Chapter 7

A RESOLUTION BASED METHOD FOR BUILDING FINITE MODELS

In the following we will present an algorithm for building finite clause sets for formulas in the class we call AM, which is the union of the Initially-extended Ackermann and Essentially Monadic Classes. The initially extended Ackermann's Class is the class of formulas with a prefix $\exists^* \forall \exists^*$; the essentially Monadic Class is a prefix class of formulas with such atoms which after Skolemization will contain no more than one nonconstant term in the set of arguments of the atom. For example, the atom (in Skolemized form) P(a, f(x,y), b, f(x,y)) will be considered to be (essentially) monadic.

The method presented here is a successor to a previous algorithm for the same class developed by the author in [Tam 91]. The basis of a new method is completely different from the one in [Tam 91], compared to which the new method is significantly faster for realization, and the correctness/completeness proofs are much simpler.

Our method is practically in no way related to the methods of building finite models described in [Wos 84]; the last methods rely totally on the human user; the theorem-prover (which does not use decision strategies) offers only a helpful housekeeping environment to keep track of the investigated search space.

The proof of the termination of our algorithm uses only the theorems about the completeness of the apriori $>_k$-ordering-refinement of resolution method presented in chapter 6 and the theorem about the completeness of narrowing given in [Sla74], thus being also the proof of the finite controllability of AM.

We have not set ourselve the task to improve the known upper bounds on the size of models for formulas in the Ackermann and Monadic class (see [DG79]). Neither can we show that our method has a better upper bound concerning the computational complexity of building a model than the one given in [Lew 80].

Nevertheless, our algorithm is capable of computing for "small formulas" considerably smaller models than the classical methods given, for example, in [DG79] (such optimisations obviously were not the aim of the authors of [DG79]); it uses more of the structure of the investigated formula, being more flexible, so to say. Since it relies on a certain resolution refinement ($>_k$-refinement), we have the opinion that it shares some of the good properties of resolution. For example, our method performs the satisfiability checking incrementally (satisfiability checking must be performed many times for generating even the smallest models) using a very efficient resolution refinement.

It would be interesting to compare the implementation of our method with the implementation of classical methods; unfortunately no implementations of the classical methods are known to us.

Last not least, we hope that the general outline of the soundness and termination proof of the algorithm might be useful for showing finite controllability and devising analogous algorithms for other solvable classes for which we have the deciding resolution refinements (Classes K and E, for example) and giving some additional insight to related problems.

7.1 THE CLASS AM

The class AM is essentially the class of clause sets corresponding to the Initially extended Ackermann Class and the Essentially Monadic Class, with arbitrary constant substitutions allowed.

DEFINITION 7.1.: Let F be a prenex form of a formula of the first order predicate calculus (without functional symbols and without the equality predicate). Let S_F be the clause form of F. Let $A = P(t_1,...,t_n)$ be an atom in S_F. Remember that by args(A) we denote the set $\{t_1,...,t_n\}$.
Then some clause set S belongs to the Class AM iff there is such a clause set S_1 which is the clause form of some formula in prenex form and for clause C in S can be split into clauses $C_1, \ldots, C_n$ so that for each C_i ($1 \leq i \leq n$) one of the following cases is true:

- (i) C belongs to S_F and for each atom atom A in C at least one of the following two possibilities holds:
  - Args(A) contains at most one non-ground term.
  - Every functional term in Args(A) is of arity ≤ 1 and A contains only one variable (this variable may have several occurences in A).
- (ii) $C = D\sigma$, where D is a clause obeying case (i) above, and $\sigma = \{c_1/x_1, \ldots, c_n/x_n\}$ for some set of constants $c_1, \ldots, c_n$ and some set of variables $x_1, \ldots, x_n$.

Obviously the clause form of any formula from either the Initially-Extended Ackermann Class or the Essentially Monadic Class belongs to Class AM. More so, clause forms of "mixes" of these classes also belong to AM. Also notice that AM corresponds to a certain subset of Maslov's Class K.

EXAMPLE 7.1.: The following clause set belongs to the Class AM:

- {P(f(x), a, g(x)), ¬S(h(x, y, z), a, h(x, y, z)},
- {R(f(a), a, b), S(h(a, y, z), b, h(a, y, z))},
- {S(f(x), x, x), ¬P(k(x, y), k(x, y), b), P(f(x), g(x), g(x))},
- {S(f(x), x, x), P(f(y), y, f(y))}
- {R(f(a), a, g(a))},
- {P(x, x, x)},
- {P(a, b, c)}.

The following clauses and atoms cannot occur in any clause set belonging to AM, as they cannot be obtained by converting a function-free prenex formula to clause form and then by splitting:

- {P(f(x), x), S(g(y, z))},
- P(f(x), g(y)),
- P(f(x), f(a)).

DEFINITION 7.2.: In the following we will say that an atom A in the clause set of Class AM is of the Ackermann type if and only if A has at least two different nonconstant arguments; if A is not of the Ackermann type, we will say it is of the Monadic type. Notice that our usage of the names "Ackermann" and "Monadic" does not exactly correspond to the "normal" usage of these names (in our usage Monadic-type atoms can't be also of the Ackermann type). For literals the same classification is defined analogously.

EXAMPLE 7.2.: The following literals in the previous example are of the Ackermann type:

P(f(x), a, g(x)), S(f(x), x, x), P(f(x), g(x), g(x)), R(f(a), a, g(a)).

The following literals in the previous example are of the Monadic type:

$\neg$S(h(x,y,z), a, h(x,y,z)), S(h(a,y,z), b, h(a,y,z)), $\neg$P(k(x,y), k(x,y), a), R(f(a), a, b), P(x, x, x), P(a, b, c).

Notice that ground instances of the Ackermann-type literals can be Monadic; that happens iff the Ackermann-type literal contains only one function symbol.

7.2 THE ALGORITHM FOR BUILDING FINITE MODELS FOR FORMULAS IN AM

Recall the $>_k$ -ordering defined in chapter 6 for deciding Maslov's Class K. Recall that by Robinson's resolution method we denote the resolution method where factoring is performed at the time of a binary resolution step only and one of the factorized literals must be the one which is resolved upon.

Throughout the current chapter we will use the refinement of resolution method presented by N. Zamov in chapter 6. We will call it k-refinement:

DEFINITION 7.3.: By k-refinement we will denote the apriori refinement of Robinson's resolution method using the following ordering $>_k$: $A >_k B$ iff A strictly dominates B (for the definition of the dominating relation see chapter 6). Let S be any set of clauses. By $R^*_{>_k}(S)$ we denote the set of clauses derivable from S by the k-refinement.

