A Satisfiability Checker for Difference Logic

Moez Mahfoudh[1], Peter Niebert[2], Eugene Asarin[1] and Oded Maler[1]

[1]VERIMAG, 2 Av. de Vignate, 38610 Gières, France
[Moez.Mahfoudh | Eugene.Asarin | Oded.Maler]@imag.fr

[2]Laboratoire d'Informatique Fondamentale, CMI, 39 rue Joliot-Curie
13453 Marseille Cedex 13, France, niebert@cmi.univ-mrs.fr

Abstract. We present a constraint solver for *difference logic*, a logic whose formulae consist of Boolean combinations of difference inequalities of the form $X - Y \geq c$ where X and Y are numerical variables and c is a constant. Such formulae are encountered in many occasions, most notably in the definition of feasible solutions for scheduling problems and in the verification of timed automata. In both cases they express constraints on the time elapsed between pairs of events.

While constraint solving for such formulae fits into more general frameworks such as of constraint propagation techniques, our aim is to provide a solver specialized for this class of formulae. To this end we take the general scheme of propositional SAT solving and extend it to difference logic.

1 Introduction

There is an increasing awareness in the domain of automatic verification and debugging that problems such that the existence of a behavior that violates a desired property of a system can be formulated as satisfiability problems. Hence, existing satisfiability checking techniques can be used to complement other verification algorithms. For instance, [BCCZ99] report remarkable results in finding faulty traces of finite state systems by translation to Boolean formulae and their resolution with generic Boolean constraint solvers.

Many satisfiability solvers are derived from the Davis-Putnam framework (see e.g. [ZS00]) which consists of two interacting components: 1) Syntactic transformations that simplify the formula toward a point where its (un)satisfiability becomes obvious. As a result of these transformations, the values of certain variables become fixed if they are the values that can lead to satisfaction; 2) A search mechanism in the space of valuations for the remaining variables. The search is conducted by choosing a variable B and "branching" on it, i.e. checking satisfiability of the formulae obtained by replacing B by T and by F, respectively. Re-application of the transformation rules to the resulting formulae may lead to a solution or, otherwise, consecutive branching on another variable should be done.

A way to avoid the inherent exponential blow-up in the size of the search tree was proposed by Stålmarck [SS90,GW99] and turned out to be rather successful for verification applications. This method is optimized for problems allowing a solution by "shallow" reasoning (bounded branching depth) with the help of the *dilemma rule*. This rule allows to join branches induced by eliminating a Boolean variable B (resulting in two sets of clauses): Conclusions drawn from both hypotheses (B or $\neg B$) can safely be added to the set of clauses before the branching. With this kind of non-backtracking branching, certain verification problems with a huge number of variables can be solved at a very low branching depth (in contrast to benchmark problems like pigeonhole problems, where inevitably the branching will essentially include all Boolean variables, making the dilemma rule highly inefficient for such cases).

In this work, we generalize the propositional procedure to a logic which allows, in addition to propositions, *difference* inequalities of the form $X - Y \geq c$ where X and Y are numerical variables (we consider both integer and real domains) and c is a constant. The logic is particularly useful for coding verification problems of timed systems (in fact, algorithms for the state exploration of timed automata [HNSY92] are based on the manipulation of clock difference constraints) and in a separate paper [NMA+02] we translate bounded reachability problems for timed systems into difference logic formulae.

While such formulae fall into the domain of applicability of *constraint propagation* techniques used, for example, in *constraint logic programming* (CLP) [JM94] as well as to that of *mixed integer linear programming* (MILP), we want to construct a specialized prover, taking advantage of the particular structure of difference constraints. Thus, we have explored the generalization of propositional methods to difference logic whose satisfiability problem is NP-complete.

The challenge for such a generalization lies, of course, in the interaction between Boolean and numerical constraints. We work on CNF formulae with clauses containing at most three literals, at most one of which is a numerical constraint. The key idea of our approach is that a conjunction of difference constraints can be represented using a *difference bounds matrix* normalized by the Floyd-Warshall algorithm. This algorithm finds contradictions in the set of numerical clauses and also allows to extract a solution from a consistent set of inequalities in polynomial time (cubic in simple implementations). As a result, the basic method branches only on Boolean variables whereas the role of numerical constraints is to cut branches. Additional heuristics serve to reduce the number of numerical variables in order to decrease the complexity of consistency tests and they can be understood as a "numerical variant" of the Davis-Putnam rule for the elimination of Boolean variables.

