Using Grammars for Finite Domain Evaluation

Robert Matzinger*
Technische Universität Wien

October 7, 1997

Abstract
In [8] we investigated representing Herbrand models via context-free grammars and found the representation power of this method to be exactly the finite models. Based on these observations we now present a clause set evaluation algorithm that operates directly on grammars, avoiding the exponential blow-up from the number of nonterminals in the grammar to the number of elements in the finite domain of the corresponding model, ending up with a not-so-obvious evaluation procedure for arbitrary clause sets over finite interpretations (specified via grammars).

1 Introduction

Currently there is considerable ongoing work in the field of automated model building (see e.g. [2–5, 10, 11]), a subfield of automated theorem proving that attempts to design algorithms for finding models of satisfiable first order logic formulas. Note that a model of a formula $\neg A$ is nothing else than a counterexample of A, which may be of enormous help for finding the reason why A failed to be valid. Clearly, the ability to represent models of first order logic formulas in a computationally feasible way is a necessary prerequisite for trying to build them. However, representation mechanisms for particular models of first order logic formulas play an important role in many other fields too, e.g. semantic resolution, model checking, etc. That's why we considered it worthwhile to investigate mechanisms for representing interpretations of first order logic formulas symbolically by their own, aiming to accompany work in the automated model building field.

From a theoretical point of view it makes sense to restrict ourselves to Herbrand models of skolemized formulas (i.e. clause sets), but this is also justified from a practical viewpoint to utilize the intuitive requirement of **understandability**, because in Herbrand models the domain and the interpretation of the function symbols are clear, fixed and intuitive. Still we know there exists a Herbrand model for any (skolemized) 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 (true) ground atoms, i.e. sets of terms (or strings, depending on how we want to look at them). This viewpoint reveals the (to our opinion) most interesting aspect of our approach: We are lead to investigating model properties in terms of syntactical properties of the corresponding true-ground-atom set.

In [7, 8] we investigated the use of context-free grammars to represent the true ground atom set of an interpretation[1] in the following sense: Consider a fixed signature Σ.

*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`, URL: `http://www.kr.tuwien.ac.at/~matzi/`

†This work was supported by the Austrian Science Foundation under FWF grant P11624–MAT.

[1]For didactical reasons we consider formulas with only monadic predicate symbols (but function symbols

HD_Σ denotes the set of terms over this signature (i.e. the Herbrand domain). Let $G = (\mathcal{A}_\Sigma, \mathcal{P} \cup \mathcal{N}, \Pi)$ be a grammar where $\mathcal{A}_\Sigma$ is the set of constant symbols plus the function symbols plus the symbols $'('$, $')'$, and $','$. $\mathcal{P}$ is the set of predicate symbols in the signature (used as nonterminals in the grammar), $\mathcal{N}$ is an appropriate set of additional nonterminal symbols and Π is a set of productions. We omit the notation of a starting symbol with the grammar, writing it as an index of the language instead, e.g. L_P denotes the language derived by G with starting symbol P. We may restrict ourselves to *flat term grammars* i.e. grammars in which 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 we do not lose generality, because any context-free[2] subset of HD_Σ can be generated also by a flat term grammar, a fact we proved in [9].

Now we can simply define the Herbrand interpretation Γ_G by stating that a ground atom $P(t)$ shall be interpreted true iff $t \in L_P$. We call any Herbrand interpretation a context-free interpretation (or context-free model) iff we can define it via a grammar in the above sense. Clearly flat term grammars are just another notation for indeterministic bottom-up tree automata (see [6]), which (beside other desirable properties) can always be translated to deterministic ones. This and the fact that regular tree languages are closed under inversion, intersection and complement enables to show that

Theorem. *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. *For any finite model of a skolemized formula (with arbitrary function symbols) the set of true ground atoms is a context-free language. A formula has a finite model iff it has a context-free one.*