DEFINITION 7.4.: We will say that some set of clauses G is in the stable form iff $G = R^*_{>_k}(G)$.

Let $S = \{C_1, \ldots, C_n\}$ be a satisfiable clause set from Class AM.
We will use the Herbrand universe $H_S = \{h_1, h_2, \ldots\}$ of S (compare Def. 2.28) as a domain of the model M for S (Herbrand's theorem implies that if a first order predicate calculus formula has a model, it has a Herbrand model - i.e.

a model with the domain of elements of Herbrand universe). For nontrivial formulas the Herbrand universe is infinite, and in order to build a finite model, we must build a finite domain and find an interpretation of the functions of the formula on this domain. Once the finite domain and interpretation of function symbols is found, finding the interpretation of predicate symbols is a relatively easy task, and we won't deal with it in the current presentation.

Whenever we speak of "a model", it is assumed that interpretations of predicate symbols may be left undetermined in it - it must only determine a finite domain and an interpretation of function symbols on that domain. Whenever we speak of a process of "model building", it is assumed that finding intepretations of predicate symbols may be left out of the process. When we speak of "the size of a model", we will mean the cardinality of the domain.

We will select a set of equations $E = \{e_1, \ldots, e_m\}$ (where each e_i is of the form $h_k = h_l$ for some k, l), that would decompose the Herbrand universe H into a finite number of different equivalence classes $\{c_1, \ldots, c_r\}$ such that the "formula" $SE = \{C_1, \ldots, C_n, e_1, \ldots, e_m\}$ would be satisfiable. In this case SE would have a finite model M with the domain D constructed by choosing one arbitrary element from each c_i and the Skolem functions interpreted by the set E. Obviously M would be a model for S.

In case we knew that any satisfiable formula in AM has a finite model, we could in principle use the following method for finding a finite model: for each possible cardinality of the model (1, 2, ...) check all possible classes of isomorphism of sets of equations, giving the domain of that size from the Herbrand universe. As we assumed that a satisfiable formula in AM always has a finite model, we will eventually find it. For example, suppose we have a clause set in AM with two one-place function symbols, f and g, and one constant symbol a. For cardinality 1 we must check the single set of equations: f(a) = a, g(a) = a. If this does not satisfy our clause set, we have to check the following eight sets for the cardinality two:

{{f(a) = a, f(g(a)) = a, g(g(a)) = a},
{f(a) = a, f(g(a)) = a, g(g(a)) = g(a)},
{f(a) = a, f(g(a)) = g(a), g(g(a)) = a},

{f(a) = a, f(g(a)) = g(a), g(g(a)) = g(a)},
{g(a) = a, g(f(a)) = a, f(f(a)) = a},
{g(a) = a, g(f(a)) = a, f(f(a)) = f(a)},
{g(a) = a, g(f(a)) = f(a), f(f(a)) = a},
{g(a) = a, g(f(a)) = f(a), f(f(a)) = f(a)}}.

Checking any such set of equations can be performed by substituting a finite set of domain elements given by the equations to all variables in the clauses in all possible ways, and computing the values of functions given by the equations. This gives a large set of propositional clauses, the satisfiability of which can be decided in standard ways (using resolution, for example).

The amount of possible sets of equations grows superexponentially on the cardinality of the model checked, and the exhaustive search method is practically applicable only for searching for very small models. Using exhaustive search (total backtracing for checking all possible interpretations for function symbols) for searching a finite model would prevent finding any but the smallest models: for example, a single functional symbol with arity 2 on a ten-element domain yields 10^{100} different interpretations (as there are ten possible values of this function for each of the 100 argument pairs).

The search algorithm we are going to present does not rely on such kind of an exhaustive search. Instead it will have the property that whenever any equation t = d is considered to be acceptable (that is, the set {S ∪ E ∪ t = d} is satisfiable, where S is our original clause set, E is the set of already accepted equations, and some additional special conditions for t = d are also fulfilled) this equation will be retained throughout the whole search. That is, the set of accepted equations grows monotonously and we never throw away any accepted equation (we do not "backtrace"). Thus at no moment during the model construction process (before the very last, the actual finding of a finite model) do we have a finite domain at hand, and for satisfiability checking we cannot use the conversion to a big propositional formula, as can be done for the exhaustive search described earlier. Therefore we will use the k-refinement of resolution (used in [Zam 89, 89a] and chapter 6) combined with Slagle's narrowing method from [Sla 74], which will yield a decision procedure for checking acceptability of new equations. We will prove that such a non-exhaustive

algorithm always terminates - when certain conditions of accepting an equation are imposed. This also gives a proof that any satisfiable formula in AM indeed has a finite model.

Let H be the Herbrand universe of a satisfiable S in AM. We will use the same main idea for building the finite model for S as is used for building finite models in [DG79], for example: we will traverse the Herbrand tree (compare def. 2. 31) moving from level 1 to deeper levels and trying to add equations between terms on the current level and some earlier level. Each equation cuts short some branches, until the branches in H are pruned.

The minimal consistent criteria for accepting some equation t = d between Herbrand terms is the satisfiability of a union {S ∪ E ∪ t = d} where E is the set of equations found so far. Such a criteria gurantees that S will have a model with the constructed interpretation of function symbols (but it generally does not guarantee finding a finite model, since we do not backtrack to try all possible combinations of equations). We will give a simple example of the process.

EXAMPLE 7.3.: Clause set S = {{P(x, f(x))}, {¬P(f(y), y)}} has a Herbrand tree with a single branch: a → f(a) → f(f(a)) → f(f(f(a))) → f(f(f(f(a)))), ... At first try to add the equation f(a) = a. But {S ∪ f(a) = a} is unsatisfiable, therefore this equation cannot be added. At the next step try to add f(f(a)) = a, but again {S ∪ f(f(a)) = a} is unsatisfiable. Then trying f(f(a)) = f(a) one will fail again. The first equation which can be added consistently is f(f(f(a))) = a, which prunes our single branch, thus giving a finite domain with four elements.

In general we have more than one branch and therefore need more than one equation.

One of the most important properties of our model- building algorithm is the absence of backtracing. Unfortunately, as we do not use backtracing, for testing the acceptability of an equation it is not possible to rely just on testing the satisfiability of the union of our original clause set, already obtained equations and the tested equation: we have no means for a proof that such a process of finding the equations will terminate. In order to

guarantee termination of the search for a finite domain we will use somewhat stronger criteria. The criteria we have chosen differ from those given in [DG79] for the Ackermann Class, for example, in such a way that as one of the consequences we can almost always build much smaller models for small formulas, and need much less search steps (the number of steps of checking acceptability of single equations are much *smaller*).