We have completed a preliminary implementation of our procedure with a special emphasis on low memory data-structures and an efficient treatment of numerical constraints. Currently, the implementation only treats the integer case and not all heuristics have been implemented yet. We have also established and implemented a translation of reachability problems for timed automata into difference logic and are currently testing the performance of our solver on examples taken from this domain.

The rest of the paper is structured as follows: In Section 2 we fix the formal framework of *difference logic* and sketch a procedure for transforming its formulae into a conjunctive normal form. In Section 3 we recall the framework for Boolean satisfiability checking. In Section 4 we present the proof rules for difference constraints and their integration into the procedure. In Section 5 the overall procedure and tool architecture is explained along with remarks on the first experiments with our running prototype.

2 Difference Logic

In this section we define formally the class of formulae that we consider. This class has already been used elsewhere (see e.g. [MLAH99,LPWY99,ABK+97,BM00]) and is probably part of folklore, but to give it a name let us call it *difference logic* (DL). We use two slightly different variants, DLZ and DLR depending on the domain of the numerical variables.

Definition 1 (Difference Logic). Let $\mathcal{B} = \{B_1, B_2, \dots\}$ be a set of Boolean variables and $\mathcal{X} = \{X_1, X_2, \dots\}$ be a set of numerical variables. The set of atomic formulas of $DL(\mathcal{B}, \mathcal{X})$ consists of the Boolean variables in $\mathcal{B}$ and numerical constraints of the following forms:

$$X_i - X_j \geq c$$

(with $c \in \mathbb{Z}$ for DLZ and $c \in \mathbb{Q}$ for DLR) and

$$X_i - X_j > c$$

with $c \in \mathbb{Q}$ for DLR only.[1]

The set $\mathcal{F}$ of all DL formulae is the smallest set containing the atomic formulae which is closed under negation ($\varphi \in \mathcal{F}$ implies $\neg\varphi \in \mathcal{F}$) and a collection of binary Boolean connectives ($\varphi_1, \varphi_2 \in \mathcal{F}$ implies $\varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2, \varphi_1 \rightarrow \varphi_2, \dots \in \mathcal{F}$).

[1] Over integers, the constraint $X - Y > c$ is of course equivalent to $X - Y \geq c + 1$, hence $>$-constraints are obsolete in the integer case and life is easier without them.

An $(\mathcal{X},\mathcal{B})$-*valuation* consists of two functions (overloaded with the name v) $v : \mathcal{B} \to \{T, F\}$ and $v : \mathcal{X} \to \mathbb{Z}$ for DLZ or $v : \mathcal{X} \to \mathbb{R}$ for DLR. The valuation v is extended to all $DL(\mathcal{B},\mathcal{X})$ formulae by letting

$$v(X_i - X_j \geq c) = T \text{ iff } v(X_i) - v(X_j) \geq c$$

and applying the obvious rules for the Boolean connectives.

A formula φ is satisfied by a valuation v iff $v(\varphi) = T$ (we denote it also by $v \models \varphi$). A formula φ is satisfiable if it has a satisfying valuation.

Proposition 2. *The satisfiability problem for $DL(\mathcal{B},\mathcal{X})$ is NP-complete.*

Proof. NP-hardness is an immediate consequence of the Boolean case, Cook's theorem [GJ79]. For NP-easiness, a non-deterministic algorithm works by guessing which atomic formulas (Boolean variables and inequalities) appearing in the formula are true and which are false. A polynomial time test then has to check that this assignment renders the entire formula true (linear time in the size of the formula) and that the corresponding set of constraints on the reals is in fact satisfiable. The satisfiability of a *conjunction* of difference constraints (a special case of linear programming) can be solved in polynomial (cubic) time using a variant of the Floyd-Warshall algorithm [CLRS01]. □

Our solver works on formulae in conjunctive normal form with at most 3 literals in a clause.

Definition 3 (3CNF). A Boolean literal is a formula of the form B or $\neg B$ with $B \in \mathcal{B}$. A numerical literal is a formula of the form $X - Y \geq c$ (also $X - Y > c$ for DLR). A 3-clause is a disjunction $C = L_1 \vee L_2 \vee L_3$ of at most 3 literals *at most one of which is a numerical literal*. A formula of difference logic is in 3CNF if it is a conjunction $\bigwedge_{k \in K} C_k$ of a set of 3-clauses C_k.

