy of z . A smart expansion, i.e. one making φ satisfiable, is one that keeps switching between the two.

$$\varphi_5 = \forall x, y : S(x, y) \Rightarrow (\neg S(y, x) \wedge x \neq y \wedge \neg T(y, x))$$

$$\varphi_6 = \forall x, y : T(x, y) \Rightarrow (\neg T(y, x) \wedge x \neq y \wedge \neg V(y, x))$$

$$\varphi_7 = \forall x, y : V(x, y) \Rightarrow (\neg V(y, x) \wedge x \neq y \wedge \neg S(y, x))$$

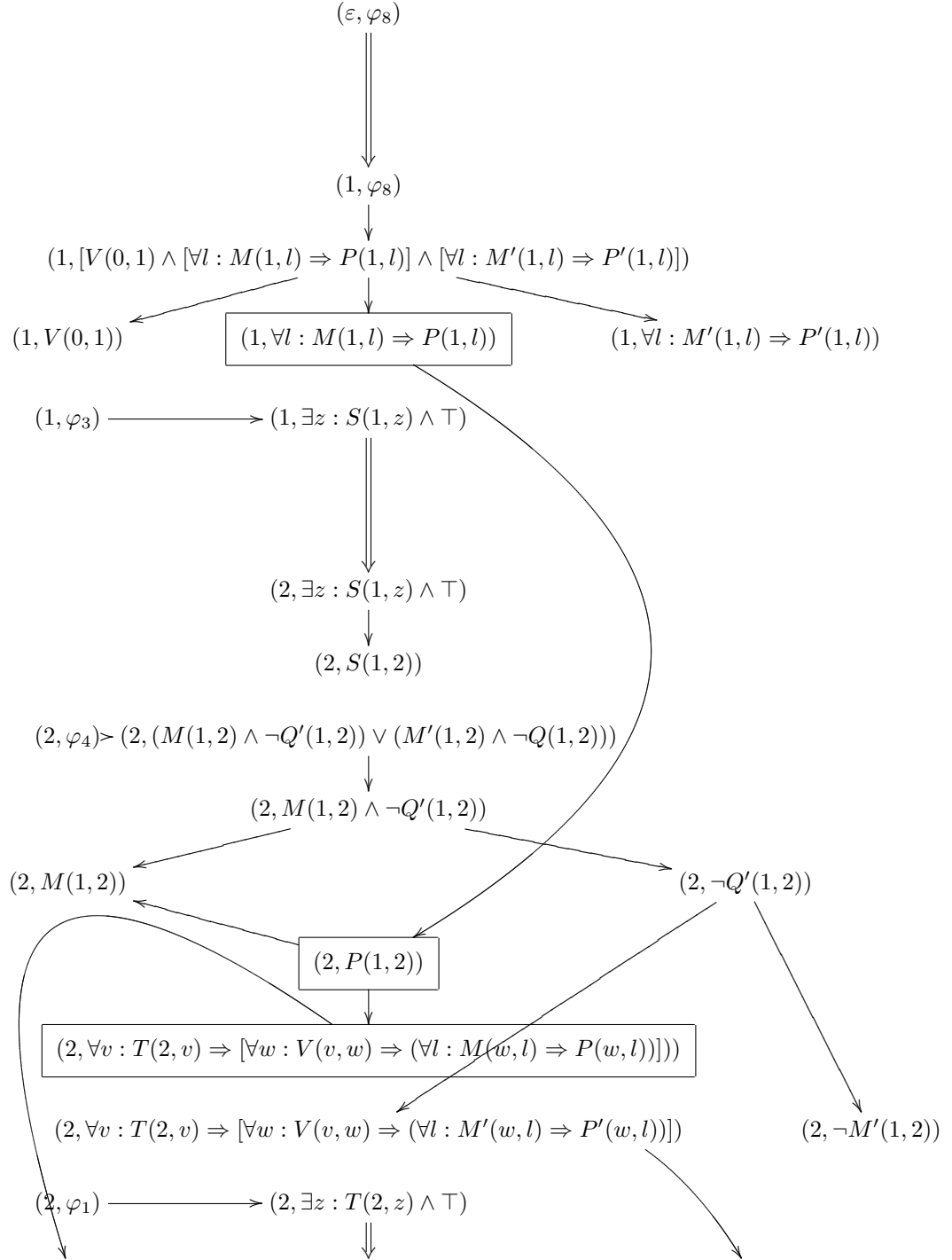
These formulas are there simply to provide some constraints, so that there isn't an easy escape route. The next formula serves to start things off in the first node, which strictly speaking isn't part of the sequence $(x_i)_{i \geq 0}$, but resembles x closely.

