A Solver for Quantified Boolean and Linear Constraints

Lucas Bordeaux
Microsoft Research, Cambridge
Cambridge, United Kingdom
lucasb@microsoft.com

Lintao Zhang
Microsoft Research, Silicon Valley
Mountain View, California
lintaoz@microsoft.com

ABSTRACT

We make a number of contributions to the understanding and practical resolution of quantified constraints. Unlike previous work in the CP literature that was essentially focused on constraints expressed as binary tables, we focus on Presburger Arithmetics, *i.e.,* Boolean combinations of linear constraints. From a theoretical perspective, we clarify the problem of the treatment of universal quantifiers by proposing a "symmetric" version of the notion of quantified consistency. This notion imposes to maintain two constraint stores, which will be used to reason on universal and existential variables, respectively. We then describe a branch & bound algorithm that integrates both forms of propagation. Its implementation is, to the best of our knowledge, the first CP solver for this class of quantified constraints.

1. INTRODUCTION

Quantified constraints. Quantified constraints allow to express many problems that cannot be modelled using classical constraint programming tools. Quantifiers (*i.e.,* $\exists$ and $\forall$) are a powerful way to express *uncertainty*: when taking a decision based on a value which is prone to some measurement error, one may prefer, for instance, robust solutions that are valid *for all* possible values of this variable. Quantifiers are also a natural tool for reasoning about *adversarial* situations: when making a decision, it is sometimes desirable to express the objectives of other agents, for instance competitors, in addition to one's own objective and constraints. Quantifier alternation allows to reason on the scenarios that can arise as all agents make a sequence of decisions interactively.

To be fair, it is still unclear, for these classes of applications, whether general-purpose quantified solvers will be useful tools in the future or whether problem-specific extensions of CP which use only bounded alternation, or syntactical variants of quantifiers, will be preferred. For instance, a number of recent works on uncertainty [12] and adversarial reasoning [8, 4] propose extensions of CP which, although expressible using quantifiers, do not require a full-fledged quantified constraint solver. As the tools for solving quantified constraints advance, we nevertheless expect to see new applications based directly on quantified constraint solvers instead of special extensions. What is clear in any case is that the better understanding we get of quantifier alternation in general, the better insight we gain on all these particular cases, and on the field of uncertain and adversarial reasoning as a whole. Following a number of works on quantified CSP [3, 7], this paper focuses on the general problem of quantified constraints, in which arbitrarily many quantifier alternations are allowed.

The problem of asymmetrical treatment of $\exists$ and $\forall$. Unlike classical (existentially quantified) constraints, quantified constraints are closed under negation[1]. As a consequence, the reasoning performed by solvers for quantified constraints should be symmetric, in the sense that dual constructs like $\forall$ and $\exists$ should be treated by dual types of reasoning. Intuitively, if a solver can easily solve a quantified formula ϕ, it should as well be able to efficiently solve the formula $\neg\phi$, in which the quantifiers are reversed. Yet most solvers fail to pass this test (see [13] for experimental evidence using the most advanced solvers for Quantified Boolean Formulae). The reason is that these solvers integrate techniques that are best adapted to only one side of the problem, namely its existential variables. For instance, the notion of quantified arc-consistency is essentially useful for existential variables only [3, 2], and all the variants of quantified consistency we are aware of (*e.g.,* [9]) are weak when it comes to pruning universally quantified variables.

The solution is to reason on the original formula (which we can think of as the "viewpoint of the existential player") and *on the negation of this formula* ("viewpoint of the adversary, or universal player") at the same time. This informal idea was suggested by several authors ([3], workshop version of [1]). In [10], the authors discussed a formulation using Quantified Boolean Formulae (QBF) of certain classes of planning problems that captures the views of both existential and universal parties. However, in their formulation, called Q-ALL SAT, only two levels of quantifier alternation are allowed. A symmetric way to solve QBF which goes beyond this restriction was proposed recently [13] (see Section 4.4). In this paper, we show how such a symmetric

[1]Given a CSP $\exists X.\phi(X)$, it is possible to encode in CSP the formula $\exists X.\neg\phi(X)$ (existence of a non-solution), but it is not possible in general to encode its negation $\forall X.\neg\phi(X)$ (absence of any solution) using an existentially quantified formula. For QCSP, the situation is, on the contrary, totally symmetric.