We won't, however, try to improve the known upper bounds on the size of the finite domains.

7.3 THE CRITERIA FOR ACCEPTING A SET OF EQUATIONS

In general we have more than one branch in the Herbrand tree and therefore need many equations to prune all the branches.

In order to guarantee termination of the model-building process without backtracking, we will use a special transformation of the stable form of the input clause set to a certain Essentially Monadic clause set.

During the whole process of search we will use the transformed clause set combined with a special disunification criterion (in respect to the stable form of the original clause set) for checking any new equation t = d.

- The minimal (correctness) criterion: {S' ∨ E ∨ t = d} must be satisfiable, where S' is the transformation of the stable form of the investigated clause set to the Essentially Monadic form and E is the set of equations obtained so far.

- The disunification criterion: if the arity of the function symbol of the left-side term t of the equation t = d is equal to one, then if some pair of atoms A, B in the stable form of the original clause set S is not unifiable, and either A and B have dual signs in S or belong to the same clause in S, then the pair A, B must not be unifiable in the equational theory {E ∨ t = d}.

The case when the arity of the function symbol of the left-side term is bigger than one is rather important. The harmlessness of this "exception" stems from the fact that for the Monadic Class the model building process will terminate without using any special disunification criterion. On the other hand, if this exception is not used, it becomes very hard to show the termination of such a model building process (if it is terminating at all).

EXAMPLE 7.4.: (disunification criterion): Suppose our input set contains Ackermann-type literals $P(f(x), x)$ and $\neg P(y, f(y))$. These literals are not unifiable. However, given an equation $f(a) = a$, the substitution instances $P(f(a), a)$ and $P(a, f(a))$ become unifiable, as then $P(f(a), a) = P(a, a)$ and $P(a, f(a)) = P(a, a)$.

Checking the correctness criterion for any new equation $t = d$ involves deciding the satisfiability of a clause set which contains the equality predicate. The simplest way to deal with the equality would be to axiomatize it in a standard way, giving axioms for transitivity, commutativity, reflexivity, and axioms for substituting into predicates and functions of the formula. Another standard way to handle equality in resolution context would be using paramodulation (see [Wos 84]). Unfortunately, neither the axiomatization (even the transitivity axiom alone) nor paramodulation can be handled by any decision strategy known to the author. Thus we have to use some other ways for handling the equality predicate.

In the following we will use the narrowing method of Slagle (see [Sla 74]): if E is a confluent term rewriting system, narrowing of any expression e is obtained by unifying a left side term h of some rewrite rule in E with some non-variable subterm t of e, applying the obtained substitution σ = m.g.u.(h, t) to e, and rewriting the expression $e\sigma$ to its normal form modulo E (see, for example, [Bar 81]). A narrowing of the narrowing of e is also considered to be a narrowing of e.

A well-known theorem of Slagle [Sla 74] states that if E is a complete term rewriting system (set of simplifiers in his terminology) and S is a set of clauses without equality, then $\{E \cup S\}$ is satisfiable if and only if the full narrowing of S using E is satisfiable. The full narrowing of S is defined as S augmented by all narrowings of all clauses in S.

Remark 1 in forthcoming section 8.4 notes that the set of equations generated by our algorithm can always be oriented to a confluent term rewriting system. It is easy to see that a full narrowing of any set of clauses in AM with any set of equations generated by the the model construction algorithm presented later will also be in the class AM. Thus we have a method for checking the acceptability of sets of equations for formulas in AM: perform full narrowing and then use k-refinement of resolution.

EXAMPLE 7.5.: Let E = {f(a) → a, g(b) → a}. The single narrowing of the atom P(f(x), x) is P(a, a). The atom P(g(x)) has a single narrowing P(a). The atom Q(g(x), f(x)) has two narrowings, Q(g(a), a) and Q(a, f(b)). Consider the set of clauses S = {{P(f(x), x), S(g(x)), {Q(g(x)), ¬S(f(x)), {¬P(f(x), f(x)), P(x,x).

The full narrowing of S using E is the following set:

S ∪
{{P(a, a), S(g(a)},
{P(f(b),b), S(a)},
{Q(g(a)), ¬S(a)},
{Q(a), ¬S(f(b))},
{¬ P(a, a), P(a, a)}}.

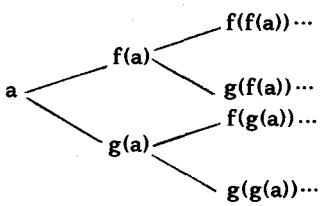
As can be easily checked by application of the k - refinement, this full narrowing is satisfiable.

DEFINITION 7.5.: We say that some set of reductions R preserves disunification in S iff the following holds: if two atoms A and B in the set S are not unifiable, then no narrowing A' of A (using E) is unifiable with B or with any narrowing B' of B.

In the following algorithm the disunification preservation is checked by the procedure ok - disunific.

EXAMPLE 7. 6.: Take the clause set
S = {{P(x,f(x))}, {¬P(f(y),y)}, {¬P(x,g(x))}, {Q(z), Q(g(z))}, {¬Q(u), ¬Q(g(u))}.

Recall that the Herbrand tree for S



is ordered in the following way: $\langle a, f(a), g(a), f(f(a)), \ldots \rangle$.

We will see how the disunification condition, given S, will treat several reductions.

- f(a) a is not OK, as narrowing gives P(a, a) and $\neg$ P(a,a) from literals P(x,f(x)) and $\neg$P(f(x), x), respectively.
- f(f(a)) → a is not OK, as narrowing gives P(f(a),a) and $\neg$P(a,f(a)), accordingly, and P(f(a),a) is unifiable with $\neg$P(f(x),x).
- f(f(f(a))) → a is OK: narrowing gives literals P(f(f(a)),a) and $\neg$P(a,f(f(a)), which are neither unifiable with each other nor with P(x,f(x)) and $\neg$P(f(x),x), respectively.
- g(a) → a is not OK, as narrowing gives a clause {Q(a)}, from the clause {Q(z), Q(g(z))}, where Q(z) and Q(g(z)) are not unifiable.
- g(g(a)) → a is OK: narrowing gives

  {$\neg$P(g(a),a)}, {Q(g(a)), Q(a)} and {$\neg$Q(g(a)), $\neg$Q(a)}.

