Model-Generation Theorem Proving for First-Order Logic Ontologies

Peter Baumgartner and Fabian M. Suchanek

Max-Planck Institute for Computer Science, Saarbrücken, Germany
[baumgart|suchanek]@mpi-sb.mpg.de

Abstract. Formal ontologies play an increasingly important role in demanding knowledge representation applications like the *Semantic Web*. Automated reasoning support for these ontologies is mandatory for tasks like debugging, classifying or querying the knowledge bases, and description logic (DL) reasoners have been shown to be very effective for that. Yet, as language extensions beyond (decidable) DLs are being discussed, more general first-order logic systems are required, too. In this paper, we pursue this direction and consider automated reasoning on full first-order logic knowledge bases. We put forward an optimized approach of transforming such knowledge bases to clause logic. The transformations include a Brand-like transformation to eliminate equality, and a transformation that incorporates a *blocking* technique to "checks loops" in derivations. The latter transformation lets theorem provers terminate more often on satisfiable input formulas. It thus enables more robust automated reasoning support on ontologies, where disproving is a common task. While the transformations are applicable to any clause set, we concentrate in this paper on demonstrating their effectiveness on a standard test suite devised by the Semantic Web community.

1 Introduction

Recent years have seen an increasing interest in formal ontologies. In particular, the vision of the *Semantic Web* requires the capability of handling huge formal knowledge bases by means of automated reasoning. To this end, the Semantic Web community has been developing a series of formal ontology languages, with OWL being the most recent one. OWL comes in three different versions - OWL-lite, OWL-DL and OWL-full - with increasing expressiveness. For OWL-DL, which is designed to be equivalent to the description logic language $\mathcal{SHOIN}(\mathcal{D}_n^-)$, efficient automated reasoning support is already available [1]

Yet it becomes clear that many applications (in the Semantic Web context or in general) demand for extensions of DLs that go beyond decidability. For instance, one of the currently discussed extensions is the integration of logic programming style *rules*, which prove to be necessary even for basic knowledge representation tasks [GHVD03].

However, even simple rule languages are expressive enough to express role-value maps, which are known to cause undecidability in conjunction even with basic description logic languages. But then, if one is willing to accept undecidability, it becomes a

[1] For instance FaCT [Hor99] and RACER [HM01].

viable option to transform the knowledge base to first order logic altogether and apply a general purpose first-order theorem prover.

Some efforts already have been been made to apply first-order theorem provers for ontological reasoning. The web-site [OWL04] summarizes comparative results of various systems applied to OWL knowledge bases (see Section 5 for more details). Of the systems tested, the perhaps most advanced system built on top of a theorem prover is the Hoolet system, which uses Vampire [RV01a]. It was demonstrated in [TRBH04] that this approach indeed can compete with dedicated DL reasoner like FaCT.

Yet, one of the conclusions from the available experimental results is that there is still a need for improving the approach based on theorem provers. Specifically, and not surprisingly, the provers typically fall behind for satisfiable input formulas. This is because general first order theorem provers do not automatically act as *decision procedures* for common description logics (at least with the standard relational translation[2]).

From a technical viewpoint, addressing this weakness is the main topic of this paper: to improve the termination behavior of first-order theorem provers on satisfiable input formulas, with a focus on knowledge base applications. As our approach is based on transformations on the given formula, no modifications of the provers are necessary. The obvious advantage over tailoring a dedicated system then is that it enables the application of available high-performance first-order theorem provers.

While in principle any sound and (refutational) complete first-order prover can be applied in conjunction with our transformation, we concentrate on provers that return *models* of satisfiable input formulas in case of termination. These provers have the advantage that, in a refutational setting, a model provides a counterexample to a conjectured entailment or to a conjectured concept subsumption that does not hold. We will refer to such systems as *model generation procedures*. Examples include systems basing on hyper-resolution [FLHT01], as for instance SATCHMO [MB88], the Hyper Tableaux [BFN96] prover KRHyper [Wer03], or instance based methods, like the Model Evolution [BT03] system Darwin [BFT05]

Many ontologies (such as for instance large parts of SUMO/MILO [NP01]) are given in first order logic. Most others, including all OWL ontologies, can be translated to first order logic (see [BCM$^+$02]). Furthermore, it is well known that most common reasoning tasks like knowledge base satisfiability, subsumption tests between two given concept descriptions or concept retrieval can be formulated easily as first-order logic satisfiability or unsatisfiability problems (see again [BCM$^+$02]). Thus, first order logic seems a very general input format. Since first order logic can be transformed efficiently to clause logic [NRW98] and most state-of-the-art automated theorem provers accept (only) clause logic as their input language, we assume that all reasoning problems are given in clause logic.

This paper makes two contributions: first, we present a method that eliminates equality from a given clause logic set. This is important because most model generation procedures do not include inference rules for equality. Our transformation is a

[2] There exist advanced techniques available for translating description logics to clausal logic [SH03,dNHS00]. These methods draw their sophistication on exploiting specific properties of modal logics. It seems not obvious how to extend these methods to general first-order logic specifications, the setting we are interested in.

simple one, but we will argue why it is actually better suited for typical ontological reasoning problems than Brand's transformation or the improved one in [BGV98].

Our second contribution is a transformation that adds a "loop check" into the clause set. More precisely, the clause set obtained in the first step is transformed in such a way that model generation procedures applied to it will search in preference for a finite Herbrand-model of the transformed input clause set. Thus, it enables theorem provers to terminate more often on satisfiable input problems.

Our main theoretical results are the soundness and the completeness of our transformations. On the practical side, we show that our approach compares favorably with others on a standard test suite of OWL description logic problems [OWL04]. Moreover, we show that exploiting *non-monotonic negation* can lead to additional improvements on larger knowledge bases.

2 Preliminaries

We use standard terminology from automated reasoning. We assume as given a signature Σ of constant symbols, and function symbols and predicate symbols of given arities. As we are working with equality, we assume Σ contains a distinguished binary predicate symbol $\approx$, which is written infix. As the only non-standard definition, we distinguish between 0-ary function symbols and constants (see UNA below). But otherwise the terms, atoms, literals and formulas over Σ and a given (denumerable) set of variables V are defined as usual.

A clause is a (finite) implicitly universally quantified disjunction of literals, as usual. We write clauses in a logic-programming style notation $H_1 \vee \cdots \vee H_m \leftarrow B_1, \ldots, B_k$, where $m, k \geq 0$, corresponding to the disjunction $H_1 \vee \cdots \vee H_m \vee \neg B_1 \vee \cdots \vee \neg B_k$. Each H_i is called a *head atom*, and each B_j is called a *body atom*. When writing expressions like $H \vee \mathcal{H} \leftarrow B, \mathcal{B}$ we mean any clause whose head literals are H and those in the disjunction of literals $\mathcal{H}$, and whose body literals are B and those in the list of literals $\mathcal{B}$. A *clause set* is a finite set of clauses.

A *(Herbrand) interpretation* I is a set of ground atoms—those that are true in the interpretation. Satisfiability/validity of ground literals, clauses, and clause sets in a Herbrand interpretation is defined as usual. Also, as usual, a clause set stands semantically for the set of all its ground instances. We write $I \models F$ to denote the fact that I satisfies F, where F is a ground literal or a (possibly non-ground) clause (set). An *E-interpretation* is an interpretation that is also a congruence relation on the ground terms and ground atoms.[3] If I is an interpretation, we denote by I^{E} the smallest congruence relation on the ground terms and ground atoms that includes all equations in I, which is an E-interpretation. We say that I *E-satisfies* F iff $I^{\mathrm{E}} \models F$. Instead of $I^{\mathrm{E}} \models F$ we write $I \models_{\mathrm{E}} F$. We say that F *E-entails* F', written $F \models_{\mathrm{E}} F'$, iff every E-interpretation that satisfies F also satisfies F'. We say that F and F' are *E-equivalent* iff $F \models_{\mathrm{E}} F'$ and $F' \models_{\mathrm{E}} F$.

A *UNA-E-interpretation* is an E-interpretation that does not contain the equation $c \approx d$, for any different constants c and d (we say it "satisfies the unique name assump-

[3] We mean that for any E-interpretation I and ground atom A: whenever $I \models A[s]$ and $I \models s \approx t$, then $I \models A[t]$.

tion (UNA)").[4] In other words, $c \approx d \notin I^{\mathrm{E}}$ for any UNA-E-interpretation I. We consider the UNA because it seems useful in the context of KBs coupled with databases, where different constants are usually meant to stand for different things. However, we allow constants to be declared as nullary functions, so that our approach below is fully compatible with the standard semantics.

3 Equality Transformation – Simple is Better

First-order knowledge bases typically make use of equality. For example, equality is used to define function results, to define that two objects are different or to state that two objects must be equal under certain circumstances. Furthermore, the translation of certain DL constructs to first order logic introduces equality. Equality comes in, e.g., for DL number restrictions, as in this formula from the Tambis Ontology [Tam]:

$$\mathsf{Cation} \sqsubseteq \leq 4\ \mathsf{hasCharge}$$

becomes

$$x_1 \approx x_2 \vee x_1 \approx x_3 \vee \cdots \vee x_4 \approx x_5 \leftarrow \mathsf{Cation}(x), \mathsf{hasCharge}(x,x_1), \ldots, \mathsf{hasCharge}(x,x_5) \quad (\star)$$

Numerous other DL constructs translate to equality constraints, the most prominent being concept disjunctions ("union" in OWL) or extensional concept definitions ("oneOf" in OWL, "nominals" in DLs).

In contrast to the resolution calculus, where efficient methods for the treatment of equality have been developed [BG98], most model generation procedures do not include built-in treatment of equality. One option then is to use equality axioms. However, as it is well-known, the search space induced by the resulting clause set is prohibitively high. Even worse, achieving termination of a model-generation system on satisfiable clause sets is practically impossible then. The most cumbersome axioms in this regard are the functional substitution axioms, like $f(x) \approx f(y) \leftarrow x \approx y$. Another option is to "compile away" equality. The probably most well-known method in this direction is the *STE-transformation* in [Bra75], which was later improved in [MS97,BGV98]. Our approach is similar, but it is specifically tailored to ontological reasoning: First, it supports the Unique Name Assumption (see Section 2), and, second it performs much better on typical ontological problems.

Definition 1 (Flat Term). *A* flat basic *term is a constant or a variable. A term is* flat *iff it is a flat basic term or a function term $f(t_1,\ldots,t_n)$, where f is a n-ary function symbol and $t_1,\ldots,t_n$ are flat basic terms. An atom $P(t_1,\ldots,t_n)$ is* flat *iff all terms $t_1,\ldots,t_n$ are flat. A literal is* flat *iff its atom is flat, and a clause is* flat *iff all of its literals are flat.*

For instance if c is a constant and a is a 0-ary function symbol, then $P(x,c,a)$, $P(f(x),f(c))$ and $f(x) \approx g(c)$ are flat, while, say, $P(f(a),f(c))$ and $f(a) \approx g(c)$ are not.

[4] Actually, this is a slight misnomer.

Any clause set can be transformed to a set of flat clauses by "pulling out" offending subterms. This is achieved by applying the following transformation rules to a given clause set as long as possible:

1. Replace a clause of the form

$$P(t_1,\ldots,f(s_1,\ldots,s_i,\ldots,s_m),\ldots,t_n) \vee \mathcal{H} \leftarrow \mathcal{B}$$
$$\text{by} \qquad P(t_1,\ldots,f(s_1,\ldots,x,\ldots,s_m),\ldots,t_n) \vee \mathcal{H} \leftarrow \mathcal{B}, x \approx s_i$$

   if s_i is not a flat basic term, where x is a fresh variable.
2. Replace a clause of the form

$$\mathcal{H} \leftarrow \mathcal{B}, P(t_1,\ldots,f(s_1,\ldots,s_i,\ldots,s_m),\ldots,t_n)$$
$$\text{by} \qquad \mathcal{H} \leftarrow \mathcal{B}, P(t_1,\ldots,f(s_1,\ldots,x,\ldots,s_m),\ldots,t_n), x \approx s_i$$

   if s_i is not a flat basic term, where x is a fresh variable.

