Unfounded Sets and Autarkies

John S. Schlipf[1] and Ryan Flannery[1⋆]

University of Cincinnati, Dept. of ECE&CS, Cincinnati, OH 45221-0030, USA
John.Schlipf@Uc.Edu, ryan.flannery@gmail.com

Abstract. We identify a new relationship between classical and non-monotonic logic; specifically, we relate autarkies for CNF formulas [Monien and Speckenmeier] to unfounded sets for logic normal logic programs [Van Gelder, Ross, Schlipf]. Using a natural way to construct a logic program P_C from a CNF formula C, we show that that the all-negative autarkies for C are exactly the unfounded sets of P_C.
In CNF satisfiability there is no preference for setting variables true or false. Thus polarities of some variables may be flipped in a CNF formula C, giving different logic programs P_C and thus different unfounded sets. Seeking autarkies has been pursued in CNF satisfiability engines; we were inspired by Sean Weaver's adding of "safe" constraints: new constraints known not to destroy consistency. By using unfounded sets as above, we have a different, and generally more aggressive, way of seeking autarkies. Our hope is that this approach can also give a way to use failed stochastic local searches for satisfying truth assignments to find autarkies.
Keywords: satisfiability, CNF, autark, minimal model, unfounded set, and safe constraint.

1 Introduction

There has been much development in recent years in both CNF Satisfiability (SAT) solvers and Answer Set Programming (ASP) Solvers, which are extensions of stable model solvers. The currently fastest solvers for both use Davis-Putnam-Loveland-Logeman (DPLL) style backtracking searches [1]. SAT solving is the more developed field, and ASP solvers have borrowed from SAT solvers. Here we identify a previously unidentified similarity: a very close connection between the *autarkies* of [4] and the *unfounded sets* of [6].

For CNF formulas, autarkies [4], when they exist — and can be found efficiently — provide an efficient way to simplify CNF satisfiability problems: all literals in an autarky may be set to true, simplifying the SAT problem without affecting satisfiability. Below we use ideas from logic programming motivate further ways to think about autarkies; we anticipate that this may lead to ways to simplify CNF satisfiability computation.

Some of the ideas for this paper arose from conversations with to John Franco, Victor Marek, Mark Vanfleet, Allen Van Gelder, and Sean Weaver. Some recent related results can be found in [3, 7].

⋆ Ryan Flannery's research and the purchase of computers for experiments was supported by Ohio Board of Regents.

2 Background and Vocabulary

Our vocabulary is fairly standard except that we focus on partial truth assignments, which we shall refer to as "interpretations." A CNF formula is a conjunction of *clauses*, formulas of the form $(a_1 \vee \cdots \vee a_j \vee \neg b_1 \vee \cdots \vee \neg b_k)$, where the a_i's and b_i's are all atoms (*i.e.*, proposition letters). (As usual in satisfiability theory, we shall ignore the order of conjuncts and disjuncts, but, following logic programming tradition, we shall still write them as if they were ordered.)

An *interpretation* is a set of literals not containing any contradictory literals $a, \neg a$. We think of the interpretation $I = \{a_1, \neg a_2, a_4\}$ as a partial function: a_1 and a_4 are assigned to *true*, a_2 is assigned to *false*, and all other other atoms are unassigned. Extending the functional language, we shall say that the *domain* of I is the set of atoms occurring, positively or negatively, in I. An interpretation I is *total* for a CNF formula $\mathcal{C}$ if every atom occurring in $\mathcal{C}$ is in the domain of I. Interpretation I *hits* a clause $c = (a_1 \vee \cdots \vee a_j \vee \neg b_1 \vee \cdots \vee \neg b_k)$ some a_i or b_i is in the domain of I; I *satisfies* c if some literal in c is in I; and I *falsifies* c if the negation of every literal in c is in I. I *satisfies* a CNF formula $\mathcal{C}$ if it satisfies every clause in $\mathcal{C}$, and it *falsifies* c if it falsifies some clause in $\mathcal{C}$.

