Using Grammars for Computational Representations of Herbrand Models

Robert Matzinger*

June 21, 1996

Abstract. Finding computationally valuable representations of models of predicate logic formulas is an important task in many fields related to automated theorem proving, e.g. automated model building or semantic resolution. For representing a single (Herbrand) model we need a mechanism to describe all true ground atoms in the model, i.e. to characterize potentially infinite sets of terms. In this article we investigate the use of context-free languages for this purpose. We focus on expressive power (which we find out to be exactly the finite models) and on algorithmic issues like clause evaluation, equivalence test (which we solve by methods of automated theorem proving), etc., proving our approach to be an interesting base for investigating connections between term schematizations, formal language theory and automated model building.

1 Introduction

Representing single models of predicate logic formulas plays an important role in various subfields of automated theorem proving. We just mention two of them:

1. An interesting branch of automated theorem proving called *automated model building* not only tries to decide the satisfiability of some PL formula (or clause formula), but also attempts to generate a model in case satisfiability is found; see e.g. [Tam91, CZ92, Sla92, FL93, BCP94, FL96]. Note that a model of a formula $\neg A$ is in fact nothing else than a counterexample for A[1], which may be of enormous help for a human engineer who tries to find the reason for A not being a tautology (imagine A to be e.g. a faulty system description).

2. *Model based resolution refinements* restrict the number of clauses generated by a resolution theorem prover by evaluating the clauses in a given model and resolving true against false clauses only (see e.g. [Lov78, CL73, Sla93]). For this purpose the availability of computationally feasible model representations is an important issue.

However, as soon as we don't restrict ourselves to finite models, we have to ask what "representing a model" actually should mean. But even if we cope with finite models a concise symbolic representation may be a lot easier to deal with than the explicit representation of a finite model with, say, some thousand domain elements. Still techniques for representing potentially infinite models symbolically are rarely explored.
From [FL96] we take the following three requirements for model representations[2]:

Ground Atom Test. For a given model representation it should be (computationally) easy to decide whether a given ground atom is true or false in the represented model. See Section 4.

Equivalence Test. It should be possible to decide whether two given representations represent the same model or not. This is done in Section 6.

Clause Test. There should be an algorithm to decide whether a given clause is true or false in the represented model. Refer to Section 5.

*Technische Universität Wien, Institut für Informationssysteme, Abteilung für Wissensbasierte Systeme; Treitlstr. 3/E184-3, A-1040 Wien/Austria/Europe; phone: +43(1)58801/8196; telefax: +43(1)5817966; email: `matzi@kr.tuwien.ac.at`

[1] As usual in the resolution literature we refute the negation of a given formula A.

[2] Note that many model building algorithms in the literature lack methods for dealing with these requirements.

It is clear that especially for using a model as a counterexample to assist a human engineer, we have to deal with the intuitive requirement of

Understandability. The model representation should be intuitively readable by a human engineer.

Because of this requirement we decided to restrict ourselves to Herbrand models, so both domain and interpretation of function symbols are clear, fixed and intuitive. Still we know we can find a Herbrand model for any[3] satisfiable formula. To describe a certain Herbrand model over a fixed signature we just have to find a method for specifying potentially infinite sets of ground atoms (which we can arbitrarily read as terms or as strings themselves), i.e. we need an appropriate *term schematization technique*. Among different possibilities we could base our approach on

- Formal Language Theory (see e.g. [Sal73]). It offers a great tradition of representing infinite sets of strings. Context-free grammars in particular appear to be very usual in computer science (e.g. for specifying the syntax of programming languages) so we expect them to be intuitively readable for many engineers in the field.
- Term schematization techniques coming from Automated Deduction (see e.g. [CHK90, Sal92, Com95, GH96]). However these ideas mostly focus on different objectives than we do (e.g. the availability of an unification algorithm) and don't appear to be so widespread in computer science, so we postponed this approach.

In this paper we focus on the first idea and use context free languages[4] for representing Herbrand models. Having chosen a particular model representation we are interested in the

Characterization Question, i.e. to characterize the models we can represent or the formulas we can satisfy with a representable model, which we settle in Section 5.

Unfortunately room is too limited to tackle other representation techniques and investigate the

Comparison Question, i.e. to compare different representation mechanisms with respect to their representation power and their computational feasibility,

so we just scratch the surface for these topics; see Section 7.

2 A Triggering Example

In [FL93, FL96] a model representation technique called *atom representation* is introduced (along with a model building algorithm for obtaining linear[5] atom representations for formulas in interesting clause classes). Atom representations define a Herbrand model by stating that all ground instances of a finite set of atoms should be true in the model, the rest should be false. E.g. the atom representation $\{P(a, f(x)), Q(g(x, y))\}$ represents the Herbrand model in which $P(a, f(t_1))$ and $Q(g(t_2, t_3))$ are true for any ground terms t_1, t_2, t_3 and all other ground atoms are false. Unfortunately there are very simple satisfiable sets of clauses that do not have a model with an atom representation:

Example 1. $\mathcal{C}_{ex} := \{\ \{P(x) \vee P(f(x))\},\ \{\neg P(x) \vee \neg P(f(x))\}\ \}$

It is easy to see that $\mathcal{C}_{ex}$ has exactly two Herbrand models: $\left\{P\left(f^{2i}(a)\right)\right\}$ and $\left\{P\left(f^{2i+1}(a)\right)\right\}$ for $i \geq 0$, none of which has an atom representation. However, the structure of true atoms is very simple in this example and could be expressed by using some intuitive kind of regular expression. As we don't want to restrict ourselves to unary function symbols we proceed to context-free languages

[3] In some sense; see Section 3.