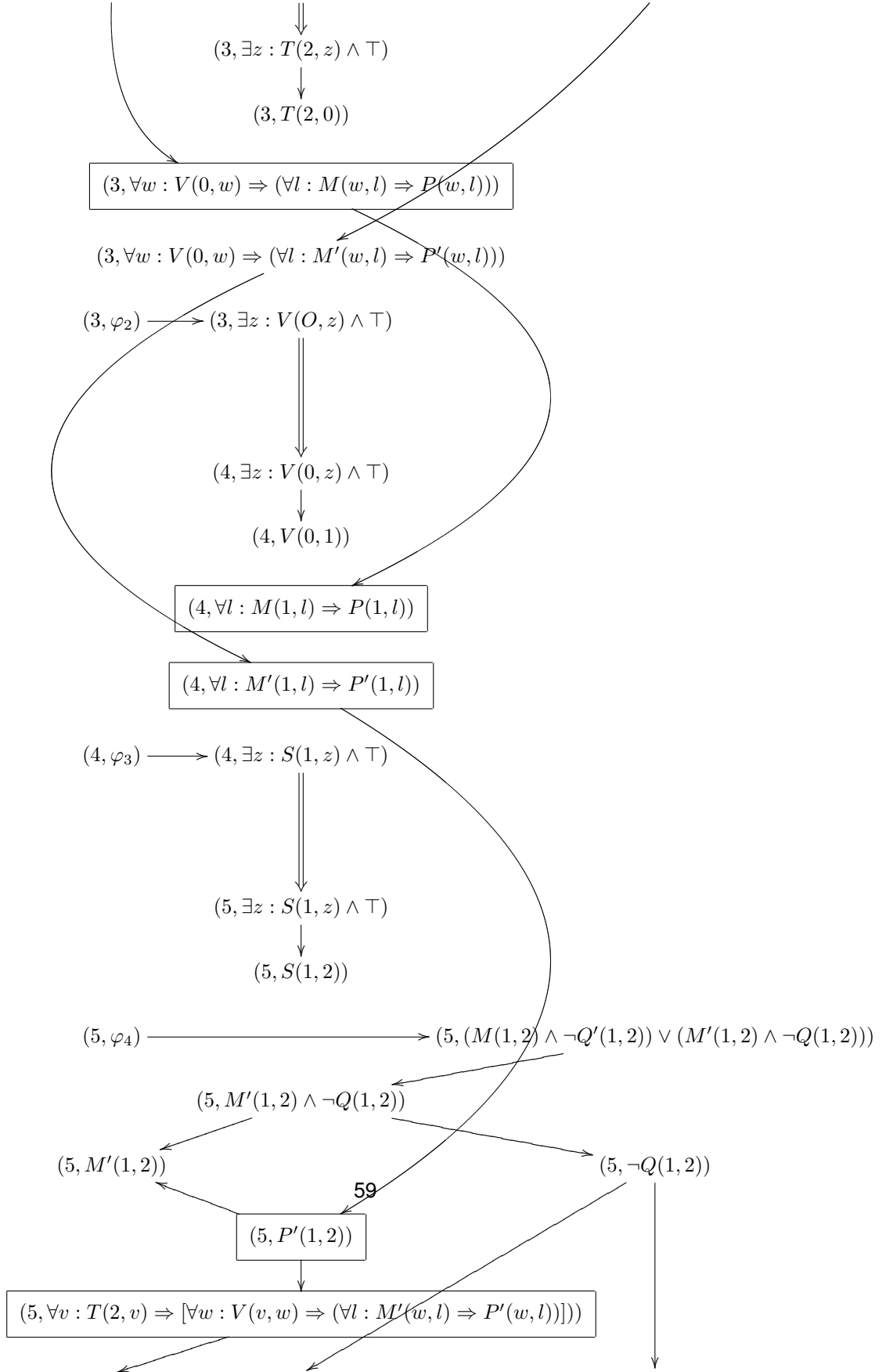
$$\varphi_8 = \exists x, y : [V(x, y) \wedge [\forall l : M(y, l) \Rightarrow P(y, l)] \wedge [\forall l : M'(y, l) \Rightarrow P'(y, l)]]$$

