Automated Reasoning Support for First-Order Ontologies

Peter Baumgartner[1] and and Fabian M. Suchanek[2]

[1] National ICT Australia (NICTA), Peter.Baumgartner@nicta.com.au
[2] Max-Planck Institute for Computer Science, Germany, suchanek@mpi-sb.mpg.de

Abstract. Formal ontologies play an increasingly important role in demanding knowledge representation applications like the Semantic Web. Regarding automated reasoning support, the mainstream of research focusses on ontology languages that are also Description Logics, such as OWL-DL. However, many existing ontologies go beyond Description Logics and use full first-order logic.
We propose a novel transformation technique that enables model generation provers to be applied to first-order ontologies. Our transformation features the handling of equality, the Unique Name Assumption (optional) and default values, and it can be configured to avoid unnecessary Skolem terms. Our main result is the completeness of our transformation, and we provide a precise characterization of the models computed.

1 Introduction

Recent years have seen an increasing interest in formal knowledge bases (KBs). Demanding application areas – notably the *Semantic Web* – will have to remain a vision without powerful automated reasoning support. The mainstream of research focuses on ontology languages that are also Description Logics, such as OWL-DL.

However, there exist numerous KBs that go beyond first-order logic. One example is the largest formal public ontology available today, the Suggested Upper Merged Ontology SUMO [NP01]. SUMO is written in KIF, the Knowledge Interchange Format [KIF], which is basically first-order logic with equality and some higher-order features. Together with its domain-specific extensions, SUMO contains more than 20'000 terms and 60'000 axioms. Unfortunately, only limited automated reasoning is available today for first-order KBs. For instance, to our knowledge, the only theorem prover applied to SUMO so far is Vampire [RV01].

This situation seems somewhat surprising, given the demonstrated usefulness of *description logic (DL) systems* for KBs written in $\mathcal{ALC}$-like languages [BCM$^+$02]. Why has this success story not been repeated for KBs in first-order logic? The answer from a technical point of view might be that DL systems are so successful because they usually *decide* the satisfiability problem of their input language. This is an important feature, as it allows, for instance, to prove that a speculated subsumption relation between concepts does *not* hold. Furthermore, it allows the "debugging" of KBs.

Although such decision procedures cannot exist for first-order KBs, reasoning support by automated theorem provers may be attempted nevertheless. Indeed, within the

semantic web framework a number of off-the-shelf first-order theorem provers have been tested on various KBs [3]. The provers generally performed well in solving the unsatisfiable test cases. However, they often could not solve the satisfiable ones.

To address this problem, we propose a novel transformation technique on first-order logic KBs that allows to compute models with readily available systems. We target at model computation systems as developed within the logic programming community or at bottom-up clausal theorem provers as long as they support a (weak) default negation principle. The rationale is to capitalize on these well-investigated techniques and lift them to a more general language, viz., first-order logic, and strengthen the model-building capabilities of such systems. Among the systems that are suitable for combination with our transformations are dlv [CEF$^+$97] (for the stable model semantics), smodels [NS96] and KRHyper [Wer03] (both for the possible model semantics). In our experiments we have chosen KRHyper, as we know it best, it is a state-of-the-art system[4] and the possible model semantics is appropriate for our purpose. All these systems feature default negation. Indeed, we do make use of default negation, but not a very deep one.

Compared to standard transformation to clause logic, our approach offers a number of advantages. First, our translation treats equality in a way that is satisfactory in combination with the mentioned systems, which do not include built-in equality handling (although it is certainly weaker than in state-of-the-art theorem provers like Vampire). Furthermore, it respects the Unique Name Assumption (UNA), restricted to constants. This is particularly useful in the context of database-like KBs, when different constants are best considered to denote different objects. However, we allow constants to be declared as nullary functions, such that our approach is fully compatible with the standard semantics. Perhaps more significant, our approach supports defining schemes for the creation of default values for existentially quantified variables. Standard Skolemization then is just a particular instance. These default values are only brought into play if there is no other possible filler for the variable. This keeps the resulting models slim and meaningful. Optionally, the scheme can be configured to handle existential sentences as *integrity constraints*. In this case, the model building process is instructed to fail if an existential sentence is not fulfilled in the KB. This can be useful for debugging a KB. Last, our transformation allows for a "loop check", by which infinite models can be represented finitely in some cases.

Related Work

From a methodological point of view, we were helped to achieve our results by considering insights and combining results and techniques from automated theorem proving, description logic and logic programming. For instance, we employ default negation, as available in logic programming systems, as a tool to realize the "loop check" modeled after the "blocking" technique commonly found in Description Logic systems.

[3] http://www.w3.org/2003/08/owl-systems/test-results-out

[4] To be precise, a simple, polynomial transformation of the given program is required to have KRHyper compute possible models for *stratified* programs. smodels does not have this limitation to stratified programs. In return, KRHyper computes the grounded version of the program *dynamically*, which may be advantageous for large KBs.

Regarding the relation to Description Logics we are considering full first-order logic as an input language and work under the assumption that giving up decidability is acceptable in certain applications. Under this assumption it is easy to extend Description Logics (expressed in first-order logic) with arbitrary (first-order expressible) "rule" languages. [5] An example is the statement: individuals who live and work at the same location are home workers. This can be expressed as a Horn rule (clause) $\mathsf{homeWorker}(x) \leftarrow \mathsf{work}(x,y) \wedge \mathsf{live}(x,z) \wedge \mathsf{loc}(y,w) \wedge \mathsf{loc}(z,w)$, but is not expressible in current Description Logic systems.

Related methods for model computation can be classified as those that directly search for a finite model, like the extended PUHR tableau method [BT98], the method in [Bez05] and the methods in the SEM-family [Sla92,Zha95,McC03]. These methods search for finite models, essentially, by searching the space of interpretations with domain sizes $1, 2, \ldots$, in increasing order, until a model is found. The same is true for MACE-style model builders (e.g. Paradox [CS03]), which reduce searching for a model to testing propositional satisfiability. Being based on translation, the MACE-style approach is thus conceptually related, but different, to our approach. Perhaps the most significant difference in operation to both SEM- and MACE-style model building is that our transformations are *not* parametrized by a domain size. Consequently, there is no iterative deepening over the domain size, and the search for finite models works differently. This way, we address a problem often found with models computed by these methods: from a pragmatic perspective, they tend to identify too many terms. For instance, for the two unit clauses $\mathsf{P}(\mathsf{a})$ and $\mathsf{Q}(\mathsf{b})$ there is a model that identifies a and b with the same object. Such models can be counterintuitive, for instance, in a description logic setting, where concepts are just unary predicates. Furthermore, logic programs are typically understood wrt. Herbrand semantics, and it is desirable to develop compatible model building techniques. Our transformations are more careful at identifying objects than the methods mentioned and thus work closer to a Herbrand semantics.

Our approach is also somewhat related to model construction by hyper resolution e.g. [FL93,GHS02,GHS03]. One difference is our use of default negation, which is not availaible in hyper resolution like systems (except for KRHyper). The perhaps closest related work is the translation scheme in [BB04]. However, that work is concerned with one specific ontology, FrameNet, and it is shown how to translate it to a logic program. The approach in this paper is thus much more general.[6]

The rest of this paper is structured as follows. Section 2 contains preliminaries, including the definition of possible models with equality, which provides the intended semantics. Section 3 is the main part, it contains the transformations. In Section 4 we turn to the treatment of equality. In Section 5 we report on first experiments carried out on the SUMO ontology. Finally, in Section 6 we draw some conclusions.

[5] "Rules" provide a connection to (deductive) databases and are being used to represent information that is currently not expressible in the description logics associated with, say, OWL-DL.

[6] A further paper on transformation techiques related to the ones presented here has been submitted for conference publication, with the first author as a co-author. We guarantee the approach in that paper is sufficiently different. It is not a double submission.

2 Preliminaries

We use standard terminology from first-order logic and automated reasoning. Our formulas, and specifically clauses, are built over a signature Σ, usually left implicit in the following. We assume that Σ contains a distinguished nullary predicate symbol false and a 2-ary predicate symbol $\approx$, equality, used infix. We deviate from the standard definitions by distinguishing between constants and nullary function symbols (see UNA below).

If $\boldsymbol{x}$ is a sequence of variables $x_1,\ldots,x_k \in \mathcal{X}$, for some $k \geq 0$, then $\forall \boldsymbol{x}$ denotes the sequence $\forall x_1 \cdots \forall x_k$. The expression $\exists \boldsymbol{x}$ is defined analogously, and $\mathbf{Q}\boldsymbol{x}$ stands for any sequence $Q_1 x_1 \cdots Q_k x_k$, where $Q_i \in \{\forall, \exists\}$, for all $i = 1,\ldots,k$, $k \geq 0$. When ψ is a formula, the notation $\psi(\boldsymbol{x})$ means that ψ contains no more free variables than those in the sequence of variables $\boldsymbol{x}$.