[4] In particular we use languages produced by term grammars similar as in [GS84] and [McA92]; see Section 4.

[5] An expression is called linear iff every variable occurs at most once within the expression. A clause is called linear iff all of its literals are linear.

(so we can also keep the syntactical structure of the terms). It is easy to see that the set of terms t for which $P(t)$ is true in the first model of Example 1 forms a context-free language represented by the grammar[6] $G_{ex} := (\{f, a, `(' , `)' \}, \{P\}, \{P \longrightarrow f(f(P)), P \longrightarrow a\}, P)$. Representing Herbrand models via context-free languages like in above example is the basic idea we will follow in the next sections.

3 Preliminaries

We assume the reader to be familiar with the terms first order logic, clause form, model, Herbrand model, resolution, etc. as well as with formal languages, grammars (context-free, regular, ...) and so on. Unless stated different (all the symbols may appear with arbitrary indices) $u, v, w, x, y, z, \ldots$ denote variables in formulas, $f, g, h, \ldots$ denote function symbols (of arbitrary arity > 0), $a, b, c, \ldots$ denote constant symbols, $r, s, t, \ldots$ denote (ground) terms, $A, B, C, \ldots, M, N, O, P, Q, \ldots$ denote clauses, predicate symbols, or nonterminal symbols in grammars, $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ denote sets of clauses, formulas, etc. We will arbitrarily omit the starting symbol of a grammar, meaning the languages that are produced per nonterminal.

Example 2. The grammar $G_1 := (\mathcal{A}_1, \{P, Q\}, \Pi_1)$ with $\mathcal{A}_1 := \{f, a, '(' , ')'\}$ and $\Pi_1 := \{ P \longrightarrow f(Q),\ P \longrightarrow a,\ Q \longrightarrow f(P) \}$ generates $L_P = \left\{f^{2i}(a) | i \geq 0\right\}$ from P and $L_Q = \left\{f^{2i+1}(a) | i \geq 0\right\}$ from Q.

Throughout this article we talk about first order logic with arbitrary function symbols but without equality. W.l.o.g. we assume all formulas to be in negation normal form[7]. We call any such formula to be in *skolemized form* iff it does not contain existential quantifiers. By Herbrand's theorem (see standard text books like [BJ74]) we know that there is a sat-equivalent[8] skolemized formula for any predicate logic formula (in negation normal form). Note that for model building we cannot be satisfied with pure sat-equivalence among the formulas. Additionally we have to require a translation of the models, i.e. we need a simple method to obtain a model of the original formula from a model of its sat-equivalent pendant. For skolemization this is trivial as by the standard method a Herbrand model of the skolemized formula is always a model of the original formula itself (simply by interpreting the existential quantifiers according to the skolem functions we introduced). Therefore it makes sense to restrict ourselves to building and representing Herbrand models of skolemized formulas or – equivalently – of clause sets.

For a clause $\{L_1 \vee \ldots \vee L_n\}$ (the L_i being literals) we sometimes use the notational shortcut $\{L_1 \ldots L_k \longleftarrow \neg L_{k+1} \ldots \neg L_n\}$ or $\{\neg L_1 \ldots \neg L_k \longrightarrow L_{k+1} \ldots L_n\}$, always deleting occurrences of $\neg\neg$ and occasionally omitting the braces. For a skolemized formula $\mathcal{F}$ we call $\Sigma_{\mathcal{F}} = (\mathcal{P}_{\mathcal{F}}, F_{\mathcal{F}}, K_{\mathcal{F}})$ the *signature* of $\mathcal{F}$ where $\mathcal{P}_{\mathcal{F}}$ is the set of predicate symbols, $F_{\mathcal{F}}$ is the set of function symbols (with arity > 0) and $K_{\mathcal{F}}$ is the set of constant symbols occurring in $\mathcal{F}$. W.l.o.g. we always assume $K_{\mathcal{F}} \neq \emptyset$. $\mathrm{HD}_{\mathcal{F}}$ denotes the set of terms that can be built from constant symbols $K_{\mathcal{F}}$ and function symbols $F_{\mathcal{F}}$, i.e. the Herbrand domain of $\mathcal{F}$. Analogously we will use HD_{Σ} to denote the set of terms built from a signature Σ.

Since we want to describe Herbrand models by describing the ground terms for which a certain predicate is true via mechanisms of formal language theory, it is convenient to restrict ourselves to one-place (monadic) predicate letters. As we allow full functional structure in terms, we do not lose generality because we can translate every skolemized formula to a sat-equivalent one that contains monadic predicates only. This can be done in a way that allows easy translation of the models in both directions:

Assume for a skolemized formula $\mathcal{F}$ we obtain a formula $\mathcal{F}'$ by replacing every literal $(\neg)P(t_1, \ldots, t_n)$ in $\mathcal{F}$ by subformulas $(\neg)T(p(t_1, \ldots, t_n)) \vee N(t_1) \vee \ldots \vee N(t_n)$ (p being a new n-place function symbol for every predicate P, T and N being two new one-place predicate symbols which in-

[6] As usual we denote a grammar as a 4-tuple $G = (\mathcal{A}, \mathcal{N}, \Pi, S)$ where $\mathcal{A}$ is the set of terminal symbols, $\mathcal{N}$ is the set of nonterminal symbols, $\Pi \subseteq \mathcal{N} \times (\mathcal{A} \cup \mathcal{N})^*$ is a finite set of productions and $S \in \mathcal{N}$ is the starting symbol.