$$\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5 \wedge \varphi_6 \wedge \varphi_7 \wedge \varphi_8$$

In Figure 9 we show a clash-free completion structure T for φ , and its dependency graph G before the closing rule is applied, where double arrows indicate that we move to a new node in T and single arrows indicate edges in G . We do not show every node of G , since this would give a very crowded figure, but restrict ourselves to the essential nodes. Also, we skip edges between nodes containing the same universal formula. The boxed nodes are the ones showing paths containing a positive defined atom, the first path containing $P(1, 2)$ and the second one containing $P'(1, 2)$. Node 7 is a copy of node 4, and node 1 can be considered a copy of these as well, since it's clear that we can expand even further so that node 10 would be a copy. Thus the sequence $(x_i)_{i \geq 0}$ corresponds to $(1 + i * 3)_{i \geq 0}$. Nodes 2 and 5 correspond to nodes y and z respectively. From 1 to 4 there's a path from a formula to itself containing the positive defined atom $P(1, 2)$. If we were to stop at 4, then clearly the closure of G wouldn't be cycle-free. From 4 to 7 there's a path from another formula (the P' version) to itself containing the positive defined atom $P'(1, 2)$. By expanding T until we have three copies of x rather than two, we have succeeded in making sure that the closure of G cycle-free, thereby showing that one cannot get

by with simpler applicability rules.