Definition 1. *An* autarky *for a CNF formula $\mathcal{C}$ is an interpretation A that satisfies every clause it hits.*

For example, for

$$\mathcal{C} = (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee d) \wedge (\neg c \vee \neg d), \tag{1}$$

$A = \{a, b\}$ is an autarky: A satisfies the first two clauses, and variables a, b do not occur in the third clause. For any CNF formula $\mathcal{C}$, $\emptyset$ is an autarky for $\mathcal{C}$, and the total autarkies for $\mathcal{C}$ are just the satisfying assignments for $\mathcal{C}$.

Given formula $\mathcal{C}$ and interpretation I, we can form the *residual* formula $\mathcal{C}_I$ by replacing, in $\mathcal{C}$, each atom in the domain of I with its truth value, simplifying each clause, and deleting *true* clauses. For example, formula (1), with $I = \{a, b\}$, substitute to form $(\mathit{true} \vee \neg \mathit{false} \vee c) \wedge (\neg \mathit{true} \vee \mathit{false} \vee d) \wedge (\neg c \vee \neg d)$ and then simplify to $(\neg c \vee \neg d)$.

For interpretations A, I, define $\mathrm{subst}(A, I) = A \cup \{a \in I : \neg a \notin A\} \cup \{\neg a \in I : a \notin A\}$. That is, if variable a is in the domain of A, use its "truth value" in A, and otherwise use its truth value in I.

Theorem 1. [4]. *Suppose A is an autarky for a CNF formula $\mathcal{C}$. Then $\mathcal{C}$ is satisfiable iff $\mathcal{C}_A$ is satisfiable. Moreover, If interpretation I satisfies $\mathcal{C}$, so does* $\mathrm{subst}(A, I)$*, and if I satisfies $\mathcal{C}_A$, then* $\mathrm{subst}(A, I)$ *satisfies $\mathcal{C}$.* □

A *normal logic program* consists of a set of "normal rules" ρ of the form

$$a \leftarrow b_1, \ldots, b_k, \neg c_1, \ldots, \neg c_m$$

Here, a is called the *head* of the rule and $b_1, \ldots, b_k, \neg c_1, \ldots, \neg c_m$, the *body*.

The *classical interpretation* of such a rule ρ is the clause $(a \vee \neg b_1 \vee \cdots \vee \neg b_k \vee c_1 \vee \cdots \vee c_m)$. However, in addition to the classical interpretation, logic programming has the intuition that no atom a should be true unless it is *caused* to be true by a rule ρ — a rule with a as its head and whose body is satisfied. Exactly what that means is clarified by the logic programming semantics chosen.

Definition 2. [6] *Let $\mathcal{P}$ be a normal logic program and let I be an interpretation. A set U of atoms is* unfounded over I with respect to $\mathcal{P}$ *if, for every rule*

$$a \leftarrow b_1, \ldots, b_k, \neg c_1, \ldots, \neg c_m$$

with head in U, some $c_i \in I$ or some $b_i \in U$.[1]

One intuition here is that, if U is unfounded, then no atom in U could be the first to be proved true, so every atom in U should be inferred to be false.

The most commonly used semantics for normal logic programs is the *stable semantics* [2]. We shall not define it here, instead giving an equivalent characterization. An interpretation I is *total* for a logic program $\mathcal{P}$ if every atom in $\mathcal{P}$ is in the domain of I.

Theorem 2. [5] *Let $\mathcal{P}$ be a normal logic program and I be a total interpretation for $\mathcal{P}$. Then I is a* stable model *of $\mathcal{P}$ if (i) I satisfies the classical interpretation of every rule in $\mathcal{P}$ and (ii) if any set U of atoms is unfounded over I w.r.t. $\mathcal{P}$, $\{\neg b : b \in U\} \subseteq I$.*

3 Autarkies and Unfounded Sets

Definition 3. *Let $\mathcal{C}$ be a CNF formula. Let $\mathcal{P}_\mathcal{C}$ be the normal logic program with the following rules: for each clause $(a_1 \vee \cdots \vee a_m \vee \neg b_1 \vee \cdots \vee \neg b_k)$ of $\mathcal{C}$, $\mathcal{P}_\mathcal{C}$ contains rules*

$$\begin{aligned} a_1 &\leftarrow b_1, \ldots, b_k, \neg a_2, \neg a_3, \ldots, \neg a_m \\ a_2 &\leftarrow b_1, \ldots, b_k, \neg a_1, \neg a_3, \ldots, \neg a_m \\ &\vdots \\ a_m &\leftarrow b_1, \ldots, b_k, \neg a_1, \neg a_2, \ldots, \neg a_{m-1}. \end{aligned}$$