- Now suppose we have already taken f(f(f(a))) → a into the set of accepted equations. Thus we have the narrowings P(f(f(a)), a) and $\neg$P(a,f(f(a))) from P(x,f(x)) and $\neg$P(f(y),y). Consider the reduction g(f(f(a))) → a: it gives the narrowing $\neg$P(f(f(a)),a) from $\neg$P(x,g(x)), thus not preserving disunification with respect to the set E (as P(x,f(x)) and $\neg$P(x,g(x)) are not unifiable).

7. 4 THE ALGORITHM FOR BUILDING FINITE MODES: DETAILS

We present an algorithm for building a finite domain and an interpretation of the function symbols for a clause set S in Class AM. The full model for S (i.e. also an interpretation of predicate symbols on elements of domain) can then be built from the domain and the interpretation of function symbols in different, simple ways (for example, applying unrestricted resolution - with elimination of subsumed clauses - to the set obtained from S by substituting elements of a domain to variables in S in all possible ways and normalizing all atoms) which are not investigated in our book.

The following algorithm actually constructs a finite domain and interpretation for a clause set $R^*_{\succ k}(S)$. Notice that since $R^*_{\succ k}(S)$ is a set of clauses derived from S, $R^*_{\succ k}(S) - S$ might subsume several clauses in S and several function and predicate symbols in S might not appear in $R^*_{\succ k}(S) - S$. In the model of S these symbols can be interpreted in an arbitrary way.

7. 4. 1 The model construction process

Let S be the investigated satisfiable set of clauses. At first the stable form $R^*_{\succ k}(S)$ for S is computed . $R^*_{\succ k}(S)$ is then transformed to a certain Essentially Monadic clause set SNEW, which will be used during the whole model construction process instead of S or $R^*_{\succ k}(S)$.

Consider the ordering of the Herbrand tree in the top-down, left-to-right direction. The previous example: given two one-place function symbols f and g and a single constant symbol a, the first three levels of the tree can be ordered in the following way: <a, f(a), g(a), f(f(a)), g(f(a)), f(g(a)), g(g(a))>. For the k-th element h_k of such an enumeration the algorithm will try to add an equation $h_k = h_l$; ($l < k$), starting with $l = 1$ and continuing up to $l = k_1$, unless the acceptable equation is found earlier. The check for acceptability of any equation $h_k = h_i$ contains following two steps:

(1) in case the arity of the function symbol of the term h_k is equal to one, check disunification preservation,

(2) check satisfiability of {SNEW ∪ E ∪ h_k = h_i} . The satisfiability check is performed by computing a full narrowing of SNEW with {E ∪ $h_k \rightarrow h_i$ }. Actually only the narrowing of the formula SNEW' with the reduction $h_k \rightarrow h_i$ has to be computed, where SNEW' is the narrowing of SNEW using E, obtained ealier. In case the acceptable equation h_k = h_i has been found for some i, one "branch" in the Herbrand tree has been cut. If such an equation is not found, the term h_k is taken as a new element of the constructed domain. In order to speed up future computations, it is useful to introduce a new constant symbol c_k for the term h_k and to compute the stable form of SNEW' with the defining reduction $h_k \rightarrow c_k$. In the following this new stable form will be used instead of SNEW' (obviously this stable form is always satisfiable). One of the important consequences of replacing a possibly deep ground term with a new constant is that the narrowed clause set is always in the Class AM - respectively in the Essentially Monadic subset of AM .

In the following algorithm the satisfiability check is performed by the function satisfiable, which is implemented by the k-refinement. The disunification check is performed by the function ok-disunific okdisunific (S^*, E ∪ {f a → f b}) could be computed in the following way: for all pairs of nonunifiable dual literals in S^* generate all narrowings of both elements of the pair with {E ∪ {fa → fb}} and test whether any narrowing of the first element (or the first element itself) doesn't unify with any narrowing of the second element (or with the second element itself). In fact, one should perform this narrowing-computing incrementally during the process of finding new elements of E.

In implementing the following algorithm one should avoid the duplicate calculation of $R^*_{\nearrow k}$(narrow(SNEW, fa → fb)) by using the value computed earlier during a computation of satisfiable(narrow(SNEW, fa → fb)).

```
{COMMENT: S is the satisfiable input set of clauses};
S* := R*>k (S); SNEW :=transform–to–monadic(S*);
D:= {c | c is a constant symbol in SNEW};
IF D = {} THEN D := {a};
DNEW :=D; DLAST :=D; E = {};
WHILE DNEW == {}
 BEGIN
  DNEW := {};
  FOR EACH f IN {h | h is a functional symbol in SNEW }
   BEGIN
    FOR EACH fa IN {f(t1,...,tn) | ∀1≤i≤n: ti∈D and ∃j: tj∈DLAST }
      BEGIN
        foundflag :=FALSE;
        FOR EACH c in D WHILE foundflag = FALSE
          BEGIN
            IF ((arity(f) > 1) OR
              (ok disunific (S*, E ∪ {fa → c})))
              AND
              satisfiable (narrow(SNEW, {fa → c})))
            THEN
              BEGIN
                E:=E ∪ {fa → c};
                SNEW := R*>k(narrow(SNEW, {fa → c}));
                foundflag :=TRUE;
              END
          END
        IF foundflag=FALSE
          BEGIN
            c:=newconstant ();
            E:=E ∪ {fa → c};
            DNEW:=DNEW ∪ {c};
           SNEW := R*>k (narrow(SNEW, {fa → c}));
          END
      END
   END
  DLAST:= DNEW; D:= D ∪ DNEW
END.
```

Remark 1:

The set E of all equations generated at each step of our algorithm is always such that orienting all equations $h_k = h_l$ in E into rewrite rules $h_k \rightarrow h_l$ produces a complete term rewriting system.

In view of remark 1, in the following we will treat E as a set of rewrite rules or as a set of equality units, depending on the context.

7.5 TRANSFORM-TO-MONADIC PROCEDURE

In the following we will present a transformation of a stable form S of clause set in the Class AM to a clause set S' in the Essentially Monadic Class so that the following two assertions hold:

- (1) if S is satisfiable, S' is also satisfiable;
- (2) for any set of equations E computed by our finite-model-building algorithm, if $\{E \cup S'\}$ is satisfiable and E satisfies the disunification condition for S, then $\{E \cup S\}$ is satisfiable, too.