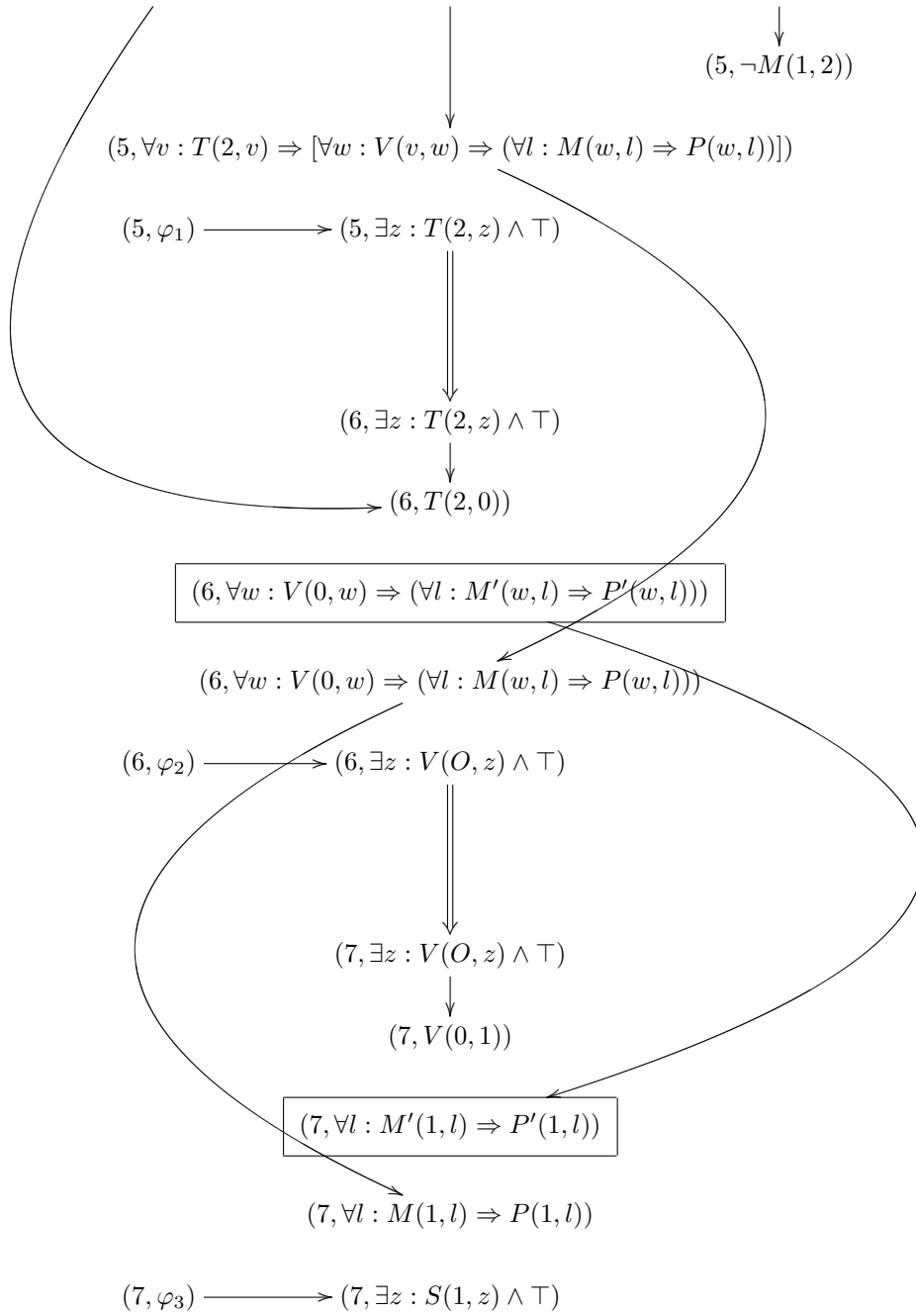


Figure 9: Example 1

7.2 Proof of Even or Uneven

We give a simple example of an application of our reasoning procedure to prove that no number can be both even and uneven. As with many other basic concepts of arithmetic, the predicates even and uneven are inductively defined, and therefore the language of FO(ID) is perfectly suited to express them. They can be defined as follows, where the binary predicate S replaces the usual successor function with a successor predicate, which appears in our theory as an open predicate.

$$\begin{cases} \forall x : E(x) \leftarrow x = 0 \vee \exists y : S(y, x) \wedge \neg E(y) \\ \forall x : U(x) \leftarrow S(0, x) \vee \exists y : S(y, x) \wedge \neg U(y) \end{cases}$$

In order to prove that no number is both even and uneven, we add the negation of this claim to our theory and try to find a model. If this fails, the claim is entailed by our theory. In order not to overcomplicate matters too much, not all Peano Axioms, which characterise the natural numbers, will be added. For example, we do not define the equality predicate, but implicitly assume that its usual properties hold. Neither do we add the Domain Closure Axiom, which was mentioned already on page 3. It states that the set of all numbers is the transitive closure of the successor relation. We will not need the axiom as such for our proof, but we do take it implicitly to hold that every strictly positive number has a predecessor, which is a weaker constraint. We do need the Unique Name Property, which states that every number has at most one predecessor, given by³⁵

$$\forall x, y : S(x, y) \Rightarrow (\forall z : S(z, y) \Rightarrow z = x)$$

Without this property it would be impossible to prove our claim, since it wouldn't hold. A simple counterexample is given by the model

$$E(a), U(a), E(0), \neg U(0), \neg E(1), U(1), S(1, a)S(0, a), S(0, 1), \neg S(0, 0), \dots$$

However it is impossible to express the Unique Name Property in Guarded FO(ID), since x appears as a free variable in the nested universal formula but not in its guard $S(z, y)$. Fortunately there is a way to circumvent this. We cannot

³⁵Remark that, since S is given by a predicate rather a function, the uniqueness of a successor for every number would require an extra axiom. We will not need this property so we leave it out.

express that any two predecessors of the same number have to be identical, but we can express that they are indiscernible, i.e. they have the exact same properties.