It is obvious that for any clause set this *flattening* terminates with a uniquely determined set of flat clauses (up to renaming of variables and ordering of body atoms).

As an example consider the clause $P(g(y),g(a),g(c)) \vee f(g(y),c) \approx y \leftarrow P(y,g(b))$. Assume that c is a constant and a and b are 0-ary function symbols. Flattening then results in the clause

$$P(g(y),g(x),g(c)) \vee f(x',c) \approx y \leftarrow P(y,g(x'')), x \approx a, x' \approx g(y), x'' \approx b$$

The purpose of flattening is to achieve the effect of the function substitution axioms. Notice that, unlike 0-ary function symbols, constants are not "pulled out".

Definition 2 (Equality Transformation). *Let $\mathcal{P}$ be a Σ-clause set. The* equality transformation *of $\mathcal{P}$, denoted as $\mathcal{P}^{eq}$, is the clause set obtained by flattening of $\mathcal{P}$ and by adding the following clauses:*

$$\leftarrow c \approx d \qquad \textit{for any two different } \Sigma\textit{-constants } c \textit{ and } d$$
$$x \approx x \leftarrow$$
$$x \approx y \leftarrow y \approx x$$
$$x \approx z \leftarrow x \approx y, y \approx z$$
$$P(x_1,\ldots,y,\ldots,x_n) \leftarrow P(x_1,\ldots,x_i,\ldots,x_n), y \approx x_i$$
$$\textit{for each } n\textit{-ary predicate symbol } P \textit{ from } \Sigma \textit{ different from } \approx, \textit{ and all } i \textit{ with } 1 \leq i \leq n$$

When ignoring the dis-equality axioms $\leftarrow c \approx d$, the only difference between the equality transformation and the axiomatic equality treatment lies in the function substitution axioms, i.e. axioms of the form $f(x_1,\ldots,y,\ldots,x_n) \approx f(x_1,\ldots,x_i,\ldots,x_n) \leftarrow y \approx x_i$, for each n-ary Σ-function symbol and all $i = 1,\ldots,n$. Due to flattening, these axioms can be dispensed with.

The difference might seem negligible, but it isn't: for instance, a unit clause $a \approx b \leftarrow$ together with the axiom $f(y) \approx f(x_1) \leftarrow y = x_1$ will cause non-termination of model generation procedures. This is avoided with the equality transformation.[5]

[5] We also have some preliminary experimental evidence for that: of the 236 *satisfiable* problems in the NLP category ("Natural Language Processing") of the TPTP Library [SSY94], 156 are

Relation to Other Transformations. One difference between our transformation and the STE-transformation [Bra75], as well as the improved ones in [MS97,BGV98] comes from the use of constants with their UNA semantics and the fact that our transformation does not eliminate the predicate substitution axioms.

For instance, the clause $\leftarrow P(f(x), f(c))$ is flat, and hence our transformation leaves it unchanged. By contrast, its STE-transformation yields the clause $\leftarrow P(x_1, x_2), f(x) \approx x_1, f(x_3) \approx x_2, c \approx x_3$.[6] Now consider the clause in conjunction with a collection of facts $P(f(c_1), f(d)), \ldots, P(f(c_n), f(d))$, for some (large) value of n and constants $c_1, \ldots, c_n$ and d. Notice that no $P(f(c_i), f(d))$, for any $i = 1, \ldots, n$, unifies with $P(f(x), f(c))$. As the clause $\leftarrow P(f(x), f(c))$ and each fact $P(f(c_i), f(d))$ is flat, satisfiability of this clause set can be detected quickly, with n (failed) unification attempts. In contrast, satisfiability of the clause set with the STE-transformation applied may take $O(n^3)$ unification attempts. This is, because the first three atoms $P(x_1, x_2), f(x) \approx x_1, f(x_3) \approx x_3, c \approx x_3$ of the body of the STE-transformed clause don't show a constraining effect on unification. Preliminary practical experiments we ran demonstrate that a noticeable difference already shows up with $n = 10000$.

A polynomial speedup might seem negligible. Yet, there are situations where this makes a practically relevant difference. Conceivable are database-like applications with many logically simple queries but large A-Boxes (facts).

The STE-transformation, like the others mentioned, differs from our transformation in that it not only eliminates the need for the function substitution axioms, but also the symmetry, transitivity and predicate substitution axioms (stated in Definition 2). However, dispensing with these axioms comes at the price of more equations in the transformed clauses – exponentially many in the worst case: in both Brand's transformation and the improved ones in [MS97,BGV98], each occurrence of a positive equation $s \approx t$ in a clause gives rise to a clause with the symmetric version $t \approx s$ instead of $s \approx t$. As a result, a clause with n equations produces 2^n clauses.

The effect of the different transformations on the search can be explained from a tableaux perspective. Consider a ground clause $s_1 \approx t_1 \vee \cdots \vee s_n \approx t_n \leftarrow$ with its 2^n symmetric versions. Any exhaustive application of the β-rule to these clauses even under the regularity condition, which should be assumed,[7] will result in

$$k(n) = \sum_{i=0}^{n-1} \frac{n!}{i!} \cdot (n-i)$$

branches. For instance, $k(2) = 6$, $k(4) = 196$, $k(6) = 9786$, $k(8) = 767208$ and $k(10) = 88776910$.

By contrast, with our transformation only n-fold branching will occur, and each leaf $s_i \approx t_i$ will be extended with its symmetric version $t_i \approx s_i$ without additional branching.

---

solvable by the Darwin prover [BFT05] with an axiomatic treatment of equality, while with the equality transformation strictly more are solvable, 167. More tests are in preparation.

[6] In fact, Brand's modification would introduce further body atoms, but the transformations [MS97,BGV98] don't.

[7] The regularity condition of clausal tableaux forbids to derive a branch where two or more nodes are labelled with the same literal [LS01]. It is practically very effective.

For instance, the transformation of the "description-logic" problem "inconsistent022" from the OWL test suite (see Section 5) contains a clause with 10 positive equations and becomes unsolvable due to this effect. With our transformation it takes about 20 seconds, and with an axiomatic treatment of equality 44 seconds to find a refutation.

We do not claim that our transformation is always superior to the STE-transformation or the transformation in [BGV98]. In fact, the transformations in [Bra75,BGV98] are theoretically much more sophisticated. They enjoy the desirably property that derivations with the resulting clause sets, in terms of resolution, avoid paramodulation into variables. This property cannot be guaranteed in presence of the symmetry axiom of equality, which is included in our transformation. Yet, the advantages discussed above typically apply for clause sets with several disjunctions of positive equations, which is not uncommon for practical knowledge bases due to number restrictions and certain other language constructs; it is perhaps not so common in "mathematical" theorem proving, where indeed the other mentioned transformations may well be superior.

The following theorem is the main result of this section. It expresses that the equality transformation is complete with respect to UNA-E models.

Theorem 3 (Soundness and Completeness of the Equality Transformation). *Let $\mathcal{P}$ be a clause set. Then $\mathcal{P}$ is UNA-E-satisfiable if and only if $\mathcal{P}^{eq}$ is satisfiable.*

Thus, any sound and complete theorem prover provides a method for checking UNA-E-satisfiability. The only-if direction (soundness) is easy and nothing essentially new compared to the results in [BGV98,Bra75]. The if direction (completeness) would follow easily from the results in [BGV98,Bra75], were it not for the UNA. This proof can be found in the appendix.

4 Blocking

In this section we define a transformation that lets model generation procedures terminate in more cases on satisfiable input clauses than without it. This *blocking transformation* transforms the given clause set so that it "encodes" the search for certain finite models.

As an example to illustrate the main idea consider the following excerpt from the Tambis knowledge base [Tam]

$$\begin{aligned} \mathsf{AuthoredChapter} &\sqsubseteq \exists\, \mathsf{partOf}\,.\,\mathsf{CollectionBook} \\ \mathsf{CollectionBook} &\sqsubseteq \exists\, \mathsf{hasPart}\,.\,\mathsf{AuthoredChapter} \qquad (\star\star) \end{aligned}$$

Additionally, hasPart is declared both as a transitive rôle and as the inverse of partOf.

The standard relational transformation, which we suppose having been carried out, yields the clauses

$$\begin{aligned} \mathsf{partOf}(x, f_{\mathsf{partOf}}(x)) &\leftarrow \mathsf{AuthoredChapter}(x) && \text{(AuthoredChapter-1)} \\ \mathsf{CollectionBook}(f_{\mathsf{partOf}}(x)) &\leftarrow \mathsf{AuthoredChapter}(x) && \text{(AuthoredChapter-2)} \\ \mathsf{partOf}(x, f_{\mathsf{hasPart}}(x)) &\leftarrow \mathsf{CollectionBook}(x) && \text{(CollectionBook-1)} \end{aligned}$$

$$\mathsf{AuthoredChapter}(f_{\mathsf{hasPart}}(x)) \leftarrow \mathsf{CollectionBook}(x) \qquad \text{(CollectionBook-2)}$$
$$\mathsf{partOf}(y,x) \leftarrow \mathsf{hasPart}(x,y) \qquad \text{(hasPart-inv-1)}$$
$$\mathsf{hasPart}(y,x) \leftarrow \mathsf{partOf}(x,y) \qquad \text{(hasPart-inv-1)}$$
$$\mathsf{hasPart}(x,z) \leftarrow \mathsf{hasPart}(x,y), \mathsf{hasPart}(y,z) \qquad \text{(hasPart-trans)}$$

It is easy to see that adding a fact, say, $\mathsf{AuthoredChapter}(\mathsf{a}) \leftarrow$ will render the (minimal) Herbrand model infinite. Indeed, model generation procedures will not terminate then. The model will include, e.g. $\mathsf{AuthoredChapter}(\mathsf{a})$, $\mathsf{CollectionBook}(f_{\mathsf{partOf}}(\mathsf{a}))$, $\mathsf{AuthoredChapter}(f_{\mathsf{hasPart}}(f_{\mathsf{partOf}}(\mathsf{a})))$ and so on. Now, the idea behind our transformation is to prefer avoiding the generation of the underlying infinite Herbrand base by speculating mappings between new Herbrand universe candidates and already present members of the Herbrand base. Technically, this is achieved by means of a domain predicate, the extension of which represents the current domain of an interpretation.

The transformation of the clause set is such that, in the example, model generation procedures will terminate with a finite domain, $\mathsf{dom}(\mathsf{a})$ and the mappings $f_{\mathsf{partOf}}(\mathsf{a}) \mapsto \mathsf{a}$ and $f_{\mathsf{hasPart}}(\mathsf{a}) \mapsto \mathsf{a}$. Together with the additional, implicit mapping $\mathsf{a} \mapsto \mathsf{a}$ a *non-Herbrand* interpretation results, the domain of which consists of those terms that are specified by the domain predicate, which is just a in this case.

Should the speculation of a mapping like $f_{\mathsf{hasPart}}(\mathsf{a}) \mapsto \mathsf{a}$ have not been successful, in the sense it does not lead to a model, then extended domains are tried. Such a domain could be, for instance, one that includes two elements denoted by $\mathsf{dom}(\mathsf{a})$ and $\mathsf{dom}(f_{\mathsf{partOf}}(\mathsf{a}))$.