SAC'07 March 11-15, 2007, Seoul, Korea

treatment can be adopted for arbitrary types of quantified constraints, by proposing a symmetric notion of consistency that can be used to reason on both existential and universal variables.

A solver for bounded Presburger arithmetics. We then describe a solver which uses the proposed symmetric propagation mechanisms in a branch & prune (MAC-like) algorithm. The input language of this solver is somehow different from previous implementations (*e.g.,* [7]) in that we do not use a generalisation of binary CSPs, but instead Boolean combinations of linear constraints. This language is well-known in the Automated Reasoning literature under the name *Presburger Arithmetics*.

Outline. Section 2 describes the type of quantified constraints we primarily focus on, *i.e.,* Bounded Presburger Arithmetics. Section 3 presents the symmetric notion of consistency. Section 4 shows how this abstract notion of consistency can be implemented in practice and maintained during the search. Section 5 presents more details on our implementation and Section 6 gives concluding remarks.

2. PRESBURGER ARITHMETICS

While most recent works in quantified constraints were based on the QCSP framework [7, 11], we choose to focus on a different quantified language, namely Presburger Arithmetics (PA, *i.e.,* the quantified language of Boolean, linear, and Boolean combinations of linear constraints). We think this is an interesting quantified language, because addition and Boolean constraints are the most basic constructs provided by most CP solvers, and an overwhelming majority of problems can be expressed using these constructs. Furthermore, using a simple but well-defined language prevents us from a number of drawbacks of CSPs, especially binary CSPs (non-readability, benchmarking typically based on random instances). PA is extremely complex (hard for $\mathbf{NDTIME}(2^{2^n})$, which is due the fact that it can express extremely large numbers. Since in many applications it the numbers that are involved represent physical quantities that cannot be arbitrarily large, we focus on a version of PA in which every variable ranges over a finite set of values. We call this problem *Bounded Presburger Arithmetics, or BPA*. It is immediate to see that with this restriction the problem is PSPACE-complete, and that it remains so even if we have a single quantified linear constraint.

A formula in BPA is defined by the following grammar:

$$\begin{array}{lcl} N & ::= & \mathit{int\_cst} \mid \mathit{var} \mid \mathit{int\_cst} \cdot \mathit{var} \mid N + N \\ B & ::= & \mathit{bool\_cst} \mid \mathit{bool\_var} \mid \neg B \mid B \wedge B \mid N \leq N \\ F & ::= & B \mid \forall x \in \mathit{itv}.\ F \mid \exists x \in \mathit{itv}.\ F \end{array}$$

We did not specify the syntax for Boolean and numerical constants (*bool_cst* and *int_cst*) or intervals (*itv*), but there is little mystery surrounding them. Variables (*var* and *bool_var*) are simply considered as names built over the alphabetical symbols (these are disjoint from the other mathematical symbols used in the syntax). For the sake of simplicity, we only presented a minimal complete language but the techniques of the paper generalise to connectives like $\vee$, $-$, $<$, $=$, $\neq$. Note that the previous syntax is also limited to formulae in *prenex* form; additionally it is convenient to restrict oneself to *well-formed formula*, in which every variable name appearing in the matrix is quantified exactly once.

3. SYMMETRIC TREATMENT OF ∃/∀

3.1 Preliminary definitions and notations

In this section we consider a closed formula F of the form:

$$Q_1 x_1 \in D_1 \dots Q_n x_n \in D_n.\ \phi(x_1 \dots x_n) \qquad (1)$$

Where ϕ is a quantifier-free Presburger Arithmetics[2] formula, each Q_i is a quantifier chosen among $\{\forall, \exists\}$, and each D_i is a finite set of integer values.

Let X denote the set of variables $\{x_1 \dots x_n\}$; the set of variables x_i with $i \leq k$ will be noted X_k. The set of existential variables will be noted E and the set of existential variables with index at most k will be noted E_k; symmetrically, the set of universal variables (with index at most k) will be noted A (*resp.* A_k).

Given a set of variables V, a tuple t that defines an integer value t_{x_i} for every variable $x_i \in V$ will be called a *V-tuple*. Given a subset of variables $U \subseteq V$, the *restriction of t to U*, noted $t|_U$, is the U-tuple which associates the same values as t to the variables of U and is undefined on the other variables. We say that an X-tuple satisfies the relation ϕ iff $\phi(t_{x_1} \dots t_{x_n})$ is true in the considered theory.