The theory as a whole is given by

$$\begin{cases} \forall x : E(x) \leftarrow x = 0 \vee \exists y : S(y, x) \wedge \neg E(y) \\ \forall x : U(x) \leftarrow S(0, x) \vee \exists y : S(y, x) \wedge \neg U(y) \end{cases}$$

$$\varphi_1 = \forall x : S(x, 0) \Rightarrow \perp$$

$$\varphi_2 = \forall x, y : S(x, y) \Rightarrow (E(x) \vee (\forall z : S(z, y) \Rightarrow \neg E(z)))$$

$$\varphi_3 = \forall x, y : S(x, y) \Rightarrow (\neg E(x) \vee (\forall z : S(z, y) \Rightarrow E(z)))$$

$$\varphi_4 = \forall x, y : S(x, y) \Rightarrow (U(x) \vee (\forall z : S(z, y) \Rightarrow \neg U(z)))$$

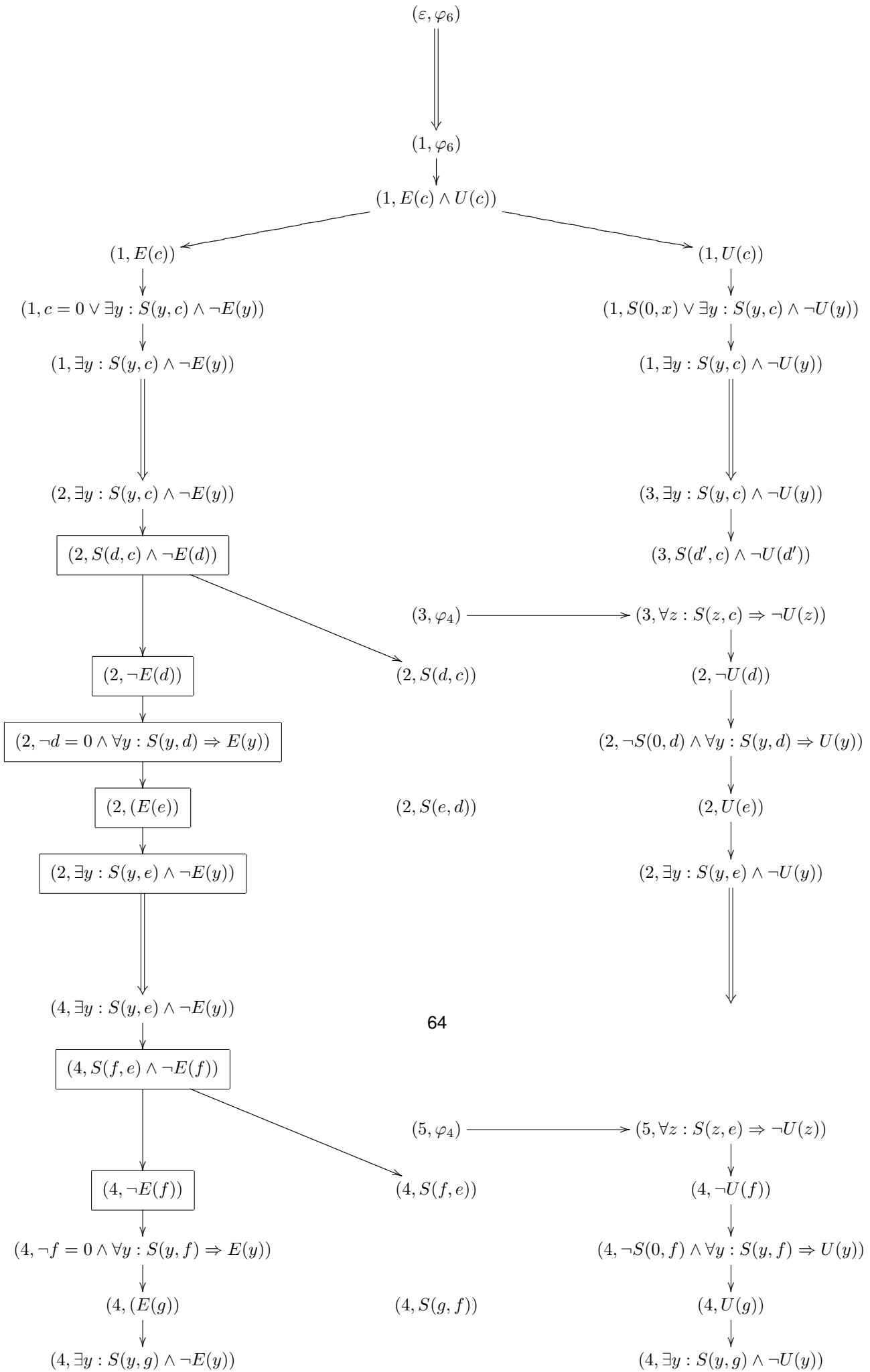
$$\varphi_5 = \forall x, y : S(x, y) \Rightarrow (\neg U(x) \vee (\forall z : S(z, y) \Rightarrow U(z)))$$