Definition 4 (Blocking Transformation). *Let $\mathcal{P}$ be a flat Σ-clause set. The* blocking transformation *of $\mathcal{P}$, denoted as $\mathcal{P}^{bl}$, is obtained in the following four steps applied in this order:*[8]

(1) Domain restriction: *Replace every rule $\mathcal{H} \leftarrow \mathcal{B}$ of $\mathcal{P}$ by the rule*

$$\mathcal{H} \leftarrow \mathcal{B}, \mathsf{dom}(x_1), \ldots, \mathsf{dom}(x_k) \tag{1}$$

where $\{x_1, \ldots, x_k\}$ is the set of variables occurring in $\mathcal{H} \leftarrow \mathcal{B}$, for some $k \geq 0$.

(2) Pulling out function terms: *In the resulting clause set, replace as long as possible each clause of the form $\mathcal{H} \leftarrow \mathcal{B}, P(t_1, \ldots, f(s_1, \ldots, s_n), \ldots, t_m)$ where f is a non-0 arity Σ-function symbol with*

$$\mathcal{H} \leftarrow \mathcal{B}, P(t_1, \ldots, x, \ldots, t_m), f(s_1, \ldots, s_n) \mapsto_{\text{ref}} x \tag{2}$$

where x is a variable not occurring elsewhere in the original clause. Finally to this step add the clauses

$$x \mapsto_{\text{ref}} x \leftarrow \tag{3}$$
$$x \mapsto_{\text{ref}} y \leftarrow x \mapsto y \tag{4}$$

[8] The (infix) predicate symbols $\mapsto$, $\mapsto_{\text{ref}}$ and $\mapsto_{\text{sub}}$ are assumed to be different to the ones in $\mathcal{P}$.

(3) Finite domain search: *Add to the resulting clause set the following clauses, for every* Σ*-constant* c*, for every n-ary* Σ*-function symbol* f *and all* $i = 1,\ldots,n$*, and for every m-ary* Σ*-predicate symbol* P *and all* $j = 1,\ldots,m$*:*

$$\mathsf{dom}(c) \leftarrow \tag{5}$$

$$\mathsf{dom}(x_i) \leftarrow \mathsf{dom}(f(x_1,\ldots,x_n)) \tag{6}$$

$$\mathsf{dom}(x_i) \leftarrow \mathsf{dom\_candidate}(f(x_1,\ldots,x_n)) \tag{7}$$

$$\mathsf{dom\_candidate}(f(x_1,\ldots,x_n)) \leftarrow \mathsf{dom}(x_1),\ldots,\mathsf{dom}(x_n) \tag{8}$$

$$f(x_1,\ldots,x_n) \mapsto_{\text{sub}} x_1 \vee \cdots \vee f(x_1,\ldots,x_n) \mapsto_{\text{sub}} x_n \vee \mathsf{dom}(f(x_1,\ldots,x_n)) \leftarrow \mathsf{dom\_candidate}(f(x_1,\ldots,x_n)) \tag{9}$$

$$x \mapsto c \leftarrow x \mapsto_{\text{sub}} c \tag{10}$$

$$x \mapsto f(x_1,\ldots,x_n) \vee x \mapsto_{\text{sub}} x_1 \vee \ldots \vee x \mapsto_{\text{sub}} x_n \leftarrow x \mapsto_{\text{sub}} f(x_1,\ldots,x_n) \tag{11}$$

$$P(x_1,\ldots,x_{j-1},y,x_{j+1},\ldots,x_m) \leftarrow x_j \mapsto y, P(x_1,\ldots,x_m) \tag{12}$$

$$\leftarrow x \mapsto y, \mathsf{dom}(x) \tag{13}$$

(4) Right uniqueness of $\mapsto$ *Add to the resulting clause set the following clauses, for every n-ary* Σ*-function symbol* f *and all* $i = 1,\ldots,n$*:*

$$\leftarrow x \mapsto y, x \mapsto z, y \neq z \tag{14}$$

$$x \neq y \leftarrow y \neq x \tag{15}$$

$$c \neq d \leftarrow \quad \textit{for any two different } \Sigma\textit{-constant } c \textit{ and } d \tag{16}$$

$$c \neq f(x_1,\ldots,x_n) \leftarrow \tag{17}$$

$$g(y_1,\ldots,y_k) \neq f(x_1,\ldots,x_n) \leftarrow \quad \textit{for every k-ary } \Sigma\textit{-function symbol } g \textit{ different from } f \tag{18}$$

$$f(x_1,\ldots,x_{i-1},y,x_{i+1},\ldots,x_n) \neq f(x_1,\ldots,x_n) \leftarrow y \neq x_i, \mathsf{dom}(f(x_1,\ldots,x_{i-1},y,x_{i+1},\ldots,x_n)), \mathsf{dom}(f(x_1,\ldots,x_n)) \tag{19}$$

Let us add some explanations. In step (1), adding the dom body atoms in clauses (1) is the key to achieve termination of model generation procedures. For illustration let us return to the example above. Model generation procedures will derive $\mathsf{dom}(\mathsf{a})$ and $\mathsf{CollectionBook}(f_{\mathsf{partOf}}(\mathsf{a}))$, but will not derive $\mathsf{dom}(f_{\mathsf{partOf}}(\mathsf{a}))$ from the transformed clause set. The blocking transformation will turn the clause CollectionBook-2 into

$$\mathsf{AuthoredChapter}(f_{\mathsf{hasPart}}(x)) \leftarrow \mathsf{CollectionBook}(x), \mathsf{dom}(x)$$

and model generation procedures will *not* "apply" this clause now to derive the new fact $\mathsf{AuthoredChapter}(f_{\mathsf{hasPart}}(f_{\mathsf{partOf}}(\mathsf{a})))$.

In step (2) function terms are "pulled out". This is necessary to replace a term $f(s_1,\ldots,s_m)$ in a body atom by a term s in presence of a mapping $f(s_1,\ldots,s_m) \mapsto s$. In essence, because the blocking transformation is applied to *flat* clause sets only, "pulling

out sub-terms" need not be applied recursively. Clauses (3) and (4) specify that the replacement can be carried out or not.

Step (3) adds clauses the purpose of which is to search for a model with a finite domain, as specified by the dom-predicate. More precisely, if a model of the transformed clause set contains $\mathsf{dom}(t)$ then t is denotes a domain element of the intended finite domain model (and all the sub-terms of t must be domain elements, too, by clause (6)).

Now, how are new domain elements generated? For constants this is obvious (cf. clause (5)). Notice that for 0-ary function symbols the clauses (6) and (7) are absent, and, if a is such a 0-ary function symbol, clause (8) and (9) take the form

$$\mathsf{dom\_candidate}(a) \leftarrow \tag{8'}$$

$$\mathsf{dom}(a) \leftarrow \mathsf{dom\_candidate}(a) \tag{9'}$$

Consequently, $\mathsf{dom}(a)$ will be contained in any model, as is the case for constants.

For non 0-ary function symbols, clause (8) assembles a "domain candidate" term $f(t_1,\ldots,t_n)$ from available domain elements $t_1,\ldots,t_n$. With clause (7) its subterms must be domain elements. This property is important, because clauses (9)-(11) realize in a nondeterministic fashion all possible ways to map $f(t_1,\ldots,t_n)$ to one of its proper sub-terms, which must be a domain element in order to define an interpretation for the symbol f at points $t_1,\ldots,t_n$. The rightmost head atom in clause (9) expresses the alternative to such a mapping, and the term $f(t_1,\ldots,t_n)$ becomes a domain element by it instead.

For better termination behavior it is desirable that the latter possibility, to make $f(t_1,\ldots,t_n)$ a new domain element, is tried after the mappings to its sub-terms. The model generation procedures we tried could be configured to achieve that.

Clause (12) expresses a substituivity property and allows to replace a term t occurring as an argument to a predicate symbol by a domain element d, provided t is mapped to d. Notice that no replacement the other way round is possible (the $\mapsto$ relation is not symmetric), and replacements only at top level term positions are possible (as above, that this is sufficient is a consequence of the assumption that the transformation is applied to flat clause sets only).

Clause (13) states that domain elements cannot be mapped to other (domain) elements. Only "domain candidates" can.

Finally, in step (4) clauses are added that define the "syntactical different" relation $\neq$ on domain elements. It is applied in clause (14) to constrain the $\mapsto$ relation to a right-unique one. This is required in order to uniquely map a term to a domain element.

Altogether we get our main result, which is as follows (a proof is in the appendix).

Theorem 5 (Soundness and Completeness of the Blocking Transformation). *Let $\mathcal{P}$ be a flat clause set. Then $\mathcal{P}$ is satisfiable if and only if $\mathcal{P}^{bl}$ is satisfiable.*

If non-monotonic negation is available, as is the case in the KRHyper prover [Wer03], the blocking transformation can be further optimized by using the following clauses instead of step (4):

$$\leftarrow x \mapsto y, x \mapsto z, \mathsf{not}\ y = z$$

$$x = x \leftarrow$$

These clauses spare the generation of quadratically many clauses otherwise (in terms of the size of the input signature). Although this seems to be a negligible issue, it is not, as our experiments demonstrate.

Related Work. Our transformation is inspired by the well-known blocking technique as used in DL systems [BDS93,HS99,HST99,HM00]. The most powerful DL blocking techniques suffice to decide certain description logics that do no have the finite model property. That is not possibly with our transformation. On the other hand, our blocking transformation is more general as it is applicable to *any* flat clause set, not only to clause sets obtained from DL knowledge bases. Our strategy should be sufficient to guarantee termination of model generation procedures for, for instance, description logics that have the finite model property.[9]

In [BFGHK04] a blocking transformation similar to ours is described. That transformation exploits *non-monotonic negation* as a language element to conveniently "program" the blocking technique in the transformed clause set. This entails two disadvantages: First, standard model generators that do not support non-monotonic negation cannot be applied. Second, unrestricted use of non-monotonic negation, as in in [BFGHK04], prevents the effective use of *back-jumping* in the KRHyper prover,[10] which otherwise is one of the most effective optimization techniques in DL systems, DPLL style SAT solvers and also in Hyper Tableaux for classical logic [BFN96].

Virtually not using back-jumping might have been irrelevant for the application in [BFGHK04], but with larger databases back-jumping is really needed. The alternative transformation to step (4) suggested above makes use of nonmonotonic negation in an inessential way only, so that back-jumping still remains in effect.[11]

Related to our approach are general methods for finite model computation. In the "SEM-style" approach, certain base calculi or systems are extended by techniques for discovering satisfiability in finite domain models (e.g. SCOTT [SLM94] and SEM [Zha95]) In [BT98] a rather general tableaux calculus in the SATCHMO tradition is described, however both refutationally complete and complete for finite domain satisfiability. In the "MACE-style" approach to model building, the search for a finite model is reduced to a propositional SAT problem [Sla92,McC94,CS03, e.g.].

One of the details of our blocking transformation is that it does not aim to be complete for minimal domain size satisfiability. The reason is, only a mapping from a term to one of its sub-terms is attempted, but not to other terms. The rationale behind is to have a small searcher space this way (model computation procedures like MACE or SEM try all possibilities and thus can be complete wrt. minimal domain size models). It would be interesting to see how all these systems perform on knowledge representation tasks like the ones considered in this paper.

---

[9] Should the paper be accepted for LPAR we hope to include such results in the final version.

[10] This is, because non-chronological backtracking is based on an analysis of what literals along a branch (in a tableaux) contributed to a derivation, and which don't. With non-monotonic negation at disposal such an analysis seems promising only in very restricted cases.

[11] In fact, we began our experiments with the transformation in [BFGHK04], but with it we did not arrive at satisfactory results. The model generation system that we tried, KRHyper [Wer03], performs much better with the new transformation.

In [PP98], a general theorem prover is coupled with a propositional SAT solver to search for finite models. The results are encouraging, although somewhat outdated in the meantime, as much improved DL systems (FaCT [Hor99], RACER [HM01]) are available than those used for comparison in [PP98].

