Computing Answer Sets Using Model Generation Theorem Provers

Orkunt Sabuncu and Ferda N. Alpaslan

Department of Computer Engineering
Middle East Technical University,
06531 Ankara, Turkey
{orkunt, alpaslan}@ceng.metu.edu.tr

Abstract. Model generation theorem provers have the capability of producing a model when the first-order input theory is satisfiable. Because grounding step may generate huge propositional instances of the program it hardens the search process of answer set solvers. We propose the use of model generation theorem provers as computational engines for Answer Set Programming (ASP). It can be seen as lifting of SAT-based ASP to the first-order level for tight programs to eliminate the grounding step of ASP or do it more intelligently.

1 Introduction

ASP is a declarative approach for solving search problems. A search problem is represented by a logic program whose models, according to the answer set semantics, correspond to the solutions of the problem. It has originated from logic programming and nonmonotonic reasoning during the late 90s [1–3]. With the emergence of the fast answer set solvers and its highly declarative nature, ASP has become an effective and widely accepted paradigm for knowledge representation and reasoning tasks [4]. Among the areas of application, we can list planning, configuration, and diagnosis.

Current answer set solvers work on grounded logic programs. The grounding step becomes a bottleneck of ASP when size of the program is relatively large, which results in a huge propositional instance of the program. This makes the search process of the answer set solver difficult. This is the main motivation behind lifting SAT-based ASP to the first-order level. We use model generation theorem provers as computational engines for ASP, which can process first-order input.

Refutational theorem provers search for the unsatisfiability of input theory usually given in first-order logic. Since the negated conjecture is added to the input theory, the unsatisfiability result proves the conjecture. However, if the theory is satisfiable many provers say only that it is satisfiable. Usually, there is no additional information which can be used as a clue to understand whether the conjecture is invalid or there is an error in the logical representation of the domain and the theorem. A model of the theory can be a valuable information

in this sense. It may help to pinpoint a possible error in the representation or convince us that the conjecture is actually invalid. Unlike traditional theorem provers, model generation theorem provers can output a model when the input is satisfiable.

The main support which makes model generation provers interesting is the paradigm of solving search problems. A search problem is represented by a first-order theory whose models correspond to the solutions of the problem. This support is especially important for knowledge representation and reasoning applications which, in turn, makes the model generation theorem provers flexible engines [5].

We have concentrated on Darwin[1], FM-Darwin[1] and Paradox[2] as model generation systems.

The ability of completing a logic program with variables and forming a first-order theory in this way is the basis of this work. The first-order theory becomes the input to the model generation prover. When the completion semantics [6] and the answer set semantics [7] coincide [8–10] for a program, we can find the answer sets of it by searching the models of the formed first-order theory. In this way, we plan to eliminate the grounding step of ASP or do it more intelligently by the model generation system.

Answer set computation based on the relationship between the completion and answer set semantics has been studied earlier and has lead to successful answer set solvers like Assat[3] and Cmodels[4]. However, these solvers complete the grounded logic program [11, 12]. Since the completed theory is propositional, they use SAT solvers to find the models of it. They have also managed to find answer sets in a sound way even for non-tight programs where the completion semantics and the answer set semantics differ. Our work does not include non-tight programs. Answer set computation experimented here can be thought as a lifting of the process of solvers, like Assat and Cmodels, to the first-order level for tight programs.

Altough not all the results we have obtained so far are very promising, our work can shed light on ASP on the first-order level. It also may increase the awareness of the ASP community in some recent theorem provers. The calculi and techniques used in the model generation theorem provers can be beneficial to ASP on the first-order level.

The brief descriptions of the model generation theorem provers used are given in the next section. Section 3 describes the lifting of the process of SAT-based answer set solvers to the first-order level. Section 4 includes the experimental results. In the final section, conclusions and future work can be found.

---

[1] http://combination.cs.uiowa.edu/Darwin/
[2] http://www.cs.chalmers.se/~koen/folkung/
[3] http://assat.cs.ust.hk/
[4] http://www.cs.utexas.edu/users/tag/cmodels.html

2 Model Generation Theorem Provers

The following is a brief description of the provers used and the calculi they are based on.

2.1 Darwin

Almost all modern SAT solvers are based on DPLL procedure. Successful techniques that are applicable in DPLL-based SAT, like unit propagation, backjumping and learning, make the modern SAT solvers very efficient. The Model Evolution (ME) calculus lifts the Davis-Putnam-Logemann-Loveland (DPLL) procedure to the first-order level [13]. It also aims that those SAT techniques will be still applicable in the lifted DPLL procedure. Darwin is an implementation of the ME calculus and has the first-order versions of successful SAT techniques [14].

The ME calculus tries to find a Herbrand model of the input set of clauses, if one exists. The derivation rules of the calculus updates an initial context until it becomes a model or the unsatisfiability of the input clauses is detected.

A *context* Λ is a finite representation of a Herbrand interpretation I_Λ. It is a set which is composed of ground and non-ground literals. An initial context represents an interpretation which falsifies all the ground atoms. When there is a clause which is not satisfied by the Herbrand interpretation represented by the current context, the derivation rules repair I_Λ by adding literals to the context. If all the clauses are satisfied, then the context actually represents a Herbrand model of the input theory. The ME calculus can also detect the situation when there are no possible repairs of the context. This implies the unsatisfiability of the input theory.