We assume the reader is familiar with basic concepts of logic programming [Bar03]. A *(program) rule* is an expression of the form $H_1 \vee \cdots \vee H_m \leftarrow B_1,\ldots,B_k, \mathsf{not}\ B_{k+1},\ldots,\mathsf{not}\ B_n$, where $m \geq 1$, $n \geq k \geq 0$ and H_i, for $i = 1,\ldots m$, and B_j, for $j = 1,\ldots,n$ are (possibly non-ground) atoms over a given first-order logic signature. Each H_i is called a *head literal*, and each B_j is called a *body literal*. The *negative* body literals are those that include the default negation operator not, the other body literals are the *positive* ones. We write $H \vee \mathcal{H} \leftarrow B, \mathcal{B}, \mathsf{not}\ B', \mathcal{B}_{\mathsf{not}}$ to mean a program rule containing the head literal H, the positive body literal B and the negative body literal $\mathsf{not}\ B'$. In a *positive* rule it holds $k = n$. We treat the terms "positive rule" and "clause" as synonyms.

A *disjunctive logic program (DLP)*, also just *program*, is a set of rules. A *positive DLP* consists of positive rules only; it is thus the same as a *clause set*. In a *normal* program each rule has exactly one head literal. We consider only *domain restricted* programs, where every variable occurring in a rule must also occur in some positive body atom $B_1,\ldots,B_k$. This is a common assumption and is present in systems like KRHyper [Wer03] and smodels [NS96].

Throughout this paper we restrict attention to *Herbrand interpretations*. We represent a Herbrand interpretation – from now on just "interpretation" – as the set of its *true* (ground) atoms, which must not contain the atom false. A DLP $\mathcal{P}$ stands for the set of all ground instances of every program rule in $\mathcal{P}$, which is denoted by $\mathcal{P}^{\mathrm{gr}}$. In our (stratified) setting, a *possible model* [Sak90] is defined as follows: We say that $\mathcal{M}$ *satisfies* a ground rule $H \leftarrow \mathcal{B}, \mathcal{B}_{\mathsf{not}}$ iff $\mathcal{B} \subseteq \mathcal{M}$ and $\mathcal{M} \cap \{B \mid \mathsf{not}\ B \in \mathcal{B}_{\mathsf{not}}\} = \emptyset$ implies $H \in \mathcal{M}$. Further, $\mathcal{M}$ is a *perfect model* of a ground normal program $\mathcal{P}$ iff it satisfies every rule in $\mathcal{P}$ and is a minimal set with that property. Now, for a given DLP $\mathcal{P}$, a *split program of* $\mathcal{P}^{gr}$ is a ground normal logic program that can be obtained from $\mathcal{P}^{\mathrm{gr}}$ by deleting from each rule all but one head literal. An interpretation $\mathcal{M}$ is a *possible model* of $\mathcal{P}$ iff it is a perfect model of some split program of $\mathcal{P}^{\mathrm{gr}}$.

For instance, the propositional program consisting of the rules

$$\begin{aligned} \mathsf{whiskey} \vee \mathsf{water} &\leftarrow \mathsf{thirsty}, \mathsf{not}\ \mathsf{hungry} \\ \mathsf{water} &\leftarrow \mathsf{whiskey} \\ \mathsf{thirsty} &\leftarrow \end{aligned}$$

has two possible models, one which assigns *true* to water but not to whiskey, and one that assigns *true* to both. Notice that the classical model that assigns *true* to hungry and thirsty is not a supported model. But it seems not very desirable from a knowledge representation point of view anyway. It is this consideration that motivates us not to rely on the model-building capabilities of "standard" automated theorem provers, but instead exploit systems whose model-computation behavior is well understood. Without going into details, we only note that with the KRHyper system [Wer03] a highly efficient system for computing possible models of domain-restricted DLPs is available.

In a *Herbrand E-interpretation* (or just *E-interpretation*) the equality symbol $\approx$ is interpreted as a congruence relation over terms. If $\mathcal{M}$ is an interpretation, then $\mathcal{M}^{\approx}$ denotes the finest congruence on terms induced by the equations in $\mathcal{M}$, which is an E-interpretation. Occasionally we call $\mathcal{M}$ itself a E-interpretation when $\mathcal{M}^{\approx}$ is meant.

A *UNA-E-interpretation* is an E-interpretation that assigns *false* to every equation $c \approx d$, where c and d are different constants (we say it "satisfies the unique name assumption (UNA)").[7] In other words, $c \approx d \notin M^{\approx}$ for any UNA-E-interpretation $\mathcal{M}$. We consider the UNA because it seems useful in the context of KBs, where constants (such as Gudrun and Gundolf) could be considered as different. However, we allow constants to be declared as nullary functions, such that our approach below is fully compatible with the standard semantics.

Above we introduced possible models for DLPs. For the case of equality, we do not treat programs with default negation.[8] For our purposes it suffices to say that an interpretation $\mathcal{M}$ *E-satisfies* a positive ground rule $\mathcal{H} \leftarrow \mathcal{B}$ iff $\mathcal{B} \subseteq \mathcal{M}^{\approx}$ implies $\mathcal{H} \subseteq \mathcal{M}^{\approx}$.

The term *(UNA-)(E-)model of a program* $\mathcal{P}$ is applied as expected, meaning a (UNA-)(E-)interpretation that assigns *true* to $\mathcal{P}^{\mathrm{gr}}$ (again, E-models are defined for positive programs only).

3 Translating First-Order Formulae to DLPs

We assume, without loss of generality, that every first-oder logic sentence is given in prenex negation normal form. Thus, the sentence is of the form $\mathbf{Q}z\ \psi(z)$, where $\mathbf{Q}z$ is the quantifier prefix and $\psi(z)$ is a quantifier-free formula, built with logical operators $\wedge$, $\vee$ and $\neg$, where $\neg$ occurs only in front of atoms.

Let ϕ be a sentence in prenex negation normal form. We define a transformation $\tau(\phi)$ to (almost) clause form as follows. The quantifier prefix $\mathbf{Q}z$ may contain an existential quantifer, or not. If it does not, set $\tau(\phi) = \{\phi\}$. Otherwise ϕ can be written as

$$\phi = \mathbf{Q}z\ \psi(z) = \forall \boldsymbol{x} \exists \boldsymbol{y} \mathbf{Q}' \boldsymbol{z}' (\Delta(\boldsymbol{x}) \vee \psi'(\boldsymbol{x}\boldsymbol{y}\boldsymbol{z}'))\ , \tag{1}$$

where $\mathbf{Q}'$ is either empty or starts with a universal quantifer. We may assume that $\psi'(\boldsymbol{x}\boldsymbol{y}\boldsymbol{z}')$ is not a disjunction such that one of its immediate subformulas contains at most the variables $\boldsymbol{x}$, because then this subformula could be part of Δ. Notice that by

[7] Actually, this is a slight misnomer.

[8] A generalized approach is in preparation.

replacing $\psi(z)$ in ϕ by $\mathsf{false} \vee \psi(z)$, the form (1) is indeed a general form ($\Delta(\boldsymbol{x})$ could be the atom false).

Suppose ϕ is of the form (1) and consider the two following sentences derived from ϕ:

$$\begin{aligned}
\phi_1 &= \forall \boldsymbol{x} \exists \boldsymbol{y} (\Delta(\boldsymbol{x}) \vee \mathit{def}_{\psi'}(\boldsymbol{x},\boldsymbol{y})) \\
\phi_2 &= \forall \boldsymbol{x} \forall \boldsymbol{y} \mathbf{Q}' \boldsymbol{z}' (\neg \mathit{def}_{\psi'}(\boldsymbol{x},\boldsymbol{y}) \vee \psi'(\boldsymbol{x}\boldsymbol{y}\boldsymbol{z}')) \\
\phi_3 &= \forall \boldsymbol{x} \forall \boldsymbol{y} \overline{\mathbf{Q}'} \boldsymbol{z}' (\mathrm{NNF}(\mathit{sat}_{\psi'}(\boldsymbol{x},\boldsymbol{y}) \vee \neg \psi'(\boldsymbol{x}\boldsymbol{y}\boldsymbol{z}'))) \ ,
\end{aligned}$$

where $\mathit{def}_{\psi'}$ and $\mathit{sat}_{\psi'}$ are fresh predicate symbols of appropriate arity, $\overline{\mathbf{Q}'}\boldsymbol{z}'$ denotes the quantifier prefix obtained from replacing every universal quantifier by an existential one and vice versa, and NNF converts its argument to negation normal form. The formula ϕ_3 could be left away without sacrificing equisatisfiability of ϕ and $\tau(\phi)$, but it will be crucial for the improvement in Section 3.2 below.

Now, Δ can be written as[9]

$$\Delta = \neg B_1(\boldsymbol{x}) \vee \cdots \vee \neg B_m(\boldsymbol{x}) \vee \Delta'(\boldsymbol{x}),$$

for some formula Δ', negative literals $\neg B_i$, for all $i = 1, \ldots, m$, $m \geq 0$, where m is chosen as large as possible. Notice we allow $m = 0$. Hence Δ can indeed be written this way. For later use observe that with ψ and hence Δ being quantifier-free, Δ' is also quantifier-free.