$$\varphi_6 = \exists x : E(x) \wedge U(x)$$

$$\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5 \wedge \varphi_6$$

In the figure on the following page, we show the expansion of a non-contradictory completion structure T for φ and its dependency graph G . As in the previous example, we only show the most important nodes of G , and we focus mainly on one branch of T since the other ones are completely similar. Further, we show the expansion only up until node 4. Because 4 and 2 contain different constants, they aren't copies of each other, but it's clear that node 6 will be a copy of 2. In between 2 and 6 there is a path containing the positive defined atom $E(e)$, that goes from $(2, \neg E(d))$ to $(6, \neg E(d))$. Therefore G would not be cycle-free after the application of the closing rule at this point, and will never be cycle-free if we continue our expansion in this manner. Of course this by itself doesn't prove anything, since our algorithm is non-deterministic, so that there might be other expansions of T which would work. However the only choices

we had to make where such that we tried to avoid ending up in a contradiction. For example, if we were to choose $c = 0$ in the disjunct in node 1, we would obtain a contradiction by φ_1 because of $S(d', c)$. The other choices can be defended similarly.



8 Conclusion

In this work we have presented a deductive reasoning procedure for the guarded fragment of FO(ID). We focussed first on what we defined as the simple guarded fragment of FO(ID), after which we generalized the approach to the whole fragment. Although the final algorithm contains pretty complex expansion and applicability rules, these were necessary largely due to exceptional instances. We predict that in practice in most cases the choices to be made will follow quite naturally out of the theory. Therefore an implementation that allows the user to guide the expansion will probably be most effective.

Unfortunately completeness of the general procedure remains a conjecture rather than a theorem. At present we see two possible ways out of this. The first one is to continue along the direction that was set forth in our proof, by producing a first order theory that formalizes the constraints and properties of the tree T in case it contains more than one branch. Then one could either try to look for a suitable automated theorem prover to falsify this theory, or use results from the field of combinatorics to do so. The second direction consists in delving into tree-automata theory, and look for theorems concerning characteristics of accepted trees of 2ATA's. We already have the correspondence between models and accepted trees of specific 2ATA's, and a lot of work has been done on analyzing the languages that are accepted by infinite tree automata [3]. It fell outside the scope of this project to perform a detailed study of the literature in this domain, since our aim was precisely to avoid the use of indirect tools such as automata and focus on the properties of FO(ID) itself.