There are two kinds of variables defined in the ME calculus; *universal variables* (simply variables) and *parametric variables* (simply parameters). Context literals are either *universal* (i.e., has only variables) or *parametric* (i.e., has only parameters). All the literals of a context, whether it is universal or parametric, stand for all of its ground instances, which, as a whole, compose a Herbrand interpretation. While an universal literal stands for all of its ground instances with no exceptions, a parametric literal represents all of its ground instances except the ones that are instances of another context literal of the same predicate with the opposite sign.

Here is an example from a presentation of the ME calculus[5]. Let $\Lambda_1 = \{p(u,v), -p(u,u), p(f(u),f(u))\}$ where u and v are parameters. The Herbrand interpretation I_{Λ_1} satisfies every instance of $p(u,v)$ except that u and v are the same. It satisfies every instance of $-p(u,u)$ except the ones of the positive literal $p(f(u),f(u))$. If we change the negative literal of Λ_1 to a universal one, we get $\Lambda_2 = \{p(u,v), -p(x,x), p(f(u),f(u))\}$ where x is a variable. The universal literal $-p(x,x)$ imposes a strict restriction that when the arguments of p are

[5] http://users.rsise.anu.edu.au/~baumgart/slides/MEGoeteborg.pdf

the same, the context produces it negatively. Since the literal $p(f(u), f(u))$ is in contradiction with this restriction, Λ_2 is contradictory.

There are two important situations which the ME calculus should detect during the derivation. First one is the case when the current context falsifies an input clause. The other is the one when the current context permanently falsifies a clause no matter how the context is repaired [13]. A context Λ *falsifies* a clause $C = L_1 \vee \cdots \vee L_n$ (i.e., $I_\Lambda \not\models C$) if there are fresh variants $K_1, \cdots, K_n$ of literals in Λ and a substitution σ, where σ is a simultaneous most general unifier of literal sets $\{K_1, \overline{L_1}\}, \cdots, \{K_n, \overline{L_n}\}$[6]. The σ is defined as a *context unifier* of the clause C against the context Λ. A context Λ *permanently falsifies* C (i.e., $I_\Lambda \not\Vdash C$) if there is a context unifier σ of C against Λ and for all i, $(Params(K_i))\sigma \subseteq P$ where $Params(K)$ denotes the set of parameters found in the literal K and P denotes the set of all parameters.

The Split rule applies when the context falsifies a clause. The rule repairs the context by adding a literal from the falsified instance of the clause (i.e., one of the literals $L_1\sigma, \cdots, L_n\sigma$) in order to satisfy it. However, some of the literals are permanently falsified by the context[7]. The clause composed of the other literals is called the *remainder*. If the remainder is $L_{m+1}\sigma \vee \cdots \vee L_n\sigma$, the left side of the Split rule tries to add $L\sigma$, a literal selected from the remainder, to Λ. The right side adds the Skolemized version of $\overline{L\sigma}$ when the derivation along the left side has been closed. The addition of the new literals should not make the context contradictory. The Split rule is the corresponding rule for DPLL's 'guess' or 'choice' rule.

The Close rule detects the case when the current context in the derivation permanently falsifies a clause. This case implies that there is no possible repair of the context. After the application of Close rule, the calculus tries to backtrack to the recent application of Split rule and continues the derivation with the right side of that application of Split. If there are no Split applications to backtrack, then all the branches are closed and the input theory is unsatisfiable.

Consider the clause $C \vee L$ where C is a (possibly empty) sub-clause and L is a literal. When the current context permanently falsifies the sub-clause C with a context unifier σ, $L\sigma$ must be valid in order to satisfy the clause. The Assert rule detects this situation and adds the literal $L\sigma$ to the context. Its precondition checks that $L\sigma$ is parameter-free (i.e., universal). This makes the application of Assert nonretractable. In the special case when C is an empty clause, the Assert rule adds the unit clause L to the context[8].

The ME calculus have other rules for a fair lifting of DPLL procedure to the first-order level. The Subsume and Resolve rules simplify the input clause set. The Compact rule simplifies the context.

---

[6] This condition is necessary but not sufficient for the unsatisfiability of C. See [13] for the necessary and sufficient condition.

[7] for all $i = 1, \cdots, m, (Params(K_i))\sigma \subseteq P$.

[8] A context permanently falsifies an empty clause with the context unifier which is an empty substitution.

Although Darwin's proof procedure is refutationally complete, it is not guaranteed to find a finite model, if one exists [14]. However, Darwin is a decider for Bernays-Schönfinkel formulas [13]. The clausal form of these formulas has no functions.

2.2 Paradox

If there is an interpretation which satisfies a first-order theory, one can easily construct another interpretation from a new domain and a bijection from the domain of the first interpretation to the new one. The constructed interpretation is also a model of the theory [15]. This fact implies that the actual members of the domain is not important in forming a model. An arbitrary set with a sufficient number of elements can be used as a domain since only the size of the domain matters.