So we can cover exactly the finite models with this representation mechanism, that well satisfies the requirements for model representations as raised in [4]: The **atom test** (i.e. evaluating $P(t)$ for a ground term t) is nothing else than a syntax check for t with respect to the term grammar, which can be solved in linear time (w.r.t. the length of the atom). The **equivalence check** (i.e. whether or not two grammars represent the same model) can be shown to be decidable with some knowledge in tree automata theory, but can also be solved e.g. by a translation to clause sets and resolution refinements (see [8]). Note also that in many practical situations specifying a model by a recursive definition of properties (under the closed world assumption), which is nothing else than a grammar's rule set, may be preferable among the explicit notation of a much bigger finite model. Still there's a very natural relation between the (indeterministic) grammar and the finite model. Although the equivalence with a finite model does yield a **clause set evaluation** procedure by enabling us to construct the equivalent finite model and evaluate all possible instances of every clause C in a clause set $\mathcal{C}$, this procedure is extremely inefficient due to the exponential blow-up from grammar nonterminals to finite domain members.[3] That's

of arbitrary arity). Note that this is no loss of generality, as we always get a sat-equivalent formula via the translation $P(t_1, \ldots, t_n)$ to $T(p(t_1, \ldots, t_n))$ (p being a new n-place function symbol for every predicate P and T being an entirely new predicate). Additionally any model of the $T(p(\ldots))$-formula can be trivially translated to one of the original formula.

[2]i.e. generated by an arbitrary context-free (string) grammar. Example: The language $\{f(a)\}$ generated by $S \longrightarrow AB, A \longrightarrow f(, B \longrightarrow a)$ can also be generated by $S \longrightarrow f(Q), Q \longrightarrow a$.

[3]Recall that in the usual construction there are exponentially many nonterminals in the deterministic grammar and there may be exponentially many finite domain members than there were nonterminals in the original grammar.

why developing more efficient procedures remained an interesting target.

2 Clause Evaluation On Context-Free Interpretations

To avoid the explicit construction of the finite model we asked ourselves whether we could evaluate clause sets directly on the grammar. Difficulties arise from the fact that the grammar resembles an indeterministic tree automaton, for which we want to avoid explicit determination. However we can utilize the connections between syntactical properties of the interpretation and the truth of formulas to design the following evaluation procedure:

Let $G = (\mathcal{A}_\Sigma, \mathcal{P} \cup \mathcal{N}, \Pi)$ be a fixed flat term grammar. $\Pi[P, f]$ is defined to be the set of productions $\subseteq \Pi$ that have the form $P \longrightarrow f(\ldots)$, where $P \in \mathcal{P} \cup \mathcal{N}$ is a nonterminal and f is a function symbol. We analogously define $\Pi[P, a]$ for constant symbols a.