9 Appendix: Nederlandstalige Samenvatting

In [5] werd eerste orde logica uitgebreid met inductieve definities (FO(ID)), om het mogelijk te maken inductief gedefinieerde concepten uit te drukken. Eerste orde logica op zich is al onbeslisbaar, waardoor het toevoegen van inductieve definities automatisch deductief redeneren bijzonder lastig maakt. Binnen het domein van de kennisrepresentatie, net zoals in de logica in haar geheel, is men steeds op zoek naar een evenwichtige balans tussen expressiviteit en beslisbaarheid. Vandaar werd naar analogie met het guarded fragment van eerste orde logica ([1]), het guarded fragment van FO(ID) geïntroduceerd in [20]. Hiermee bekomt men zowel de karakteristieken van FO(ID) als beslisbaarheid. Ondanks het bestaan van een beslisbaarheidsresultaat voor dit fragment, zijn er nog geen effectieve algoritmes ontwikkeld om hiermee te redeneren. De opzet van het huidige werk is om deze situatie recht te zetten door zulke algoritmes te ontwikkelen, die het mogelijk maken om automatisch na te gaan of een guarded FO(ID) theorie een model heeft. Om dit te bereiken bouwen we verder op bestaande technieken uit het verwante domein van Open Answer Set Programming, hetgeen een uitbreiding is van de declaratieve programmeertaal Answer Set Programming.

In het eerste hoofdstuk na de inleiding wordt FO(ID) geïntroduceerd, waarna haar syntax en semantiek worden uiteengezet. Nadat we OASP kort inleiden, maken we een vergelijking van de semantiek van beide talen. Dit zal ons toelaten om het verband tussen beide paradigma's te belichten. Op het einde van het hoofdstuk wordt er ingegaan op de notie van guardedness, meer specifiek vestigen we de aandacht op de relatie hiervan met modellen die een boomstructuur hebben.

Het derde hoofdstuk presenteert de theoretische resultaten rond guarded FO(ID) in verband met beslisbaarheid. Deze zullen we later nodig hebben om de correctheid van onze algoritmes te bewijzen. Beslisbaarheid wordt aangetoond door voor elke guarded FO(ID) theorie een alternerende automaat te ontwikkelen die bomen neemt als input, met de eigenschap dat deze automaat de lege taal bepaalt als de theorie geen model heeft.

In het vierde hoofdstuk presenteren we een algoritme voor een beperkte vorm van guarded FO(ID), met name voor simple guarded FO(ID). Dit algoritme bereidt de weg voor de algemene versie, en introduceert enkele essentiële concepten. De belangrijkste notie is die van een completion structure. Een dergelijke structuur bestaat uit de combinatie van een eindige boom met een

aantal labelling functies en een gerichte graaf, die dient om een mogelijk model voor te stellen. Bepaalde verzamelingen van knopen van de boom komen overeen met elementen van het domein, en de labels bevatten formules over deze elementen. De graaf duidt de afhankelijkheden tussen formules aan. Het algoritme gaat na of de labels en de paden in de graaf de nodige informatie bevatten om de aanwezigheid van atomen in een model te rechtvaardigen.

Het centrale resultaat wordt uiteengezet in het volgende hoofdstuk. Het algoritme uit het vorige hoofdstuk wordt uitgebreid om gebruikt te kunnen worden voor het volledige guarded fragment. Een model wordt nog steeds voorgesteld door een completion structure, maar er zullen complexere regels gelden om een dergelijke structuur op te bouwen en na te gaan of deze effectief een model voorstelt. Completion structures waarvoor dit geldt noemen we clash-free.

Hoofdstuk zes bevat de bewijzen die onze aanpak legitimeren. Eerst tonen we aan dat ons algoritme eindigt. Vervolgens bewijzen we dat elk clash-free completion structure opgebouwd vanuit een guarded FO(ID) theorie door het algoritme inderdaad een model voorstelt. Andersom zou moeten gelden dat er voor elke theorie die een model heeft door het algoritme een clash-free completion structure wordt gegenereerd. Helaas zijn we er maar gedeeltelijk in geslaagd dit hard te maken, meer specifiek is het ons niet gelukt aan te tonen dat de completion structure eindig blijft. We geven echter wel aan in welke richting een bewijs hiervan zou moeten evolueren.

Het laatste hoofdstuk bevat twee voorbeelden die het gebruik van de ontwikkelde procedure illustreren. Het eerste voorbeeld toont aan dat men wel degelijk nood heeft aan complexe regels om te bepalen wanneer de uitbreiding van een completion structure dient te stoppen. In het tweede voorbeeld passen we onze methode toe om een simpele rekenkundige eigenschap te bewijzen.