From Δ derive the formulas

$$\begin{aligned}
\Delta_1 &= \neg B_1(\boldsymbol{x}) \vee \cdots \vee \neg B_m(\boldsymbol{x}) \vee \mathit{def}_{\Delta'}(\boldsymbol{x}) \\
\Delta_2 &= \forall \boldsymbol{x} (\neg \mathit{def}_{\Delta'}(\boldsymbol{x}) \vee \Delta'(\boldsymbol{x})) \ ,
\end{aligned}$$

where again $\mathit{def}_{\Delta'}$ is a fresh predicate symbol of appropriate arity. The next step is to replace Δ in ϕ_1 by Δ_1, which yields

$$\phi_1^{\Delta_1} = \forall \boldsymbol{x} \exists \boldsymbol{y} (\neg B_1(\boldsymbol{x}) \vee \cdots \vee \neg B_m(\boldsymbol{x}) \vee \mathit{def}_{\Delta'}(\boldsymbol{x}) \vee \mathit{def}_{\psi'}(\boldsymbol{x},\boldsymbol{y})) \ .$$

Above we already defined $\tau(\phi) = \{\phi\}$ for the case that $\mathbf{Q}\boldsymbol{z}$ does not contain an existential quantifier. We are now ready to define $\tau(\phi)$ if $\mathbf{Q}\boldsymbol{z}$ does contain an existential quantifier:

$$\tau(\phi) = \{\phi_1^{\Delta_1}, \Delta_2\} \cup \tau(\phi_2) \cup \tau(\phi_3) \ .$$

To see the termination of the transformation τ, observe that both ϕ_2 and ϕ_3 are strictly smaller than ϕ in the (well-founded) ordering on formulas with quantifier prefixes of same length induced by the lexicographic ordering on quantifer sequences, where $\exists$ is greater than $\forall$.

Introducing names (like $\mathit{def}_{\psi'}(\boldsymbol{x},\boldsymbol{y})$ above) for subformulas and adding definitions for them, like our transformation does, is a standard technique used in clause normal form transformations. It is well-know that such transformations preserve satisfiability[10].

[9] Similarly to above, we allow $\Delta'(\boldsymbol{x})$ to be false.

[10] Because existential quantifiers are not eliminated, τ even preserves models, in both ways (in the sense of conservative extensions for the newly introduced symbols).

All universal formulas in $\tau(\phi)$, i.e. those sentences in $\tau(\phi)$ of the form $\forall \boldsymbol{x}\ \Delta(\boldsymbol{x})$, can be converted to clausal form (i.e. a positive DLP) easily by means of well-known techniques [11].

Each remaining formula in $\tau(\phi)$ is of the form as denoted by $\phi_1^{\Delta_1}$ above. Let Φ be a formula of this kind. We propose 4 different options to translate Φ to a DLP.

3.1 Skolemization Option

With this option a Skolem term is chosen as a default value to satisfy – in Description Logic terminology – an existentially quantified role. Technically, the formula Φ is translated to the following (domain-restricted) DLP:

$$def_{\Delta'}(\boldsymbol{x}) \vee default\_filler_{\Phi}(\boldsymbol{x}, \boldsymbol{sk}_{\Phi}(\boldsymbol{x})) \leftarrow B_1(\boldsymbol{x}), \ldots, B_m(\boldsymbol{x}) \tag{1}$$

$$def_{\psi'}(\boldsymbol{x},\boldsymbol{y}) \leftarrow default\_filler_{\Phi}(\boldsymbol{x},\boldsymbol{y})\ . \tag{2}$$

Here, $default\_filler_{\Phi}$ is a new predicate symbol associated to the formula Φ, and $\boldsymbol{sk}_{\Phi}(\boldsymbol{x})$ is a list of Skolem terms made from the variables $\boldsymbol{x}$. Thus, our transformation includes the usual Skolemization as its simplest option.

3.2 Recycling Option

This option allows to avoid the introduction of a Skolem term if there is already a role filler present in the model. This can be achieved by translating Φ as follows (instead of (1) and (2)):

$$def_{\Delta'}(\boldsymbol{x}) \vee \mathsf{check}\_sat_{\psi'}(\boldsymbol{x}) \vee default\_filler_{\Phi}(\boldsymbol{x}, \boldsymbol{sk}_{\Phi}(\boldsymbol{x})) \leftarrow B_1(\boldsymbol{x}), \ldots, B_m(\boldsymbol{x}) \tag{3}$$

$$def_{\psi'}(\boldsymbol{x},\boldsymbol{y}) \leftarrow default\_filler_{\Phi}(\boldsymbol{x},\boldsymbol{y}) \tag{4}$$

$$\mathsf{false} \leftarrow def_{\Delta'}(\boldsymbol{x}), \mathsf{check}\_sat_{\psi'}(\boldsymbol{x}) \tag{5}$$

$$\mathsf{false} \leftarrow \mathsf{check}\_sat_{\psi'}(\boldsymbol{x}), def_{\psi'}(\boldsymbol{x},\boldsymbol{y}) \tag{6}$$

$$\mathsf{false} \leftarrow \mathsf{check}\_sat_{\psi'}(\boldsymbol{x}), \mathsf{not}\ sat1_{\psi'}(\boldsymbol{x}) \tag{7}$$

$$sat1_{\psi'}(\boldsymbol{x}) \leftarrow sat_{\psi'}(\boldsymbol{x},\boldsymbol{y}) \tag{8}$$

$$\mathsf{false} \leftarrow def_{\psi'}(\boldsymbol{x},\boldsymbol{y}), sat_{\psi'}(\boldsymbol{x},\boldsymbol{z}), \mathsf{not}\ \mathsf{equal}_{|\boldsymbol{y}|}(\boldsymbol{y},\boldsymbol{z}) \tag{9}$$

$$\mathsf{equal}_{|\boldsymbol{y}|}(\mathrm{x}_1,\ldots,\mathrm{x}_n,\mathrm{y}_1,\ldots,\mathrm{y}_n) \leftarrow \mathrm{x}_1 \approx \mathrm{y}_1, \ldots, \mathrm{x}_n \approx \mathrm{y}_n \tag{10}$$

Notice that (3) contains one more head literal than (1), which is $\mathsf{check}\_sat_{\psi'}(\boldsymbol{x})$. This literal signals that there is *already* a role-filler for the existentially quantified role. (Having default negation at disposal is indeed helpful to formalize such integrity constraints.) The other rules realize certain exclusivity tests among the alternatives according to the head of the rule 3. Notice that the DLP is stratified because default negation is used only in the body of rules with false in the head.

[11] Note that our transformation singles out the literals in $\phi_1^{\Delta_1}$ that contain only the variables $\boldsymbol{x}$ but not $\boldsymbol{y}$) (*mini-scoping*), which will help to obatin a *domain-restricted* DLP more often.

For illustration of the whole transformation consider the formula

$$\phi = \forall x(\mathsf{p}(x) \rightarrow \exists y\, \mathsf{q}(x,y) \vee \mathsf{r}(x)) \quad .$$

The translation τ with the recycling option applied to ϕ yields for the rule schemes (3) and (4) the DLP

$$\mathsf{def}_r(\mathrm{x}) \vee \mathsf{check\_sat}_q(\mathrm{x}) \vee \mathsf{default\_filler}(\mathrm{x}, \mathsf{sk}(\mathrm{x})) \leftarrow \mathsf{p}(\mathrm{x})$$
$$\mathsf{def}_q(\mathrm{x},\mathrm{y}) \leftarrow \mathsf{default\_filler}(\mathrm{x},\mathrm{y}) \quad .$$

Suppose additionally the fact $\mathsf{p}(\mathsf{a})$ as given. Using description logic terminology, $\mathsf{sk}(\mathsf{a})$ is a "default filler" for the "role" q, possibly derived as $\mathsf{default\_filler}(\mathsf{a}, \mathsf{sk}(\mathsf{a}))$. From inspecting ϕ it is clear that model must satisfy the formula $\exists y\, \mathsf{q}(\mathsf{a}, y) \vee \mathsf{r}(\mathsf{a})$. This can be achieved in various ways. In the first rule above, the instantiated head $\mathsf{def}_r(\mathsf{a})$ stands for satisfying the "rest" outside the scope of the $\exists$-quantifer, i.e. $\mathsf{r}(\mathsf{a})$ (notice that $\tau(\phi)$ includes the rule $\mathsf{r}(\mathrm{x}) \leftarrow \mathsf{def}_r(\mathrm{x})$). Together with the second rule, the third alternative $\mathsf{default\_filler}(\mathsf{a}, \mathsf{sk}(\mathsf{a}))$ in the instantiated first rule instructs the model generation process to actually insert $\mathsf{q}(\mathsf{a}, \mathsf{sk}(\mathsf{a}))$ into the model candidate ($\tau(\phi)$ includes the rule $\mathsf{q}(\mathrm{x},\mathrm{y}) \leftarrow \mathsf{def}_q(\mathrm{x},\mathrm{y})$). However, rule (9) makes sure that no other filler will or has been inserted that is equal to the default filler (which justifies the name *default* filler). The test for (non-)equality is necessary, because later, the default filler could be equated to some other term. For instance, if $\mathsf{q}(\mathsf{a},\mathsf{b})$ is also present and $\mathsf{sk}(\mathsf{a}) \approx \mathsf{b}$ is not present, this model candidate will be rejected and the alternative $\mathsf{check\_sat}_q(\mathsf{a})$, the second alternative in the first rule, will succeed to build a model (unless it fails for other reasons not related to the case analysis). The formula $\exists y\, \mathsf{q}(\mathsf{a}, y)$ will thus be satisfied in one way or the other, with a preference to a filler different from the default value. Notice that the scheme ϕ_3 from the τ transformation yields the required DLP to test if, in terms of the example, $\exists y\, \mathsf{q}(\mathsf{a}, y)$ is satisfied. That DLP is just $\mathsf{sat}_q(\mathrm{x},\mathrm{y}) \leftarrow q(\mathrm{x},\mathrm{y})$.