A Mace-style finite model builder [16] searches for a model with a finite domain by enumerating the finite domains with sizes in increasing order like $\{0\}, \{0,1\}, \{0,1,2\}, \cdots$ and so on. The builder checks for each finite domain if there exists a model. The domain sizes increase in order until a model is found. The actual search for a model is achieved by instantiating the input clauses for each domain and running a SAT solver on the propositional theory.

The builder handles equality while instantiating the clauses. The equality predicate has a fixed interpretation in finite model building. The model should assign the literal $d_1 = d_2$ true only when it assigns the terms d_1 and d_2 the same domain element. For the other cases it should be false. The builder can eliminate or simplify a clause instance having equality literals.

Paradox is a Mace-style finite model builder [15]. It has many optimizations that enhance the original Mace-style finite model building method. In order to reduce the number of instances of a clause, clause splitting technique is used. It reduces the number of variables in a clause. The SAT solver used has the incremental solving capability. It does not start from scratch for every domain size. The conflict clauses that have been learned during an unsuccessful search for a model in previous iterations can be reused for the next domain sizes.

Many symmetric models in finite model building makes the job of SAT solver harder since it considers them as different models. A model assigns each constant a domain element. Any permutation of this assignment will generate symmetric models. Function interpretations can also cause symmetric models in the same way. Paradox adds some extra clauses to the propositional theory in order to reduce symmetries.

Paradox also uses sort inference as another optimization. Multiple sorts form a typed first-order logic. The quantification of the variables in a clause should be over its assigned sort. There may be less instances of the clauses respecting sorts compared to the unsorted case. This will reduce the instantiation time and ease model finding. Same domain elements can be used for interpreting functions and predicates that are of different sorts. This will decrease the size of the model. Paradox tries to find sorts of the problem by analyzing its first-order logic encoding. It uses the fact that same variables in a clause should be of the

same sort. Also, the arguments of the equality symbol should be of the same sort.

Paradox is complete for finding models with finite domains, but not refutationally complete in general. However, it is complete for first-order input in Effectively Propositional (EPR) class [15]. EPR class problems have no functions of arity greater than 0. The number of the elements in the largest sort inferred for an EPR problem is the limit for its satisfiability. There can be no models with a domain of size greater than this limit [15]. If no model can be found of size up to that number, Paradox decides that the input is unsatisfiable.

2.3 FM-Darwin

FM-Darwin[9] is also a Mace-style finite model builder [17]. Paradox transforms the problem to a propositional satisfiability problem, however FM-Darwin transforms it to a satisfiability problem of function-free clause logic without equality. The ME calculus and Darwin can decide that class of first-order logic [13]. FM-Darwin uses Darwin as an engine to solve the transformed satisfiability problem. Note that FM-Darwin implements the Mace-style finite model building without grounding, since Darwin can work on the first-order level. FM-Darwin has optimizations similar to those in Paradox for reducing symmetries.

3 Lifting SAT-based ASP to First-Order Level

The theory behind SAT-based answer set solvers is the relationship between the completion semantics and the answer set semantics. When they coincide the models of the completed program are equal to the answer sets of the program. A condition for the equivalence of these semantics have been set first by Fages' theorem [8]. This syntactic condition, which is called tightness in [18], is extended to tightness on a set of literals [9] and to programs with nested expressions [10]. For non-tight programs since the two semantics do not coincide, there can be some models of the completion that are not answer sets.

A logic program with variables can be completed. We can run a model generation theorem prover on the completed first-order theory to find the models of the program's completion. In this way, for tight programs the flow in SAT-based ASP can be lifted to the first-order level.

Let the logic program $\mathcal{P} = \{q(2)., r(a,1)., r(b,2)., p(X) \leftarrow r(X,Y), not\ q(Y).\}$. $\mathcal{P}$ is tight and has only one answer set which is $\{q(2), r(a,1), r(b,2), p(a)\}$. After completing $\mathcal{P}$, the first-order theory $\mathcal{C}$ is found as:

$$\begin{aligned}
&\forall x \quad (q(x) \Leftrightarrow (x = 2)),\\
&\forall xy \quad (r(x,y) \Leftrightarrow ((x = a \wedge y = 1) \vee (x = b \wedge y = 2))),\\
&\forall x \quad (p(x) \Leftrightarrow \exists y\ (r(x,y) \wedge \neg q(y)))\ .
\end{aligned}$$

[9] FM-Darwin comes embedded in Darwin. It can be used by running Darwin with some command line arguments.

A prover should be capable of handling equality in order to reason on the theory $\mathcal{C}$. Although equality reasoning can be embedded to a prover by adding the necessary axioms of equality predicate, it is efficient to have dedicated equality inference rules. Unfortunately, Darwin does not have efficient equality reasoning yet[10]. However, Darwin has a command line option for enabling axiomatic equality reasoning. The equality reasoning is relatively easy for finite model builders since the equality predicate has a fixed interpretation (see Section 2.2).