5 Experimental Evaluation

For our experimental evaluation, we used the KRHyper [Wer03] and Darwin [BFT05] systems. KRHyper implements the Hyper Tableaux calculus [BFN96], and Darwin implements the Model Evolution calculus [BT03]. Both systems are sound and complete for classical first-order logic. KRHyper supports non-monotonic negation, and the way it is used in the blocking transformation it is obvious that soundness and completeness are not affected. In our experiments with KRHyper we always use it.

We applied our transformation to three ontologies also used in [TRBH04] to evaluate the Vampire prover on ontological reasoning. The first one is Tambis [Tam], a knowledge bases about chemical structures, functions, processes, etc. within a cell, which contains about 345 concepts. The second one is the Galen Common Reference Model [Gal], a medical terminology ontology comprising 24.000 concepts and 913.000 relations (including transitive ones). The third one is the Wine Ontology [Win] from the OWL test suite, with 346 concepts and 16 roles. The following table shows the time (in seconds) that Darwin and KRHyper took to prove the *consistency* of the ontology:

Ontology	**System**	**w/o blocking**	**with blocking**
Tambis w/o instances	KRHyper	0.64	42
Tambis w/o instances	Darwin	0.1	20
Tambis with instances	KRHyper	∞	66
Tambis with instances	Darwin	∞	22
Galen	KRHyper	1.3	4
Galen	Darwin	0.5	timeout
Wine	KRHyper	97	timeout
Wine	Darwin	timeout	timeout

The column labeled “w/o blocking” contains the results when only the equality transformation is applied, whereas in the right column, both transformations are applied. Without blocking, all problems with a finite Herbrand model are solvable quite quickly – except for the Wine ontology, where Darwin exceeded the time-limit (1h). We presume, however, that Darwin did not fail for principal reasons.

If Tambis is populated with a suitable instance, the problem described in ($\star\star$) renders the Herbrand model infinite, and the provers do not terminate. The blocking technique solves this problem as expected. On the flip-side, due to the additional search space introduced by blocking, some problems become *practically* unsolvable. In particular, explicitly deriving the ground $\neq$-theory for a current domain in the classical formulation (cf. step (4) in the blocking transformation) may slow down the model computation considerable. Similarly to [TRBH04], we validated our transformation by pairwise concept subsumption tests. We used KRHyper and our optimized transformation (with non-monotonic negation), although we could not use the blocking technique

for the Wine Ontology, because this results in a timeout (s.a.). The samples that we run took the following average times (in seconds), and the runs on all samples terminated.

Ontology	**System**	**Satisfiable**	**Unsatisfiable**
Tambis	KRHyper with blocking	47	2
Galen	KRHyper with blocking	3.8	3.8
Wine	KRHyper w/o blocking	120	0.15

Furthermore, we applied our transformation to the OWL test cases provided by the W3 consortium [OWL04]. This is a collection of OWL files on which numerous reasoners have already been tested. We used the WonderWeb API [Won04] to translate the OWL test cases to first order logic. As it supports only OWL-DL and OWL-lite, we could not run the OWL-full tests. The distribution of the problem classes is as follows:

Test class	**# OWL-light**	**# OWL-DL**	**# total**
Consistency tests	25	27	52
Inconsistency tests	29	38	67
Entailment tests	23	29	52
			171

As has been done with the other reasoners, we distinguish three result cases: pass (the prover returned the correct answer), fail (the prover returned the wrong answer, thus entailing its incorrectness) and other cases (such as memory problems, undecided answers or timeouts). The following table contains the results of our experiments at the top. For the other systems the results were just taken from the OWL test cases web page. For that systems we list in parenthesis whether it is a native description logic (DL) or OWL reasoner, or the theorem prover it is based on. The numbers denote the percentage of problems solved of those mentioned in the previous table.

	Consistency		**Inconsistency**		**Entailment**	
System	pass	fail	pass	fail	pass	fail
KRHyper with blocking	89%	4%	90%	4%	**86%**	7%
KRHyper w/o blocking	75%	4%	93%	4%	**86%**	5%
Darwin with blocking	**89%**	4%	92%	4%	84%	5%
Darwin w/o blocking	7%	4%	**94%**	4%	**86%**	5%
Darwin ∪ KRHyper	**93%**	4%	**94%**	4%	**88%**	5%
Fact (DL)	42%	0%	85%	0%	7%	0%
Hoolet (Vampire)	78%	0%	94%	0%	72%	0%
FOWL (OWL)	53%	0%	4%	0%	32%	0%
Pellet (DL)	**96%**	0%	**98%**	0%	86%	0%
Euler (“Prover”)	0%	2%	98%	0%	**100%**	0%
OWLP (DL)	50%	10%	26%	12%	53%	4%
Cerebra (DL)	90%	0%	59%	0%	61%	0%
Surnia (Otter)	-	-	0%	0%	13%	0%
ConsVISor (OWL-full)	77%	6%	65%	1%	-	-

KRHyper and Darwin perform quite well compared to the other systems. All of the *fail* cases can be traced back to conversion failures of the OWL-to-TPTP converter or the TPTP-to-KIF converter. These failures lie outside the scope of our implementation.

Still, our systems do not return an answer for all of the test cases. In the majority of these cases, a timeout limitation (200 seconds) caused the prover to stop, but it is

unlikely that extending the time limit would help a lot. Memory problems were not an issue at all (both provers realize a memory-efficient one-branch-at-a-time approach).

For the consistent (satisfiable) problems both Darwin and KRHyper gained from the blocking transformation. There are clearly identifiable problems in the test suite where the provers will not terminate without blocking, for any timeout. Notice that the "union" of Darwin and KRHyper solves more problems than Darwin or KRHyper individually. This is because they don't solve exactly the same problems.

For the inconsistent (unsatisfiable) problems both Darwin's and KRHyper's performance degraded with the blocking transformation. This is clear, as both systems are refutationally complete, and the blocking transformation introduces *additional* model candidates to be inspected for model-ship (without success, of course).

6 Conclusion

We presented improvements for first-order model-generation style reasoning in the context of ontological knowledge bases. We proposed a transformation for equality reasoning and a "blocking" transformation that enables provers to terminate more often on satisfiable input, a useful feature e.g. when debugging knowledge bases or when computing subsumption hierarchies. The approach is motivated by the possibility to easily accommodate powerful extensions to the input language, like rule languages.

Under what conditions our transformation provides a decision procedure in conjunction with model-generation style provers is a question we are currently investigating (we expect a positive result for all description logics with a finite model property).

In this paper we evaluated our technique on description logics knowledge bases. We did this in order to compare our approach to others. It will be interesting to apply our technique to problems outside this domain, e.g. on problems stemming from natural language representation [Bos03, e.g.].

References

Baa03. F. Baader, ed. *CADE-19*, vol. 2741 of *LNAI*. Springer, 2003.

BCM+02. F. Baader, et al. eds. *Description Logic Handbook*. Cambridge U Press, 2002.

BDS93. M. Buchheit, F.M. Donini, A.Schaerf. Decidable reasoning in terminological knowledge representation systems. *JAIR*, 1:109–138, 1993.

BFGHK04. P. Baumgartner et al. Model based deduction for database schema reasoning. In S. Biundo et al eds, *KI 2004*, vol. 3238, 168–182. Springer, 2004.

BFN96. P. Baumgartner, U. Furbach, I. Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, no. 1126 in LNAI. Springer, 1996.

BFT05. P. Baumgartner, A. Fuchs, C. Tinelli. Implementing the Model Evolution Calculus. In S. Schulz, et al eds, *Special Issue of the IJAIT*, 2005. To appear.

BG98. L. Bachmair, H. Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In W. Bibel, P. Schmitt, eds, *Automated Deduction*, vol. I, 353–398. Kluwer Academic, 1998.

BGV98. L. Bachmair, H. Ganzinger, A. Voronkov. Elimination of equality via transformation with ordering constraints. In Kirchner&Kirchner [KK98].

Bos03. J. Bos. Exploring model building for natural language understanding. In *Proc. ICoS-4*, 2003.

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

BT98. F. Bry, S. Torge. A Deduction Method Complete for Refutation and Finite Satisfiability. In *Proc. JELIA*, LNAI. Springer, 1998.

BT03. P. Baumgartner C. Tinelli. The Model Evolution Calculus. In Baader [Baa03], 350–364.

Bun94. A. Bundy, ed. *CADE 12*, LNAI 814. 1994. Springer.

CS03. K. Claessen N. Sörensson. New techniques that improve mace-style finite model building. In P. Baumgartner, C. Fermüller, eds, *CADE-19 Workshop*, 2003.

dNHS00. H. de Nivelle, U. Hustadt, R. Schmidt. Resolution-based methods for modal logics. *Logic J. of the IGPL*, 8(3):265–292, 2000.

FLHT01. C. Fermüller, A. Leitsch, U. Hustadt, T. Tammet. Resolution Decision Procedures. In Robinson Voronkov [RV01b], 1791–1850.

Gal. The galen common reference model. `http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/galen.owl`.

GHVD03. B. Grosof, I. Horrocks, R. Volz, S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. WWW 2003*, 48–57. ACM, 2003.

HM00. V. Haarslev R. Möller. Expressive ABox reasoning with number restrictions, role hierarchies, and transitively closed roles. In A. Cohn, et al eds, *KR2000*, 273–284, 2000. Morgan Kaufmann.

HM01. V. Haarslev R. Möller. High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study. In *IJCAI 97*. Morgan Kaufmann, 2001.

Hor99. I. Horrocks. FaCT and iFaCT. In P. Lambrix et al. eds, *Proc. DL'99*, 133–135, 1999.

HS99. U. Hustadt R. Schmidt. On the relation of resolution and tableaux proof systems for description logics. In *Proc. IJCAI '99*, 110–117, 1999. Morgan Kaufmann.

HST99. I. Horrocks, U. Sattler, S. Tobies. A description logic with transitive and converse roles, role hierarchies and qualifying number restrictions. LTCS-Report LTCS-99-08, RWTH Aachen, 1999. Revised version.

KK98. C. Kirchner, H. Kirchner, eds. *CADE 15*, LNAI 1421, 1998. Springer.

LS01. R. Letz G. Stenz. Model elimination and connection tableau procedures. In A. Robinson, A. Voronkov, eds, *Handbook of AR*, 2017–2114. Elsevier, 2001.

MB88. R. Manthey F. Bry. SATCHMO: a theorem prover implemented in Prolog. In E. Lusk, R. Overbeek, eds, *Proc. CAD 1988*, vol. 310 of *LNCS*, 415–434. Springer.

McC94. W. McCune. A davis-putnam program and its application to finite first-order model search: Quasigroup existence problems. TR, Argonne National Lab., 1994.

MS97. M. Moser, J. Steinbach. Ste-modification revisited. TR AR-97-03, TU Munich, 1997.

NP01. I. Niles, A. Pease. Towards a standard upper ontology. In C. Welty, B. Smith, eds, *Proc. FOIS-2001*, 2001.

NR01. R. Nieuwenhuis, A. Rubio. Paramodulation-based theorem proving. In Robinson Voronkov [RV01b], 371–443.

NRW98. A. Nonnengart, G. Rock, C. Weidenbach. On generating small clause normal forms. In Kirchner&Kirchner [KK98].

OWL04. 2004. `http://www.w3.org/2003/08/owl-systems/test-results-out`.

PP98. M. Paramasivam D. Plaisted. Automated deduction techniques for classification in description logic systems. *JAR*, 20(3):337–364, 1998.

RV01a. A. Riazonov, A. Voronkov. Vampire 1.1 (system description). In *Proc. IJCAR*, vol. 2083 of *LNCS*. Springer, 2001.

RV01b. J. Robinson, A. Voronkov, eds. *Handbook of AR*. Elsevier MIT Press, 2001.

SH03. R. Schmidt U. Hustadt. A principle for incorporating axioms into the first-order translation of modal formulae. In Baader [Baa03], 412–426.

Sla92. John Slaney. Finder (finite domain enumerator): Notes and guide. TR-ARP-1/92, Australian National U, Canberra, 1992.