We use this normal form because it simplifies the description of proof methods based on the Davis-Putnam approach. As we will see later, the restriction to at most one numerical literal per clause plays an important role in the implementation of our procedure. As in propositional logic, translations from arbitrary formulae to 3CNF need not be costly if auxiliary Boolean variables are introduced:

Proposition 4. *From an arbitrary $DL(\mathcal{B},\mathcal{X})$ formula φ it is possible to derive a $DL(\mathcal{B}',\mathcal{X})$ formula φ' over an extended set of Boolean variables $\mathcal{B}' \supseteq \mathcal{B}$, such that*

- *φ' is in 3CNF;*
- *any $(\mathcal{B},\mathcal{X})$-valuation v satisfying φ can be extended to a $(\mathcal{B}',\mathcal{X})$-valuation v' satisfying φ'.*
- *(the projection of) any valuation v' satisfying φ' satisfies equally φ.*
- *$|\mathcal{B}'| = O(|\varphi|)$ and $|\varphi'| = O(|\varphi|)$.*

Proof. (Sketch) A simple construction to achieve this is to introduce for each composed sub-formula ψ of φ a fresh variable B_ψ to express whether for a given valuation ψ holds. Then the structure of φ can be broken up into local semantic constraints by coding, using 3CNF clauses, the semantic relation of ψ with its immediate sub-formulae. For instance, if $\psi = \psi_1 \vee \psi_2$ then we will add the three clauses $\{\neg B_\psi \vee B_{\psi_1} \vee B_{\psi_2}, B_\psi \vee \neg B_{\psi_1}, B_\psi \vee \neg B_{\psi_2}\}$ to our set of clauses. In addition to these structural clauses, a clause B_φ is used to express that φ should be true. It is easy to extend a satisfying valuation of φ to a satisfying valuation for the set of clauses constructed this way and, conversely, projecting the additional variables of a satisfying valuation of φ' yields a satisfying valuation of φ. □

In practice, optimized translations introducing less variables and with certain additional structural properties are possible (see [GW99] for a more detailed discussion).

As a tiny example, the formula $\varphi = (X - Y \geq 3) \wedge (Y - Z \geq -2 \vee Z - X \geq 1)$ can be translated to 3CNF using an additional Boolean variable B to represent $Y - Z \geq -2$: $\varphi' = (X - Y \geq 3) \wedge (B \vee Z - X \geq 1) \wedge (\neg B \vee Y - Z \geq -2) \wedge (B \vee Z - Y > 2)$. For the case of integer semantics, we can further translate to $\varphi' = (X - Y \geq 3) \wedge (B \vee Z - X \geq 1) \wedge (\neg B \vee Y - Z \geq -2) \wedge (B \vee Z - Y \geq 3)$.

3 Boolean Proof Methods

In this section we sketch the way we treat the Boolean part, some of which is common to other approaches for realizing the Davis-Putnam proof procedure (see, e.g. [Zha95]). This will serve as an introduction to the treatment of numerical constraints (there are some interesting analogies between the Boolean and numerical proof methods). Our presentation largely follows that of [GW99].

The proof method relies on applying satisfiability-preserving transformations to 3CNF DL formulae, some of which might reduce the number of satisfying valuations. Successive applications of these rules can lead to a satisfying valuation or to a (resolution) proof of unsatisfiability.

Simplifications: Elimination of redundant clauses which are weaker than others, independently of the rest of the formula. On the level of literals, a literal L_2 is weaker than another literal L_1 if for any valuation v, $v \models L_1$ implies $v \models L_2$. For Boolean literals this is the case for identical literals. For numerical constraints, $X - Y \geq c$ is weaker than $X - Y \geq d$ if $d \geq c$. This weaker-than relation generalizes to clauses in the obvious way.
Elimination of redundant literals by resolution: If there are two clauses $C_1 = L_1 \vee C'$ and $C_2 = \neg L_1 \vee C'$, replace C_1 and C_2 by C'.

Boolean unit resolution: If there is a clause L_1 and a clause $C = L_2 \vee C'$ such that L_1 and L_2 are contradictory, replace C by C'. Boolean literals are contradictory if one is the negation of the other.