Now let's start with a clause set $\mathcal{C}$ for which we want to know whether it evaluates to true in the model specified by the grammar G (in the sense described before) or not, i.e. we want to know whether $\Gamma_G \models \mathcal{C}$. Now observe that if $\Pi[P, f] = \{P \longrightarrow f(Q_1^1, Q_2^1, \ldots Q_n^1), \ldots, P \longrightarrow f(Q_1^k, Q_2^k, \ldots Q_n^k)\}$ this means that $\Gamma_G \models P(f(x_1, x_2, \ldots x_n)) \leftrightarrow ((Q_1^1(x_1) \wedge \ldots \wedge Q_n^1(x_n)) \vee \ldots \vee (Q_1^k(x_1) \wedge \ldots \wedge Q_n^k(x_n)))$. We can denote this formula negatively too: $\Gamma_G \models \neg P(f(x_1, x_2, \ldots x_n)) \leftrightarrow ((\neg Q_1^1(x_1) \vee \ldots \vee \neg Q_n^1(x_n)) \wedge \ldots \wedge (\neg Q_1^k(x_1) \vee \ldots \vee \neg Q_n^k(x_n)))$. Now take these formulas as rewrite rules (to be used from the left to the right) that let us reduce the literals in a clause set $\mathcal{C}$. (Clearly we have to use the law of distributivity after every application of a rewrite rule to get a clause set again). We write $\mathcal{C} \mapsto \mathcal{C}'$, if $\mathcal{C}'$ is the clause form of the result of applying one of above mentioned rewrite rules to $\mathcal{C}$. Clearly $\Gamma_G \models \mathcal{C} \Leftrightarrow \Gamma_G \models \mathcal{C}'$. As we reduce the term-depth of the literals in each step, we must reach an irreducible form when repeatedly applying the rules. Since we have only one applicable rule for every literal this irreducible form is unique.[4] Let's call it $\mathcal{C}^0$. Because every literal in $\mathcal{C}^0$ is irreducible, it is either of the form $(\neg)P(x)$ where x is a variable or $(\neg)P(a)$ where a is a constant symbol. Clearly we can decide whether $\Gamma_G \models \mathcal{C}^0$, if we can decide whether $\Gamma_G \models C$ for a single clause $C \in \mathcal{C}^0$. Let's write $C[x]$ for the set of all literals on the variable x in C. $C[\text{const}]$ shall denote the set of ground literals in C. Thus $C = C[x_1] \vee C[x_2] \vee \ldots \vee C[x_m] \vee C[\text{const}]$. Clearly $\Gamma_G \models C$ iff Γ_G entails at least one of the $C[x_i]$ or Γ_G entails at least one of the literals in $C[\text{const}]$. Now recall the definition of Γ_G to see that a ground literal $P(a)$ is true in Γ_G iff $a \in L_P$, which can easily be checked in the grammar. Thus it remains to evaluate whether $\Gamma_G \models C[x]$. Observe that $C[x]$ corresponds to the formula $\forall x C[x]$. Thus $C[x]$ evaluates to true iff every ground term makes at least one literal true in $C[x]$. In general some literals are positive and some are negative in $C[x]$, i.e. $C[x] = \neg P_1(x) \vee \ldots \vee \neg P_l(x) \vee P_{l+1}(x) \vee \ldots \vee P_m(x)$. According to the definition of the interpretation Γ_G every ground term makes at least one of the $(\neg)P_i(x)$ true iff $(\mathrm{HD}_\Sigma \setminus L_{P_1}) \cup \ldots \cup (\mathrm{HD}_\Sigma \setminus L_{P_l}) \cup L_{P_{l+1}} \cup \ldots \cup L_{P_m} = \mathrm{HD}_\Sigma$, which is the case iff $L_{P_1} \cap \ldots \cap L_{P_l} \subseteq L_{P_{l+1}} \cup \ldots \cup L_{P_m}$. If there are either no negative or no positive literals the situation is pretty much the same: the corresponding conditions are $\mathrm{HD}_\Sigma \subseteq L_{P_1} \cup \ldots \cup L_{P_m}$ or $L_{P_1} \cap \ldots \cap L_{P_m} \subseteq \emptyset$, respectively.

Example. Let the Herbrand interpretation Γ_G be given by the following grammar: $G = \{\{f, a, '(' , ')' , ',' \}, \{P, Q\}, \{P \longrightarrow f(Q), Q \longrightarrow f(P), P \longrightarrow a\}\}$, i.e. the interpretation where $\{P(f^{2i}(a)), Q(f^{2i+1}(a)) | i \geq 0\}$ is true, the rest is false. Assume we want to evaluate $C = P(x) \vee P(f(x))$ in this interpretation. The grammar rule $P \longrightarrow f(Q)$ corresponds to the fact that $P(f(x)) \leftrightarrow Q(x)$, which we use as a rewrite rule (from left to right)

[4] Modulo the AC-property of the connectives and modulo renaming of the variables.

on $P(f(x))$ to obtain $C^0 = P(x) \vee Q(x)$. Thus C evaluates to true iff $L_P \cup L_Q = \mathrm{HD}_\Sigma$. We enhance the grammar by the rules $T \longrightarrow a, T \longrightarrow f(T)$ (i.e. $L_T = \mathrm{HD}_\Sigma$) and check for $L_T \subseteq L_P \cup L_Q$. On the other hand if we want to evaluate the clause $C' = P(x) \vee P(f^2(x))$, we end up with $C'^0 = P(x) \vee P(x)$, so we have to check whether or not $L_T \subseteq L_P$.