SLM94. J. Slaney, E. Lusk, W. McCune. SCOTT: Semantically constrained Otter (system description). In Bundy [Bun94], 764–768.

SSY94. G. Sutcliffe, C. Suttner, T. Yemenis. The TPTP problem library. In Bundy [Bun94].

Tam. The Tambis ontology. `http://protege.stanford.edu/plugins/owl/owl-library/tambis-full.owl`.

TH03. D. Tsarkov, I. Horrocks. DL reasoner vs. first-order prover. In *Proc. DL 2003*, vol. 81 of *CEUR*, 152–159, 2003.

TRBH04. D. Tsarkov, A. Riazanov, S. Bechhofer, I. Horrocks. Using vampire to reason with owl. In S. McIlraith et al, eds, *ISWC*, vol. 3298 of *LNCS*, 471–485. Springer, 2004.

Wer03. C. Wernhard. System Description: KRHyper. Fachberichte Informatik 14–2003, Universität Koblenz-Landau, 2003.

Win. The w3c wine ontology. `http://www.w3.org/TR/2003/WD-owl-guide-20030331/wine.owl`.

Won04. `http://owl.man.ac.uk/api.shtml`.

Zha95. H. Zhang. Sem: a system for enumerating models. In *Proc. IJCAI-95*, 298–303, 1995.

A Proofs

A.1 Proof of Theorem 3

The formal tool to prove the completeness direction of Theorem 3 is the model construction technique introduced for the superposition calculus [BG98,NR01, e.g.].

We will need some terminology regarding term orderings and rewrite systems. A *(rewrite) rule* is an expression of the form $l \to r$ where l and r are Σ-terms. A *rewrite system* is a set of rewrite rules.

We suppose as given a reduction ordering $\succ$ that is total on ground Σ-terms. [12] Fix any such reduction ordering $\succ$ 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 clause set and that is minimal in $\succ$. That is, there is no term t with $\mathsf{true} \succ t$.

The purpose of the constant true is to enable uniform notation in the proof. More precisely, an atom A that is not an equation is read as the equation $A \approx \mathsf{true}$. This way non-equational atoms become terms. Also, a flat atom A is possibly no longer flat when written as $A \approx \mathsf{true}$. This all causes no problems, though, as the changes are made only conceptually, for the sake of leaner notation in the proof.

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 ground clauses, 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.

A ground rewrite system R is *ordered by* $\succ$ iff $l \succ r$, for every rule $l \to r \in R$. In the sequel, the letter R will always denote a ground rewrite system ordered by the given reduction ordering $\succ$. By the symbol $\to_R$ we denote the one-step rewrite relation on ground Σ-terms, and by $\to_R^\star$ we denote its transitive-reflexive closure.

As a non-standard notion, we define a *rewrite system without overlaps* to be a ground rewrite system R that is ordered by $\succ$, and whenever $l \to r \in R$ then there is no other rule in R of the form $s[l] \to t$ or $s \to t[l]$. In other words, no rule can be reduced by another rule, neither the left hand side *nor the right hand side*.

Any rewrite system without overlaps is a convergent ground rewrite system. It is well known that for any convergent ground rewrite system R, and any two terms s and t, $R \models_E s \approx t$ [13] if and only if there is a term u such that $s \to_R^\star u$ and $t \to_R^\star u$. This result thus applies in particular to ground rewrite systems without overlaps.

When talking about a ground substitution γ for a clause $\mathcal{H} \leftarrow \mathcal{B}$ below we always mean a substitution that moves (only) the variables of $\mathcal{H} \leftarrow \mathcal{B}$ to ground Σ-terms. We say that γ is *reducible* by a rewrite system R 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 by* R.

[12] A *reduction ordering* is a strict partial ordering that is well-founded and is closed unter context i.e., $s \succ s'$ implies $t[s] \succ t[s']$ for all terms t, and liftable, i.e., $s \succ t$ implies $s\delta \succ t\delta$ for every term s and t and substitution δ.

[13] Here R is read as a set of equations.

Finally, for a Σ-clause set $\mathcal{P}$ let $\mathcal{P}^{\mathrm{gr}}$ denote the set of all ground Σ-instances of all clauses in $\mathcal{P}$.

Theorem 3 (Soundness and Completeness of the Equality Transformation). *Let $\mathcal{P}$ be a clause set. Then $\mathcal{P}$ is UNA-E-satisfiable if and only if $\mathcal{P}^{eq}$ is satisfiable.*

Proof. The proof of the only-if direction is not difficult and is omitted. The main observation needed is that "pulling out" subterms preserves E-satisfiability.

For the if direction (completeness) suppose that I is a Herbrand model of $\mathcal{P}^{\mathrm{eq}}$. We will show that a certain subset $R_I \subseteq I$ is a UNA-E-model of $\mathcal{P}$ (to make the statement $R_I \subseteq I$ meaningful, every equation $s \approx t$ in I is taken as the two rules $s \to t$ and $t \to s$). More precisely, R_I will be a terminating rewrite system without overlaps.

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

$\mathbf{R_I \models_E \mathcal{P}^{eq}}$. At the beginning we assumed that I is a Herbrand model of $\mathcal{P}^{\mathrm{eq}}$. That is, $I \models (\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$. We first construct R_I and then show $R_I \models_E (\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$.

For every equation $s \approx t \in I$[14] 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 $s' \approx t' \in I$ 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[15]

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

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

By construction R_I is a rewrite system without overlaps. Because $\succ$ is a well-founded ordering R_I thus is a convergent rewrite system.

Now we show by well-founded induction that R_I is an E-model of of $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$, or, equivalently, $R_I \models_E (\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$. It suffices to chose a clause from $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$ arbitrary. It is of the form $(\mathcal{H}^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}})\gamma$, for some clause $C^{\mathrm{eq}} = (\mathcal{H}^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}}) \in \mathcal{P}^{\mathrm{eq}}$ and some ground substitution γ, the domain of which are the variables of C.

We distinguish two complementary cases.

(1) γ *is reducible by* R_I.
That is, there is a variable x in the domain of γ such that $x\gamma \to_{R_I} t$ for some (ground) 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 $R_I \models_E C^{\mathrm{eq}}\gamma'$, and by congruence $R_I \models_E C^{\mathrm{eq}}\gamma$.

[14] Recall that an atom A is written as an equation $A \approx \mathtt{true}$ for the purpose of uniform notation.
[15] The third condition is absent in the standard model construction [BG98].

(2) γ *is irreducible by R.*
If $R_I \not\models_E \mathcal{B}^{\mathrm{eq}}\gamma$ then $R_I \models_E C^{\mathrm{eq}}\gamma$ follows trivially. Hence suppose $R_I \models_E \mathcal{B}^{\mathrm{eq}}\gamma$ from now on.

Because γ is irreducible by R_I, with Lemmas 7 and 8 it follows $I \models \mathcal{B}^{\mathrm{eq}}\gamma$. Recall that I is given as a Herbrand model of $(\mathcal{P}^{\mathrm{eq}})^{\mathrm{gr}}$. From $I \models \mathcal{B}^{\mathrm{eq}}\gamma$ it thus follows $I \models \mathcal{H}^{\mathrm{eq}}\gamma$. Again by Lemmas 7 and 8, this time in the other direction, it follows $R_I \models_E \mathcal{H}^{\mathrm{eq}}\gamma$. This result implies trivially $R_I \models_E (\mathcal{H}^{\mathrm{eq}} \leftarrow \mathcal{B}^{\mathrm{eq}})\gamma$.

This concludes the case analysis. Notice that in both cases we have shown $R_I \models_E C^{\mathrm{eq}}\gamma$, which remained to be shown.

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

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 ment that results from flattening the equations $s_1 \approx x_1^s, \ldots, x_n \approx x_n^s$, 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 $\boldsymbol{t} = \boldsymbol{x}^{\boldsymbol{t}}\gamma'$.[16] 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 $R_I \models_{\approx} C^{\mathrm{eq}}\gamma'\gamma$. 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 $R_I \models_{\approx} u_i\gamma'\gamma \approx x_i\gamma'\gamma$. But then $R_I \models_{\approx} \mathcal{H}\gamma \leftarrow \mathcal{B}\gamma$ follows, which was to show.

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

R *satisfies UNA.* Suppose, to the contrary, that R does not satisfy UNA, i.e. $R_I \models_E c \approx d$ for some different constants c and d. The equation $c \approx d$ is flat. By Lemma 7 then $c \approx d \in I$. With $(\mathcal{P}^{\text{eq}})^{\text{gr}}$ containing the rule $\leftarrow c \approx d$ this is impossible, though. □

Lemma 7. *Let $s \approx t$ be a flat equation and γ a ground substitution irreducible by R_I. Then, $R_I \models_E s\gamma \approx t\gamma$ iff $s\gamma \approx t\gamma \in I$.*

Proof. For the if-direction suppose $s\gamma \approx t\gamma \in I$.

If $s\gamma = t\gamma$ then $R_I \models_E s\gamma \approx t\gamma$ follows trivially.

If both $s\gamma$ and $t\gamma$ are irreducible by R_I then $\varepsilon_{s\gamma \approx t\gamma} = \{s\gamma \to t\gamma\}$ and so $s\gamma \to t\gamma \in R_I$. From $s\gamma \to t\gamma \in R_I$ the result $R_I \models_E s\gamma \approx t\gamma$ follows easily.

Hence suppose, without loss of generality that $s\gamma$ is reducible. Recall first that I is a model of 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, I 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_I does not contain $c \to d$ or $d \to c$ either (because of $R_I \subseteq I$). Together with the restriction (1) on orderings defined above it follows that R_I 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_I.

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 0-ary) 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 by R_I.

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_I must contain a rule of the form $s\gamma \to u$, for some term u.

From $s\gamma \to u \in R_I$ 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 $s\gamma \approx t\gamma \in I$ 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 $t\gamma \to u' \in R_I$ for some irreducible term u'.

Recall any program obtained from the equality transformation contains the axioms of reflexivity, symmetry and transitivity. Further recall that I is a model of some such program.

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

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

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

The last open case, that $t\gamma$ is irreducible, is treated similarly: from $s\gamma \approx t\gamma \in I$ and $s\gamma \approx u \in I$ it follows by the symmetry and transitivity axioms that $t\gamma \approx u \in I$. 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_I, contradicting irreducibility of $t\gamma$ and of u). Thus, with $s\gamma \to u \in R_I$ and $t\gamma = u$ it follows $R_I \models_E s\gamma \approx t\gamma$.

This completes the proof for the if-direction.

For the only-if direction suppose $R_I \models_E s\gamma \approx t\gamma$. Because R_I is a convergent rewrite system there is a term w such that $s\gamma \to^\star_{R_I} w$ and $t\gamma \to^\star_{R_I} w$.

If both $s\gamma$ and $t\gamma$ are irreducible then $s\gamma = t\gamma$, $s\gamma \approx t\gamma$ is an instance of the reflexivity axiom, and so $s\gamma \approx t\gamma \in I$ 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 $s\gamma \to u \in R_I$. 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 $s\gamma \to u' \in R_I$ with $u' = w$. Because $R_I \subseteq I$ we get $s\gamma \approx w \in I$ and $t\gamma \approx w \in I$. By the symmetry and transitivity axioms, I must also satisfy $s\gamma \approx t\gamma$ and $t\gamma \approx s\gamma$. Equivalently, $s\gamma \approx t\gamma \in I$ and $t\gamma \approx s\gamma \in I$.