Detection of logical cycles: If there is a chain of two literal clauses $\neg L_1 \vee L_2, \neg L_2 \vee L_3, \ldots \neg L_n \vee L_1$, apply a renaming of $L_2, \ldots , L_n$ to L_1 (this renaming has to be remembered in order to reconstruct complete valuations of the original formula). A cycle including the same literal negatively and positively exposes a contradiction and the formula is unsatisfiable. In practice, the detection of cycles is implemented using Tarjan's linear-time algorithm for computing strongly connected components [Tar72].

Detection of Boolean contradictions: An empty clause is not satisfiable, hence the entire formula containing an empty clause is not satisfiable. Note that numerical contradictions will show up as Boolean contradictions via resolution.

The empty conjunction: If we arrive at a formula containing no more clauses, it is trivially satisfiable by any valuation.

All of the above transformations can reduce the size of a formula without affecting its semantics. Sometimes, however, checking the conditions for applying these rules might be costly (quadratic in the size of the formula).

The Davis-Putnam rules go beyond these transformations by replacing a formula by a stronger one while preserving (non)satisfiability. Instead of describing the general Davis-Putnam framework, we only mention an interesting rule which preserves the 3CNF structure without introducing new variables.

Boolean Davis-Putnam rule: Supposing that there is a Boolean literal L such that the formula can be represented as $\bigwedge_i C_i \wedge \bigwedge_j (L \vee C'_j) \wedge \bigwedge_k (\neg L \vee L_k)$ where L does not occur in any of the clauses C_i, C'_j. That is, all negative occurrences of L are in 2-clauses. Then we can set $L := \bigwedge_k L_k$ (and memorize this assignment) and substitute $\bigwedge_k L_k$ for L throughout the formula.

Note that this transformation, while preserving satisfiability, can decrease the number of solutions. Secondly, while going back to 3CNF, some clauses may have to be copied in case of more than a single negative occurrence of L. It is a question of heuristics, when the application of this rule is sensible. Obviously the case of a single negative occurrence of L will result in a simplification.

Branching When the above transformations yield neither a solution nor a contradiction we resort to branching over valuations of Boolean variables: A Boolean variable is chosen and either assumed to be false or true, thus eliminating it. By recursion, eventually all variables will be eliminated. However, the scheme as stated requires backtracking, possibly leading to an exponential-time linear-space complexity.

A variation of this scheme is a sort of backtracking-free branching using Stålmarck's *Dilemma Rule*, motivated by shallow natural deduction proofs. It works as follows:

- As the non-branching rules lead to no more simplification of the formula, a variable is chosen for branching.
- On each branch, the non-branching rules are applied until no more progress is possible.
- If on either branch contradiction or satisfaction is achieved, act as in standard branching (satisfaction=success, contradiction=continuation of the other branch).
- If on neither branch contradiction or satisfaction is achieved then *join*: Apply to the set of clauses before the branching point the *assignments common to both branches* (which are consistent with both hypotheses over the branching variable).

It is possible to iterate the branching over all variables until the joining yields no more progress (no common assignments), then to apply the dilemma rule recursively on the branches with second (third ...) variables. Again with arbitrary depth of branching, this yields a complete decision procedure. This gives rise to a hierarchy of formulae for which the Stålmarck method yields a decision at recursion level k. The complexity can then be estimated by $O(n^{2k+1})$ where n is the size of the input formula. In practice, the method yields remarkable results for branching depth ≤ 2, above which it becomes inefficient in comparison to other procedures.

4 Numerical Proof Methods

The main contribution of the paper is in the treatment of numerical difference constraints. We sketch here the major ideas as applicable to the integer interpretation (the real one is discussed in the appendix).

Numerical unit resolution When a formula contains a numerical 1-literal clause $X - Y \geq c$, we can replace numerical literals $X - Y \geq d$ with $d \leq c$ by T and numerical literals $Y - X \geq d$ with $c > -d$ by F.