Thus we are done, if we have procedures that decide (efficiently) whether or not $L_{P_1} \cap \ldots \cap L_{P_n} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_m}$ for regular tree languages L_{P_i}, L_{Q_j}. It is well known that this problem is decidable, but we are interested particularly in a solution that avoids the exponential blow-up of the state-space that comes from determination of the corresponding tree automata. In the next section we give an algorithm, that works directly on the indeterministic automata (grammar).

3 Calculating with Regular Tree Languages

So we need a procedure to decide for a given flat term grammar G whether or not $L_{P_1} \cap \ldots \cap L_{P_n} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_m}$. The idea of the procedure is to algorithmically construct a "proof by induction on term depth" for the truth or falsehood of the questioned property. We use the following rules to unfold expressions of the form $L_P \cap \ldots \subseteq L_Q \cup \ldots$:

(i): $L_{P_1} \cap \ldots \cap L_{P_n} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_m} \Leftrightarrow \bigwedge_f L_{P_1}[f] \cap \ldots \cap L_{P_n}[f] \subseteq L_{Q_1}[f] \cup \ldots \cup L_{Q_m}[f]$, where $L[f]$ denotes the restriction of the language L to f-terms and $\bigwedge_f$ ranges over all function symbols and constant symbols,

(ii): If $\Pi[P, f] = \{P \longrightarrow f(P_1, \ldots), \ldots, P \longrightarrow f(P_n, \ldots)\}$, then $L_P[f] = f(L_{P_1}, \ldots) \cup \ldots \cup f(L_{P_n}, \ldots)$,[5]

(iii): $L_A \cup L_B \subseteq L_C \iff L_A \subseteq L_C \wedge L_B \subseteq L_C$ and

(iv): $f(L_A, \ldots) \cap f(L_B, \ldots) = f(L_A \cap L_B, \ldots)$,

Clearly these rules are correct. See that they enable us to unfold $L_{P_1} \cap \ldots \cap L_{P_n} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_m}$ to a boolean formula consisting only of subformulas of the form $f(L_{A_1} \cap \ldots \cap L_{A_k}, \ldots, L_{Z_1} \cap \ldots \cap L_{Z_k}) \subseteq f(L_{A'_1}, \ldots, L_{Z'_1}) \cup \ldots \cup f(L_{A'_l}, \ldots, L_{Z'_l})$, connected by $\wedge$ and $\vee$.

We start our procedure with an empty set H of induction hypothesis on an expression $L_{P_1} \cap \ldots \cap L_{P_n} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_m}$. First we unfold it according to above rule set. We get a boolean expression containing subformulas of the type $f(\ldots) \subseteq f(\ldots) \cup \ldots \cup f(\ldots)$. In the simplest case the subformula is just $f(L_A, \ldots, L_Z) \subseteq f(L_{A'}, \ldots, L_{Z'})$. We solve this by recursively applying our method on each $L_A \subseteq L_{A'}, \ldots, L_Z \subseteq L_{Z'}$, with the enhanced induction hypothesis $H \cup \{L_{P_1} \cap \ldots \cap L_{P_n} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_m}\}$ (because we stepped one step downward in the term depth; this corresponds to the induction step). Any more complicated case of $f(\ldots L_{A_i} \ldots) \subseteq f(\ldots L_{B_j} \ldots) \cup \ldots \cup f(\ldots L_{C_k} \ldots)$ is similarly solved by reducing it to a boolean expression over language inclusion properties of the $L_{A_i}, L_{B_j}, L_{C_k}$. See appendix Appendix A for technical details. We can once again use our idea of "stepping downward" and reapply our procedure on the subproblems with an enhanced induction hypothesis set. We can stop recursing and return true or false if we come down to constant symbols (this check is trivial; it corresponds to the induction base), or we can return true if the inclusion we have to test is a consequence of the induction hypothesis, i.e. if we have to test whether $L_{P_1} \cap \ldots \cap L_{P_n} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_m}$ and there's already a formula $L_{P_1} \cap \ldots \cap L_{P_i} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_j}$ in the induction hypothesis set H for some $i \leq n, j \leq m$.