The first-order formulas in $\mathcal{C}$ can easily be input to Darwin, Paradox, and FM-Darwin in TPTP format[11]. However, an input with arbitrary formulas should be clausified first. While Darwin and FM-Darwin uses eprover[12] as a clausifier, Paradox can do the clausification itself.

When we run the finite model builders Paradox and FM-Darwin on $\mathcal{C}$, they generate a model but it is counterintuitive. The model found with a finite domain of size 1 assigns every constant to the same and only domain element. This is not intended in ASP since unique names are assumed. Let the unique names axioms $\mathcal{U} = \{a \neq b,\ a \neq 1,\ \cdots\}$. The model of $\mathcal{C} \cup \mathcal{U}$ found by Paradox (and also FM-Darwin) has a finite domain $\{1',\ 2',\ 3',\ 4'\}$. This model coincides with the unique answer set of the program $\mathcal{P}$ and defined as

$$
\begin{array}{l}
1 = 1' \\
2 = 2' \\
a = 3' \\
b = 4'
\end{array}
\qquad
\begin{array}{c|cccc}
p & 1' & 2' & 3' & 4' \\
\hline
 & F & F & T & F
\end{array}
\qquad
\begin{array}{c|cccc}
q & 1' & 2' & 3' & 4' \\
\hline
 & F & T & F & F
\end{array}
\qquad
\begin{array}{c|cccc}
r & 1' & 2' & 3' & 4' \\
\hline
1' & F & F & F & F \\
2' & F & F & F & F \\
3' & T & F & F & F \\
4' & F & T & F & F
\end{array}
\quad .
$$

Darwin does not halt for the theory $\mathcal{C} \cup \mathcal{U}$ because of the Skolem function introduced during clausification and the equality reasoning. The existential variable in the last formula of $\mathcal{C}$ cause the introduction of a Skolem function f during clausification. The substitution axiom $f(x) = f(y) \leftarrow x = y$ is added by Darwin for equality reasoning. When an equality like $f(a) = 1$ is in the Herbrand model of the theory, the substitution axiom endlessly generates new equations that make the Herbrand model infinite. For the theory $\mathcal{C} \cup \mathcal{U}$, Darwin generates the equations $\{f(a) = 1,\ f(f(a)) = f(1),\ f(f(f(a))) = f(f(1)),\ \cdots\}$ endlessly and can not output a model.

An n-ary Skolem function can be eliminated by introducing an $n + 1$-ary predicate. The unary Skolem function f introduced for the existential variable in the last formula of $\mathcal{C}$ can be eliminated by a binary predicate F. Here are the implicative clauses $\mathcal{D}$ generated after clausifying the theory $\mathcal{C}$ and eliminating

[10] Although the ME calculus with equality has been defined in [19], it has not been implemented in Darwin yet.

[11] http://www.cs.miami.edu/~tptp/TPTP/SyntaxBNF.html

[12] http://www.eprover.org/

the Skolem functions.

$$
\begin{array}{ll}
q(2). & r(a,1). \\
x = 2 \leftarrow q(x). & r(b,2). \\
 & r_1(x,y) \vee r_2(x,y) \leftarrow r(x,y). \\
 & x = a \leftarrow r_1(x,y). \\
p(x) \leftarrow r(x,y),\ \neg q(y). & y = 1 \leftarrow r_1(x,y). \\
r(x,y) \leftarrow p(x),\ F(x,y). & x = b \leftarrow r_2(x,y). \\
\neg q(y) \leftarrow p(x),\ F(x,y). & y = 2 \leftarrow r_2(x,y).
\end{array}
$$

In order to define the new predicate F, we need to add the following axioms $\mathcal{T}$ which state its totality over the set of all constants appearing in the theory and its uniqueness in the last argument.

$$
\begin{array}{l}
F(x,a) \vee F(x,b) \vee F(x,1) \vee F(x,2). \\
y = y' \leftarrow F(x,y),\ F(x,y').
\end{array}
$$

The clausification used in $\mathcal{D}$ is similar to the naive clausification process which converts arbitrary formulas to formulas in clausal normal form. However the naive clausification can generate exponential number of clauses when a predicate in the logic program has many generating rules having many literals in the premises. The new literals r_1 and r_2 are introduced during the clausification for keeping the size of the clauses polynomial. Similarly, Cmodels also introduces new propositional symbols for the rule bodies for polynomial-spaced clausification [20].

Darwin can find a model for the theory $\mathcal{D} \cup \mathcal{U} \cup \mathcal{T}$ which corresponds to the unique answer set of the program $\mathcal{P}$. The found model represented as disjunction of implicit generalizations (DIG) is

$$
\begin{array}{ll}
x = x & q(2) \\
F(a,1) & r(a,1) \\
F(x,2) & r(b,2) \\
\quad \text{with exception: } F(a,2) & r_1(a,1) \\
p(a) & r_2(b,2)
\end{array}
\quad .
$$