Theorem 3. *Let A be an all-negative interpretation, and let $U = \{a : \neg a \in A\}$. Let $\mathcal{C}$ be a CNF formula. Then A is an autarky for $\mathcal{C}$ if and only if U is unfounded over $\emptyset$ with respect to $\mathcal{P}_\mathcal{C}$.*

Proof. First suppose that A is an autarky and show that U is unfounded over $\emptyset$ w.r.t. $\mathcal{P}_\mathcal{C}$. Let $\rho = a \leftarrow b_1, \ldots, b_k \wedge \neg c_1, \ldots, \neg c_m$ be any rule of $\mathcal{P}_\mathcal{C}$ with head $a \in U$; to show U is unfounded, we show that some $b_i \in U$. Rule ρ is derived from a clause $(a \vee c_1 \vee \cdots \vee c_m \vee \neg b_1 \vee \cdots \vee \neg b_k)$ in $\mathcal{C}$. Since A is an autarky

[1] The original paper added a third alternative, that some $\neg b_i \in I$, but the definition here gives the same semantics.

and hits this clause, it must satisfy this clause, so some $\neg b_i \in A$. So $b_i \in U$, as desired.

Conversely, suppose U is unfounded and suppose A hits a clause $c = (a_1 \vee \cdots \vee a_m \vee \neg b_1 \vee \cdots \vee \neg b_k)$. We must show A satisfies the clause. Since A is all-negative and hits c, some $\neg a_i \in A$ or some $\neg b_j \in A$. In the latter case, the clause is satisfied. Otherwise: For notational convenience, assume $i = 1$. So $\mathcal{P}_\mathcal{C}$ contains rule $a_1 \leftarrow b_1, \ldots, b_k, \neg a_2, \ldots, \neg a_m$. Since $a_i \in U$, and U is unfounded over $\emptyset$, some $b_i \in U$. So $\neg b_i \in A$, and A satisfies clause c. □

4 Logic programming tools for satisfiability?

We generalize a term from [7]: For sets of formulas S, T, we say T is *safe* for S if either S is unsatisfiable or $S \cup T$ is satisfiable. That is, if we replace S with $S \cup$T, we do not change from satisfiable to unsatisfiable. So if we are interested only in testing whether $\mathcal{C}$ is satisfiable, or in finding one satisfying truth assignment, then we can freely replace S with $S \cup T$. The simplest example is the well-known result that, if literals $\{\lambda_1, \ldots, \lambda_k\}$ occur pure in CNF formula $\mathcal{C}$, then $\{\lambda_1, \ldots, \lambda_k\}$ is safe for $\mathcal{C}$. In this language, Theorem 1 says that if T is an autarky for as set $\mathcal{C}$ of clauses, then T is safe for $\mathcal{C}$. Of course, what is significant about autarkies is that we can easily recognize them when we see them — although searching for them may be difficult.

Theorem 3 shows that some forced inferences in the stable semantics for logic programming correspond to safe additional constraints in satisfiability. We start out with the all-negative interpretation for a CNF formula $\mathcal{C}$, form $\mathcal{P}_\mathcal{C}$, and find the greatest unfounded set U for $\emptyset$ w.r.t. $\mathcal{P}_\mathcal{C}$ — getting the greatest all-negative autarky for $\mathcal{C}$. The greatest unfounded set w.r.t. a logic program $\mathcal{P}$ can be constructed in time $O(|\mathcal{P}|^2)$ [6].

Since classical logic — unlike the stable semantics — has no preference for setting variable to *false* over setting them to *true*, we can use essentially the same construction to find — and assume — the greatest autark subinterpretation of any interpretation I for a CNF formula $\mathcal{C}$. The algorithm amounts to just removing from I all literals that can't be in an autarky.

Algorithm 1.