The inverse of the assertion (2) is not generally true, however: satisfying the disunification condition and S' will not imply satisfying S. This is the reason why we have to use the transformation in the actual model-building process - if the computed set of equations (satisfying the disunification condition) satisfies S iff it satisfies S', the transformation could be used only in the termination proof and not in the actual algorithm.

At first we will present a pseudo-code version of the transformation, to be followed by a natural-language description and examples.

```
BEGIN
Let us initially have S={C1,..., Cn} such that R*>k(S)=S;
S1 :={};
MON := {R | R is a monadic-type literal in some Ci};
FOR EACH CLAUSE Ci IN S DO
  BEGIN
    S1:= S1 ∪ Ci;
    ACK:={L | L is an Ackermann-type literal Ci};
    FOR EACH LITERAL L IN ACK DO
      BEGIN
        FOR EACH R IN MON DO
          BEGIN
            IF EXISTS m.g.u(L, Rd)
             THEN
               BEGIN
                σ:=m.g.u.(L, Rd);
                S1:= S1 ∪ Ci σ ;
               END
          END
      END
  END
ACK := {L | L is an Ackermann-type literal in some Ci}
FOR EACH Li IN ACK DO
  BEGIN
    sgn:=sign(Li);
    p[i]:=new—monadic—predicate—symbol();
    {COMMENT: p[] is an array of monadic predicate symbols};
    F(Li) := {f | f is a function symbol in Li;
    {COMMENT: let F(Li), now have a form f1, ... ,fm }}
    t:= argument of a functional term in Li;
    {COMMENT: for Ackermann-type literals t is unique};
```

```
      newmon1[i]:= {sgn (p[i](f1 (t))), ..., sgn(p[i](fm (t)))}
                    for each fj  in F(Li );
      {COMMENT: newmon1[] is an array of sets of literals};
    END
  FOR EACH Li in ACK  DO
    BEGIN
      newmon2[i]:=newmon1[i];
      {COMMENT: newmon2[] is an array of sets of literals};
      FOR EACH Rj IN ACK  SUCH THAT i ≠ j DO
        BEGIN
          IF EXISTS m.g.u(Li, Rj^d);
           THEN
           BEGIN
             σ := m.g.u(Li, Rj^d);
             IF NOT(ground(Li σ ))
              THEN σ:= {y/x},
                  where x is a variable in Rj,
                  and y is a variable in Lj;
             newmon2[i] := newmon2[i] ∪ (newmon1[j]σ);
             {COMMENT: newmon1[j]σ ={e1 σ, ... ,eN σ}
           END
        END
    END
```

To form the set of clauses S' from S_1, replace each clause C in S_1 by a set of new clauses obtained by replacing all Ackermann-type literals L_i in C with the atoms from corresponding newmon2[i] in all possible ways. During that replacement, whenever a non-ground literal L_i with a single variable x in C is replaced by a ground unary atom L', where a constant symbol c is the argument of some functional term, apply the substitution c/x to the result of transformation.

From the resulting set of transformations remove all such transformed clauses, which contain at least two such transformed literals $A(f_i(c_j))$, $B(f_k(c_l))$, where the corresponding original literals A(...) and B(...) both contained variables and the substituted constants c_j and c_l are different.

END.

In the following we will present the same transformation algorithm in natural language, combined with examples.

Let S be the stable form of a set of clauses in the Class AM (that is, $S = R^{*}_{>k}(S)$). Form the new set of clauses S' = transform–to–monadic(S) by replacing each old clause C from S by a set of new clauses in the following two-step way:

First step:
Form the new set S_1 by replacing each clause C in S by a set of new clauses in the following way:

(1) C contains Ackermann-type literals which are unifiable with some dual Monadic-type literals in S: Let L be such an Ackermann-type literal in C, and let G be some Monadic-type literal in S, the dual of which is unifiable with L. Then $m.g.u(L, G^{d})$ will assign a certain constant symbol c for a single variable x in L. Let Cset = $\{c_1, \ldots, c_n\}$ be the set of all constants assigned to x by unification with different Monadic-type literals. Form the set of clauses $C, \{C\sigma_1, \ldots, C\sigma_n\}$, where $\sigma_i = c_i/x$, where c_i belongs to Cset, x is the single variable in Ackermann-type literals in C.

EXAMPLE 7.7.: Let us transform a clause {P(f(x), x), S(x, c)}, and let the set S contain Monadic literals ¬P(y, a) and ¬P(z, b). Then, as an Ackermann-type literal P(f(x), x) is unifiable with the duals of ¬P(y, a) and ¬P(z, b), we have to generate the set { {P(f(x), x), S(x, c)}, { P(f(a), a), S(a, c)}, { P(f(b), b), S(b, c)}}. Notice that the instances of the Acker-mann-type atoms in the new clauses generated this way are always of the Monadic type.

(2) C does not contain Ackermann-type atoms, unifiable with negation of Monadic-type atoms in S: Let the new set of clauses contain the unchanged clause C alone; that is, in this case additional clauses are not generated.

Thus S_1 will be equal to S ∪ {SINST}, where SINST is a certain set of instances of clauses from S.

EXAMPLE 7. 8.: Consider the clause set S:

{{R(f(x), g(x), x), ¬P(x, f(x))},
{P(a, y)},
{R(f(a), g(a), a)},
{S(f(x), x, x), L(x)},
{¬S(f(x), x, a)},
{¬S(f(x), x, b)},
{L(a)},
{L(b)}}.

As there is a single pair ¬P(x, f(x)), P(a, y) of unifiable dual Ackermann-Monadic literals in S, the first step of the transformation will give us a following set S_1:

{{R(f(x), g(x), x), ¬P(x, f(x))},
{R(f(a), g(a), a), ¬P(a, f(a))},
{P(a, y)},
{R(f(a), g(a), a)},
{S(f(x), x, x), L(x)},
{¬S(f(x), x, a)},
{¬S(f(x), x, b)},
{L(a)},
{L(b)}}.

Second step:

Let ACK be the set of all Ackermann-type atoms in S_1. For each atom in ACK take a new unique unary predicate symbol. Also, for each literal L in ACK find the set of all dual literals in ACK, which are unifiable with L. For each element R in such a set one of the following holds: the literal R is syntactically equal to L up to renaming variables, or the unification of R with L produces a substitution σ so that Lσ is ground. Take the set of all function symbols in L - let the set be denoted by F(L). Form a new set of atoms newmon2[i], where i is the number of L in some enumeration of ACK:

For each new monadic predicate symbol P, corresponding to some Ackermann-type literal R, unifiable with the dual of L, form the set of atoms of the form $P(f_i(t))$, where f_i is some element of F(L) and t is determined in the following way:

- In case R is syntactically equivalent to L (up to renaming variables), let t be the argument of any functional term in L.
- In case R is not syntactically equivalent to L (up to renaming variables), then m.g.u.(R, L^d) = {c/x, c/y}, where x and y are the single variables of R and L, correspondingly, and c is some constant symbol (or m.g.u(R, L^d) = {c/x} or m.g.u.(R, L^d) = {c/y}). Take t to be the constant symbol c.