[7] Formulas with quantifiers $\forall$ and $\exists$, connectives $\vee$ and $\wedge$, where $\neg$ is only allowed in front of atoms.

[8] Two formulas are called sat-equivalent if one is satisfiable iff the other is satisfiable.

tuitively abbreviate "true" and "nonsense") and after all adding clauses (or the respective disjuncts) $\{N(p(x_1,\ldots,x_n))\}$, $\{N(x_i) \longrightarrow N(f(x_1,\ldots,x_k))\}$, $\{\neg N(a)\}$ and $\{\neg N(x_1),\ldots,\neg N(x_k) \longrightarrow \neg N(f(x_1,\ldots,x_k))\}$ for any new function symbol p, for every "old" function symbol f, for any constant symbol a and for any appropriate index i. Now we can state:

Lemma 1. *For any Herbrand model Ω of a skolemized formula $\mathcal{F}$ there is a Herbrand model Ω' of $\mathcal{F}'$ such that*

the interpretations of $P(t_1,\ldots,t_n)$ in Ω and $T(p(t_1,\ldots,t_n))$ in Ω' are the same for any ground terms $t_1,\ldots,t_n$ that are in both domains, (1)

and vice versa, i.e. for any Herbrand model Ω' of $\mathcal{F}'$ there is a Herbrand model Ω of $\mathcal{F}$ such that (1) *holds.*

Proof. Observe that the clauses we added for N ensure that $N(t)$ is true exactly on terms $\in \mathrm{HD}_{\mathcal{F}'} \setminus \mathrm{HD}_{\mathcal{F}}$ (i.e. terms that contain one of the newly introduced function symbols) for any Herbrand model Ω' of $\mathcal{F}'$. Thus for terms $t_1,\ldots,t_n \in \mathrm{HD}_{\mathcal{F}}$ the interpretation of $(\neg)T(p(t_1,\ldots,t_n)) \vee N(t_1) \vee \ldots \vee N(t_n)$ and $(\neg)T(p(t_1,\ldots,t_n))$ is equal (w.r.t. Ω'), whereas otherwise $(\neg)T(p(t_1,\ldots,t_n)) \vee N(t_1) \vee \ldots \vee N(t_n)$ is always true. From this and the fact that the subformulas we invented occur purely positive it is not hard to verify the lemma. □

This construction provides the required easy translation of the models, as any model of $\mathcal{F}'$ can be translated to a model of $\mathcal{F}$ simply by stripping all "nonsense terms", i.e. all terms containing one of the newly introduced function symbols (or in other words all terms for which N is true) and use the correspondence between $T(p(\ldots))$ and $P(\ldots)$.

4 Formulating Herbrand Models as Languages

If we restrict ourselves to monadic skolemized formulas, we can fully describe a Herbrand model Ω of a formula $\mathcal{F}$ (with fixed signature) by assigning a set of ground terms L_P to every predicate symbol P and stating that $P(t)$ should be interpreted as true iff $t \in L_P$. The basic idea of this article is to describe each L_P as a formal language $\subseteq \mathrm{HD}_{\mathcal{F}}$. We consider an arbitrary, but fixed signature $\Sigma = (\mathcal{P}, F, K)$. We abbreviate $\mathcal{A}_\Sigma := F \cup K \cup \{$'(' , ')' , ','$\}$.

Definition 2. A context-free grammar $G = (\mathcal{A}_\Sigma, \mathcal{P} \cup \mathcal{N}, \Pi)$ ($\mathcal{N}$ being a finite set of additional nonterminal symbols disjoint with $\mathcal{P}$, F and K) is said to be a *(flat) term grammar* iff each production is either of the form $N \longrightarrow a$ with a being a constant or is of the form $N \longrightarrow f(N_1,\ldots,N_n)$ with f being a function symbol of arity n and $N, N_1,\ldots,N_n \in \mathcal{P} \cup \mathcal{N}$ being nonterminals.

Note that if we stick to our notational conventions, it suffices to specify the production set of a term grammar. Clearly if a language L_P is produced by a term grammar G, $P \in \mathcal{P}$, then $L_P \subseteq \mathrm{HD}_\Sigma$, i.e. the grammar produces Herbrand terms only. However there is strong evidence for the conjecture that every context-free subset of HD_Σ can be generated by a term grammar.

Definition 3. The Herbrand interpretation (or Herbrand model) Ω_G *defined via the term grammar* G is defined by: $P(t)$ is interpreted as true iff $t \in L_P$ where L_P is generated by the grammar G (P being a predicate symbol as well as a nonterminal symbol in the term grammar). Any Herbrand model of a (skolemized) monadic formula which can be defined via a term grammar is called a *context-free model.*

Representing Herbrand models by term grammars well fulfils our requirement of **understandability** as we already mentioned. We also get an efficient **atom test** for free as evaluating $P(t)$ (for a ground term t) is nothing else than a syntax check for t with respect to the term grammar (a context-free grammar), which can be done in at most polynomial time. (Based on Section 5 and on [GS84] it can be shown that it's even possible in linear time.)