3.2 Game-theoretic viewpoint on alternation

Quantifier alternation can be defined precisely using a game-theoretic viewpoint, in which a fictitious "existential player" chooses the values for the existential variables and tries to make the formula true, while an adversary interactively assigns the universal variables and tries to make the formula false.

DEFINITION 1 (GAME-THEORETIC SEMANTICS [5, 2]).

strategy: *An $\exists$-strategy s is defined by giving, for each existential variable x_i, a function s_{x_i} whose domain is the set of A_{i-1}-tuples and which returns values in D_{x_i}.*

scenario: *The set of scenarios for an $\exists$-strategy s, noted $\mathsf{sce}(s)$, is the set of X-tuples t which are such that $t_{x_i} = s_{x_i}(t|_{A_{i-1}})$, for each $x_i \in E$.*

winning strategy: *An $\exists$-strategy s is a winning $\exists$-strategy if every scenario $t \in \mathsf{sce}(s)$ satisfies the matrix ϕ. The set of winning $\exists$-strategies of the formula F will be noted $\mathsf{WIN}^{\exists}(F)$.*

Intuitively, an $\exists$-*strategy* can be thought of as a tree defining how the existential player is planning to react to every sequence of choices for the universal variables; this tree is formally defined by a set of functions that specify, for each existential variable, which value should be assigned to it depending on the values assigned to its universal predecessors. The *scenarios* are the X-tuples that are constrained to "follow the rules" imposed by a strategy, they correspond to the branches of the tree. In a *winning strategy*, all the branches correspond to solutions to the formula ϕ. Winning strategies are a central concept in understanding quantifier alternation; it is not hard to show the following:

FACT 1. *A formula F is true (i.e., $\mathbf{PA} \models F$) iff it has a winning $\exists$-strategy.*

[2]The results in this section can be adapted to other classes of quantified constraints.

The previous definitions were mentioned in the literature, without explicitly mentioning that they model the viewpoint of the existential player. To allow a more symmetric reasoning, we propose the dual notion of (winning) $\forall$-strategy:

DEFINITION 2 (NEW, DUAL DEFINITIONS).

strategy: *A $\forall$-strategy s is defined by giving, for each universal variable x_i, a function s_{x_i} whose domain is the set of E_{i-1}-tuples and which returns values in D_{x_i}.*

scenario: *The set of scenarios for a $\forall$-strategy s, noted $\mathsf{sce}(s)$, is the set of X-tuples t which are such that $t_{x_i} = s_{x_i}(t|_{E_{i-1}})$, for each $x_i \in A$.*

winning strategy: *A $\forall$-strategy s is a winning $\forall$-strategy if for every $t \in \mathsf{sce}(s)$, t* does not *satisfy the matrix ϕ. The set of winning $\forall$-strategies of the formula F will be noted $\mathsf{WIN}^\forall(F)$*

Basically the main thing to understand about $\forall$-strategies is that they correspond to the $\exists$-strategies of the *negation* of the formula. In particular:

FACT 2. *A formula F is false ($\boldsymbol{PA} \models \neg F$) iff it has a winning $\forall$-strategy.*

Note that, since every formula is either true or false in classical logic, a formula will *either* have one or several winning $\exists$-strategies, *or* (exclusive or) one/several winning $\forall$-strategies.

3.3 A Symmetric Notion of Consistency

In a classical (*i.e.,* existentially quantified) CP world, the meaning of consistency is clear: a value is consistent for a given variable iff at least one solution assigns the value to this variable. In a quantified setting the meaning of consistency is less trivial: under which conditions can we remove a value from the domain of a quantified variable with proven guarantees that the truth of the whole quantified expression will not be altered? The notion of quantified consistency, suggested in [3] and stated formally in [6, 2], answers this question for the case of existential variables: a value is consistent if at least one winning strategy has a scenario that assigns the value to this variable. We complete the picture and use our symmetric notion to define a notion of consistency which allows to deal with both existential and universal variables:

DEFINITION 3 (SYMMETRIC CONSISTENCY).