3.3 Model Checking Option

Sometimes, it is useful to regard existential formulae as integrity constraints for a KB. For instance, to check if the objects mentioned in a given database suffice to extend it to a model for a given KB. Instead of creating fillers by means of Skolem terms, the model construction process must check that fillers are already present. This can be achieved by replacing (3) and (4) above by the following scheme:

$$def_{\Delta'}(\boldsymbol{x}) \vee \mathsf{check\_}sat_{\psi'}(\boldsymbol{x}) \leftarrow B_1(\boldsymbol{x}), \ldots, B_m(\boldsymbol{x})$$

This transformation ensures that no Skolem terms can be inserted by the model computation. The only way to satisfy the existentially quantified part then is by proving that it is already satisfied.

3.4 Loop Check Option

The introduction of Skolem terms leads easily to nontermination of model-generation systems. Instead of such Skolem terms one could choose constants as default values.

This will ensure termination (supposing no other function symbols are present), and in case of satisfiability a finite domain model of the original knowledge base exists. As a less drastic means, we propose to "re-use" existing Skolem terms that qualify as role fillers instead of creating new ones, similarly to the blocking techniques found in description logic systems (however, more general). This can be achieved by translating Φ according to the "recycling option", however the rule (3) is replaced by the following rules:

$$\begin{aligned} &\mathit{def}_{\Delta'}(\boldsymbol{x}) \vee \mathsf{check\_sat}_{\psi'}(\boldsymbol{x}) \vee \mathit{choose\_default\_filler}_{\Phi}(\boldsymbol{x}) \vee \\ &\qquad \mathit{default\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{sk}_{\Phi}(\boldsymbol{x})) \leftarrow B_1(\boldsymbol{x}), \ldots, B_m(\boldsymbol{x}) \end{aligned} \tag{11}$$

$$\begin{aligned} &\mathit{other\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{sk}_{\Phi}(\boldsymbol{y})) \vee \mathit{default\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{sk}_{\Phi}(\boldsymbol{y})) \leftarrow \\ &\qquad \mathit{choose\_default\_filler}_{\Phi}(\boldsymbol{x}), \mathit{sat}_{\psi'}(\boldsymbol{x_1}, \boldsymbol{sk}_{\Phi}(\boldsymbol{y})) \end{aligned} \tag{12}$$

$$\mathsf{false} \leftarrow \mathit{default\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{y}), \mathit{other\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{z}) \tag{13}$$

$$\mathsf{false} \leftarrow \mathit{default\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{y}), \mathit{default\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{z}), \mathsf{not\ equal}_{|\boldsymbol{y}|}(\boldsymbol{y}, \boldsymbol{z}) \tag{14}$$

$$\mathsf{false} \leftarrow \mathit{choose\_default\_filler}_{\Phi}(\boldsymbol{x}), \mathsf{not}\ \mathit{some\_default\_filler}_{\Phi}(\boldsymbol{x}) \tag{15}$$

$$\mathit{some\_default\_filler}_{\Phi}(\boldsymbol{x}) \leftarrow \mathit{default\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{y}) \tag{16}$$

Notice that rule (3) contained the head literal $\mathit{default\_filler}_{\Phi}(\boldsymbol{x}, \boldsymbol{sk}_{\Phi}(\boldsymbol{x}))$ which is responsible for inserting new Skolem terms into the domain. Compared to rule (3) the new rule (11) contains once more an additional head literal, $\mathit{choose\_default\_filler}_{\Phi}(\boldsymbol{x})$. Together with rule (12) this has the effect of nondeterministically selecting a default filler among all Skolem terms previously introduced to satisfy the existential quantification of (another instance of) the formula. The nondeterministic selection process is realized by the $\mathit{other\_filler}_{\Phi}$-alternative in the head, which allows to choose a default filler – or not. The remaining rules achieve that exactly one default filler will be generated.
For illustration, consider the following example from the Tambis Ontology [SPB$^+$04]:

$$\forall x\ \mathsf{chapter}(x) \rightarrow \exists y\ \mathsf{in\_book}(x, y) \wedge \mathsf{book}(y) \tag{17}$$

$$\forall x\ \mathsf{book}(x) \rightarrow \exists y\ \mathsf{has\_chapter}(x, y) \wedge \mathsf{chapter}(y) \tag{18}$$

Notice the terminological cycle. To get the model computation started, suppose an additional fact $\mathsf{chapter}(\mathsf{a})$. Leaving away many uninteresting facts, the model generation process will first satisfy (17) by deriving

$$\mathsf{book}(f_1(a)) \tag{19}$$

$$\mathsf{in\_book}(a, f_1(a))\ . \tag{20}$$

Next, it will satisfy (18) by deriving

$$\mathsf{chapter}(f_2(f_1(a))) \tag{21}$$

$$\mathsf{has\_chapter}(f_1(a), f_2(f_1(a)))\ . \tag{22}$$

Now, (17) requires the existence of a book for the newly created chapter $f_2(f_1(a))$. Instead of creating a new Skolem term, rule (12) will find that $f_1(a)$ can be used as a default filler. Thus, the model generation process terminates by deriving

$$\mathsf{in\_book}(f_2(f_1(a)), f_1(a))\ . \tag{23}$$

In summary, the natural infinite model will be avoided by the loop check option. Thus, the loop check option can allow for a finite model in cases where a naive translation to clauses may only have an infinite model.

3.5 The Loop Check Option in Practice

Up to now, the loop check option has been introduced in a purely declarative way. More considerations are necessary to make it effective in practice. First of all, the loop check is not designed to prove the *unsatisfiability* of a set of formulae. Unsatisfiability can be proven more easily without the loop check option, because the search space is much smaller without the additional rules. Instead, the loop check aims at the more difficult problem of proving the *satisfiability* of a set of formulae.

If the loop check-transformation of a set of formulae has a finite model, then the original set of formulae also has a finite model. Unfortunately, model generation systems may have difficulties in finding this finite model, even if it exists. We are currently investigating ways to facilitate the search for the finite model.

The issue is to realize a *fair* search for a model. This is not trivial, as, in general, the Herband universe of the programs obtained by the translation is infinite. For instance, the search strategy of KRHyper is fair in the sense that it guarantees *refutational* completeness (in particular when the Herbrand universe is infinite). In contrast, even for very simple satisfiable programs obtained with the loopchecking option of Section 3.4 KRHyper will not terminate – the search strategy is just not fair for (finite) model building. The iterative deepening scheme KRHyper may lead into an infinite branch in the search tree and thereby miss an alternative leading to a model.

Other systems, like smodels, require full grounding-out of their input clause set, which is obviously, in general, not possible in presence of function symbols.

A solution to these problems is to generate (finite) interpretations as candidates, check them explicitly for being a model of the program and stop this search as soon as a model has been found. A systematic way to do so is to run the systems with a bound on the resources allowed, checking if a model has been found and increasing these resources in a fair way on failure (iterative deepening). For KRHyper, for instance, this can be achieved by running it with a limit on the term depth on the generated terms. Regarding smodels, one could work with growing approximations of the infinite set of all ground instances.

For the check for modelship the following rules are added:

$$\mathit{unsatisfied}_{\Phi}(\boldsymbol{x}) \leftarrow B_1(\boldsymbol{x}), \ldots, B_m(\boldsymbol{x}), \mathsf{not}\ \mathit{sat1}_{\Psi'}(\boldsymbol{x}) \tag{24}$$

$$\mathsf{unsatisfied\_some} \leftarrow \mathit{unsatisfied}_{\Phi}(\boldsymbol{x}) \tag{25}$$

Last, one adds the rule

$$\mathsf{satisfiable} \leftarrow \mathsf{not\ unsatisfied\_some}\ . \tag{26}$$

Now, the idea is to conclude if a model contains the atom satisfiable then the set of formula is indeed satisfiable (in a finite model) and no further deepening is necessary. However, this conclusion is not true if the given formula, and hence the obtained translated program, contains function symbols other than Skolem functions and constants.