5 Finiteness and Clause Evaluation

Before tackling clause test and equivalence test we ask ourselves what kind of models we are able to express as context free models. We are especially interested in the question whether (essentially) infinite models can be expressed or not. Although the answer is no, which we show by transforming an arbitrary context-free model to a finite one, we find it interesting that also the opposite is possible, i.e. to transform any finite model to a context-free one (in some sense).

Definition 4. A *Herbrand image* of an interpretation Ω of a skolemized formula is a Herbrand interpretation Ω_H such that there exists a (not necessarily surjective) homomorphism $h : \Omega_H \rightarrow \Omega$ (with respect to both functions and predicates in the obvious sense). We call Ω and Ω_H *Herbrand-equivalent*, if h is surjective.

Note that although the mapping works from the Herbrand interpretation to the original one, the Herbrand image is still uniquely fixed by the interpretation of the constant symbols (w.l.o.g. we assume there is at least one). Furthermore the interpretation of ground atoms is always the same in a model and its Herbrand image. Thus Herbrand-equivalent models are truth-equivalent for arbitrary skolemized formulas, so we call them just *equivalent*. Note that every model of a skolemized formula must contain a submodel that is equivalent to the Herbrand image.
For this section we shall just talk about skolemized monadic formulas and we consider the signature Σ as fixed. Now if we succeed to produce a grammar that has the property that every ground term $t \in \mathrm{HD}_\Sigma$ is generated by exactly one nonterminal symbol, we can read the nonterminal symbols as elements of a finite domain of a model which indeed is equivalent to the original Herbrand model (we will have a closer look on that soon). Thus we proceed by showing that we can transform any term grammar G to an (in a certain sense) equivalent grammar G^0_{unique} that satisfies above property.

Definition 5. For a set of nonterminals $\mathcal{N} = \{N_1, \ldots, N_n\}$, $\mathcal{N}^{\neg\wedge}$ denotes the equivalence classes of the terms that can be built from $\mathcal{N}$ and $\neg\mathcal{N} = \{\neg N_1, \ldots, \neg N_n\}$ with a binary function symbol $\wedge$ with respect to idempotency, commutativity and associativity of $\wedge$.

Clearly this is isomorphic to the powerset of $\mathcal{N} \cup \neg\mathcal{N}$, thus it's a finite set; in the sequel we will not distinguish between different representations of elements of $\mathcal{N}^{\neg\wedge}$. We naturally associate a language with every nonterminal in $\mathcal{N}^{\neg\wedge}$:

Definition 6. Let $\mathcal{N}$ be a set of nonterminals of a grammar. The *language corresponding to*

- $N \in \mathcal{N}$ is L_N, i.e. the language produced by the grammar from the nonterminal N,
- $\neg N$ is $\mathrm{HD}_\Sigma \setminus L_N$ for $N \in \mathcal{N}$,
- $N_1 \wedge N_2$ is $L_{N_1} \cap L_{N_2}$ for $N_1, N_2 \in \mathcal{N}^{\neg\wedge}$.

See that this is indeed a function from symbols in $\mathcal{N}^{\neg\wedge}$ to languages $\subseteq \mathrm{HD}_\Sigma$. We are going to enhance our grammar G by additional nonterminals and additional productions. As their definitions require a lot of combinations we just indicate the construction by restricting the actual number of productions and nonterminals. It should be obvious how to extend the definition to arbitrary cases.

Definition 7. Let's start with $G = (\mathcal{A}_\Sigma, \mathcal{N}, \Pi)$. As a first step we consider $G' = (\mathcal{A}_\Sigma, \mathcal{N}^{\neg\wedge}, \Pi)$. For convenience we put an additional nonterminal symbol T into this grammar and add rules $T \longrightarrow a$ and $T \longrightarrow f(T, T, \ldots, T)$ to the production set for every constant symbol a and every function symbol f (with the right arity) thus obtaining G''. To G'' we apply the following rules in any order as often as possible (i.e. until no new productions can be built) always considering $\neg\neg A = A$. Let $\Pi[A, f]$ be the set of rules of the form $A \longrightarrow f(\ldots)$ in Π (and similar $\Pi[A, a]$ for constant symbols a).

- For every $\Pi[A,f]$ and $\Pi[A,a]$ with $A \in \mathcal{N}$ that is **empty**, we add rules $\neg A \longrightarrow f(T,\ldots,T)$ and $\neg A \longrightarrow a$ to the production set of the grammar.
- Let $\Pi[A,f] = \{A \longrightarrow f(B,C), A \longrightarrow f(D,E)\}$ $(A \in \mathcal{N})$, then we add the productions $\neg A \longrightarrow f(\neg B \wedge \neg D, T)$, $\neg A \longrightarrow f(\neg B, \neg E)$, $\neg A \longrightarrow f(\neg D, \neg C)$, $\neg A \longrightarrow f(T, \neg C \wedge \neg E)$ to the production set (and we do similarly for $\Pi[A,f]$ containing more or less than two elements and f having an arity $\neq 2$).

On the result we apply the following two rules to all $A, B \in \mathcal{N}^{\neg\wedge}$ (!) as often as possible and call the resulting grammar $G^{\neg\wedge}$:

- Let $\Pi[A,a] = \{A \longrightarrow a\}$ and $\Pi[B,a] = \{B \longrightarrow a\}$, then we add the production $A \wedge B \longrightarrow a$ to the grammar.
- Let $\Pi[A,f] = \{A \longrightarrow f(D_1,E_1), A \longrightarrow f(F_1,G_1)\}$ and $\Pi[B,f] = \{B \longrightarrow f(D_2,E_2), A \longrightarrow f(F_2,G_2)\}$, then we add the productions $A \wedge B \longrightarrow f(D_1 \wedge D_2, E_1 \wedge E_2)$, $A \wedge B \longrightarrow f(D_1 \wedge F_2, E_1 \wedge G_2)$, $A \wedge B \longrightarrow f(F_1 \wedge D_2, G_1 \wedge E_2)$, $A \wedge B \longrightarrow f(F_1 \wedge F_2, G_1 \wedge G_2)$ to the production set (and we once again do similarly when the arity of above sets or the arity of f is $\neq 2$).