To form the set of clauses S' from S_1, replace each clause C in S_1 by a set of new clauses obtained by replacing all Ackermann- type literals L_i in C by the atoms from corresponding newmon2[i] in all possible ways. During that replacement, whenever a non-ground literal L_i with a single variable x in C is replaced by a ground unary atom L', where a constant symbol c is the argument of some functional term, apply the substitution {c/x} to the result of the transformation.

From the resulting set of transformations remove all such transformed clauses, which contain at least two such transformed literals $A(f_i(c_j))$ and $B(f_k(c_l))$, where the corresponding original literals A(...) and B(...) both contained variables and the substituted constants c_j and c_l are different.

An example of the last "filtration" step: Let us have a clause {P(f(x), x), R(f(x), x)}, and let the corresponding sets of new atoms for the original atoms in the clause be {P1(f(x)), P2(f(a)), P3(f(b))} and {R1(f(x)), R2(f(a)), R3(f(b))}, respectively. Then the transformations {P2(f(a)), R3(f(b))} and {P3(f(b)), R2(f(a))} must be removed, and the resulting set of transformations will be {P1(f(x)), R1(f(x))}, {P2(f(a)), R2(f(a))}, {P3(f(b)), R3(f(b))}.

EXAMPLE 7. 9.: We continue the example from the "first step": The set ACK for S would be

{R(f(x), g(x), x), P(x, f(x)), R(f(a), g(a), a), S(f(x), x, x), S(f(x), x, a), S(f(x), x, b)}.

Let the set of new predicate symbols for ACK be {R1, P1, R2, S1, S2, S3}. The corresponding sets of newmon2[i], are:

R(f(x),g(x),x) - {R1(f(x)), R1(g(x)), R2(f(a)), R2(g(a))},
P(x,f(x)) - {P1(f(x))},
R(f(a),g(a),a) - {R2(f(a)), R2(g(a)), R1(f(a)), R1(g(a))},
S(f(x),x,x) - {S1(f(x)), S2(f(a)), S3(f(b))},
S(f(x),x,a) - {S2(f(x)), S1(f(a))},
S(f(x),x,b) - {S3(f(x))}, S1(f(b))}.

Here is the resulting S' of the second and final transformation step:

{{R1(f(x)), ¬P1(f(x))},
{R1(g(x)), ¬P1(f(x))},
{R2(f(a)), ¬P1(f(a))},
{R2(g(a)), ¬P1(f(a))},
{R2(f(a)), ¬P(a,f(a))},
{R2(g(a)), ¬P(a,f(a))},
{R1(f(a)), ¬P(a,f(a))},
{R1(g(a)), ¬P(a,f(a))},
{P(a,x)},
{R2(f(a))},
{R2(g(a))},
{R1(f(a))},
{R1(g(a))},
{S1(f(x)), L(x)},
{S2(f(a)), L(a)},
{S3(f(b)), L(b)},
{S2(f(a))},
{S3(f(b))},
{L(a)},
{L(b)}}.

7.6 SOUNDNESS

Let S be a stable clause set in the Class AM, clause set S' be equal to transform–to–monadic(S), and E be a certain set of ground equations, computed at any stage during our finite-model-building algorithm for S. By the definition of our model-construction algorithm E satisfies the disunification condition for S.

We will say that a clause C' corresponds to the clause C if C' is a transformed version of C, or the derivation of C can be rebuilt into a derivation of C' by replacing clauses in the original derivation with transformations, if needed. We will use the notion of correspondence also in the inverse sense.

THEOREM 7. 1.: If S is satisfiable then, S' is satisfiable.

Proof:

Recall that during the transformation of S to S' ground substitutions could be applied during the derivation of a transformed clause from the original one. For such new clauses the k-refinement may allow more literals to be resolved upon than for the original clause without ground substitution applied. In order to avoid complications stemming from the k-refinement, we will use the following construction: Build a new set of clauses SG = S ∪ G, where

G = {Cσ | (C belongs to S and

C contains a literal L, which contains a single variable x, and

σ = {c/x}, where c is some constant in S)}.

Obviously S is satisfiable iff SG is satisfiable.

We will show that for any clause C' derivable from S' by $R_{>k}$ (without using subsumption) there is a corresponding clause C derivable from SG by $R_{>k}$ (without using subsumption).

Induction basis: The construction of S' from S.

Induction step: We will give the proof for the binary resolution only, as this proof can be easily modified for factorization.

Let $C_1' = \{L_1', ..., L_n'\}$ and $C_2' = \{G_1', ..., G_m'\}$ be two clauses derivable from S' by $R_{>k}$. Let a clause $C_3' = \{L_2', ..., L_n', G_2', ..., G_m'\}\sigma$, where $\sigma = m.g.u(L_1', G_1'^{d})$, be derivable from C_1' and C_2' by binary resolution. As an induction hypothesis suppose that there are corresponding clauses $C_1 = \{L_1, ..., L_n\}$ and $C_2 = \{G_1, ..., G_m\}$ derived from SG by $R_{>k}$. We will show that then the corresponding clause $C_3 = \{L_2, ..., Ln, G_2, ..., G_m\}\sigma$ or a more general clause $C_3 = \{L_2, ..., L_n, G_2, ..., G_m\}\rho$

where $\sigma = \rho\mu$ for some substitution μ , can be derived from C_1 and C_2. Actually, as the next theorem demonstrates, more general clauses cannot be derived; for the current theorem this fact is not needed.