outcome: *The set of $\exists$-outcomes (resp. $\forall$-outcomes) of a QCSP is the set which contains the scenarios of all winning $\exists$-strategies (resp. $\forall$-strategies):*

$$\mathsf{out}^\exists(F) = \bigcup_{s \in \mathsf{WIN}^\exists(F)} \mathsf{sce}(s) \quad \mathsf{out}^\forall(F) = \bigcup_{s \in \mathsf{WIN}^\forall(F)} \mathsf{sce}(s)$$

consistency: *A value $a \in D_i$ is $\exists$-consistent w.r.t. x_i iff there is a winning $\exists$-strategy which, in one scenario at least, assigns x_i to a, i.e., if there exists $t \in \mathsf{out}^\exists$ such that $t_{x_i} = a$. Symmetrically, the value a is $\forall$-consistent w.r.t. x_i iff there is a winning $\forall$-strategy which, in one scenario at least, assigns x_i to a, i.e., if there exists $t \in \mathsf{out}^\forall$ such that $t_{x_i} = a$.*

The essential property of this symmetric definition of consistency is that it does indeed allow us to remove values from the domains of the variables. In particular, the notion of $\forall$-*consistency correctly expresses the possibility of removing a value from the domain of a universally quantified variable*:

PROPOSITION 1 (CORRECTNESS). *Given a formula $F : Q_1x_1 \in D_1 \dots Q_nx_n \in D_n.\ \phi(x_1 \dots x_n)$ and a value $a \in D_{x_i}$:*

- *if $Q_i = \exists$, then (1) if a is $\exists$-inconsistent w.r.t. x_i, then removing value a is equivalence-preserving, i.e., F is equivalent to the formula $Q_1x_1 \in D_1 \dots \exists x_i \in D_i \setminus \{a\} \dots Q_nx_n \in D_n.\ \phi(x_1 \dots x_n)$; (2) if a is $\forall$-inconsistent w.r.t. x_i then the formula is true.*
- *if $Q_i = \forall$, then (1) if a is $\forall$-inconsistent w.r.t. x_i, then removing value a is equivalence-preserving, i.e., F is equivalent to the formula $Q_1x_1 \in D_1 \dots \forall x_i \in D_i \setminus \{a\} \dots Q_nx_n \in D_n.\ \phi(x_1 \dots x_n)$; (2) if a is $\exists$-inconsistent w.r.t. x_i then the formula is false.*

(Proofs are omitted due to lack of space.) It was noted in [2] that determining whether a value is ($\exists$-)consistent is PSPACE-complete; it is straightforward to see that the same complexity result holds for $\forall$-consistency, and remains true even when the matrix is reduced to a single quantified linear constraint. Last, we note the following:

FACT 3. *A value a is $\forall$-inconsistent for variable x_i w.r.t. a quantified formula F iff it is $\exists$-inconsistent for x_i w.r.t. $\neg F$.*

4. DETECTING INCONSISTENT VALUES DURING THE SEARCH

Since the problem of detecting inconsistent values is complex, it is wise, within a branch & prune framework, to detect inconsistent values only incompletely. We focus on one approach, which is to use *local* consistency techniques for this purpose. In this approach, classical in CP, a complex formula is broken into a set of small constraints for which the solver natively defines a set of propagators. Some new technicalities arise, however, if we want to prune both existential and universal variables.

In this section we explain the approach in detail; we first recall how the matrix of a BPA formula can be syntactically decomposed, then analyse how the information obtained from each constraint of the decomposition can be used to make deductions on the original formula. This analysis calls for a framework in which two constraint stores are maintained. We conclude this section with a comparison with a technique recently proposed in QBF.

4.1 Decomposition of BPA

Since it is impossible to deal with a complex Boolean expression (in the sense of Section 2) as a whole, we can decompose it using the following (informal) rules:

$\phi[k.x] \rightsquigarrow \exists z.\ \phi[z] \wedge z = (k \cdot x)$;
$\phi[x + y] \rightsquigarrow \exists z.\ \phi[z] \wedge z = (x + y)$;
$\phi[x \vee y] \rightsquigarrow \exists z \in [0, 1].\ \phi[z] \wedge z = (x \vee y)$;
$\phi[\neg x] \rightsquigarrow \exists z \in [0, 1].\ \phi[z] \wedge z = (\neg x)$;
$\phi[x \leq y] \rightsquigarrow \exists z \in [0, 1].\ \phi[z] \wedge z = (x \leq y)$.