[5]With $f(L_A, L_B, \ldots)$ we denote the language that consists of all f terms that have a member of L_A on the first place, a member of L_B on the second place and so forth.

So regardless whether $L_{P_1} \cap \ldots \cap L_{P_n} \subseteq L_{Q_1} \cup \ldots \cup L_{Q_m}$ is true or false, the procedure's run can be seen as an inductive proof that this is the case, from which we can conclude the correctness of the procedure. However, as our grammar has only finitely many nonterminal symbols, there are only finitely many possible induction hypothesis. As we always enlarge the set of possible induction hypothesis, we can guarantee termination of the procedure. Note that evaluations in the different recursion branches are totally independent, giving way for various search algorithms and heuristics as well as for parallelization. Note furthermore that the grammar we evaluate upon remains fixed, so every proposition about it, once proven, remains true. It's an obvious and substantial improvement to maintain a database of $L_P \cap \ldots \subseteq L_Q \cup \ldots$-propositions that are already proven and to cut off recursive calls that would otherwise just re-prove known facts. In many practical examples the algorithm terminates quickly on the constant symbol check. However a worst-case exponential blow-up seems to be unavoidable due to the inherent difficulty of the problem – note that the language inclusion problem "is $\Sigma^* \subseteq L_R$?" for regular expressions R (which resemble grammars with only unary function symbols) is PSPACE-complete (see e.g. [1]) and the language intersection nonemptiness problem for regular tree languages (i.e. whether $L_{P_1} \cap \ldots \cap L_{P_n} \subseteq \emptyset$) was shown to be EXPTIME-complete (e.g. in [12]).
Example. (continued) We denote our procedure as $L_A \overset{?}{\subseteq} L_B | H$, if we check whether or not $L_A \subseteq L_B$ under the induction hypothesis H. To check whether or not $L_T \subseteq L_P \cup L_Q$ we call $L_T \overset{?}{\subseteq} L_P \cup L_Q | \emptyset \;=\; f(L_T) \overset{?}{\subseteq} f(L_Q) \cup f(L_P) | \emptyset \;\wedge\; \{a\} \overset{?}{\subseteq} \{a\} | \emptyset \;=$
$= \; L_T \overset{?}{\subseteq} L_Q \cup L_P | \{L_T \subseteq L_P \cup L_Q\} \;\wedge\; \text{true} \;=\; \text{true} \wedge \text{true} \;=\; \text{true}.$
On the other hand if we want to check $L_T \subseteq L_P$, we get
$L_T \overset{?}{\subseteq} L_P | \emptyset \;=\; f(L_T) \overset{?}{\subseteq} f(L_Q) | \emptyset \;\wedge\; \{a\} \overset{?}{\subseteq} \{a\} | \emptyset \;=\; L_T \overset{?}{\subseteq} L_Q | \{L_T \subseteq L_P\} \;\wedge\; \text{true} \;=$
$= \; f(L_T) \overset{?}{\subseteq} f(L_P) | \{L_T \subseteq L_P\} \;\wedge\; \{a\} \overset{?}{\subseteq} \emptyset | \{L_T \subseteq L_P\} \;\wedge\; \text{true} \;=$
$= \; L_T \overset{?}{\subseteq} L_P | \{L_T \subseteq L_P, L_T \subseteq L_Q\} \;\wedge\; \text{false} \;\wedge\; \text{true} \;=\; \text{true} \wedge \text{false} \wedge \text{true} \;=\; \text{false}.$

With this algorithm we conclude our evaluation procedure for clause sets on a given context-free interpretation. Knowing that every context-free interpretation is equivalent to a finite one and vice-versa, we got an interesting and not-so-obvious method for evaluating arbitrary clause sets on finite interpretations. Furthermore we think that our algorithm will be of benefit for a complexity analysis of the problem of interpreting clauses over context-free interpretations because of its close relation to regular tree languages for which many complexity results are known.