Convince yourself that this is indeed a well-formed definition as definitions for different representations of the same nonterminal in $\mathcal{N}^{\neg\wedge}$ are equal. Clearly the new nonterminal T produces all terms, i.e. HD_Σ. Never $\neg$ must be applied to "nonatomic" nonterminals, thus we never construct nonterminals outside $\mathcal{N}^{\neg\wedge}$. From its finiteness termination of the process is obvious. The following lemma can be shown by induction on term depth:

Lemma 8. *In $G^{\neg\wedge}$ every nonterminal produces exactly its corresponding language.*

Now we consider only those elements of $\mathcal{N}^{\neg\wedge}$ that are *complete*, i.e. those which contain **all** elements of $\mathcal{N} \cup \{T\}$ either negated or unnegated. We proceed by reformulating our grammar to operate over complete nonterminals only:

Definition 9. Assume we have started with a grammar G and have obtained $G^{\neg\wedge}$. Now we apply the rule

> "Replace a production $A \longrightarrow f(\ldots,X,\ldots)$, B not occurring in X, by productions $A \longrightarrow f(\ldots,X \wedge B,\ldots)$ and $A \longrightarrow f(\ldots,X \wedge \neg B,\ldots)$."

as often as possible and call the result $G_c^{\neg\wedge}$. From $G_c^{\neg\wedge}$ we obtain G_{unique} by deleting all noncomplete nonterminals and all productions containing one of them.

Once again by induction on term depth, Lemma 8 can be shown for $G_c^{\neg\wedge}$ too (and thus we can as well delete every occurrence of T and every production with an occurence of $\neg T$). Now observe that any derivation of a term in $G_c^{\neg\wedge}$ from a complete nonterminal never uses anything else than complete nonterminals, as we have just replaced all the incomplete ones in the right sides of the productions. Thus we can restrict the grammar to G_{unique} without hurting the applicability of Lemma 8 (so we can as well delete all productions containing $\ldots P \wedge \neg P \ldots$). Let's finally obtain G^0_{unique} by stripping all nonterminals and all productions from G_{unique} that do not produce any ground terms at all (note that this can be done in time linear with the number of nonterminals in G_{unique}, so we don't add significant complexity here). We conclude:

Corollary 10. *For both G_{unique} and G^0_{unique} it holds that:*

- *All generated languages are pairwise disjoint and their union is HD_Σ, i.e. all ground terms.*
- *Every ground term of HD_Σ can be derived from exactly one nonterminal symbol.*
- *If t_i are ground terms derivable from N_i $(1 \leq i \leq n)$, then there must be exactly one production $X \longrightarrow f(N_1,\ldots,N_n)$ for every function symbol f.*

Note that beside the model-theoretic properties we are interested in, our construction also proves language-theoretic results, e.g. the fact that the class of languages produced by term grammars is closed under union, intersection and inversion (w.r.t. HD_Σ).

Definition 11. Now let Γ_G be the Herbrand interpretation of a skolemized monadic formula defined via the grammar G. We define a finite interpretation Γ_G^{finite}: For the domain we take the nonterminal symbols of G^0_{unique}. Every ground term is interpreted as the unique nonterminal it is derivable from. Every ground atom $P(t)$ is interpreted as true iff t can be derived from a nonterminal in G^0_{unique} that contains P positively (i.e. not as $\neg P$).

Lemma 12. *Γ and Γ_G^{finite} are equivalent interpretations,*

because Γ_G^{finite} is a homomorphic image of Γ_G per definition. Thus we have succeeded to construct a finite model for every context free model of a monadic skolemized formula (but with arbitrary function symbols).
On the other hand any (finite) model must have a submodel that is equivalent to some Herbrand model (i.e. that is the picture of a homomorphic mapping of a Herbrand model). We can read this finite submodel as a grammar in the following way: The elements of the domain are the nonterminals of the grammar and if ground terms t_i are interpreted to the domain elements N_i $(1 \leq i \leq n)$ and $f(t_1, \ldots, t_n)$ is interpreted to N, then we put the production $N \longrightarrow f(N_1, \ldots, N_n)$ into the grammar. Clearly all ground terms must be derivable from exactly one nonterminal in this grammar. According to whether a predicate is true or false on a certain element of the finite domain, we can rewrite the nonterminals as complete nonterminals in $\mathcal{P}^{\neg\wedge}$. Thus we have constructed a grammar with the same properties as G^0_{unique}. We can conclude the next theorem:

Theorem 13. *For any context-free model of a skolemized monadic formula (with arbitrary function symbols) there is an equivalent finite model and every finite model contains a submodel that is equivalent to a context-free model.*

Corollary 14. *For any finite model of a skolemized formula (with arbitrary function symbols) the set of true ground atoms for a particular predicate is a context-free language.*

Corollary 15. *A (skolemized monadic) formula has a finite model iff it has a context-free one.*

Note that the translation between Herbrand models of monadic skolemized formulas and models of arbitrary formulas we already discussed can easily be taken over to (finite) homomorphic images. Thus the restriction to skolemized monadic formulas in Corollary 15 is inessential and we can say that we can indeed represent models of exactly those formulas having a finite model.
The fact that every context-free model is (truth-)equivalent to a finite one immediately settles the problem of **clause evaluation** as we can always evaluate over a finite domain. However it will still be interesting to investigate methods for speeding up clause evaluation. Among other possibilities this may be achieved by making methods of automated theorem proving benefit from the particular structure of a clause set that represents exactly one context-free model. In the next section we use this idea to solve the equivalence test for context-free models in an elegant way.