Computing numerical implications: Consider a chain of clauses $X_1 - X_2 \geq c_1, \ldots, X_{n-1} - X_n \geq c_n$. For every j, k such that $1 \leq j < k \leq n$ we can conclude that $X_j - X_k \geq \sum_{j \leq i \leq k} c_i$. These implied inequalities can be used to remove weaker numerical literals from the formula. In particular, when the chain involves a cycle (X_j and X_k are identical) then it can be replaced by T if $\sum_{j \leq i \leq k} c_i$ is negative and by F if it is positive. Moreover, if this sum is zero, one can eliminate the variables $X_{j+1}, \ldots, X_{k-1}$ and express them using X_j.

Similarly to the treatment of Boolean chains and cycles, the above numerical implications are computed using a graph algorithm, in this case the Floyd-Warshall *all vertex shortest path algorithm with negative weights* [CLRS01]. The data-structure for representing conjunctions of difference constraints is the *difference bounds matrix* used extensively in the verification of timed automata. In this matrix an inequality of the form $X_i - X_j \geq c$ is represented by putting $-c$ in the (i, j)-entry of the matrix (when two variables are not related by a constraint c is set to $-\infty$).

In the following, we develop an analogue of the Davis-Putnam rule for numerical constraints. We say that in a numerical literal $X - Y \geq c$ the variable X occurs positively and Y occurs negatively. Let v and v' be two valuation which are identical except for one variable X. One can see that if $v'(X) \geq v(X)$ then $v \models L$ implies $v' \models L$ for every literal L where X occurs positively and $v' \models L$ implies $v \models L$ when X occurs negatively in L.

Davis-Putnam for literal $\geq$-constraints: Assume, we have only a few (ideally, one) positive occurrences of X in literal clauses $X - Y_i \geq c_i$ with $i \in I$ and all other occurrences – in particular those in mixed clauses – are negative. Then we can set $X := max\{Y_i + c_i \mid i \in I\}$ (or $X := Y_i + c_i$

in the ideal case) and eliminate X from all clauses: Each literal $Z - X \geq d$ will be replaced by a conjunction $\bigwedge_{i \in I} Z - Y_i \geq d + c_i$. Analogously, if we have only a few negative occurrence of X in the one-literal clauses $Y_i - X \geq c$ with $i \in I$, we can safely set $X := min\{Y - c_i \mid i \in I\}$.

This rule is correct in not affecting the satisfiability of the formula because any valuation v satisfying it must satisfy $v(X) \geq v(Y_i) + c_i$. The valuation $v' = v[X := max\{v(Y_i) + c_i\}]$ where only the value of X is changed such that $v'(X) \leq v(X)$ also satisfies the formula, because it satisfies $X - Y_i \geq c_i$ for all $i \in I$ and all other clauses containing negative occurrences of X will also remain true.

5 Implementation Architecture

Data structures During the execution of the algorithm we keep Boolean and mixed clauses in a clause table while purely numerical clauses are kept in a difference bounds matrix. The interaction between the two happens when, following a simplification, a mixed clause is transformed into a numerical literal and passes from the clause table to the matrix. Numerical implications are computed on the matrix and may affect Boolean clauses, e.g. by numerical unit resolution.

Due to branching we need to organize these data structures in a stack. Heuristic improvements can be used to render stack operations less expensive, e.g. a special technique for storing stacks of sparse matrices.

The overall algorithm. The overall algorithm consists of two major parts: The branching procedure (either backtracking or Dilemma branching) and the non-branching rules. Since the cost of Davis-Putnam rules is potentially higher than that of the other rules, Boolean and numerical rules are performed in the following order:

1. Boolean simplifications;
2. Unit resolution;
3. Strongly connected component algorithm on the two literal Boolean clauses with subsequent detection of contradictions or elimination of equivalent literals;
4. Shortest path algorithm for the normalization of numerical constrains with subsequent detection of contradictions or elimination of tightly coupled variables and numerical unit resolution;
5. Application of the Boolean Davis-Putnam rule;
6. Application of the numerical Davis-Putnam rule;
7. Pass to branching.

Whenever a change in the clause set occurs at one of the stages (1)-(6), the algorithm restarts at (1).

Additional remarks. Technically, we have separated the translation from DL to 3CNF from the 3CNF solver by an intermediate file format: This allows to bypass the automatic translation and to directly generate clause sets.

Other variants of satisfiability procedures work on clauses containing more than three literals, but we believe that the restriction to at most one numerical literal per clause is very useful for the integration of numerical constraint resolution within a Boolean satisfiability solver.