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

Lemma 8. *Let $P(t_1, \ldots, t_n)$ be a flat non-equational atom and γ a ground substitution irreducible by R_I. Then, $R_I \models_E P(t_1, \ldots, t_n)$[17] iff $P(t_1, \ldots, t_n) \in I$.*

Proof. The proof is similar to the proof of Lemma 7 and is omitted. An important detail is that I is a model of the predicate substitution axioms, which are part of the equality transformation (cf. Definition 2, the last clause scheme stated there). □

A.2 Proof of Theorem 5

Lemma 9. *Let $\mathcal{P}$ be a flat clause set. If $\mathcal{P}^{bl}$ is unsatisfiable then $\mathcal{P}$ is unsatisfiable.*

Proof. Suppose $\mathcal{P}^{\mathrm{bl}}$ is unsatisfiable. We directly show that $\mathcal{P}$ is unsatisfiable.

Consider the clause set $(\mathcal{P}^{\mathrm{bl}})_1$ which is obtained from $\mathcal{P}^{\mathrm{bl}}$ by replacing every clause of the form (9) by its subclause

$$\mathsf{dom}(f(x_1, \ldots, x_n)) \leftarrow \mathsf{dom\_candidate}(f(x_1, \ldots, x_n)) \qquad (9')$$

[17] Or, more precisely, $R_I \models_E P(t_1, \ldots, t_n) \approx \mathsf{true}$ according to the convention that non-equational atoms are represented as rewrite rules in R_I.

With $\mathcal{P}^{\text{bl}}$ being unsatisfiable, $(\mathcal{P}^{\text{bl}})_1$ is unsatisfiable, too. We consider a hyper-resolution refutation of $(\mathcal{P}^{\text{bl}})_1$, which exists by the completeness of hyper-resolution. We take this refutation as a starting point to argue that certain clauses can be deleted from $(\mathcal{P}^{\text{bl}})_1$ without affecting unsatisfiability.

Observe that with the move from (9) to (9') the only occurences of $\mapsto_{\text{sub}}$-literals are those in clauses (10) and (11). It is easy to see that no hyper-resolution inferences from these clauses exist. Consequently, they can be removed from $(\mathcal{P}^{\text{bl}})_1$ without affecting unsatisfiability. Let $(\mathcal{P}^{\text{bl}})_2$ be the resulting clause set. With the removal of clauses (10) and (11), the clause set $(\mathcal{P}^{\text{bl}})_2$ contains no positive occurence of $\mapsto$-literals any more.[18] In the clauses (4), (12), (13) and (14), all the occurences of $\mapsto$-literals are negative. Therefore these clauses can be removed from $(\mathcal{P}^{\text{bl}})_2$ without affecting unsatisfiability. Let $(\mathcal{P}^{\text{bl}})_3$ be the resulting clause set. The only positive occurence of $\mapsto_{\text{ref}}$-literals in $(\mathcal{P}^{\text{bl}})_3$ is in clause (3), and the only negative occurences of $\mapsto_{\text{ref}}$-literals is in clause (2). It is clear that all hyper-resolution inferences from these clauses can be done exhaustively and clauses (2) can be removed afterwards, thus undoing the transformation that led to clauses (2). Clearly, the resulting clause set $(\mathcal{P}^{\text{bl}})_4$ is unsatisfiable, too.

Next we consider the clauses (15)-(19), which axiomatize a theory of syntactic equality on the domain elements. It is easy to see that these clauses are consistent with any set of dom-atoms. Further, as no clause from the other clauses in $(\mathcal{P}^{\text{bl}})_4$ contains a negative occurence of an $\neq$-literal. It follows that with $(\mathcal{P}^{\text{bl}})_4$ being unsatisfiable, removal of the clauses (15)-(19) preserves unsatisfiability. Let $(\mathcal{P}^{\text{bl}})_5$ be the resulting clause set.

Now, $(\mathcal{P}^{\text{bl}})_5$ is the same clause set as the clause set obtained from $\mathcal{P}$ by applying only step (1) of Defintion 4 and augment the resulting clause set with the following clauses, for every Σ-constant c, and for every n-ary Σ-function symbol f and all $i = 1,\ldots,n$:

$$\mathsf{dom}(c) \leftarrow \tag{5}$$

$$\mathsf{dom}(x_i) \leftarrow \mathsf{dom}(f(x_1,\ldots,x_n)) \tag{6}$$

$$\mathsf{dom}(x_i) \leftarrow \mathsf{dom\_candidate}(f(x_1,\ldots,x_n)) \tag{7}$$

$$\mathsf{dom\_candidate}(f(x_1,\ldots,x_n)) \leftarrow \mathsf{dom}(x_1),\ldots,\mathsf{dom}(x_n) \tag{8}$$

$$\mathsf{dom}(f(x_1,\ldots,x_n)) \leftarrow \mathsf{dom\_candidate}(f(x_1,\ldots,x_n)) \tag{9'}$$

As the dom_candidate-predicate does not appear outside these clauses, it is not difficult to see that these clauses can be replaced in an unsatisfiability preserving way by the following clauses, for every Σ-constant c, and for every n-ary Σ-function symbol f:

$$\mathsf{dom}(c) \leftarrow$$

$$\mathsf{dom}(f(x_1,\ldots,x_n)) \leftarrow \mathsf{dom}(x_1),\ldots,\mathsf{dom}(x_n)$$

Let $(\mathcal{P}^{\text{bl}})_6$ be the resulting clause set. Of course, these clauses just enumerate the Herbrand universe of Σ.

Recall that for every clause $\mathcal{H} \leftarrow \mathcal{B}$ in $\mathcal{P}$ there is a clause

$$\mathcal{H} \leftarrow \mathcal{B}, \mathsf{dom}(x_1),\ldots,\mathsf{dom}(x_k) \tag{2'}$$

[18] I.e., no clause contains a head atom with the predicate symbol $\mapsto$.

in $(\mathcal{P}^{\mathrm{bl}})_4$, which is also in $(\mathcal{P}^{\mathrm{bl}})_6$. By definition of Herbrand-satisfiability, unsatisfiability of $(\mathcal{P}^{\mathrm{bl}})_6$ entails that each clause (2') can be replaced by all its ground instances, i.e., by

$$\mathcal{H}\gamma \leftarrow \mathcal{B}\gamma, \mathsf{dom}(x_1)\gamma, \ldots, \mathsf{dom}(x_k)\gamma$$

for all ground substitutions γ. The resulting clause set $(\mathcal{P}^{\mathrm{bl}})_7$ is unsatisfiable, too. Because of the presence of the clauses above that enumerate the Herbrand universe, each clause

$$\mathcal{H}\gamma \leftarrow \mathcal{B}\gamma, \mathsf{dom}(x_1)\gamma, \ldots, \mathsf{dom}(x_k)\gamma$$

in $(\mathcal{P}^{\mathrm{bl}})_7$ can be replaced by

$$\mathcal{H}\gamma \leftarrow \mathcal{B}\gamma$$

and the axioms above enumerating the Herbrand universe can be deleted. Let $(\mathcal{P}^{\mathrm{bl}})_8$ be the resulting clause set, which is unsatisfiable, too. Notice that $(\mathcal{P}^{\mathrm{bl}})_8$ can be obtained also from $\mathcal{P}$ by replacing every clause $\mathcal{H} \leftarrow \mathcal{B}$ in $\mathcal{P}$ by all its ground instances. Hence, with $(\mathcal{P}^{\mathrm{bl}})_8$ being unsatisfiable, $\mathcal{P}$ is unsatisfiable, too. □

Non-Herbrand Interpretations. In most parts of this paper we are working with Herbrand interpretations. An exception is below, in the completeness proof of the blocking transformation. Let us therefore introduce our notation regarding (not necessarily Herbrand) interpretations; everything is complete standard.

A *(Σ-)interpretation I* consists of a domain Δ, which is a non-empty set, and mappings for each Σ-constant to a domain element, for each *n*-ary Σ-function symbol to a function from Δ^n to Δ, and for each *n*-ary Σ-predicate symbol to a function from Δ^n to $\{true, false\}$. We denote these mappings by c^I, f^I and P^I, respectively.

A *valuation* is a mapping from the set of variables V to Δ. We write $v(x)$ to denote the value of x under v.

Given an interpretation I, a valuation v and a $\Sigma-$term t we write $t^{I,v}$ to denote the result of evaluating t under the usual homomorphic extension of $\cdot^I$ and v to terms. For a Σ-atom $P(t_1,\ldots,t_n)$ we define, completely standard, $(P(t_1,\ldots,t_n))^{I,v} = P^I(t_1^{I,v},\ldots,t_n^{I,v})$. We write $I,v \models^\star A$ iff I,v satisfies the atom A.[19] The only quantified formulas we are concerned with are clauses. We therefore define that *I satisfies* a clause $\mathcal{H} \leftarrow \mathcal{B}$, written as $I \models^\star \mathcal{H} \leftarrow \mathcal{B}$, iff for all valuations v it holds $I,v \models^\star \mathcal{H} \leftarrow \mathcal{B}$. The latter is defined as $I,v \models^\star \mathcal{H} \leftarrow \mathcal{B}$ iff whenever $I,v \models B$ for all body atoms B of $\mathcal{B}$ then $I,v \models H$ for some head atom H of $\mathcal{H}$. Finally, I is a *model* of clause set $\mathcal{P}$ iff I satisfies all clauses in $\mathcal{P}$.

Theorem 5 (Completeness). *Let $\mathcal{P}$ be a flat clause set. Then $\mathcal{P}$ is satisfiable if and only if $\mathcal{P}^{bl}$ is satisfiable.*

[19] We always use the symbol $\models^\star$ for satisfaction by (not necessarily Herbrand) interpretations, whereas the symbol $\models$ is reserved for satisfaction by Herbrand interpretations.

Proof. The only-if direction follows immediately from Lemma 9.

Regarding the if-direction, suppose $\mathcal{P}^{\mathrm{bl}}$ is satisfiable. We have to show that $\mathcal{P}$ is satisfiable.

Let I^{bl} be a Herbrand Σ-model of $\mathcal{P}^{\mathrm{bl}}$. We show that I^{bl} determines a (possibly non-Herbrand) Σ-model I of $\mathcal{P}$. The proof proceeds in two steps. In the first step we construct the domain of I and the interpretation of the constants, function symbols and predicate symbols. In the second step we then show that I is a model of $\mathcal{P}$.

The domain of I is defined as the set $\Delta = \{d \mid \mathsf{dom}(d) \in I^{\mathrm{bl}}\}$. Regarding the interpretation function $\cdot^I$, define $c^I = c$ for every constant c; for every n-ary function symbol f and all domain elements $d_1, \ldots, d_n \in \Delta$ define

$$f^I(d_1, \ldots, d_n) = \begin{cases} d & \text{if } f(d_1, \ldots, d_n) \mapsto d \in I^{\mathrm{bl}} \text{ and} \\ & d \text{ is a proper subterm of } f(d_1, \ldots, d_n) \\ f(d_1, \ldots, d_n) & \text{otherwise} \end{cases}$$

We have to make sure that this definition is well-defined. This is immediate for constants, as with clause (5) it follows $c \in \Delta$. Regarding function symbols, let $d_1, \ldots, d_n \in \Delta$ arbitrary. By definition of Δ it follows $\mathsf{dom}(d_1), \ldots, \mathsf{dom}(d_n) \in I^{\mathrm{bl}}$. By clause (8) we have $\mathsf{dom\_candidate}(f(d_1, \ldots, d_n)) \in I^{\mathrm{bl}}$. With clauses (9), (10) and (11) this implies

(i) $\mathsf{dom}(f(d_1, \ldots, d_n)) \in I^{\mathrm{bl}}$ (cf. the rightmost head atom in (9)), or
(ii) $f(d_1, \ldots, d_n)) \mapsto d \in I^{\mathrm{bl}}$ for some proper subterm d of $f(d_1, \ldots, d_n))$.