The test not $sat1_{\Psi'}(\boldsymbol{x})$ in the body of the first clause is too weak then. We are currently working on a solution for this general case.[12]

4 Equality

Ontologies typically make use of equality. For example, equality is used to define functions or in *integrity constraints* to state that certain objects are different. Another common use of equality is to state that two objects must be equal under certain circumstances. For example, the "age" of twins must be "equal".

The most advanced techniques to *efficiently* treat equality have been developed in the field of *automated theorem proving* for refutational theorem provers (see [BG98]). Unfortunately, none of these methods has been implemented in the above-mentioned model generation systems.

One generic option to treat equality is by means of adding equality axioms. However, the search space induced by the resulting clause set is prohibitively high and achieving termination is practically impossible. The most problematic axioms in this regard are substitution axioms, like $\mathsf{f}(\mathrm{x}) \approx \mathsf{f}(\mathrm{y}) \leftarrow \mathrm{x} \approx \mathrm{y}$. Another option is to "compile away" equality. The probably most well-known method in this direction is the "modification method" in [Bra75], which was later improved in [BGV97]. We follow this direction and propose below an *equality transformation* for DLPs.

We say that a rule is *flat* if (i) the only proper subterms of terms in equations are either variables or constants, and (2) all arguments to predicate symbols are either variables or constants. Every rule can be turned into a flat one by recursively replacing an offending subterm t by a fresh variable x and adding the equation $t \approx x$ to the rule body (See again [BGV97]). As a running example consider the program consisting of the two rules $\mathsf{f}(\mathsf{g}(\mathrm{x})) \approx x \leftarrow \mathsf{p}(\mathrm{x},\mathsf{a})$ and $\mathsf{p}(\mathsf{c},\mathsf{a}) \leftarrow$. Assume that c is a constant and a is a nullary function symbol. A flat version then is

$$\mathsf{f}(\mathrm{y}) \approx \mathrm{x} \leftarrow \mathsf{p}(\mathrm{x},\mathrm{z}), \mathsf{g}(\mathrm{x}) \approx \mathrm{y}, \mathsf{a} \approx \mathrm{z}$$
$$\mathsf{p}(\mathsf{c},\mathrm{x}) \leftarrow \mathsf{a} \approx \mathrm{x} \ .$$

The purpose of flattening is to achieve the effect of the substitution axioms. If the UNA is desired, it is axiomatized by the rules $\mathsf{false} \leftarrow c \approx d$, for each pair c,d of different constants. Next, $\approx$ has to be confined to an equivalence relation by means of the rules[13]

$$\mathrm{x} \approx \mathrm{x} \leftarrow$$
$$\mathrm{x} \approx \mathrm{y} \leftarrow \mathrm{y} \approx \mathrm{x}$$
$$\mathrm{x} \approx \mathrm{z} \leftarrow \mathrm{x} \approx \mathrm{y}, \mathrm{y} \approx \mathrm{z} \ .$$

These axioms together with the flattened rules of a positive DLP $\mathcal{P}$ constitute the *equality transformation of* $\mathcal{P}$, denoted by $\mathcal{P}^{\mathrm{eq}}$.

The following theorem is the main result of this section.

[12] We intend to have a solution for the general case ready for the final version of the paper (should it be accepted).

[13] Strictly speaking, the reflexivity rule $\mathrm{x} \approx \mathrm{x} \leftarrow$ is not domain-restricted. But this case is harmless and usually poses no problems.

Theorem 1 (Soundness and Completeness). *Let $\mathcal{P}$ be a positive disjunctive logic program. If $\mathcal{P}^{eq}$ does not have a possible model then there is no UNA-E-model of $\mathcal{P}$ (soundness). If $\mathcal{M}$ is a possible model for $\mathcal{P}^{eq}$ then $\mathcal{M}$ contains a UNA-E-model of $\mathcal{P}$ (completeness).*

Thus, any sound and complete method to generate possible models of DLPs provides a reliable method to compute UNA-E-models. The (easy) soundness result is nothing essentially new. The more difficult completeness proof is contained in the appendix.[14]

The completeness result says: if the system computes a possible model $\mathcal{M}$ of $\mathcal{P}^{\text{eq}}$, then no more atoms than those in $\mathcal{M}$ have to be made true in order to obtain an UNA-E-model for $\mathcal{P}$. Notice that this result entails refutational completeness by its contrapositive direction. However, our completeness result is somewhat stronger than the refutational completeness results in [BGV97,Bra75] as it makes a claim in terms of possible models.

For the simple example above, any reasonable bottom-up model computation system will terminate on its equality transformation and report $\mathcal{M} = \{\mathsf{c} \approx \mathsf{f}(\mathsf{g}(\mathsf{c})),\ \mathsf{f}(\mathsf{g}(\mathsf{c})) \approx \mathsf{c}, \mathsf{p}(\mathsf{c},\mathsf{a}), \mathsf{x} \approx \mathsf{x}\}$, which is the expected UNA-E-model $\mathcal{M}^{\approx}$ of the original program. Notice that undesired models (non-minimal ones) like ones that also include $\mathsf{p}(\mathsf{a},\mathsf{a})$ will not be computed. Further notice that any such system will not terminate on the original program when equipped with the equality axioms.

5 Preliminary Experiments

We applied our transformation to the core of the Suggested Upper Merged Ontology SUMO [NP01]. SUMO contains *meta-predicates*, i.e. predicates that define the properties of other predicates. We translated these predicates appropriately to first-order logic. For example, we translated the (higher-order) sentence $\mathsf{disjoint\_classes}(\mathsf{Man},\mathsf{Woman})$ to the rule

$$\mathsf{false} \leftarrow \mathsf{instance}(x,\mathsf{Man}), \mathsf{instance}(x,\mathsf{Woman}) \ . \tag{27}$$

SUMO occasionally uses other higher order formulae, which we had to filter out. The resulting first-order KB contains about 1800 formulae.

Running KRHyper on the DLP transformation revealed numerous inconsistencies in SUMO. These included misspelled and hence unbound variables as well as semantic inconsistencies in connection with the Mid-level-ontology extensions. For example, one can derive that planetEarth is a geographicArea. Since each geographicArea is a geographicSubregion of planetEarth, it follows that planetEarth is a geographicSubregion of itself. This contradicts the irreflexivity of geographicSubregion. We reported the errors to the developers of SUMO and removed them. Then, KRHyper can calculate a model for our DLP translation within a few seconds. The model consists of roughly 2000 facts.

[14] We do not consider this appendix part of the workshop paper, proper. Should the paper be accepted we make publicly available a long version containing the proof and omit from the workshop version the proof.

To test the equality transformation, we added the following facts to SUMO: France lies west of Germany and Germany's biggest trading partner lies east of Germany.

$$\mathsf{orientation}(\mathsf{germany}, \mathsf{france}, \mathsf{west})$$
$$\mathsf{orientation}(\mathsf{germany}, \mathsf{biggestTradingPartner}(\mathsf{germany}), \mathsf{east}) \ .$$

By help of the axioms in SUMO, KRHyper deduces (among others) the following facts:

$$\mathsf{orientation}(\mathsf{france}, \mathsf{germany}, \mathsf{east})$$
$$\mathsf{orientation}(\mathsf{biggestTradingPartner}(\mathsf{germany}), \mathsf{germany}, \mathsf{west})$$
$$\mathsf{between}(\mathsf{germany}, \mathsf{france}, \mathsf{biggestTradingPartner}(\mathsf{germany}))$$

Now, we add the fact $\mathsf{biggestTradingPartner}(\mathsf{germany}) \approx \mathsf{france}$. As a result, KRHyper derives a contradiction, as expected, because France cannot lie both east and west of Germany.

To test our default value transformation, we added the following facts to SUMO:

$$\mathsf{instance}(\mathsf{p}, \mathsf{judicialProcess})$$
$$\mathsf{agent}(\mathsf{p}, \mathsf{a})$$

In SUMO, each judicial process is a political process. Furthermore, each political process requires an agent. Hence the model generation produces the fact $\mathsf{agent}(\mathsf{p}, \mathsf{f}(\mathsf{p}))$. However, if the recycling option is chosen, *a* qualifies as a default filler for the agent role. Consequently, the above fact is *not* derived with the recycling option.

SUMO contains numerous axioms that induce infinite models. Unfortunately, most of them cannot be detected by the current version of our loop check option.

6 Conclusions

We presented a transformation from first-order logic formulae to disjunctive logic programs. Our transformation allows to apply logic programming model generation systems to first-order logic ontologies. As special features, our transformation allows the efficient treatment of equality, including the unique name assumption. Furthermore, our transformation allows a flexible handling of existential roles, including the avoidance of unnecessary Skolem terms or the re-use of existing Skolem terms.