Experiments At this stage, we have a running prototype with a low memory profile. There are several simple improvements that are still to be done, notably in the representation of sets of clauses and in the implementation of branching. We started experimenting with formulae obtained from translations of timed-automata reachability problems. Although the following table must not be confused with anything as a benchmark, it should give the reader an impression of the current performance of our implementation. Apart from the inefficient data structures we have nevertheless preliminary findings concerning the rapid (in terms of proof phases) elimination of redundant variables, which we consider encouraging for the overall methodology.

Test	# bool clauses	# mixed clauses	# bool vars	# real vars	Time (s)
RING1	111	52	59	6	1.33
RING2	224	104	116	9	15.47
RING3	337	156	173	12	100.98
RING4	450	208	230	15	224.65
RING5	563	260	287	18	409.43

Table 1. Initial experiments with formulae derived from timed automata, conducted on a 600 MHz Pentium III.

References

[ABK$^+$97] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data structures for the verification of timed automata. In *Hybrid and Real-Time Systems, LNCS 1201*, pages 346–360. Springer, 1997.

[BCCZ99] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS 1999, LNCS 1579*, pages 193–207. Springer, 1999.

[BM00] O. Bournez and O. Maler. On the representation of timed polyhedra. In *ICALP 2000, LNCS 1853*, pages 793–807. Springer, 2000.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, McGraw-Hill, 2001.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.

[GW99] J.F. Groote and J.P. Warners. The propositional formula checker HeerHugo. Technical Report 691, Centrum voor Wiskunde en Informatica (CWI) Amsterdam, 1999.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *LICS'92*, pages 394–406. IEEE, 1992.

[JM94] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[LPWY99] K. Larsen, J. Pearson, C. Weise, and W. Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[MLAH99] J. Møller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Difference decision diagrams. In *Proc. CSL'99*, 1999.

[NMA$^+$02] P. Niebert, M. Mahfoudh, E. Asarin, N. Jain, and O. Maler. Model checking of timed automata via SAT solving. submitted for publication, 2002.

[Rob96] A. Robinson. *Non-standard Analysis*. Princeton University Press, 1996.

[SS90] G. Stålmarck and M. Saflund. Modelling and verifying systems and software in propositional logic. In *SAFECOMP'90*, pages 31–36, 1990.

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[Zha95] G. Zhang. The Davis-Putnam resolution procedure. In *Advances in Logic Programming and Automated Reasoning*, volume 2. Ablex Publishing Corporation, 1995. chap. 4.

[ZS00] H. Zhang and M. Stickel. Implementing the Davis-Putnam method. *Journal of Automated Reasoning*, 24:277–296, 2000.

A Appendix: The Case of Real Domains

For DLR, everything is essentially the same as for DLZ, except that we have to deal with constraints of the form $X - Y > c$ and solutions over the reals. This affects

- The numerical unit resolution rule.
- The Davis-Putnam rule for the case $X - Y > c$.

We cannot eliminate $X - Y > c$ by setting $X = Y + c$, of course. But we can nevertheless eliminate X by defining it to be arbitrarily close to $Y + c$, say $X = Y + c + \varepsilon$. Then, of course, the substitution may propagate ε into other literals (like $Y - Z = d + c + \varepsilon$).

In order to formalize it in a consistent manner[2] we introduce a generalized normal form. The formal variable ε stands for a small real value, is allowed to occur as parameter in the real constraints.

Definition 5 (ε-3CNF). The syntactic form ε-3CNF is defined exactly as 3CNF except for real literals which can be of the form

$$X - Y > c + n \cdot \varepsilon$$

or

$$X - Y \geq c + n \cdot \varepsilon$$

where $c \in \mathbb{Q}$, $n \geq 0 \in \mathbb{N}$ and ε is a special symbol.

Obviously, 3CNF is a special case of ε-3CNF. The semantics of ε-3CNF formulae is defined as follows:

Definition 6 (Satisfaction of ε-3CNF formulae). Let φ be an ε-3CNF formula, $v : \mathcal{B} \rightarrow \{T, F\}$ and $v : \mathcal{X} \rightarrow \mathbb{R} \times \mathbb{N}$ be an valuation. Let furthermore $s > 0$ be a real number. We say that $v, s \models \varphi$ iff $v' \models \varphi'$ where φ' is the 3CNF formula obtained from φ by replacing every $c + n \cdot \varepsilon$ in a constraint by the real number $c + n \cdot s$ and by setting $v'(X) = d + m \cdot s$ for $v(X) = (d, m)$. We say that $v \models \varphi$ iff there exists a $\delta > 0$ such that for all r with $0 < r \leq \delta$ we have $v, r \models \varphi$.