where we use a notation $\phi[s]$ to denote the fact that formula ϕ has a subterm s (in practice ϕ will typically be the matrix

of the formula). The initial domains of the variables introduced when decomposing a multiplication or addition are computed in the usual way, by simple interval evaluation. Applying these rules it is easy to see that any BPA formula rewrites into the "conjunctive primitive form":

$$Q_1 x_1 \dots Q_n x_n . \exists z_1 \dots \exists z_p . \ \left(\ \bigwedge_{i \in 1..m} c_i \ \right)$$

where each c_i is a primitive constraint of the form $z = (k.x)$, $z = (x + y)$, $z = (x \vee y)$, $z = (\neg x)$, or $z = (x \leq y)$. The interest of this decomposition is that treating each of these primitive constraints becomes tractable.

4.2 Detecting inconsistency locally

How do deductions made on each formula of a decomposition relate to the original formula? The following proposition is a symmetric version of a result in [2]; it states under which conditions it is possible to detect inconsistent values by reasoning on each constraint independently:

PROPOSITION 2 (CONDITIONS FOR LOCAL INCONSISTENCY). *Given a formula F of the form $Q_1 x_1 .. Q_n x_n . \ \phi$:*

- *If $\phi \equiv \bigwedge_{j \in 1..m} c_j$, then the following is true for every i: if a value is $\exists$-inconsistent w.r.t. $Q_1 x_1 .. Q_n x_n . c_i$ then it is also $\exists$-inconsistent w.r.t. F.*
- *If $\phi \equiv \bigvee_{j \in 1..m} c_j$, then the following is true for every i: if a value is $\forall$-inconsistent w.r.t. $Q_1 x_1 .. Q_n x_n . c_i$ then it is also $\forall$-inconsistent w.r.t. F.*

Given a *conjunction* of constraints, it is therefore not possible in general to detect $\forall$-inconsistency by having a look at a subset of the constraints of the formula. What is possible, however, is to use Fact 3 and to reason on the negation of the formula. *Any means to detect $\exists$-inconsistent values on formula $\neg F$ will allow us to detect $\forall$-inconsistent values on formula F.*

4.3 Maintaining Consistency During the Search

Our approach therefore maintains two constraint stores: let S_1 denote the store of constraints obtained by decomposing the original formula F using the rules of Section 4.1, and let S_2 denote the store obtained by decomposing formula $\neg F$ using these rules, and in which the quantification is reversed. Note that both stores are *conjunctions* of constraints. Each of these stores has its own copy of the original variables of the problem $(x_1 \dots x_n)$.

We detect $\exists$-inconsistent values by running a propagation algorithm on store S_1 and $\forall$-inconsistent values by running a propagation algorithm on store S_2. A classical (domain or interval) propagation algorithm can be used in each case. For certain quantifier patterns one can design slightly improved propagation rules. For instance, if a store contains a primitive constraint of the form $x + y = z$ (x, y, and z appearing in this order in the prefix) where x is quantified existentially, y universally and z existentially, we can use the following propagation rules:

$$\begin{array}{rcl} I_x & \leftarrow & I_x \cap [z^- - y^-, z^+ - y^+] \\ I_z & \leftarrow & I_z \cap [x^- + y^-, x^+ + y^+] \end{array}$$

where $I_x = [x^-, x^+]$ denotes the interval of values for variable x. These rules compute a correct interval approximation of the values that are $\exists$-consistent *w.r.t.* formula $\exists x \forall y \exists z . \ x + y = z$, and the bounds computed for variable x are tighter than the ones obtained using the classical rule, *i.e.,* $I_x \cap [z^- - y^+, z^+ - y^-]$. This is an improvement of the same kind as what is done in QBF, where quantified unit propagation can be used to do more deductions than in SAT.