Our main result is the completeness of our equality transformation. It is not difficult, although somewhat tedious, to prove the soundness and (refutational) completeness of the other transformations from first-order logic to logic programs.

Our transformation allows to generate finite models in certain cases. We are currently investigating ways to strengthen the transformation so that finite models can be detected more often.

References

[Bar03] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[BB04] Peter Baumgartner and Aljoscha Burchardt. Logic Programming Infrastructure for Inferences on FrameNet. In José Alferes and João Leite, editors, *Logics in Artificial Intelligence, Ninth European Conference, JELIA'04*, volume 3229 of *Lecture Notes in Artificial Intelligence*, pages 591–603. Springer Verlag, Berlin, Heidelberg, New-York, 2004.

[BCM+02] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *Description Logic Handbook*. Cambridge University Press, 2002.

[Bez05] M. Bezem. Disproving distributivity in lattices using geometry logic. In *Proc. CADE-20 Workshop on Disproving*, 2005.

[BG98] Leo Bachmair and Harald Ganzinger. Chapter 11: Equational Reasoning in Saturation-Based Theorem Proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction. A Basis for Applications*, volume I: Foundations. Calculi and Refinements, pages 353–398. Kluwer Academic Publishers, 1998.

[BGV97] Leo Bachmair, Harald Ganzinger, and Andrej Voronkov. Elimination of equality via transformation with ordering constraints. Research Report MPI-I-97-2-012, Max-Planck-Institut fÃ¼r Informatik, Im Stadtwald, D-66123 SaarbrÃ¼cken, Germany, December 1997.

[Bra75] D. Brand. Proving theorems with the modification method. *SIAM Journal on Computing*, 4:412–430, 1975.

[BT98] François Bry and Sunna Torge. A Deduction Method Complete for Refutation and Finite Satisfiability. In *Proc. 6th European Workshop on Logics in AI (JELIA)*, LNAI. Springer, 1998.

[CEF+97] Simona Citrigno, Thomas Eiter, Wolfgang Faber, Georg Gottlob, Christoph Koch, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The dlv system: Model generator and advanced frontends (system description). In *Workshop Logische Programmierung*, pages 0–, 1997.

[CS03] Koen Claessen and Niklas SÃ¶rensson. New techniques that improve mace-style finite model building. In Peter Baumgartner and Christian G. FermÃ¼ller, editors, *CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications*, 2003.

[FL93] C. Fermüller and A. Leitsch. Model building by resolution. In *Computer Science Logic: CSL'92*, volume 702 of *LNCS*, pages 134–148. Springer, 1993.

[GHS02] L. Georgieva, U. Hustadt, and R. A. Schmidt. A new clausal class decidable by hyperresolution. In *Automated Deduction: CADE-18*, volume 2392 of *LNAI*. Springer, 2002.

[GHS03] L. Georgieva, U. Hustadt, and R. A. Schmidt. Hyperresolution for guarded formulae. *J. Symbolic Computat.*, 36(1–2):163–192, 2003.

[KIF] Kif - knowledge interchange format. http://www.csee.umbc.edu/kse/kif/.

[McC03] W. McCune. Mace4 reference manual and guide. Technical Memorandum 264, Argonne National Laboratory, 2003.

[NP01] I. Niles and A. Pease. Towards a standard upper ontology. In Chris Welty and Barry Smith, editors, *In Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, 2001.

[NS96] Ilkka Niemelä and Patrik Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, Bonn, Germany, 1996. The MITPress.

[RV01] Alexandre Riazonov and Andrei Voronkov. Vampire 1.1 (system description). In *Proc. International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[Sak90] C. Sakama. Possible Model Semantics for Disjunctive Databases. In W. Kim, J.-M. Nicholas, and S. Nishio, editors, *Proceedings First International Conference on Deductive and Object-Oriented Databases (DOOD-89)*, pages 337–351. Elsevier Science Publishers B.V. (North–Holland) Amsterdam, 1990.

[Sla92] John Slaney. Finder (finite domain enumerator): Notes and guide. Technical Report TR-ARP-1/92, Australian National University, Automated Reasoning Project, Canberra, 1992.

[SPB+04] R. D. Stevens, N. W. Paton, S. K. Bechhofer, G. K. Ng, M. Peim, P. G. Baker, C. A. Goble, and A. M. Brass. Tambis: Transparent access to multiple bioinformatics services. *Genetics, Genomics, Proteomics, and Bioinformatics*, January 2004. ISBN 0470849746.

[Wer03] Christoph Wernhard. System Description: KRHyper. Fachberichte Informatik 14–2003, Universita"t Koblenz-Landau, UniversitÃ¤t Koblenz-Landau, Institut fÃ¼r Informatik, Rheinau 1, D-56075 Koblenz, 2003.

[Zha95] Hantao Zhang. Sem: a system for enumerating models. In *IJCAI-95 — Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal*, pages 298–303, 1995.

A Proof of Theorem 1

Theorem 1 (Completeness). *Let $\mathcal{P}$ be a positive disjunctive logic program. If $\mathcal{P}^{eq}$ does not have a possible model then there is no UNA-E-model of $\mathcal{P}$ (soundness). If $\mathcal{M}$ is a possible model for $\mathcal{P}^{eq}$ then $\mathcal{M}$ contains a UNA-E-model of $\mathcal{P}$ (completeness).*

The formal tool used below is the model construction technique introduced for superposition calculi (see [BG98] for some details left away here).

Before turning to the proof, proper, let us introduce some notation. Let $\mathcal{P}_1$ a ground, *normal* positive DLP, i.e. one without disjunction in the head of its rules. Its program rules thus are of the form $H \leftarrow \mathcal{B}$, where H is an atom. (A rule $H \leftarrow \mathcal{B}$ is not necessarily a definite program rule in the standard sense, as H could be false.). We write $\mathcal{M} \models^p \mathcal{P}_1$ for the fact that $\mathcal{M}$ is a minimal model of $\mathcal{P}_1$. Notice that minimal and perfect models coincide for any such program $\mathcal{P}_1$.

Recall that for a given interpretation $\mathcal{M}$, the finest congruence induced by its equations is denoted by $\mathcal{M}^{\approx}$. For a given atom A we write $\mathcal{M} \models_{\approx} A$ iff $A \in \mathcal{M}^{\approx}$, and we write $\mathcal{M} \models_{\approx,\mathrm{UNA}} A$ if additionally $\mathcal{M}$ satisfies UNA.

As we are dealing with ground programs, we may leave away the term "ground" when talking about terms, atoms and program rules most times. When non-ground expressions are involved, we will notice that explicitly.

Below we consider non-ground rules C and ground substitutions γ for C. Without loss of generality we may always assume that the domain of γ[15] is just the variables occuring in C. When R is a term rewrite system (TRS), we say that γ *is reducible for* C if there is a variable x such that $x\gamma \to_R t$. That is, some term in the range of γ can be rewritten by some rule in R to the smaller term t. Otherwise, γ *is irreducible for* C. As the proof works with a specific TRS R defined below, there is no need to have R as parameter to the definition.

Proof. (Completeness) Suppose that $\mathcal{M}$ is a possible model of $\mathcal{P}^{\mathrm{eq}}$. This means $\mathcal{M}$ is a minimal model of some split program $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}_1$ of $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$. We will show there is a certain subset $R \subseteq \mathcal{M}$ that is a UNA-E-model of $\mathcal{P}$. More precisely, R will be a terminating rewrite system without critical pairs, i.e. a convergent rewrite system.

The proof that R is a UNA-E-model of $\mathcal{P}$ has three parts. First we will show that $R \models_{\approx} \mathcal{P}^{\mathrm{eq}}$. The subsequent (easy) step is to conclude $R \models_{\approx} \mathcal{P}$. In the final step we will show that R satisfies UNA, which will complete the proof.

$R \models_{\approx} \mathcal{P}^{eq}$. For the construction of R we need a reduction ordering that is total on ground terms.[16] Let $\succ$ be any such reduction ordering that additionally satisfies the following two conditions:

1. For any constant c and term t, if $c \succ t$ then t is a constant, too. That is, no non-constant term can be smaller that a non-constant term.
2. There is a designated constant true, not occuring in the given program and that is minimal in $\succ$. That is, there is no term t with $\mathsf{true} \succ t$.

[15] I.e. $\{x \mid x\gamma \neq x\}$.

[16] A *reduction ordering* is a strict partial ordering that is well-founded and is closed under context, i.e. $s \succ s'$ implies $t[s] \succ t[s']$ for all terms t.

The purpose of the constant true is to enable uniform notation, where atoms are read as equations. More precisely, an atom A different from false and that is not an equation is read as the equation $A \approx \mathsf{true}$. This way atoms become terms. The change of signature involved causes no problems in the proof. Notice that it follows from the definition of "flat" that the atom A is flat iff the equation $A \approx \mathsf{true}$ is flat then.