References

- [1] Andréka H., Németi I., and van Benthem J.; Modal languages and bounded fragments of predicate logic. In *J. of Philosophical Logic*, 27(3):217–274, (1998).
- [2] van Benthem J.; Dynamic bits and pieces. ILLC research report, University of Amsterdam, (1997).
- [3] Comon H. et al.; Tree Automata Techniques and Applications, Available on: <http://www.grappa.univ-lille3.fr/tata>, (2007).
- [4] Dantsin E., Eiter Th., Gottlob G. and Voronkov A.; Complexity and expressive power of logic programming. In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity*, Ulm, Germany, 82–101, (1997), IEEE Computer Society Press.
- [5] Denecker M.; Extending Classical Logic with Inductive Definitions. In *Proceedings of the 8th International Workshop on Nonmonotonic Reasoning*, Breckenridge, Colorado, USA, 1-10, (2000).
- [6] Feier C. and Heymans S.; A sound and complete algorithm for simple conceptual logic programs. Technical Report INFSYS RESEARCH REPORT 184-08-10, KBS Group, Technical University Vienna, Austria, (2008). <http://www.kr.tuwien.ac.at/staff/heyman/priv/projects/fwf-doasp/alpsws2008-tr.pdf>.
- [7] Gelfond M. and Lifschitz V.; The Stable Model Semantics for Logic Programming. In *Proc. of ICLP'88*, 1070–1080, Cambridge, Massachusetts, MIT Press, (1988).
- [8] Grädel E. and Walukiewicz I.; Guarded fixed point logic. In *Logic in Computer Science*, 45–55, (1999).
- [9] Grädel E.; On the Restraining Power of Guards. In *Journal of Symbolic Logic*, 64(4):1719–1742, (1999).
- [10] Heymans S., Van Nieuwenborgh D., and Vermeir D.; Semantic Web Reasoning with Conceptual Logic Programs. In *Proc. of RuleML 2004*, 113–127. Springer, (2004).

- [11] Heymans S., Van Nieuwenborgh D., and Vermeir D.; Conceptual logic programs. In *Annals of Mathematics and Artificial Intelligence (Special Issue on Answer Set Programming)*, 47(1–2):103–137, (2006).
- [12] Heymans S., Van Nieuwenborgh D., and Vermeir D.; Open answer set programming with guarded programs. In *ACM Transactions on Computational Logic (TOCL)*, 9(4), (2008).
- [13] Heymans S., Van Nieuwenborgh D., and Vermeir D.; Extending Conceptual Logic Programs with Arbitrary Rules. In *Proceedings of Answer Set Programming: Advances in Theory and Implementation (ASP 2005)*, vol 142 , CEUR-WS.org, (2005).
- [14] Heymans S., Van Nieuwenborgh D., and Vermeir D.; Hybrid Reasoning with Forest Logic Programs. In *Proceedings of 6th Annual European Semantic Web Conference (ESWC 2009)*, vol 5554 of Lecture Notes in Computer Science, 338-352, (2009).
- [15] Hou P., Wittocx J., Denecker M.; A Sequent Calculus Proof System for PC(ID), Unpublished.
- [16] Lifschitz, V.; Introduction to Answer Set Programming.
- [17] Lifschitz, V., Pearce, D., and Valverde, A.; Strongly equivalent logic programs. *ACM Trans. Comput. Syst.* 2(4), 526–541, (2001).
- [18] Mariën M., Gilis D., and Denecker M.; On the relation between ID-Logic and Answer Set Programming. In José Júlio Alferes and João Alexandre Leite, editors, JELIA'04, LNCS 3229, 108–120. Springer, (2004).
- [19] Vennekens J. and Denecker M.; FO(ID) as an extension of DL with rules. In *European Semantic Web Conference, Lecture Notes in Computer Science*, (2009). To appear.
- [20] Vennekens J. and Denecker M.; A Guarded Fragment of FO(ID) and its Decidability, Unpublished.