Let the logic program $\mathcal{N} = \{p(a).,\ k \leftarrow not\ p(X).,\ \leftarrow not\ k.\}$. Although $\mathcal{N}$ has no answer sets, the non-clausal theory formed after completion $\mathcal{NC} = \{\forall x\,(p(x) \Leftrightarrow (x = a)),\ k \Leftrightarrow \exists y\,(\neg p(y)),\ false \Leftarrow \neg k\}$ has a non-Herbrand model. $\{k,\ p(1'),\ \neg p(2')\}$ is a model of $\mathcal{NC}$ with a domain $\{1',\ 2'\}$. It is reasonable to expect that the model generation theorem provers find the unsatisfiability of the theory $\mathcal{NC}$, however they can output the mentioned non-Herbrand model. The domain of the Skolem function introduced for the existentially quantified variable in $\mathcal{NC}$ is not restricted to $\{a\}$, which is the set of all constants in the program $\mathcal{N}$. In order to force the provers to search for Herbrand models, we should add the domain closure axiom to the theory. For $\mathcal{N}$, the domain closure axiom $\mathcal{DC} = \{\forall x\ \ (x = a)\}$. The theory $\mathcal{NC} \cup \mathcal{DC}$ has no models.

There is no need to add a domain closure axiom to a theory formed by eliminating Skolem functions as explained in this section. The totality axioms

added for the introduced predicates actually restrict the domains of the Skolem functions to the set of all constants appear in the program. For example, there is no need to add a domain closure axiom to the theory $\mathcal{D} \cup \mathcal{U} \cup \mathcal{T}$.

The theories formed by completing the *domain-restricted programs*[13], whether they are in non-clausal or clausal form, satisfy the domain closure axiom implicitly. All of the rules of a domain-restricted program are domain-restricted (i.e., every variable appearing in a rule also appears in a positive domain literal in the body). Lparse[14] uses domain predicates to restrict the domains of variables and requires domain-restricted programs. Since the domain of every variable in a domain-restricted program is restricted to a subset of all constants appearing in the program, the need for adding the domain closure axiom to the theory formed by completing such a program is eliminated. Note that the second rule of the program $\mathcal{N}$ is not domain-restricted. All the logic programs used in Section 4 are domain-restricted.

4 Experiments

We have carried out the following experiments on the blocks world planning problem and the quasigroup existence problem. All the tests in this section have been performed on an 1.8Ghz Core2Duo machine with 2GB of RAM running Linux 2.6.20.

4.1 Blocks World Problem

The logic program used here is due to Ilkka Niemelä. It has been shown in [9] that the models of its completion correspond to its answer sets.

The encoding of the blocks world problem is suitable for sort inference optimization. Both FM-Darwin and Paradox have sort inference optimization (see Section 2.2). Time values and block names (including table) constitute important concepts of the problem. These concepts can be considered as different sorts. The constants in different sorts can share the same domain elements in the finite model. This will help searching for a model since the size of its finite domain is less than the size of the unsorted one.

In order to take advantage of sort optimization we need to separate the unique names axioms for each sort (i.e., the axioms $\{a \neq table, a \neq b, \cdots\} \cup \{0 \neq 1, 0 \neq 2, \cdots\}$ where inequalities like $a \neq 0$ are eliminated). Otherwise, finite model builders will infer only one sort. The totality axioms of the predicates that are introduced to eliminate the Skolem functions should also respect the sort information. Theses axioms will not be over all the constants appearing in the theory, but over the elements of the sort of the existential variable for which the Skolem function is introduced.

The experiments use 4 blocks world problem instances: an instance encoding of Susman anomaly case (bw1) with 3 blocks, instances that correspond to the

[13] See Section 4.5 of Lparse manual http://www.tcs.hut.fi/Software/smodels/lparse.ps
[14] http://www.tcs.hut.fi/Software/smodels/

problems PLA026+1 (bw2) and PLA025+1 (bw3) from TPTP library with 5 and 9 blocks respectively, and the large.c instance (bw4) from [2] with 15 blocks. They have plans of 3, 3, 4 and 8 steps respectively.

Table 1. The run times of Smodels, Cmodels and Clasp on Blocks World problems. The entries are CPU times in seconds, '–' means that the solver could not output a result within a time limit of 600 seconds. The 'Grounding' column gives the CPU times of grounding the problem instances by Lparse.

Problem	Grounding	Smodels	Cmodels	Clasp
bw1	0.02	0.01	0.01	0.01
bw2	0.02	0.02	0.02	0.01
bw3	0.1	0.1	0.1	0.04
bw4	0.4	1.5	0.5	0.3

Table 1 shows the run times of Smodels (version 2.32), Cmodels (version 3.70) and Clasp (version 1.01) on the blocks world problems. The time values are CPU times in seconds that correspond to the sum of user and system time values given by the shell's time command[15]. All the solvers have solved the problems instantly even with the addition of grounding times of Lparse (version 1.0.17).

Table 2. The run times of FM-Darwin, Paradox and Darwin on Blocks World problems. The entries are CPU times in seconds, '–' means that the prover did not halt within a time limit of 600 seconds. The 'non-clausal' column is for non-clausal first-order theory formed by completing the logic program. Both 'clausal' and 'no Skolem' columns are for the completed and clausified theories, but the Skolem functions introduced during clausification are eliminated in the latter one.