Equations are compared reading an equation $s \approx t$ as a multiset $\{s,t\}$ and using the extension of $\succ$ to multisets, which is also denoted by $\succ$. To compare (possibly disjunctive, ground) positive program rules, it is sufficient to define $(\mathcal{H} \leftarrow \mathcal{B}) \succ (\mathcal{H}' \leftarrow \mathcal{B}')$ iff $(\mathcal{H} \cup \mathcal{B}) \succ (\mathcal{H}' \cup \mathcal{B}')$, where $\succ$ again denotes its own extension to multisets.

An equation $s \approx t$ such that $s \succ t$ is also written as a rule[17] $s \to t$. It is well known that for any convergent rewrite system R, and any two terms s and t it holds $s \approx t \in R^{\approx}$ if and only if there is a term u such that $s \to_R^\star u$ and $t \to_R^\star u$.

We are now ready to define the rewrite system R. First, for every equation $s \approx t \in \mathcal{M}$ we define by induction on the term ordering $\succ$ sets of rewrite rules $\varepsilon_{s \approx t}$ and $R_{s \approx t}$ as follows. Assume that $\varepsilon_{s \approx t}$ has already been defined for all $s' \approx t' \in \mathcal{M}$ with $s \approx t \succ s' \approx t'$. Let $R_{s \approx t} = \bigcup_{s \approx t \succ s' \approx t'} \varepsilon_{s' \approx t'}$ and define

$$\varepsilon_{s \approx t} = \{s \to t\} \text{ if } \begin{cases} s \succ t, \\ s \text{ is irreducible by } R_{s \approx t}, \text{ and} \\ t \text{ is irreducible by } R_{s \approx t}. \end{cases}$$

Otherwise $\varepsilon_{s \approx t} = \emptyset$. Finally let $R = \bigcup_{s \approx t} \varepsilon_{s \approx t}$.

By construction R has no critical pairs. Because $\succ$ is a well-founded ordering R thus is a convergent rewrite system.

Now we show by well-founded induction that that R is an E-model of $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$, or, equivalently, $R \models_{\approx} (\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$. Any rule in $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$ can be written as $(\mathcal{H}^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}})\gamma$, for some rule $C^{\mathrm{eq}} = (\mathcal{H}^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}}) \in \mathcal{P}^{\mathrm{eq}}$ and some ground substitution γ. Conversely, any such rule $C^{\mathrm{eq}}\mathcal{P}^{\mathrm{eq}}$ and any ground substitution γ is a rule in $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$. Thus chose any such rule C^{eq} and γ arbitrary.

We distinguish two complementary cases.

(1) γ *is reducible.*
Then there is a variable x in the domain of γ such that $x\gamma \to_R t$ for some term t. Let γ' be the substitution such that

$$y\gamma' = \begin{cases} t & \text{if } y = x \\ y\gamma & \text{otherwise} \end{cases}$$

Because x occurs in C^{eq} it follows $C^{\mathrm{eq}}\gamma \succ C^{\mathrm{eq}}\gamma'$. By the induction hypothesis $R \models_{\approx} C^{\mathrm{eq}}\gamma'$, and by congruence $R \models_{\approx} C^{\mathrm{eq}}\gamma$.

(2) γ *is irreducible.*
If $R \not\models_{\approx} \mathcal{B}^{\mathrm{eq}}\gamma$ then $R \models_{\approx} C^{\mathrm{eq}}\gamma$ follows trivially. Hence suppose $R \models_{\approx} \mathcal{B}^{\mathrm{eq}}\gamma$. Because γ is irreducible for C^{eq}, γ irreducible for every body literal in $\mathcal{B}^{\mathrm{eq}}$. From Lemma 3 $\mathcal{M} \models \mathcal{B}^{\mathrm{eq}}\gamma$ follows easily. By definition, $(\mathcal{P}^{\mathrm{eq}})_1^{\mathrm{gr}}$ includes the program rule $(H^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}})\gamma$ for

[17] In the sense as used in the term rewrite literature, not a *program* rule of a logic program.

some atom H^{eq}, where H^{eq} is among the head literals of $\mathcal{H}^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}}$. Recall that $\mathcal{M}$ is given as a minimal model of $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}_1$. From $\mathcal{M} \models \mathcal{B}^{\mathrm{eq}}\gamma$ it thus follows $\mathcal{M} \models H^{\mathrm{eq}}\gamma$, which is the same as to say $H^{\mathrm{eq}}\gamma \in \mathcal{M}$. Again by Lemma 3, this time in the other direction, it follows $R \models_{\approx} H^{\mathrm{eq}}\gamma$. This result implies trivially $R \models_{\approx} (\mathcal{H}^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}})\gamma$.

This concludes the case analysis. Notice that in both cases we have shown $R \models_{\approx} C^{\mathrm{eq}}\gamma$, which entails $R \models_{\approx} (\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$.

$R \models_{\approx} \mathcal{P}$. Let $C = (\mathcal{H} \leftarrow \mathcal{B}) \in \mathcal{P}$ and γ a ground substitution for C. It suffices to show $R \models_{\approx} C\gamma$.

Let $C^{\mathrm{eq}} = (\mathcal{H}^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}}) \in \mathcal{P}^{\mathrm{eq}}$ be the rule obtained from C by the equality transformation. The rules C and C^{eq} can be written as

$$\begin{aligned} C &= \mathcal{H}[\boldsymbol{s}] \leftarrow \mathcal{B}[\boldsymbol{t}] \\ C^{\mathrm{eq}} &= \mathcal{H}[\boldsymbol{x}^{\boldsymbol{s}}] \leftarrow \mathcal{B}[\boldsymbol{x}^{\boldsymbol{t}}], \mathrm{flatten}(\boldsymbol{s} \approx \boldsymbol{x}^{\boldsymbol{s}}), \mathrm{flatten}(\boldsymbol{t} \approx \boldsymbol{x}^{\boldsymbol{t}}) \ , \end{aligned}$$

where $\boldsymbol{s}$ ($\boldsymbol{t}$) are the terms occuring in $\mathcal{H}$ (in $\mathcal{B}$) that prevent the literals in $\mathcal{H}$ (in $\mathcal{B}$) from being flat. The variables $\boldsymbol{x}^{\boldsymbol{s}}$ are those that replace the terms $\boldsymbol{s}$ in the head literals by flattening. By $\mathrm{flatten}(\boldsymbol{s} \approx \boldsymbol{x}^{\boldsymbol{s}})$ the list of equations is meant that results from flattening the equations $s_1 \approx x_1^s, \ldots, s_n \approx x_n^s$, where $\boldsymbol{s} = s_1, \ldots, s_n$ and $\boldsymbol{x}^{\boldsymbol{s}} = x_1^s, \ldots, x_n^s$, for some $n \geq 0$. The expression $\mathrm{flatten}(\boldsymbol{t} \approx \boldsymbol{x}^{\boldsymbol{t}})$ is defined in the same way, as expected.

The equations $\mathrm{flatten}(\boldsymbol{s} \approx \boldsymbol{x}^{\boldsymbol{s}}), \mathrm{flatten}(\boldsymbol{t} \approx \boldsymbol{x}^{\boldsymbol{t}})$ can be written as $u_1 \approx x_1, \ldots, u_m \approx x_m$, for some $m \geq 0$, where $u_1, \ldots, u_m$ are (flat) terms and $x_1, \ldots, x_m$ are variables (pairwise different and fresh wrt. the variables in C). These equations can be seen as a unification problem in solved form. Now consider the substitution

$$\gamma' = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\} \ .$$

By inspection of the flattening process one convinces oneself that $\boldsymbol{s} = \boldsymbol{x}^{\boldsymbol{s}}\gamma'$ and $\boldsymbol{t} = \boldsymbol{x}^{\boldsymbol{t}}\gamma'$.[18] Thus we obtain

$$\begin{aligned} C^{\mathrm{eq}}\gamma' &= \mathcal{H}[\boldsymbol{x}^{\boldsymbol{s}}\gamma'] \leftarrow \mathcal{B}[\boldsymbol{x}^{\boldsymbol{t}}\gamma'], u_1\gamma' \approx x_1\gamma', \ldots, u_m\gamma' \approx x_m\gamma' \\ &= \mathcal{H}[\boldsymbol{s}] \leftarrow \mathcal{B}[\boldsymbol{t}], u_1\gamma' \approx x_1\gamma', \ldots, u_m\gamma' \approx x_m\gamma' \end{aligned}$$

Now apply the given substitution γ to $C^{\mathrm{eq}}\gamma'$ and obtain

$$\begin{aligned} C^{\mathrm{eq}}\gamma'\gamma &= (\mathcal{H}[\boldsymbol{s}] \leftarrow \mathcal{B}[\boldsymbol{t}], u_1\gamma' \approx x_1\gamma', \ldots, u_m\gamma' \approx x_m\gamma')\gamma \\ &= \mathcal{H}\gamma \leftarrow \mathcal{B}\gamma, u_1\gamma'\gamma \approx x_1\gamma'\gamma, \ldots, u_m\gamma'\gamma \approx x_m\gamma'\gamma \end{aligned}$$

With the result of the preceeding part conclude $R \models_{\approx} C^{\mathrm{eq}}\gamma'\gamma$. Because $u_i\gamma' = x_i\gamma'$, for all $i = 1, \ldots, m$, it follows trivially $u_i\gamma'\gamma = x_i\gamma'\gamma$ and $R \models_{\approx} u_i\gamma'\gamma \approx x_i\gamma'\gamma$. But then $R \models_{\approx} \mathcal{H}\gamma \leftarrow \mathcal{B}\gamma$ follows, which was to show.