Consider the following cases:

- L_1 and G_1 are Monadic: Then the corresponding literals L_1' and G_1' are syntactically equal to L_1 and G_1 (as Monadic-type literals are not transformed), and derivability of C_3 from C_1 and C_2 is obvious.
- L_1 and G_1 are of the Ackermann type: By the definition of the transformation algorithm, unification of the transformed literals can only be restricted when compared to the unification of original literals. Thus, as L_1' and G_1' are unifiable with m.g.u σ, L_1 and G_1 are also unifiable either with the same m.g.u. σ or a m.g.u. ρ, more general than σ ($\sigma = \rho\mu$ for some substitution μ).
- L_1 is of the Ackermann type, G_1 is of the Monadic type: Notice that in this case L_1' must be a ground Monadic-type literal by the definition of transformation (during the transformation to L_1' the substitution m.g.u(L_1, G_1) = ρ is applied to L_1, and ρ is for the current case always ground; it has the form {c/x, f(c)/y}, where x is a single variable in L_1, y is a single variable in G_1 (if G_1 is ground, ρ has the simple form{c/x}), f is a single function symbol in L_1). Thus, L_1 and G_1^d are obviously unifiable and C_3 is derivable from C_1 and C_2.

Q.E.D.

THEOREM 7.2.: If {E ∪ S'} is satisfiable then {E ∪ S} is also satisfiable.

Proof:
We will show that for any clause in the full narrowing SN of S using E and for any clause derivable from SN by $R_{>k}$, there is a corresponding clause derivable from the full narrowing S'N of S' by E using $R_{>k}$. Notice that for any non-transformed clause the k-refinement ordering of that clause either exactly corresponds to or is stronger than k-ordering of the corresponding transformed clause.

Induction basis: The full narrowing of S by E. Due to the fact that for any equation t → d in E there is a single possible narrowing of any Ackermann-type atom in S by this equation, for any narrowable clause in S by E there is a corresponding clause in S' which can be narrowed in the same way.

Induction step: We will give the proof for the binary resolution only, as this proof can be easily modified for factorization.

Let $C_1 = \{L_1, ..., L_n\}$ and $C_2 = \{G_1, ..., G_m\}$ be two clauses derivable from SN by $R_{>k}$. Let a clause $C_3 = \{L_2, ..., L_n, G_2, ..., G_m\}\sigma$, where σ = m.g.u.(L_1, G_1^d), be derivable from C_1 and C_2 by binary resolution. As an induction hypotheses suppose that there are corresponding clauses $C_1' = \{L_1', ..., L_n'\}$ and $C_2' = \{G_1', ..., G_m'\}$ derived from S'N by $R_{<k}$. We will show that then the corresponding clause $C_3' = \{L_2', ..., L_n', G_2', ... , G_m'\}\sigma$ can be derived from C_1' and C_2'.

Consider the following cases:

(1) L_1 and G_1 are Monadic: Then the corresponding literals L_1' and G_1' are syntactically equal to L_1 and G_1, and derivability of C_3' from C_1' and C_2' is obvious.

(2) L_1 and G_1 are of the Ackermann type: As they are unifiable, and the set E preserves disunification, then there must be such transformations of C_1 and C_2 which have literals L_1' and G_1' corresponding to L_1 and G_1, so that L_1' and G_1' are unifiable (notice that this is generally true only due to disunification preservation by E). We have to investigate m.g.u(L_1, G_1^d):

(2.1) m.g.u.(L_1, G_1^d) = {x/y}, where x is a single variable in L_1, y is a single variable in G_1: There are such corresponding C_1' and C_2' that m.g.u(L_1', G_1^d) = {x/y} for corresponding x, y.

(2.2) m.g.u.(L_1, G_1^d) = {c/x}, where x is a single variable in L_1:
Consider the "leaf" clause S (contained in S'N) in the derivation of C_1 containing the literal L_1. There is a transformation S' of S in S'N with a substitution {c/x} applied. The derivation of C_1 can be easily rebuilt into a derivation of C_1' from S'N, replacing corresponding literals and applying the substitution {c/x} where needed.
Now we have got a half of C_3', namely $\{L_2', ..., LN'\}\sigma$, with a right substitution. The other half, $\{G_2', ... , GM'\}\sigma$ is obtained by applying a substitution m.g.u$(L_1', G_1'^d)$ = {c/y}.

(2.3) m.g.u.(L_1, G_1^d) = {c/x, c/y}, where c is some constant, x is a single variable in L_1, y is a single variable in G_1: This case is similar to the previous case, with the main difference that the argumentation from the last case could be carried out symmetrically. We can freely pick one of the possible two ways of analysis:

Consider the "leaf" clause (contained in SN) S in the derivation of C_1 containing the literal L_1. There is a transformation S' of S in S'N with a substitution {c/x} applied. The derivation of C_1 can be easily rebuilt into a derivation of C_1' from S'N, replacing corresponding literals and applying the substitution {c/x} where needed.
In this way we get a half of C_3', namely {L_2', ..., LN'}σ , with a right substitution. The other half, {G_2', ... , GM'}σ is obtained by applying a substitution m.g.u.(L_1', $G_1'^d$) = {c/y}.

(2.4) m.g.u.(L_1, G_1^d) is an empty set. Similarly to case (2.1) L_1 is syntactically equal to G_1 and derivability of C_3' is trivial.

(3) L_1 is of the Ackermann type and G_1 is of the Monadic type (e.g, P(f(x),x) and ¬P(x,b)). Obviously the corresponding literal L_1' has a new predicate symbol, different from the one in L_1, and thus L_1' and $G_1'^d$ are not unifiable. But, during the first step of the transformation, for all clauses C containing an Ackermann-type atom L, unifiable with some Monadic-type atom P in S with some σ = m.g.u.(L, P^d), a new Monadic instance Cσ was generated. Consider the derivation of a clause C_1 from a full narrowing SN. Let σ = m.g.u.(L_1, G_1^d). Clearly the derivation of C_1 can be rebuilt into the derivation of C_1 σ by replacing the clause C in SN containing the literal L_1 (as L_1 is of the Ackermann-type and unifiable with a Monadic atom, it must be non-ground) with a clause Cσ , found in S'N, and applying a substitution σ where needed. Therefore C_3' = C_3 can be derived from S'N, like in case (1) of two Monadic literals.

Q.E.D.

7.7 TERMINATION PROOF

Given that our original formula S belongs to Class AM, the termination of every subprocedure in our program is easy to prove.

We have to show that $R^*_{>k}$ always terminates. We use the fact that $R_{<k}$ decides Maslov's Class K, thus also the Class AM.

If $R^*_{>k}$ terminates on a set S, then procedure *satisfiable* terminates on S also.

The transformation algorithm transform-to-monadic oviously terminates. The same is true for okdisunific.

7.7.1 Termination of the whole model construction algorithm

We prove the termination by giving an upper bound on the length of the branches of the "pruned" Herbrand tree. Recall that the finite domain D is constructed by finding a set of rewrite rules E such that terms containing elements of D are reduced by rewrite rules in E to terms in D.