The main algorithm for the resolution of a quantified formula $Q_1 x_1 \in D_1 \dots Q_n x_n \in D_n . \ \phi(x_1 \dots x_n)$ is described below; it shows how to exploit $\exists$-propagation (propagation on original store, or S_1) and $\forall$-propagation (propagation on negated store, or S_2) during the search.

```
Algorithm eval
    return rec_eval(1)
where
    function rec_eval (i:variable_index): bool
        propagate()
        if inconsistent(S1) then return false
        if inconsistent(S2) then return true
        if Qi = ∃ then
            for each a ∈ Di do
                if instantiate(xi, a, S1) then
                    if ¬instantiate(xi, a, S2) ∨ rec_eval(i + 1) then
                        return true
            done
            return false
        else  % Qi = ∀
            for each a ∈ Di do
                if instantiate(xi, a, S2) then
                    if ¬instantiate(xi, a, S1) ∨ ¬rec_eval(i + 1) then
                        return false
            done
            return true
        fi
    end
end algorithm
```

A last remark needed to prove the correctness of the algorithm is the following:

FACT 4. *Let $F[x_i \leftarrow v]$, where $v \in D_i$ represent the BPA formula obtained by fixing the value of x_i to v (i.e., fixing the domain D_i to the singleton $\{v\}$). We have $(\neg F)[x_i \leftarrow v] \equiv \neg(F[x_i \leftarrow v])$.*

4.4 Comparison with combined disjunctive/conjunctive forms

A related approach to the symmetric treatment of $\exists$ and $\forall$ was recently proposed by the second author in the context of QBF [13]. We briefly describe how it can be formulated in the context of BPA and compare it to our approach. The idea is, instead of maintaining a classical conjunctive form, to preprocess the problem in order to obtain a disjunction of two matrices: one in conjunctive form and one in disjunctive form. More precisely, let $\phi(x_1..x_n)$ be quantifier free formula; using the decomposition technique sketched in Section 4.1 we see that:

- $\phi(x_1..x_n)$ can be decomposed into a primitive conjunctive form: we can compute an equivalent formula of the form $\exists z_1..z_p . \ A(x_1..x_n, z_1..z_p)$ where A is a conjunction of primitive constraints;
- $\neg\phi(x_1..x_n)$ can also be decomposed into a primitive conjunctive form: we can compute an equivalent formula of the form $\exists z_1..z_q . \ B(x_1..x_n, z_1..z_q)$ where B is a conjunction of primitive constraints.

Now taking a quantified formula F of the form $Q_1x_1 \dots Q_nx_n.\ \phi(x_1..x_n)$, we can rewrite it in a *Combined Conjunctive Disjunctive Normal Form*: because $\phi(x_1..x_n)$ is equivalent to $\phi(x_1..x_n) \vee \neg\neg\phi(x_1..x_n)$, we can express F as:

$Q_1x_1 \dots Q_nx_n.\ (\ \phi(x_1..x_n) \vee \neg(\neg\phi(x_1..x_n))\ )$

which rewrites to:

$$Q_1x_1 \dots Q_nx_n. \left(\begin{array}{cc} & (\exists z_1..z_p.\ \bigwedge_i a_i(x_1..x_n, z_1..z_p)) \\ \vee & \neg(\exists z_1..z_q.\ \bigwedge_i b_i(x_1..x_n, z_1..z_q)) \end{array} \right)$$

and ultimately to:

$$Q_1x_1 \dots Q_nx_n. \left(\begin{array}{cc} & (\exists z_1..z_q.\ \bigwedge_i a_i(x_1..x_n, z_1..z_q)) \\ \vee & (\forall z_1..z_q.\ \bigvee_i \overline{b_i}(x_1..x_n, z_1..z_q)) \end{array} \right)$$

where $\overline{b_i}$ denotes the negation of the primitive constraint b_i. In this approach, $\forall$-inconsistent values are detected by reasoning on the *disjunctive* part of the matrix; a dual form of reasoning is needed because in the disjunctive part we deal with the *negation* of primitive constraints (in QBF these are known as *cubes i.e.,* conjunctions of literals, and dual techniques to unit propagation have been developed to make deductions on cubes).

The approach we have developed throughout Section 4.2, on the other hand, will maintain the two formulae, representing respectively F and $\neg F$:

$$\begin{array}{ll} Q_1x_1 \dots Q_nx_n.\exists z_1..z_p. & \bigwedge_i a_i(x_1..x_n, z_1..z_p) \\ \overline{Q_1}x_1 \dots \overline{Q_n}x_n.\exists z_1..z_q. & \bigwedge_i b_i(x_1..x_n, z_1..z_q) \end{array}$$

where $\overline{Q_i}$ denotes the dual quantifier to Q_i. Compared to [13] this approach is more lightweight in the sense that the language of primitive constraints does not need to be closed under complement. The two stores that are maintained are conjunctions and the deduction technique applied to both of them is a classical propagation (possibly enhanced to take into account quantifier patterns as suggested in Section 4.3). To use the approach of [13] we would require to define propagators of $\forall$-inconsistent values for each of the negated primitive constraints $z \neq (k.x)$, $z \neq (x+y)$, $z \neq (x \vee y)$, $z \neq (\neg x)$, and $z \neq (x \leq y)$.