R *satisfies UNA.* Suppose, to the contrary, that R does not satisfy UNA, i.e. $R \models_{\approx} c \approx d$ for some different constants c and d. The equation $c \approx d$ is flat. By Lemma 3 then $c \approx d \in \mathcal{M}$. With $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}_1$ containing the rule $\mathsf{false} \leftarrow c, d$ it follows $\mathsf{false} \in \mathcal{M}$. But then $\mathcal{M}$ is no interpretation, a contradiction to the fact that $\mathcal{M}$ was given as a possible model (of $\mathcal{P}^{\mathrm{eq}}$). □

[18] The somewhat tedious formal proof would not provide any additional insights.

Lemma 3. *Let $s \approx t$ be a flat equation and γ a ground substitution irreducible for $s \approx t$. Then, $R \models_{\approx} s\gamma \approx t\gamma$ iff $s\gamma \approx t\gamma \in \mathcal{M}$.*

Proof. For the if-direction suppose $s\gamma \approx t\gamma \in \mathcal{M}$.

If $s\gamma = t\gamma$ then $R \models_{\approx} s\gamma \approx t\gamma$ follows trivially.

If both $s\gamma$ and $t\gamma$ are irreducible[19] then $\varepsilon_{s\gamma \approx t\gamma} = \{s\gamma \to t\gamma\}$ and so $s\gamma \to t\gamma \in R$. From $s\gamma \to t\gamma \in R$ the result $R \models_{\approx} s\gamma \approx t\gamma$ follows easily.

Hence suppose, without loss of generality that $s\gamma$ is reducible. Recall frist that $\mathcal{M}$ is a possible model for a program that was obtained by the equality transformation. Any such program contains, by construction, the program rules $\mathsf{false} \leftarrow c, d$ for any pair of different constants c and d. Hence, $\mathcal{M}$ cannot contain any equation $c \approx d$. Nor can it contain $d \approx c$ because the equality transformation adds a program rule for symmetry. Notice that consequently R does not contain $c \approx d$ or $d \approx c$ either (because of $R \subseteq \mathcal{M}$). Together with the restriction (1) on orderings defined above it follows that R does not contain any rule of the form $c \to t$, where c is a constant and t is any term. In other words, constants are irreducible by R.

By this consideration, $s\gamma$ cannot be a constant. The term $s\gamma$ thus is of the form $f(\boldsymbol{v})\gamma$ where f is some (possibly nullary) function symbol and $\boldsymbol{v}$ is some list of terms. More specifically, because $s \approx t$ is given as a flat equation, each term v in $\boldsymbol{v}$ must be a constant or a variable. Now, if v is a constant then $v = v\gamma$ is irreducible, as just concluded. And if v is a variable then $v\gamma$ is irreducible, too, because γ is given as irreducible for $s \approx t$.

Recall we are considering the case that $s\gamma = f(\boldsymbol{v})\gamma$ is reducible. Because $v\gamma$ is irreducible, for each v in $\boldsymbol{v}$, $f(\boldsymbol{v})\gamma$ must be reducible at the top position. That is, R must contain a rule of the form $s\gamma \to u$, for some term u.

From $s\gamma \to u \in R$ it follows $\varepsilon_{s\gamma \approx u} = \{s\gamma \to u\}$. By definition of ε, $R_{s\gamma \approx u}$ cannot contain a rule that rewrites u. Further, the ordering $\succ$ on equations is defined in such a way that any rule that could rewrite u must precede the rule $s\gamma \to u$. Together, thus, u is irreducible. In other words, deriving the normal form of $s\gamma$ takes exactly one step. Notice this fact is independent from whether $s\gamma \approx t\gamma \in \mathcal{M}$ or not. It holds for any flat term s and irreducible substitution γ. This result will be used also in the proof of the only-if direction below.

Next consider $t\gamma$. If $t\gamma$ is reducible then by the same arguments as for $s\gamma$ it must be of the form $g(\boldsymbol{w})\gamma$ where g is some function symbol and $\boldsymbol{w}$ is a list of constants or variables. Further, there is a rule $t\gamma \to u' \in R$ for some irreducible term u'.

Recall that any program obtained from the equality transformation contains the axioms of reflexivity, symmetry and transitivity. Further recall that $\mathcal{M}$ is a model of some such program.

Because of $R \subseteq \mathcal{M}$, from $s\gamma \to u \in R$ and $t\gamma \to u' \in R$ it follows $s\gamma \approx u \in \mathcal{M}$ and $t\gamma \approx u' \in \mathcal{M}$. The symmetric versions are also contained in $\mathcal{M}$ by the symmetry axioms.

Because $s\gamma \approx t\gamma \in \mathcal{M}$, $s\gamma \approx u \in \mathcal{M}$ and $t\gamma \approx u' \in \mathcal{M}$ and the fact that $\mathcal{M}$ must be a model in particular for the symmetry and transitivity axioms it follows $u \approx u' \in \mathcal{M}$.

Next we will show that $u = u'$. Suppose, to the contrary that u and u' are different terms. But then, either $u \succ u'$ or $u' \succ u$ holds. Without loss of generality suppose $u \succ u'$.

[19] As usual, we say that a term s is *reducible (by R)* iff there is a term t such that $s \to_R t$, otherwise s is *irreducible (by R)*.

Recall that both u and u' are irreducible. But then $\varepsilon_{u \approx u'} = \{u \to u'\}$ and so $u \to u' \in R$, contradicting the irreducibility of u. Hence it follows $u = u'$. Consequently we have $t\gamma \to u \in R$. Together with $s\gamma \to u \in R$ it follows trivially $s\gamma \to_R u$ and $t\gamma \to_R u$. Because R is convergent it follows $R \models_\approx s\gamma \approx t\gamma$ as desired.

The last open case, that $t\gamma$ is irreducible, is treated similarly: from $s\gamma \approx t\gamma \in \mathcal{M}$ and $s\gamma \approx u \in \mathcal{M}$ it follows by the symmetry and transitivity axioms that $t\gamma \approx u \in \mathcal{M}$. By the same arguments as above it must hold $t\gamma = u$ (because both terms are irreducible, and if they were different, a rule $t\gamma \to u$ or $u \to t\gamma$ would have been added to R, contradicting irreducibility of $t\gamma$ and of u. Thus, with $s\gamma \to u \in R$ and $t\gamma = u$ it follows $R \models_\approx s\gamma \approx t\gamma$.

This completes the proof for the if-direction.

For the only-if direction suppose $R \models_\approx s\gamma \approx t\gamma$. Because R is a convergent rewrite system there is a term w such that $s\gamma \to_R^\star w$ and $t\gamma \to_R^\star w$.

If both $s\gamma$ and $t\gamma$ are irreducible then $s\gamma = t\gamma$ so $s\gamma \approx t\gamma$ is an instance of the reflexivity axiom, and so $s\gamma \approx t\gamma \in \mathcal{M}$ follows.

Hence suppose that $s\gamma$ or $t\gamma$ is reducible. Without loss of generality suppose $s\gamma$ is reducible. By exactly the same arguments as made in the proof of the if-direction, $s\gamma$ can only be rewritable at the top position. Thus, there is a rule of the form $s\gamma \to u \in R$. In the if-part of the proof we concluded that deriving the normal form of $s\gamma$ takes exactly one step. This implies $u = w$.

If $t\gamma$ is reducible, by the same arguments as for $s\gamma$, there is a rule of the form $s\gamma \to u' \in R$ with $u' = w$. Because $R \subseteq \mathcal{M}$ we get $s\gamma \approx w \in \mathcal{M}$ and $t\gamma \approx w \in \mathcal{M}$. By the symmetry and transitivity axioms, $\mathcal{M}$ must also satisfy $s\gamma \approx t\gamma$ and $t\gamma \approx s\gamma$. Equivalently, $s\gamma \approx t\gamma \in \mathcal{M}$ and $t\gamma \approx s\gamma \in \mathcal{M}$.

If $t\gamma$ is irreducible, we have $t\gamma = w$. From $s\gamma \to w \in R$, $R \subseteq \mathcal{M}$ and $t\gamma = w$ it follows (with the symmetry axiom) $s\gamma \approx t\gamma \in \mathcal{M}$ and $t\gamma \approx s\gamma \in \mathcal{M}$. □