In case (i) we first show $f(d_1, \ldots, d_n)) \mapsto d \notin I^{\mathrm{bl}}$, for any term d. This however follows easily from $\mathsf{dom}(f(d_1, \ldots, d_n)) \in I^{\mathrm{bl}}$ and clause (13). Thus, in the definition of $\cdot^I$ the second case applies. With $\mathsf{dom}(f(d_1, \ldots, d_n)) \in I^{\mathrm{bl}}$ and the definition of Δ it follows $f(d_1, \ldots, d_n) \in \Delta$. Thus $f^I(d_1, \ldots, d_n) \in \Delta$, which means that $\cdot^I$ is well-defined in case (i).

In case (ii) we have to show two things: the first is right-uniqueness, i.e. there is no proper subterm d' of $f(d_1, \ldots, d_n))$ such that $d \neq d'$ and $f(d_1, \ldots, d_n)) \mapsto d' \in I^{\mathrm{bl}}$, and the second is $d \in \Delta$. The second follows easily from $f(d_1, \ldots, d_n)) \mapsto d \in I^{\mathrm{bl}}$, the clauses (6) and (7), which imply $\mathsf{dom}(d) \in I^{\mathrm{bl}}$, and the definition of Δ. In fact, no assumption about d was made except that it is a proper subterm of $f(d_1, \ldots, d_n))$. Thus, regarding right-uniqueness, if there were a proper subterm d' of $f(d_1, \ldots, d_n))$ such that $d \neq d'$ and $f(d_1, \ldots, d_n)) \mapsto d' \in I^{\mathrm{bl}}$, then it would also hold $\mathsf{dom}(d') \in I^{\mathrm{bl}}$. By clauses (14)-(19), however, this is impossible. Thus, also in case (ii) the definition of $\cdot^I$ is well-defined.

To conclude the first step of the proof, for every n-ary predicate symbol P and domain elements $d_1, \ldots, d_n \in \Delta$ define

$$P^I(d_1, \ldots, d_n) = \begin{cases} \mathit{true} & \text{if } P(d_1, \ldots, d_n) \in I^{\mathrm{bl}} \\ \mathit{false} & \text{otherwise} \end{cases}$$

Now we turn to the second step of the proof. Let $\mathcal{H} \leftarrow \mathcal{B}$ be an arbitrarily chosen clause from $\mathcal{P}$ and let ν be an arbitrary valuation. It suffices to show $I, \nu \models^\star \mathcal{H} \leftarrow \mathcal{B}$. From now on suppose $I, \nu \models^\star B$ for all body atoms B of $\mathcal{B}$, because otherwise the claim holds trivially.

We are given that I^{bl} is a Herbrand model of $\mathcal{P}^{\text{bl}}$. Therefore, in particular, $I^{\text{bl}} \models (\mathcal{H} \leftarrow \mathcal{B}', \mathsf{dom}(x_1), \ldots, \mathsf{dom}(x_k))\gamma$, for all ground substitutions γ, where the dom body atoms result from step (1) of Definition 4 applied to the clause $\mathcal{H} \leftarrow \mathcal{B}$, and $\mathcal{B}'$ is obtained from $\mathcal{B}$ by "pulling out function terms", as described in step (2).

The first subgoal of the proof is to show there is some such ground substitution γ that satisfies the rule body. That is, we are going to show there is a ground substitution γ such that

$$B'\gamma \in I^{\text{bl}} \text{ , for all body atoms } B' \text{ of } \mathcal{B}', \text{ and} \tag{20}$$

$$\mathsf{dom}(x_1)\gamma, \ldots, \mathsf{dom}(x_k)\gamma \in I^{\text{bl}} \tag{21}$$

The substitution γ is defined as $\gamma := \gamma_{B_1} \cdots \gamma_{B_m}\gamma_v$, for certain substitutions $\gamma_{B_1}, \ldots, \gamma_{B_m}$, where $\mathcal{B} = B_1, \ldots, B_m$. All these substitutions will have disjoint domains. With that disjointness property, (21) follows easily by the following argumentation: recall that I^{bl} is a Herbrand model of $\mathcal{P}$. Therefore Δ consists of ground (Σ-)terms, and for each domain element $d \in \Delta$ it holds $\mathsf{dom}(d) \in I^{\text{bl}}$. With Δ consisting of ground terms (only), the valuation v must map the variables to ground terms. This allows to view v also as a ground substitution. Formally define the substitution γ_v as the substitution with domain $\{x_1, \ldots, x_k\}$ such that $x\gamma_v = v(x)$ for all variables $x_1, \ldots, x_k$. Finally, with $\mathsf{dom}(d) \in I^{\text{bl}}$ for all $d \in \Delta$ it follows in particular $\mathsf{dom}(x_i)\gamma_v \in I^{\text{bl}}$, for all $i = 1, \ldots, k$. It thus remains to prove (20).

Let $P(t_1, \ldots, t_n)$ be any body atom of $\mathcal{B}$. Pulling out function terms transforms it to a body atom $P(t'_1, \ldots, t'_n)$ in $\mathcal{B}'$, where each t'_i is the same as t_i, or else t'_i is a variable y_i and $\mathcal{B}'$ includes an atom $t_i \mapsto_{\text{ref}} y_i$ (for systematic notation we note the variable as y_i but not as x). Let $J \subseteq \{1, \ldots, n\}$ be those indicies corresponding to the latter case.

The substitution $\gamma_{P(t_1,\ldots,t_n)}$ mentioned earlier will be defined below with the domain $\{y_j \mid j \in J\}$ and in such a way that

$$(t_j \mapsto_{\text{ref}} y_j)\gamma_{P(t_1,\ldots,t_n)}\gamma_v \in I^{\text{bl}} \text{ , for all } j \in J, \text{ and} \tag{22}$$

$$t_i^{I,v} = t'_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v \text{ , for all } i = 1, \ldots, n. \tag{23}$$

We get

$$\begin{array}{lll}
 & I, v \models^{\star} P(t_1, \ldots, t_n) & \\
\text{iff} & P^I(t_1^{I,v}, \ldots, t_n^{I,v}) = \mathit{true} & \text{(by definition of } \models^{\star}\text{)} \\
\text{iff} & P^I(t'_1\gamma_{P(t_1,\ldots,t_n)}\gamma_v, \ldots, t'_n\gamma_{P(t_1,\ldots,t_n)}\gamma_v) = \mathit{true} & \text{(by (23))} \\
\text{iff} & P(t'_1\gamma_{P(t_1,\ldots,t_n)}\gamma_v, \ldots, t'_n\gamma_{P(t_1,\ldots,t_n)}\gamma_v) \in I^{\text{bl}} & \text{(by definition of } P^I\text{)} \\
\text{iff} & P(t'_1, \ldots, t'_n)\gamma_{P(t_1,\ldots,t_n)}\gamma_v \in I^{\text{bl}} & \text{(trivial)}
\end{array}$$

By definition of "pulling out function terms" all the variables y_j, for all $j \in J$, are pairwise different, different to all other variables introduced by pulling out function terms of other body atoms of $\mathcal{B}$, and different to all the variables $x_1, \ldots, x_k$. The substitutions $\gamma_{B_1}, \ldots, \gamma_{B_m}$ thus may all be composed, e.g. in this order, and also composed with γ_v, and the resulting substitution $\gamma = \gamma_{B_1} \cdots \gamma_{B_m}\gamma_v$ can be used equivalently instead of $\gamma_{P(t_1,\ldots,t_n)}\gamma_v$ in the chain of equivalences just derived, and also in (22) and (23).

Recall from above the assumption $I, v \models^\star B$, for all body atoms B of $\mathcal{B}$. With the just said and from the equivalences above it follows $P(t'_1, \ldots, t'_n)\gamma \in I^{\mathrm{bl}}$, and from (22) it follows $(t_j \mapsto_{\mathrm{ref}} y_j)\gamma \in I^{\mathrm{bl}}$, for all $j \in J$. Together this entails (20).

In order to complete the proof of the first subgoal stated above, it remains only to define $\gamma_{P(t_1,\ldots,t_n)}$ in such a way that (22) and (23) hold. To this end, set inititally $\gamma_{P(t_1,\ldots,t_n)} := \varepsilon$ (the empty substitution) and extend it by considering the terms t_i, for $i = 1, \ldots, n$. We distinguish two main cases.

In the first case t_i is a variable, a constant or a function term with a 0-ary function symbol. As such terms are not pulled out, it follows $t_i = t'_i$. We consider the three subcases, and in all of them the substitution $\gamma_{P(t_1,\ldots,t_n)}$ is kept unmodified:

– if t_i is a variable, it is one of the variables $x_1, \ldots, x_k$. To prove (23) observe

$$\begin{aligned} t_i^{I,v} &= v(t_i) && (t_i \text{ is a variable}) \\ &= t_i\gamma_v && (\text{by definition of } \gamma_v) \\ &= t'_i\gamma_v && (t_i = t'_i) \\ &= t'_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v \ . \end{aligned}$$

The last identity follows from the easy to check invariant in the construction of $\gamma_{P(t_1,\ldots,t_n)}$, that $\gamma_{P(t_1,\ldots,t_n)}$ will not move any variable $x_1, \ldots, x_k$.

– if t_i is a constant c we have $c^I = c$ by definition of $\cdot^I$. To prove (23) observe

$$\begin{aligned} t_i^{I,v} &= t_i && (t_i = c^I = c) \\ &= t'_i && (t_i = t'_i) \\ &= t'_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v \ . \end{aligned}$$

– if t_i is a 0-ary function symbol a it cannot have a proper subterm. By definition of $\cdot^I$ it follows $a^I = a$. The proof of (23) is the same then as in the preceeding case.

In the second case t_i is a function term $f(s_1, \ldots, s_m)$ for some non 0-ary function symbol f and some term $s_1, \ldots, s_m$, where $m > 0$. The term t'_i then is a variable y_i and $\mathcal{B}'$ includes an atom $f(s_1, \ldots, s_m) \mapsto_{\mathrm{ref}} y_i$.

Recall that $\mathcal{P}$ is given as a *flat* clause set. This implies s_l is a constant or a variable, for each $l = 1, \ldots, m$. If s_l is a constant we have

$$s_l^{I,v} = s_l^I = s_l \tag{24}$$

Likewise, if s_l is a variable it must be one of $x_1, \ldots, x_k$ and we have

$$s_l^{I,v} = v(s_l) = s_l\gamma_v \tag{25}$$

Because by the invariant stated above, that $\gamma_{P(t_1,\ldots,t_n)}$ will not move any variable $x_1, \ldots, x_k$, we obtain easily in both cases

$$s_l^{I,v} = s_l\gamma_{P(t_1,\ldots,t_n)}\gamma_v \tag{26}$$

Next we consider the term t_i and its value

$$t_i^{I,v} = (f(s_1, \ldots, s_m))^{I,v} = f^I(s_1^{I,v}, \ldots, s_m^{I,v}) \overset{(26)}{=} f^I(s_1\gamma_{P(t_1,\ldots,t_n)}\gamma_v, \ldots, s_m\gamma_{P(t_1,\ldots,t_n)}\gamma_v) \tag{27}$$

For slightly lighter notation define $d_l = s_l\gamma_{P(t_1,\ldots,t_n)}\gamma_v$.

We distinguish two cases according to the definition of f^I.

In the first case $f^I(d_1,\ldots,d_m) = d$, for some proper subterm d of $f(d_1,\ldots,d_m)$. From the definition of f^I it follows $f(d_1,\ldots,d_m) \mapsto d \in I^{\mathrm{bl}}$. Recall that the domain Δ is comprised of terms. Because y_i does not occur in any term $t_1,\ldots,t_{i-1}$ by construction, we may assume that the substitution $\gamma_{P(t_1,\ldots,t_n)}$ constructed so far does not move y_i. Therefore we can define

$$\gamma_{P(t_1,\ldots,t_n)} := \gamma_{P(t_1,\ldots,t_n)}\{y_i/d\}$$

and it follows $y_i\gamma_{P(t_1,\ldots,t_n)} = d$.