6 The Equivalence Problem

We solve the equivalence problem, i.e. the question whether two term grammars generate the same languages among some fixed predicate symbols $P_1, \ldots, P_n$, by translating each of the grammars to a clause set that has exactly the one Herbrand model defined via the grammar and checking the union of the two clause sets for consistency. We prove that an A-ordering resolution prover, which is known to be complete, must terminate on this union. Althoug equivalence of context-free languages in general is undecidable, an equivalence test for term grammars could be obtained by reading them as tree-automata (see e.g. [GS84]) for which corresponding results are achieved. However it is still interesting to see how techniques of automated theorem proving can efficiently be used for this purpose. The translation to clause-sets shows another interesting aspect of our approach: If we are able to represent our grammar (model) as a clause set itself, there is no need to switch to different representations and algorithms, instead we can stay in the world of clauses a reasonable theorem prover has to deal with anyway. For this section keep in mind that we are still working on a fixed signature.

Definition 16. Let G be a flat term grammar. Then we define the clause set

$$\begin{aligned}\mathcal{C}_G^+ := \{\ & P\ (f\,(x_1,\ldots,x_k)) \longleftarrow N_1(x_1),\ldots,N_k(x_k) \mid \\ & \quad P \longrightarrow f\,(N_1,\ldots,N_k) \text{ is a production in } G, x_1,\ldots,x_k \text{ are distinct variables }\} \bigcup \\ & \{\ P(a) \mid P \longrightarrow a \text{ is a production in } G,\ a \text{ is a constant symbol }\}\ .\end{aligned}$$

Example 3. Let's take the grammar $G_{ex} := (\mathcal{A}_\Sigma, \{S, Q\}, \Pi)$ with $\Pi_{ex} := \{\ S \longrightarrow g(S,Q),\ Q \longrightarrow f(S),\ S \longrightarrow a\ \}$. Then we obtain $\mathcal{C}_{G_{ex}}^+ = \{\{S(g(x,y)) \longleftarrow S(x), Q(y)\}, \{Q(f(x)) \longleftarrow S(x)\}, \{S(a)\}\}$.

By an induction argument on term depth it is not hard to prove that for any ground term t and any term grammar G it holds that $t \in L_P \Leftrightarrow \mathcal{C}_G^+ \models P(t)$.
Note that the specific syntactical structure of $\mathcal{C}_G^+$ lets us view it as a kind of "linear atom representation with side conditions". In fact it is easily verified that we can transform every linear atom representation[9] to a term grammar representing the same model, thus showing that term grammars are a natural extension of linear atom representations.
What is missing to construct a clause set that has exactly the one intended model is modelling the closed world assumption (*CWA*) for $\mathcal{C}_G^+$. Although modelling the CWA is impossible for general clause sets, the specific structure of $\mathcal{C}_G^+$ enables us to do so. To make this definition a bit simpler we first consider only *deterministic* flat term grammars, i.e. grammars in which for every nonterminal P and every function symbol f (and analogously for constants a) there is at most one production of the type $A \longrightarrow f(\ldots)$; or in other words $|\Pi[A, f]| \leq 1$ for all A and f. We will indicate how the construction can easily be taken over to arbitrary term grammars.

Definition 17. Let $G = (\mathcal{A}_\Sigma, \mathcal{P} \cup \mathcal{N}, \Pi)$ be a deterministic flat term grammar, then we define

$$\begin{aligned}\mathcal{C}_G^{-1} &:= \{\ \neg P(a) \longleftarrow\ \mid \Pi[P,a] = \emptyset\ \} \cup \{\ \neg P(f(x_1,\ldots,x_n) \longleftarrow\ \mid \Pi[P,f] = \emptyset\ \}\,, \\ \mathcal{C}_G^{-2} &:= \{\ \neg P(f(x_1,\ldots,x_i,\ldots,x_n) \longleftarrow \neg Q(x_i) \mid \\ &\qquad P \longrightarrow f(N_1,\ldots,N_{i-1},Q,N_{i+1},\ldots,N_n) \in \Pi\ \}\,, \\ \mathcal{C}_G^- &:= \mathcal{C}_G^{-1} \cup \mathcal{C}_G^{-2}\ .\end{aligned}$$

Example 4. For the grammar from Example 3 we obtain

$$\begin{aligned}\mathcal{C}_G^{-1} &:= \{\ \{\neg S(f(x))\}, \{\neg Q(a)\}, \{\neg Q(g(x,y))\}\ \} \\ \mathcal{C}_G^{-2} &:= \{\ \{\neg S(g(x,y)) \longleftarrow \neg S(x)\}, \{\neg S(g(x,y)) \longleftarrow \neg Q(y)\}, \{\neg Q(f(x)) \longleftarrow \neg S(x)\}\ \}\ .\end{aligned}$$