The idea of this definition is to exploit the dense character of the order on the real or rational numbers: Between two distinct real numbers, we can fit an arbitrary multiple of a sufficiently small number r. Making r even smaller will not change this property. With the help of this normal form, we can adapt the Davis-Putnam rule to $>$-constraints. We elaborate only the special case of one-literal $>$-constraints.

Davis-Putnam for one-literal $>$-constraints: Assume we have only one positive occurrence of X in the clause $X - Y > c + n \cdot \varepsilon$ and all other occurrences are negative. Then we set X:= $Y + c + (n + 1) \cdot \varepsilon$. Literals $Z - X \geq d + m \cdot \varepsilon$ (or $Z - X > d + m \cdot \varepsilon$) with negative occurrences of X will then be replaced by $Z - Y \geq c + d + (m + n + 1) \cdot \varepsilon$ (or $Z - Y > c + d + (m + n + 1) \cdot \varepsilon$). In the analog case of a negative occurrence $Y - X > c + n \cdot \varepsilon$ we set $X := Y(n + 1) \cdot \varepsilon$.

The latter rule requires some justification:

Proposition 7. *Let φ be a ε-3CNF formula, v a valuation such that $v \models \varphi$, and $X - Y > c + n \cdot \varepsilon$ a one-literal clause, the only positive occurrence of X. Let φ' be the formula obtained from φ by elimination of X as described above. Then $v \models \varphi'$.*

Conversely, let v be a valuation such that $v \models \varphi'$ and let $v' = v$ except for $v'(X) = (c + r; k + n + 1)$ where $v(Y) = (r; k)$. Then $v' \models \varphi$.

[2] The ideas underlying this formalization are inspired from *Non-standard Analysis* [Rob96].

Proof. Let δ be such that $v, s \models \varphi$ for all s with $0 < s < \delta$. Let $Z - X > d + m \cdot \varepsilon$ be one of the literals with negative occurrences of X in φ which is replaced by $Z - Y > c + d + (m + n + 1) \cdot \varepsilon$. If $v, s \models Z - X > d + m \cdot \varepsilon$ an d $v, s \models X - Y > c + n \cdot \varepsilon$, $v(X) = (r_X, n_X)$, $v(Y) = (r_Y, n_Y)$ and $v(Z) = (r_Z, n_Z)$. Then $r_X - r_Y - c > (n_X + n_Y + n) \cdot s$ for all s with $0 < s < \delta$. Then either $r_X \cdot r_Y \cdot c > 0$, in which case there also exists δ' such that $r_X \cdot r_Y \cdot c > (n_X + n_Y + (n + 1)) \cdot s$, or $r_X \cdot r_Y \cdot c = 0$ in which case $0 \geq (n_X + n_Y + (n + 1))$. With the same reasoning we obtain $r_Z - r_X - d > (n_Z + n_X + (m + 1)) \cdot s$ or $0 \geq (n_Z + n_X + (m + 1))$.

Adding up inequalities we obtain $r_Z - r_Y - c - d > (n_Z + n_Y + (n + m + 1)) \cdot s$ for sufficiently small s or $0 \geq (n_Z + n_Y + (n + m + 2))$ and equally $0 > (n_Z + n_Y + (n + m + 1))$. In either case $v, s \models Z - Y > c + d + (m + n + 1) \cdot \varepsilon$ for sufficiently small s.

For the converse direction, $v' \models X - Y > c + n \cdot \varepsilon$ by definition of $v'(X)$. Simple arithmetics suffice to conclude $v' \models Z - X > d + m \cdot \varepsilon$ from $v \models Z - Y > c + d + (m + n + 1) \cdot \varepsilon$. □

The proposition justifies to do deduction with a sufficiently small constant ε, without instantiating it. In the case of a satisfiable formula with rational constraints, it is easy to instantiate ε as a sufficiently small fraction. Furthermore, we do not discuss the case of $>$-constraints that concern mixed clauses or more than one clause. It is tedious but not difficult to fill out these details.