Because y_i is a variable introduced by pulling out, it must be different to all terms $s_1,\ldots,s_m$. Therefore $d_l = s_l\gamma_{P(t_1,\ldots,t_n)}\gamma_v$ still holds.

Above we defined the index set J as comprised of those indices that are subject to pulling out terms in $P(t_1,\ldots,t_n)$. It follows $i \in J$.

Recall we have to show (22) and (23). First we turn to (22):

$$\begin{aligned}
&(f(s_1,\ldots,s_m) \mapsto y_i)\gamma_{P(t_1,\ldots,t_n)}\gamma_v \\
&= f(s_1,\ldots,s_m)\gamma_{P(t_1,\ldots,t_n)}\gamma_v \mapsto y_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v \\
&= f(d_1,\ldots,d_m) \mapsto y_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v && (d_l = s_l\gamma_{P(t_1,\ldots,t_n)}\gamma_v) \\
&= f(d_1,\ldots,d_m) \mapsto d\gamma_v && (y_i\gamma_{P(t_1,\ldots,t_n)} = d, \text{ as derived above}) \\
&= f(d_1,\ldots,d_m) \mapsto d && (\text{trivial})
\end{aligned}$$

But then, from $f(d_1,\ldots,d_m) \mapsto d \in I^{\mathrm{bl}}$, as concluded further above, these identities and clause (4) it follows $(f(s_1,\ldots,s_m) \mapsto_{\mathrm{ref}} y_i)\gamma_{P(t_1,\ldots,t_n)}\gamma_v \in I^{\mathrm{bl}}$ as desired.

Finally to this case, (23) is proven as follows:

$$\begin{aligned}
t_i^{I,v} &= f^I(s_1\gamma_{P(t_1,\ldots,t_n)}\gamma_v,\ldots,s_m\gamma_{P(t_1,\ldots,t_n)}\gamma_v) && (\text{by (27)}) \\
&= f^I(d_1,\ldots,d_m) && (d_l = s_l\gamma_{P(t_1,\ldots,t_n)}\gamma_v) \\
&= d && (\text{assumption of current case}) \\
&= y_i\gamma_{P(t_1,\ldots,t_n)} && (\text{see above}) \\
&= y_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v && (\text{trivial}) \\
&= t_i'\gamma_{P(t_1,\ldots,t_n)}\gamma_v && (t_i' = y_i, \text{ see above})
\end{aligned}$$

In the second case $f^I(d_1,\ldots,d_m) = f(d_1,\ldots,d_m)$. The proof is similar to the first case.

Because y_i does not occur in any term $t_1,\ldots,t_{i-1}$ by construction, we may assume that the substitution $\gamma_{P(t_1,\ldots,t_n)}$ constructed so far does not move y_i. Therefore we can define

$$\gamma_{P(t_1,\ldots,t_n)} := \gamma_{P(t_1,\ldots,t_n)}\{y_i/f(d_1,\ldots,d_m)\}$$

and it follows $y_i\gamma_{P(t_1,\ldots,t_n)} = f(d_1,\ldots,d_m)$.

Because y_i is a variable introduced by pulling out, it must be different to all terms $s_1, \ldots, s_m$. Therefore $d_l = s_l\gamma_{P(t_1,\ldots,t_n)}\gamma_v$ still holds.

Above we defined the index set J as comprised of those indices that are subject to pulling out terms in $P(t_1, \ldots, t_n)$. It follows $i \in J$.

Recall we have to show (22) and (23). First we turn to (22):

$$
\begin{aligned}
& (f(s_1,\ldots,s_m) \mapsto y_i)\gamma_{P(t_1,\ldots,t_n)}\gamma_v \\
&= f(s_1,\ldots,s_m)\gamma_{P(t_1,\ldots,t_n)}\gamma_v \mapsto y_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v \\
&= f(d_1,\ldots,d_m) \mapsto y_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v && (d_l = s_l\gamma_{P(t_1,\ldots,t_n)}\gamma_v) \\
&= f(d_1,\ldots,d_m) \mapsto f(d_1,\ldots,d_m)\gamma_v && (y_i\gamma_{P(t_1,\ldots,t_n)} = f(d_1,\ldots,d_m), \text{ as derived above}) \\
&= f(d_1,\ldots,d_m) \mapsto f(d_1,\ldots,d_m) && (\text{trivial})
\end{aligned}
$$

But then, from $f(d_1,\ldots,d_m) \mapsto f(d_1,\ldots,d_m) \in I^{\mathrm{bl}}$, as concluded further above, these identities and clause (3) it follows $(f(s_1,\ldots,s_m) \mapsto_{\mathrm{ref}} y_i)\gamma_{P(t_1,\ldots,t_n)}\gamma_v \in I^{\mathrm{bl}}$ as desired.

Finally to this case, (23) is proven as follows:

$$
\begin{aligned}
t_i^{I,v} &= f^I(s_1\gamma_{P(t_1,\ldots,t_n)}\gamma_v, \ldots, s_m\gamma_{P(t_1,\ldots,t_n)}\gamma_v) && (\text{by (27)}) \\
&= f^I(d_1,\ldots,d_m) && (d_l = s_l\gamma_{P(t_1,\ldots,t_n)}\gamma_v) \\
&= f(d_1,\ldots,d_m) && (\text{assumption of current case}) \\
&= y_i\gamma_{P(t_1,\ldots,t_n)} && (\text{see above}) \\
&= y_i\gamma_{P(t_1,\ldots,t_n)}\gamma_v && (\text{trivial}) \\
&= t_i'\gamma_{P(t_1,\ldots,t_n)}\gamma_v && (t_i' = y_i, \text{ see above})
\end{aligned}
$$

This concludes the proof of the first subgoal.

Now that we have shown (20) and (21), with $I^{\mathrm{bl}} \models (\mathcal{H} \leftarrow \mathcal{B}', \mathsf{dom}(x_1), \ldots, \mathsf{dom}(x_k))\gamma$ it follows $H\gamma \in I^{\mathrm{bl}}$ for some head atom H of $\mathcal{H}$. The second subgoal of the proof now is to show $I, v \models^\star H$. This suffices to obtain $I, v \models \mathcal{H} \leftarrow \mathcal{B}$ and the proof will thus be complete.

Instead of γ we can work now with γ_v, as defined above, and it still holds $H\gamma_v \in I^{\mathrm{bl}}$. This holds, because each variable of H is among $\{x_1, \ldots, x_k\}$, which is the domain of γ_v.

The head atom H can be written as $P(t_1, \ldots, t_n)$. Thus we have $P(t_1, \ldots, t_n)\gamma_v = P(t_1\gamma_v, \ldots, t_n\gamma_v) \in I^{\mathrm{bl}}$. We will first show there is a (ground) atom $P(d_1, \ldots, d_n)$, for some domain elements $d_1, \ldots, d_n \in \Delta$ such that

$$P(d_1,\ldots,d_n) \in I^{\mathrm{bl}} \text{ , and} \tag{28}$$

$$t_i^{I,v} = d_i \text{ , for all } i = 1,\ldots,n \tag{29}$$

This will actually suffice to prove $I, v \models^\star H$:

$$\begin{array}{lll} & P(d_1,\ldots,d_n) \in I^{\mathrm{bl}} & \\ \text{iff} & P^I(d_1,\ldots,d_n) = \mathit{true} & \text{(by definition of } P^I\text{)} \\ \text{iff} & P^I(t_1^{I,v},\ldots,t_n^{I,v}) = \mathit{true} & \text{(by (29))} \\ \text{iff} & I, v \models^\star P(t_1,\ldots,t_n) & \end{array}$$

It thus only remains to prove (28) and (29). The proof will be by proceeding along $i = 1,\ldots,n$, where we show how d_i can be obtained from $t_i\gamma_v$ so that (29) holds. In each intermediate stage i we will have $P(d_1,\ldots,d_{i-1},t_i\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$, which in the end implies (28).

Recall that $\mathcal{P}$ is given as a *flat* clause set. Therefore, t_i is a variable (one of $x_1,\ldots,x_k$), a constant, or a function term $f(s_1,\ldots,s_m)$ such that $s_1,\ldots,s_m$ all are variables or constants. We consider all these cases.

- if t_i is a variable, it is one of the variables $x_1,\ldots,x_k$. To prove (29) consider

  $$\begin{array}{ll} t_i^{I,v} = v(t_i) & (t_i \text{ is a variable}) \\ = t_i\gamma_v & \text{(by definition of } \gamma_v\text{)} \\ =: d_i \ . & \end{array}$$

  Clearly, from $P(d_1,\ldots,d_{i-1},t_i\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$ it follows $P(d_1,\ldots,d_{i-1},d_i,t_{i+1}\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$.
- if t_i is a constant c we have $c^I = c$ by definition of $\cdot^I$. To prove (29) consider

  $$\begin{array}{ll} t_i^{I,v} = t_i & (t_i = c^I = c) \\ = t_i\gamma_v & \text{(trivial)} \\ =: d_i & \end{array}$$

  As above, from $P(d_1,\ldots,d_{i-1},t_i\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$ it follows $P(d_1,\ldots,d_{i-1},d_i,t_{i+1}\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$.
- If none of the previous cases applies, t_i must be a function term $f(s_1,\ldots,s_m)$. As said above, $s_1,\ldots,s_m$ all are variables or constants. With the same argumentation that led to (27) above, we have here

  $$t_i^{I,v} = (f(s_1,\ldots,s_m))^{I,v} = f^I(s_1^{I,v},\ldots,s_m^{I,v}) = f^I(s_1\gamma_v,\ldots,s_m\gamma_v) \qquad (30)$$

  We distinguish two cases according to the definition of f^I.
  In the first case $f^I(s_1\gamma_v,\ldots,s_m\gamma_v) = d$, for some proper subterm d of $f(s_1\gamma_v,\ldots,s_m\gamma_v)$. From the definition of f^I it follows $f(s_1\gamma_v,\ldots,s_m\gamma_v) \mapsto d \in I^{\mathrm{bl}}$. Recall we assume $P(d_1,\ldots,d_{i-1},t_i\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$. With $t_i\gamma_v = f(s_1,\ldots,s_m)\gamma_v = f(s_1\gamma_v,\ldots,s_m\gamma_v)$ and the clause (12) conclude $P(d_1,\ldots,d_{i-1},d,t_{i+1}\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$. Thus we define $d_i := d$ to prove the invariant.
  The equation (29) is obtained as follows:

  $$\begin{array}{ll} t_i^{I,v} = f^I(s_1\gamma_v,\ldots,s_m\gamma_v) & \text{(by (30))} \\ = d & \text{(assumption of this case)} \\ = d_i & \text{(definition of } d_i\text{)} \end{array}$$

If the first case $f^I(s_1\gamma_v,\ldots,s_m\gamma_v) = d$ does not apply, then, by definition of f^I, $f^I(s_1\gamma_v,\ldots,s_m\gamma_v) = f(s_1\gamma_v,\ldots,s_m\gamma_v)$. We define $d_i := f(s_1\gamma_v,\ldots,s_m\gamma_v)$.
Recall we assume $P(d_1,\ldots,d_{i-1},t_i\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$. With $t_i\gamma_v = f(s_1,\ldots,s_m)\gamma_v = f(s_1\gamma_v,\ldots,s_m\gamma_v) = d_i$ the invariant $P(d_1,\ldots,d_{i-1},d_i,t_{i+1}\gamma_v,\ldots,t_n\gamma_v) \in I^{\mathrm{bl}}$ follows.
The equation (29) is obtained as follows:

$$\begin{aligned} t_i^{I,v} &= f^I(s_1\gamma_v,\ldots,s_m\gamma_v) && \text{(by (30))} \\ &= f^I(s_1\gamma_v,\ldots,s_m\gamma_v) && \text{(assumption of this case)} \\ &= d_i && \text{(definition of } d_i\text{)} \end{aligned}$$

This completes the proof. □