	FM-Darwin			Paradox			Darwin
Prb.	non-clausal	clausal	no Skolem	non-clausal	clausal	no Skolem	no Skolem
bw1	3.1	0.3	0.4	0.1	0.2	0.2	0.6
bw2	22.4	7.7	7.7	0.4	0.5	0.4	7.4
bw3	–	–	–	8.1	9.8	2.9	–
bw4	–	–	–	–	–	109.5	–

Table 2 shows the run times of FM-Darwin (version 1.4), Paradox (version 2.1) and Darwin (version 1.4) on these problems. We have completed the logic program with variables manually. The results of the corresponding non-clausal first-order theories are shown in the *non-clausal* columns of Table 2. The *clausal*

[15] Since the evaluation machine has a multi-core CPU, we used the sum of user and system time values instead of real time value. The real time may be less than the user time since the programs of shell pipes in the test scripts and other programs like eprover called within a prover may run on different cores in parallel.

columns are representing the results of the inputs that are formed by clausification of the completed theory. They have Skolem functions introduced during the clausification. We have also eliminated these Skolem functions (see Section 3 for the elimination method) and obtained another input. The *no Skolem* columns are for results of this type of input. The run times of Darwin are only for the *no Skolem* column. For the others, because of the Skolem functions and the equality reasoning, Darwin may not halt and try to generate an infinite model.

If we take the program $\mathcal{P}$ in Section 3 as an example, here are the theories which the columns in Table 2 represents. The non-clausal column represents the theory $\mathcal{C} \cup \mathcal{U}$. Here is the theory $\mathcal{S}$ which is formed after clausifying $\mathcal{C}$.

$$
\begin{array}{ll}
q(2). & r(a,1). \\
x = 2 \leftarrow q(x). & r(b,2). \\
 & r_1(x,y) \vee r_2(x,y) \leftarrow r(x,y). \\
 & x = a \leftarrow r_1(x,y). \\
p(x) \leftarrow r(x,y),\ \neg q(y). & y = 1 \leftarrow r_1(x,y). \\
r(x, f(x)) \leftarrow p(x). & x = b \leftarrow r_2(x,y). \\
\neg q(f(x)) \leftarrow p(x). & y = 2 \leftarrow r_2(x,y).
\end{array}
$$

The theory $\mathcal{S}$ is same as $\mathcal{D}$ except that the Skolem function f is not eliminated. The clausal column represents the theory $\mathcal{S} \cup \mathcal{U}$. The no Skolem column represents the theory $\mathcal{D} \cup \mathcal{U} \cup \mathcal{T}$ for the program $\mathcal{P}$.

Darwin (FM-Darwin as well) uses eprover as a clausifier for the non-clausal inputs. The results in the non-clausal and clausal columns of FM-Darwin in Table 2 show that the clausification used in our work, explained in Section 3, leads to better results compared to the results obtained from eprover. The corresponding analysis for Paradox show that the clausification done by Paradox leads to slightly better performance compared to the one used in our work. However, all the results in Table 2 show that all the model generation theorem provers tested do not scale up well for the blocks world problems whether it is non-clausal, clausal with Skolem functions, or clausal without Skolem functions.

Both Darwin and FM-Darwin have displayed weak performance in the experiments. [14] states that Darwin is weak for problems with equality. The axiomatic equality reasoning can be the bottleneck for Darwin in the results shown in Table 2. FM-Darwin transforms the problem to a satisfiability problem over function-free clause logic without equality. It uses Darwin as an engine to build a finite model. Since the transformation eliminates equality, only the inefficient equality reasoning can not explain the weak results. The transformation used in FM-Darwin introduces new predicate symbols and add new clauses for the totality constraints needed over the finite domain. These increase the size of the theory and the search space which, in turn, hardens the job of Darwin.

4.2 Quasigroup Existence Problem

We have also done experiments on the quasigroup existence (QG) problem in algebra. We have selected the QG5 variation. QG problems impose constraints

on a relation whose multiplication table is a order n Latin square The QG5 problem's constraints on the relation $\circ$ are $a \circ a = a$ and $a \circ b \circ a \circ a = b$ for all a and b in a order n Latin square defined for $\circ$. The QG problem is selected since the rules in the logic program representation has many instances [21]. The grounding step of ASP can generate a huge propositional program which makes the job of answer set solvers hard. Since the model generation theorem provers accept first-order theories, they may show better performance relatively.

Here is the tight[16] logic program in Lparse format that is used to encode the QG5 problem.

```
range(1..n).

val(X,Y,Z) :- not nval(X,Y,Z), range(X;Y;Z).
:- val(X,Y,Z), val(X,Y,Z1), Z != Z1, range(X;Y;Z;Z1).

hasval(X,Y) :- val(X,Y,Z), range(X;Y;Z).
:- not hasval(X,Y), range(X;Y).

nval(X,Y1,Z) :- val(X,Y,Z), Y != Y1, range(X;Y;Z;Y1).
nval(X1,Y,Z) :- val(X,Y,Z), X != X1, range(X;Y;Z;X1).

:- val(X,X,X1), X != X1, range(X;X1).
:- val(X,Y,Z), val(Z,X,T), val(T,X,Y1), Y != Y1, range(X;Y;Z;T;Y1).
```

Table 3. The run times of Smodels, Cmodels and Clasp on QG5 problem. The entries are CPU times in seconds, '–' means that the solver could not output a result within a time limit of 600 seconds. The structure of the table is analog to Table 1.