Appendix A

We illustrate that any expression of the form $f(\ldots L_{A_i} \ldots) \subseteq f(\ldots L_{B_j} \ldots) \cup \ldots \cup f(\ldots L_{C_k} \ldots)$ can be reduced to a boolean expression over language inclusion properties of the $L_{A_i}, L_{B_j}, L_{C_k}$ by solving the following example: We show that $f(L_A, L_{A'}) \subseteq f(L_C, L_{C'}) \cup f(L_D, L_{D'})$ iff $(L_A \subseteq L_C \cup L_D \vee L_{A'} = \emptyset) \wedge (L_A \subseteq L_D \vee L_{A'} \subseteq L_{C'}) \wedge (L_A \subseteq L_C \vee L_{A'} \subseteq L_{D'}) \wedge (L_A = \emptyset \vee L_{A'} \subseteq L_{C'} \cup L_{D'})$.
Proof. $f(L_A, L_{A'}) \subseteq f(L_C, L_{C'}) \cup f(L_D, L_{D'})$ holds iff for all ground terms t, t' it holds that $t \in L_A \wedge t' \in L_{A'} \to (t \in L_C \wedge t' \in L_{C'}) \vee (t \in L_D \wedge t' \in L_{D'})$.
For notational convenience let us abbreviate "for all ground terms t it holds that" by $\forall t$ and let's write X for $t \in L_X$ and X' for $t' \in L_{X'}$. Thus the previous proposition reads $\forall t \forall t' A \wedge A' \to (C \wedge C') \vee (D \wedge D')$, which is the case iff $\forall t \forall t' \neg A \vee \neg A' \vee (C \wedge C') \vee (D \wedge D')$, which we can transform to the equivalent expression
$\forall t \forall t' (\neg A \vee \neg A' \vee C \vee D) \wedge (\neg A \vee \neg A' \vee C \vee D') \wedge (\neg A \vee \neg A' \vee C' \vee D) \wedge (\neg A \vee \neg A' \vee C' \vee D').$

We get an equivalent expression when we shift the quantifiers over the $\wedge$ and $\vee$, so we obtain $((\forall t' \neg A') \vee (\forall t \neg A \vee C \vee D)) \wedge ((\forall t \neg A \vee C) \vee (\forall t' \neg A' \vee D')) \wedge ((\forall t \neg A \vee D) \vee (\forall t' \neg A' \vee C')) \wedge ((\forall t \neg A) \vee (\forall t' \neg A' \vee C' \vee D'))$,
which expresses nothing else than $(L_{A'} = \emptyset \vee L_A \subseteq L_C \cup L_D) \wedge (L_A \subseteq L_C \vee L_{A'} \subseteq L_{D'}) \wedge (L_A \subseteq L_D \vee L_{A'} \subseteq L_{C'}) \wedge (L_A = \emptyset \vee L_{A'} \subseteq L_{C'} \cup L_{D'})$. $\Box$

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, Reading, Ma., 1974.

[2] R. Caferra and N. Peltier. Decision procedures using model building techniques. In *Computer Science Logic (9th Int. Workshop CSL'95)*, pages 131–144, Paderborn, Germany, 1995. Springer Verlag. LNCS 1092.

[3] 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.

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

[5] C. Fermüller and A. Leitsch. Decision procedures and model building in equational clause logic. *Journal of the IGPL*, 1997. to appear.

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

[7] R. Matzinger. Comparing computational representations of Herbrand models. In A. Leitsch, editor, *Computational Logic and Proof Theory, 5th Kurt Gödel Colloquium, KGC'97*, volume 1289 of *LNCS*, pages 203–218, Vienna, 1997. Springer.

[8] R. Matzinger. Computational representations of Herbrand models using grammars. In D.v. Dalen, editor, *Computer Science Logic, 10th International Workshop, CSL'96*, volume 1258 of *LNCS*, pages 334–348, Utrecht, 1997. Springer.

[9] R. Matzinger. Context free term sets are regular - and some applications to logic. Technical Report TR-WB-Mat-97-2, TU Wien, Vienna/Austria, 1997.

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

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

[12] M. Veanes. *On Simultaneous Rigid E-Unification.* PhD thesis, Computing Science Department, Uppsala University, 1997.