Clearly if t is a ground term and G is a deterministic flat grammar, then we have $t \notin L_P \Leftrightarrow \mathcal{C}_G^- \models \neg P(t)$. Note that the construction of $\mathcal{C}_G^-$ for a non-deterministic grammar G is only slightly more difficult. We keep $\mathcal{C}_G^{-1}$ and indicate the construction of $\mathcal{C}_G^{-2}$ by the following example:

Example 5. Assume in a term grammar $\Pi[S, g] = \{S \longrightarrow g(M,N), S \longrightarrow g(P,Q)\}$. To construct $\mathcal{C}_G^{-2}$ that makes above proposition applicable we must put $\{\neg S(g(x,y)) \longleftarrow \neg M(x), \neg P(x)\}$, $\{\neg S(g(x,y)) \longleftarrow \neg N(y), \neg P(x)\}$, $\{\neg S(g(x,y)) \longleftarrow \neg M(x), \neg Q(y)\}$, $\{\neg S(g(x,y)) \longleftarrow \neg N(y), \neg Q(y)\}$ into $\mathcal{C}_G^{-2}$ for the predicate (nonterminal) symbol S and the function symbol g.

By putting together $\mathcal{C}_G^+$ and $\mathcal{C}_G^-$ we obtain

Proposition 18. *For any flat term grammar G, $\mathcal{C}_G := \mathcal{C}_G^+ \cup \mathcal{C}_G^-$ has exactly one Herbrand model which is the one defined via the grammar G.*

For deciding the equivalence of two (flat) term grammars $G_1 = (\mathcal{A}_\Sigma, \mathcal{P} \cup \mathcal{N}_1, \Pi_1)$ and $G_2 = (\mathcal{A}_\Sigma, \mathcal{P} \cup \mathcal{N}_2, \Pi_2)$, with respect to the languages generated from symbols in $\mathcal{P}$ (we ensure that $\mathcal{P}$, $\mathcal{N}_1$ and $\mathcal{N}_2$ are pairwise disjoint by renaming nonterminals), we must just decide the consistency of $\mathcal{C}_{G_1} \cup \mathcal{C}_{G_2}$. From [Joy76, Fer90, FLTZ93], etc. we know the method of A-ordered resolution: I.e. in resolution calculus we do not allow resolvents that contain literals strictly bigger than the literal that was resolved upon (after applying the most general unifier) with respect to an A-ordering.

[9] See Section 2 and [FL93, FL96].

This method is known to be complete[10] for an arbitrary A-ordering, i.e. an irreflexive, transitive and substitution-preserved[11] relation on atoms. We further know that this resolution refinement is compatible with condensation; i.e. replacing every clause C by the smallest subset $A \subseteq C$ such that A is a factor of C, does not violate completeness[12]. For deciding the equivalence problem we only need the following subterm ordering, which can easily be verified to be an A-ordering:

$$P(r) <_{\text{subt}} Q(t) \Leftrightarrow \text{r is a proper subterm of t} \ .$$

Theorem 19. *An A-ordering resolution prover using $<_{\text{subt}}$ and condensation must terminate on $\mathcal{C}_{G_1} \cup \mathcal{C}_{G_2}$ for two (flat) term grammars G_1 and G_2. Thus the equivalence problem for term grammars is decidable.*

Proof. Let's have a close look on what happens when resolving clauses in $\mathcal{C}_{G_1} \cup \mathcal{C}_{G_2}$: Assume we resolve $P(f(\ldots x \ldots y \ldots)) \vee \ldots \neg P_1(x) \vee \neg P_2(y)$ with $\neg P(f(\ldots u \ldots v \ldots)) \vee \ldots P_3(u) \vee P_4(v)$ (the only admissible resolvent must resolve on the two deepest literals). Then the resolvent is: $\ldots \vee \neg P_1(x) \vee \neg P_2(y) \vee P_3(x) \vee P_4(y)$. And if we resolve this resolvent with one of the input clauses we get a clause containing just variable disjoint subparts of the type of the input clauses. Similarly it can be shown that the invariant

> "Any clause that can be derived from $\mathcal{C}_{G_1} \cup \mathcal{C}_{G_2}$ consists of variable-disjoint subparts, in each of which there occurs a leading term $f(x_1, \ldots, x_n)$ in one or more literals $(\neg)P(f(\ldots))$ and every other literal in the subpart is of the form $(\neg)Q(x_i)$ for some i - or - all the literals in the subpart are of the form $(\neg)Q(x)$ or $(\neg)Q(a)$."

holds for any obtained resolvent. As condensation prevents the occurrence of different variants of the same (leading) literal in one clause, we see that there are only finitely many condensed clauses that satisfy the invariant. Thus the number of potentially derivable resolvents is finite and thus our "$<_{\text{subt}}$ +condensation"-prover must terminate. □

7 Summary and Directions for Further Investigation

We have introduced and motivated the use of context-free grammars to represent Herbrand models and have shown some basic results. We also argued why we don't lose generality when restricting ourselves to Herbrand models of skolemized monadic formulas (with arbitrary function symbols), as those models can directly be taken over to models of general formulas. We can satisfy the four requirements from the beginning of this article: The **ground atom test** can be performed by a simple syntax check in the grammar. The **equivalence test** can be performed by translating the grammars to clause sets and using methods of automated theorem proving. The requirement of **understandability** is (to our opinion) well fulfilled for representing structures via context-free grammars. For **clause evaluation** we can effectively translate the grammar to an equivalent finite model and evaluate the clause there. It's still desirable to look for more efficient algorithms for clause evaluation. To approach this problem we suggest to either investigate optimizations of the transformation to finite models or to try to benefit from grammar-to-clauses-set translations like we used for the equivalence test, which is especially promising if we restrict the class of clauses we want to evaluate.