Order	Grounding	Smodels	Cmodels	Clasp
5	0.03	0.03	0.02	0.02
6	0.1	0.2	0.1	0.1
7	0.1	0.2	0.2	0.1
8	0.2	4.3	0.3	0.3
9	0.4	–	540.1	274.4
10	0.7	–	–	–
11	1.2	–	24.4	8.8

Similar to the blocks world problem we have 3 types of input for every instance of the QG5 problem: The first-order completion of the logic program, its clausal form with Skolem functions, and its clausal form after eliminating the Skolem functions. We have tested the QG5 problem instances of order 5 to 11. For orders 5,7,8 and 11 there are QGs satisfying the QG5 constraints. The instances of order 6,9 and 10 are unsatisfiable.

[16] The tightness condition [9] is satisfied by the existence of a function λ which assigns every range atom to 0, every val atom to 1 and every nval and hasval atoms to 2.

Table 3 shows the run times of Smodels, Cmodels and Clasp on the QG5 problem instances. Smodels could not solved the instances of order 9,10 and 11 within a time limit of 600 seconds. Clasp has better results than Cmodels.

Table 4. The run times of FM-Darwin, Paradox and Darwin on QG5 problem. The entries are CPU times in seconds, '–' means that the prover did not halt within a time limit of 600 seconds. The structure of the table is analog to Table 2.

	FM-Darwin			Paradox			Darwin
Ord.	non-clausal	clausal	no Skolem	non-clausal	clausal	no Skolem	no Skolem
5	0.3	0.2	0.3	0.1	0.1	0.1	0.4
6	–	–	4.6	–	–	0.1	10.3
7	93.6	78.7	37.6	0.1	0.2	0.3	98.6
8	–	–	–	0.2	0.4	0.5	–
9	–	–	–	–	–	567.4	–
10	–	–	–	–	–	–	–
11	–	–	–	3.0	6.7	6.5	–

The run time results of Paradox in Table 4 are similar to those of Cmodels and Clasp in Table 3 for the small and satisfiable instances of the QG5 problem. However, these results show that Paradox seems to perform better for the satisfiable instances as the order increase, especially with the non-clausal encoding of the problem. The rules of the logic program for this problem has many instances. The results in Table 3 show that the propositional instance of the logic program formed by the grounder becomes harder for the solver as the order increases (especially the results for Smodels). Note that Paradox also performs grounding before calling the SAT solver but it is done incrementally. It calls the SAT solver for every finite domain starting with the one of size 1 and increase incrementally. It is interesting to compare Cmodels and Paradox since Cmodels is a SAT-based answer set solver. The order of grounding and completion steps is interchanged in Paradox and Cmodels. The completed first-order theory is given to Paradox, but in the case of Cmodels the input is the grounded logic program. Paradox performs the grounding itself, while Cmodels performs the completion. The incremental nature of finite model building and the other optimizations of Paradox seem to be efficient for this problem.

The clausal theory formed after eliminating the Skolem functions has a special importance for the finite model builders. When the input is in this form, it is in the EPR problem class. A finite model builder can be refutationally complete for this class [15]. Paradox have found the unsatisfiability of the input for the problem instances of order 6 and 9 (i.e., no quasigroup exists for them). Similarly, FM-Darwin found the unsatisfiability result for the instance of order 6 as depicted in the *no Skolem* columns of the Table 4. Note that in the columns *non-clausal* and *clausal*, where the input is not in the EPR class, both FM-Darwin and Paradox could not halt for the unsatisfiable instances (orders 6,9 and 10).

The results depicted in Tables 4 show that both FM-Darwin and Darwin have weak performances for this problem. The comments that have been stated in the case of blocks world problem also apply to this problem.

5 Conclusions and Related Work

We proposed and experimented a way of lifting SAT-based ASP to the first-order level for tight programs. The completion step in the lifted case produces a first-order theory. The place of the SAT solver should be filled by a system which can process first-order input and output a model when it is satisfiable. The model generation theorem provers can be suitable to play this role. One important outcome of this lifting is that grounding can be either eliminated or done intelligently in the model generation prover. Model generation theorem provers are gaining increasing popularity in the automated reasoning community. We have used Darwin, FM-Darwin and Paradox in the experiments.

The experiments with Paradox showed good results for the quasigroup existence problem. Darwin and FM-Darwin displayed weak performance for our benchmark problems. Axiomatic equality reasoning, which is used by Darwin, is inefficient. The equality literals generated by completion and the unique names axioms increase the importance of equality reasoning.

The ME calculus is a lifting of the DPLL procedure to the first-order level. Note that the search procedures of successful answer set solvers, like smodels and dlv, are based on the DPLL procedure on the propositional level. We think that the ME calculus and its implementation Darwin are interesting for the future answer set solvers working on the first-order level.