5. THE SOLVER

We developed a solver for bounded Presburger arithmetics that implements the search algorithm described throughout the paper. Our goal in the development of the solver was twofold: first, we wanted to provide a first implementation that could be distributed for comparison with other approaches to quantified constraints, and that we could use on the longer term as a platform for the development of new algorithms. Second, we wanted to start gathering a collection of instances in a simple format. Both the executable and the instance set are available at the web-page of the first author `www.research.microsoft.com/~lucasb`.

6. CONCLUSION

Summary of the contributions. We have formalised a "symmetric" notion of consistency that captures the meaning of a pruning of both existential and universal variables; we feel that this extension of the formalism of [2] was necessary since the notion of quantified arc-consistency was so far essentially restricted to existential variables. We have then proposed an approach to maintain the two symmetric forms of consistency during the search, using two constraint stores. This proposal is closely related to recent work on Quantified Boolean Formulae, and we have discussed the respective benefits of the two approaches.

Extending the formalism of [2], we have made an effort to make all definitions as formal as possible, and to prove the algorithms we proposed. We believe this is important since the use of quantifiers makes many notions harder to get right; a number of properties that are taken for granted on classical constraints are less obvious, or do simply not hold, when we reason on quantified constraints (see Proposition 2).

We have, lastly, proposed an implementation of the proposed techniques for the language of (bounded) Presburger Arithmetics. To the best of our knowledge, this is the first implementation of a solver based on CP techniques for this important class of problems. Because of the difficulty to solve quantified constraints, the solver is far from the efficiency which would make it widely usable, but the implementation of the core of the solver is robust and the solver should be extensible.

Extensions of the work. Our goal is now to integrate in the platform more advanced techniques; experiments show that propagation techniques alone deal poorly with some instances, and more evolved algorithms inspired from the QCSP framework [7] or QBF have to be considered.

7. REFERENCES

[1] C. Ansótegui, C. P. Gomes, and B. Selman. The Achilles' heel of QBF. In *AAAI*, pages 275–281, 2005.

[2] L. Bordeaux, M. Cadoli, and T. Mancini. CSP properties for quantified constraints: Definitions and complexity. In *AAAI*, pages 360–365, 2005.

[3] L. Bordeaux and E. Monfroy. Beyond NP: arc-consistency for quantified constraints. In *CP*, pages 371–386, 2002.

[4] K. N. Brown, J. Little, P. J. Creed, and E. C. Freuder. Adversarial constraint satisfaction by game-tree search. In *ECAI*, pages 151–155, 2004.

[5] H. Chen. Quantified constraint satisfaction and bounded treewidth. In *ECAI*, pages 161–165, 2004.

[6] H. Chen and M. Pál. Optimization, games, and quantified constraint satisfaction. In *MFCS*, pages 239–250, 2004.

[7] I. P. Gent, P. Nightingale, and K. Stergiou. QCSP-Solve: A solver for quantified constraint satisfaction problems. In *IJCAI*, pages 138–143, 2005.

[8] S. Manandhar, A. Tarim, and T. Walsh. Scenario-based stochastic constraint programming. In *IJCAI*, pages 257–262, 2003.

[9] P. Nightingale. Consistency for quantified constraint satisfaction problems. In *CP Workshop on Quantification in Constraint Programming*, 2005.

[10] C. Otwell, A. Remshagen, and K. Truemper. An effective QBF solver for planning problems. In *AMCS*, pages 311–316, 2004.

[11] K. Stergiou. Repair methods for quantified CSPs. In *CP*, pages 652–666, 2005.

[12] N. Yorke-Smith and C. Gervet. Certainty closure: a framework for reliable constraint reasoning with uncertainty. In *CP*, pages 769–783, 2003.

[13] L. Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *AAAI*, 2006. To appear.