The idea of the proof is the standard pigeonhole argument, used in [DG 79] for most termination proofs of the kind. In fact, one could "read off" the current termination proof from some proofs in [DG79]. However, we will give a termination proof in order to have a complete presentation.

During the proof we will consider the modification of the model construction algorithm where new constant symbols are not introduced for new domain elements, and possibly deep ground terms are used instead. Obviously there is no difference in the process of model construction between the modification with new constant symbols and the modification without them.

Let $B_n = \langle t_1,\ldots,t_n\rangle$ be a tuple of m first elements of the constructed domain D. Let $B_m = \langle d_1,\ldots,d_i\rangle$ be such a subtuple of B_n, which contains exactly the elements of a certain branch of Herbrand universe. For example, $\langle$b, f(b), g(c,f(b)), f(g(c,f(b))$\rangle$

could be such a branch. As our algorithm has added all elements of B_n to D, it also has added all elements of the subtuple (branch) B_m, and therefore cannot have added any equation of the form $d_i = t$ (where t is any term and $1 < i < m$) to E, since adding any such equation to E would have meant pruning the branch B_m. Nevertheless, during the generation of D the algorithm has for every equation of the form $d_i = d_j$ (where d_i and d_j belong to the branch B_m and $1 < i < m$ and $1 < j < i$) examined the possibility to add this equation to E. There are two possible causes for rejecting the examined equation $d_i = d_j$:

(A) equation $d_i = d_j$ does not preserve disunification in S.

(B) {SNEW $\cup$ E $\cup$ $d_i = d_j$} is not satisfiable.

In the following we show that both of the abovementioned conditions, (A) and (B), decompose every branch of the Herbrand tree of S into a finite number (n_A, n_B, accordingly) of classes so that if an element d_i belongs to the same class (for both of the conditions A, B) as an element d_j, then the equation $d_i = d_j$ would be accepted by our algorithm. From this it immediately follows that if the number m (which represents the number of domain elements constructed) $> n_A * n_B$, the set E will contain one equation of the form $d_i = t$ ($1 < i < m$, t is some term), thus pruning the branch B_m in D. The upper bound on the size of the model which will arise from the upper bound on the length of branches is by no means the best: we give it only in order to prove the termination of the search. We say that a term contains itself at depth 0, arguments of its leading functional symbol at depth 1, etc.

By the inequality of structures up to some depth n we mean that when we replace the subterms deeper than n both in d_i and d_j by a new common constant c, the resulting terms are not syntactically equal. For example, the structures of terms f(a, g(h(b,c), c)) and f(a, g(g(d,c),c)) are equal up to depth 1, but inequal up to depth 2.

(A): disunification

Equation $d_i = d_j$ may not preserve disunification for atoms in S only if one of the following holds:

- the leading function symbol of d_i has an arity equal to one and d_i contains d_j as a subterm at depth less than 3: We can say that "the cause" for this

condition is the occur check: take atoms P(f(x),x), P(y,f(y)) and equation f(f(f(f(a)))) = f(f(f(a))), for example. The chosen atoms are not unifiable, but their narrowings with the chosen equation are both P(f(f(f(a))),f(f(f(a)))). The depth boundary 3 is sufficient since Ackermann-type atoms in S do not contain more than one variable and depth of terms in these atoms is 0 or 1.

- the structure of d_i is up to depth 1 not equal to the structure of d_j. The number of classes of terms with different structure up to depth 1 is equal to $(n_f * (n_c + n_f)^m$ where n_f is the number of the functional symbols in S, n_c is the number of constant symbols in S, and m is the maximal arity of function symbols in S.

Overall, we can take the number for classes of disunification $n_A = 3 * (n_f * (n_c + n_f)^m$ for the following reasons. Consider the branch B_m. Divide it into parts consisting of three adjacent elements. For example, given a branch < b, f(b), g(c, f(b)), f(g(c, f(b)), f(f(g(c, f(b))), h(f(f(g(c, f(b))))), ...> we will divide it into <<b, f(b), g(c,f(b))>, <f(g(c,f(b)), f(f(g(c,f(b))), h(f(f(g(c,f(b)))))>, ... > . Now for any two such parts of the branch the "occur check" reason for disunification cannot hold for any pair of elements taken from these parts, accordingly. Thus only the structural differences have to be considered for testing the disunification property between elements of any two three-element parts of the branch. In fact, we could organize the search for an equation guaranteeing disunification preservation by investigating a single branch only, and considering only the first elements of three-element parts of the branch.

(B): satisfiability

$n_B = 2^w$, where w is the number of different atoms in the set of clauses transform-to-monadic($R^*_{>k}$(S)). The following explains our selection of the number n_B.

Differently from formulas in the Ackermann Class, the satisfiability of a set of ground equations for any formula in a Monadic Class is equivalent to the following criterion (see [DG79] for the well-known, slightly different version of this criterion):

(1) Form the set of all instantiations of all atoms in a clause set $R^*_{>k}$(S),

obtained by applying substitutions given by unifying the left sides of all equations t → d with terms in the atoms. For example, given literals {P(f(x,y)), R(g(x))} and equations {f(a,b) → c, g(e) → l} we will get a set {P(f(a, b)), R(g(e))}.

(2) Rewrite the terms in the set obtained after the previous step (1) to normal forms given by the set of ground equations. From the previous example we will get a tuple {P(c), R(l)}.

Steps (1) and (2) give us two tuples of atoms with one-to-one correspondence between elements: atoms with certain ground substitutions and corresponding normal forms.

(3) Give all the atoms in the tuples signs (either positive or negative), so that corresponding atoms in two tuples would have the same sign. For example, {¬P(f(a,b)), R(g(e))} and {¬P(c), R(l)}.
Now the investigated set of equations is satisfiable for the investigated clause set iff it is possible to give such a combination of signs in the described way, so that the conjunction of all literals in two signed tuples with the investigated formula is satisfiable.

For the Monadic Class the described procedure gives a well-known upper bound on the size of the finite domain: 2^m, where m is the number of different predicate symbols in the clause set. For the Essentially Monadic Class the upper bound is obtained in an analogous way, and the number of different atoms in the stable form of a formula can be taken as an upper bound for m.

For the Ackermann Class the reasoning above cannot be applied. For example, consider the reduction f(a) → f(b) and an atom P(f(x), x). The instance of P(f(x),x) obtained from the left side of the reduction is P(f(a), a), which is reduced to P(f(b), a). The instance P(f(b), b) obtained from the right side of reduction is different from P(f(b), a).