Paradox and FM-Darwin are refutationally complete for theories in the EPR class. The clausal theories formed in our work by eliminating the Skolem functions are in the EPR class. The relation between the EPR formulas and the logic programs under the answer set semantics has been examined in [22] for an application area of textual inference. It has a translation from any EPR formula to a logic program such that the formed logic program has an answer set iff the EPR formula is satisfiable.

Recently, a definition of the stable models of an arbitrary first-order formula has been given in [23]. That work also extends the tightness condition to first-order formulas. Using that definition, it can be decided whether a logic program is tight or not without referring to the original tightness condition. The tightness condition based on Fages' theorem refers to grounding, but the new one does not.

One limitation of the provers used is that they can only output one model for an input. The modification of Darwin needed for finding all models should be easy. After a model is found, it can backtrack to the last application of the Split rule and continue with the right side of that rule. The symmetric models that correspond to the same answer set should be eliminated when we are searching for all models using a finite model builder.

We have concentrated on Mace-style finite model building. One future work can be examining the Sem-style finite model builders like Sem[17] and Finder[18]. A Sem-style finite model builder searches for a model by working on the input directly. There is no translation of the problem into a satisfiability problem [15]. During the lifting of SAT-based ASP, we formed the completed theory and added the necessary axioms manually. Automating this process is another subject of future work.

Acknowledgments

This work is supported by the Scientific and Technical Research Council of Turkey, Project No: EEEAG 105E068. We are grateful to Esra Erdem, Vladimir Lifschitz, Ilkka Niemelä, Tomi Janhunen and anonymous referees for useful comments and discussions related to the subject of this paper.

References

1. Lifschitz, V.: Answer set planning. In Proceedings of the 16th International Conference on Logic Programming (1999) 23–37
2. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25** (1999) 241–273
3. Marek, V., Truszczynski, M.: Stable models and an alternative logic programming paradigm. In the Logic Programming Paradigm: a 25-Year Perspective (1999) 375–398
4. Anger, C., Konczak, K., Linke, T., Schaub, T.: A glimpse of answer set programming. Künstliche Intelligenz **19** (2005) 12–17
5. Baumgartner, P., Furbach, U., Yahya, A.: Automated reasoning, knowledge representation and management. Künstliche Intelligenz **19** (2005) 5–11
6. Clark, K.: Negation as failure. In Logic and Data Bases (1978) 293–322
7. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–386
8. Fages, F.: Consistency of Clark's completion and existence of stable models. Journal of Methods of Logic in Computer Science **1** (1994) 51–60
9. Babovich, Y., Erdem, E., Lifschitz, V.: Fages' theorem and answer set programming. In Proceedings of 8th International Workshop on Non-Monotonic Reasoning (2000)
10. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming **3** (2003) 499–518
11. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157** (2004) 115–137
12. Lierler, Y., Maratea, M.: Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In Proceedings of the 7th International Conference on Logic Programming and Non-Monotonic Reasoning (2004) 346–350

[17] http://www.cs.uiowa.edu/~hzhang/sem.html
[18] http://users.rsise.anu.edu.au/~jks/finder.html

13. Baumgartner, P., Tinelli, C.: The Model Evolution Calculus. Fachberichte Informatik 1–2003, Universität Koblenz-Landau, Institut für Informatik (2003)
14. Baumgartner, P., Fuchs, A., Tinelli, C.: Implementing the Model Evolution Calculus. International Journal of Artificial Intelligence Tools **15** (2006) 21–52
15. Claessen, K., Sörensson, N.: New Techniques that Improve MACE-style Finite Model Finding. In Proceedings of Workshop: Model Computation – Principles, Algorithms, Applications at the 19th International Conference on Automated Deduction (2003)
16. McCune, W.: A Davis-Putnam Program and Its Application to Finite First-Order Model Search: Quasigroup Existence Problems. Technical Report ANL/MCS-TM-194, Argonne National Laboratory (1994)
17. Baumgartner, P., Fuchs, A., de Nivelle, H., Tinelli, C.: Computing Finite Models by Reduction to Function-Free Clause Logic. In Proceedings of 3rd Workshop on Disproving - Non-Theorems, Non-Validity, Non-Provability at the 3rd International Joint Conference on Automated Reasoning (2006)
18. Lifschitz, V.: Foundations of logic programming. Principles of Knowledge Representation (1996) 69–127
19. Baumgartner, P., Tinelli, C.: The Model Evolution Calculus with Equality. In Proceedings of the 20th International Conference on Automated Deduction (2005)
20. Babovich, Y., Lifschitz, V.: Computing answer sets using program completion. Unpublished draft (2003)
21. Shirai, Y., Hasegawa, R.: Answer Set Computation Based on a Minimal Model Generation Theorem Prover. In Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence (2004) 43–52
22. Lierler, Y., Lifschitz, V.: Logic Programs vs. First-Order Formulas in Textual Inference. Unpublished draft
23. Ferraris, P., Lee, J., Lifschitz, V.: A New Perspective on Stable Models. In Proceedings of the 20th International Joint Conference on Artificial Intelligence (2007) 372–379