Input: *An interpretation I and a CNF formula $\mathcal{C}$.*
Output: *The greatest autarky $A \subseteq I$.*

Set $A = I$.
Repeat until a fixed point is reached:
  For each clause $c \in \mathcal{C}$:
    if c is not satisfied in A
      for each literal $\lambda \in c$, set $A = A - \{\neg\lambda\}$.

We have experimented using hillclimbing to construct interpretations I satisfying most clauses of a CNF formula and then assuming literals in the greatest autark subinterpretation. So far the results have been fairly poor; more often

than not, the time needed to find the autarkies is greater than the time saved in a later DPLL or stochastic search.

However, using logic programming intuitions, we may be able to identify other safe constraints to add to a CNF formula. One result in this direction is the following, related to the concern for minimal models in logic programming.

Theorem 4. *Let $\mathcal{C}$ be a CNF formula, I be an interpretation for $\mathcal{C}$, and literals $\beta_1, \beta_2, \ldots, \beta_k \in I$. For $1 \leq i \leq k$, let A_i be the greatest autarky $\subseteq I$ for $\mathcal{C}_{\{\neg\beta_i\}}$. Let $T = \{\alpha \vee \beta_i : 1 \leq i \leq k \text{ and } \alpha \in A_i\}$. If $\mathcal{C}$ is satisfiable, so is $\mathcal{C} \cup T$.*

Proof. Without loss of generality, assume I is all-negative (otherwise just reverse the polarities of all variables occurring positively in A).

For assignments (total interpretations) I_1, I_2, let $I_1 \sqsubseteq I_2$ mean that every positive literal in I_1 is also in I_2. Clearly $\sqsubseteq$ is a partial ordering. There are only finitely many assignments for the variables in $\mathcal{C}$, so if $\mathcal{C}$ is satisfiable, it has $\sqsubseteq$-minimal satisfying assignments, called a *minimal models of* $\mathcal{C}$.

We show all minimal models of $\mathcal{C}$ satisfy T. Let i be a minimal model of T, and let $(\alpha \vee \beta_i) \in T$. If I satisfies β_i we are done. Otherwise, clearly, I is minimal model of $\mathcal{C} \wedge \neg\beta_i$. By Theorem 1, α holds in all minimal models of $\mathcal{C} \wedge \neg\beta_i$, so I satisfies $(\alpha \vee \beta_i)$, as desired. □

The importance of the Theorem 4 is that it is safe to assume *all* the formula in T simultaneously. Some such 2-clauses can be found relatively simply:

Algorithm 2. Shortcut algorithm to derive some safe 2-clauses.

For each literal $\lambda \in A$, count how many times $\neg\lambda$ appears in $\mathcal{C}$.
For each literal λ where $\neg\lambda$ appears only once,
Identify the clause containing $\neg\lambda$ — say $(\neg\lambda \vee \kappa_1 \vee \cdots \vee \kappa_m)$
For each κ_i where $\neg\kappa_i \in A$, assert $(\lambda \vee \neg\kappa_i)$.

We continue experimenting with these ideas to identify circumstances when adding such safe constraints results in formulas which are, with significant probability, easier to solve than the original formulas.

References

1. M. Davis, G. Logemann, and D. Loveland. "A machine program for theorem proving." *Communications of the ACM* 5 (1962), 394-397.
2. M. Gelfond and V. Lifschitz. The stable models semantics for logic programming. *Logic Programming: Proceedings of the Fifth International Conference and Symposium* (1988), 1070-1080.
3. V. Marek and M. Trusczcyński. Algorithmic properties of autarkies. Circulated manuscript, 2006.
4. B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n Steps. *Discrete Applied Mathematics* 10, pp. 287-295, 1985.

5. C. Zaniolo and D. Saccà. Stable models and non-determinism in logic programs with negaton. *ACM Principles of Database Systems* (1990), 205-217.
6. A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM* 38(3), pp. 620-650, 1991.
7. S. Weaver, J. Franco, and J. Schlipf. Extending Existential Quantification in Conjunctions or BDDs. *Journal on Satisfiability, Boolean Modeling, and Computation* 1 (2006), 89-110.