We also answered the **characterization question**: We could show that the properties of having a finite model and of having a context-free one coincide. This is a pity as we hoped to overcome finiteness, but on the other hand we obtain a strong characterization result also answering some questions connected with the **comparison question**.

We have already seen that term grammars are a strict extension of linear atom representation as introduced in [FL93, FL96]. With atom representations in general, they appear to be incomparable

[10] i.e. the empty clause can be derived whenever the initial clause set is unsatisfiable.

[11] i.e. $A <_A B \Rightarrow A\sigma <_A B\sigma$ for any substitution σ.

[12] This idea was first used in [Joy76] to control the size of resolution-derived clauses helping to prove the decidability of several classical recursive subclasses of first order logic via resolution refinements.

as the model represented by the atom representation $\{P(f(x,x))\}$ could not be achieved with term grammars; on the other hand the model generated by the grammar $\{P \longrightarrow f(f(P)), P \longrightarrow a\}$ has no atom representation. For a comparison with "ground atoms plus ground equations" (i.e. take ground atoms that can be derived from the atoms with the help of the equations for true, the others for false) there is strong evidence that they yield the same model class as term grammars. Things are likely not to change if we allow linear atoms instead. Many other interesting representations are still not investigated. It would be interesting to add equations to the grammars or to switch to other language classes like (certain) context sensitive languages, L-systems, etc. (see e.g. [Sal73, RS92]). Finally we think that on the base of our investigations it should be possible to construct **model building algorithms** for interesting classes of formulas. However we observed that thinking about models and formal language theory in parallel yields another interesting benefit: Whenever it suits we can arbitrarily switch between the language aspects and the logical (model theoretic) aspects and use the better suited representation along with well-known results of both fields.

References

[BCP94] C. Bourely, R. Caferra, and N. Peltier. A method for building models automatically. Experiments with an extension of otter. In Alan Bundy, editor, 12^{th} *International Conference on Automated Deduction*, pages 72–86, Nancy/France, June 1994. Springer Verlag, LNAI 814.

[BJ74] G. Boolos and R. Jeffrey. *Computability and Logic*. Cambridge University Press, 1974.

[CHK90] H. Chen, J. Hsiang, and H.-C. Kong. On finite representations of infinite sequences of terms. In S. Kaplan and M. Okada, editors, *Proceedings 2nd International Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada)*, volume 516 of *Lecture Notes in Computer Science*, pages 100–114. Springer, June 1990.

[CL73] C.-L. Chang and R. C. T. Lee. *Symbolic Logic an Mechanical Theorem Proving*. Academic Press, New York, 1973.

[Com95] H. Comon. On unification of terms with integer exponents. *Mathematical Systems Theory*, 28(1):67–88, 1995.

[CZ92] R. Caferra and N. Zabel. A method for simultanous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13(6):613–641, June 1992.

[Fer90] C. Fermüller. Deciding some Horn clause sets by resolution. In *Yearbook of the Kurt-Gödel-Society 1989*, pages 60–73, Vienna, 1990.

[FL93] C. Fermüller and A. Leitsch. Model building by resolution. In *Computer Science Logic (CSL'92)*, pages 134–148, San Miniato, Italy, 1993. Springer Verlag. LNCS 702.

[FL96] C. Fermüller and A. Leitsch. Hyperresolution and automated model building. *J. of Logic and Computation*, 6(2):173–203, 1996.

[FLTZ93] C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution Methods for the Decision Problem*. Springer Verlag, 1993. LNAI 679.

[GH96] R. Galbavý and M. Hermann. Unification of infinite sets of terms schematized by primal grammars. *Journal of Symbolic Computation*, 1996. To appear.

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[Joy76] W.H. Joyner. Resolution strategies as decision procedures. *Journal of ACM*, 23(1):398–417, 1976.

[Lov78] D. Loveland. *Automated Theorem Proving — A Logical Basis*. North Holland, 1978.

[McA92] D. McAllester. Grammar rewriting. In D. Kapur, editor, *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs, (New York, USA)*, volume 607 of *Lecture Notes in Computer Science (in Art. Inteligence)*. Springer, 1992.

[RS92] G. Rozenberg and A. Salomaa, editors. *Lindenmayer Systems*. Springer, 1992.

[Sal73] A. Salomaa. *Formal languages*. Academic Press, Orlando, 1973.

[Sal92] G. Salzer. The unification of infinite sets of terms and its applications. In A. Voronkov, editor, *Proceedings Logic Programming and Automated Reasoning, St. Petersburg (Russia)*, volume 624 of *Lecture Notes in Computer Science (in Art. Inteligence)*. Springer, 1992.

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

[Sla93] J. Slaney. SCOTT: A model-guided theorem prover. In *Proceedings of the 13th international joint conference on artificial intelligence (IJCAI '93)*, volume 1, pages 109–114. Morgan Kaufmann Publishers, 1993.

[Tam91] T. Tammet. Using resolution for deciding solvable classes and building finite models. In *Baltic Computer Science*, pages 33–64. Springer Verlag, 1991. LNCS 502.