

Compilation for QCSP

Igor Stéphan

LERIA, University of Angers, France
email: igor.stephan@info.univ-angers.fr

Abstract. We propose in this article a framework for compilation of quantified constraint satisfaction problems (QCSP). We establish the semantics of this formalism by an interpretation to a QCSP. We specify an algorithm to compile a QCSP embedded into a search algorithm and based on the inductive semantics of QCSP. We introduce an optimality property and demonstrate the optimality of the interpretation of the compiled QCSP.

1 Introduction

A constraint satisfaction problem (CSP) requires a value, selected from a given finite domain, to be assigned to each variable in the problem, so that all constraints relating the variables are satisfied [14,8]. A quantified constraint satisfaction problem (QCSP) [9,6] is an extension of a constraint satisfaction problem in which some of the variables are universally quantified (since the remaining variables are still existentially quantified). In this latter framework, variables take value in discrete domains. Universally quantified variables may be considered to represent certain kind of uncertainty: a choice of nature or an opponent. A QCSP can formalize many AI problems including planning under uncertainty and playing a game against an opponent. In this second application, the goal of the QCSP is to make a robust plan against the opponent. Whereas finding a solution of a CSP is generally NP-complete, finding a solution for a QCSP is generally PSPACE-complete [9].

Most of the recent decision procedure for QCSP [3,5,12,7] are based on a search algorithm (except [18] which is based on a bottom-up approach and [13] which is based on a translation to quantified boolean formulas) and off-line procedures (except [2] which is an on-line real-time algorithm based on Monte Carlo game tree search and [17] which is based on standard game tree search techniques). Such an algorithm chooses a variable, branches on the different values of the domain, verifies if the subproblems have some solutions and combines, according to the semantics of the quantifier associated to the variable, those solutions into a solution to the problem.

Knowledge compilation is considered in many AI applications where quick on-line responses are expected. In general, a knowledge base is compiled off-line into a target language which is then used on-line to answer some queries. The goal is to have a lesser complexity for the query computation of the compiled knowledge

base than for the initial knowledge base. This principle is for example applied in product configuration where the set of possible configurations is compiled [1].

As far as we know, the problem of compiling a knowledge base represented as a QCSP has not been treated but only for the related domain of quantified Boolean formulas [16,11]. Our first contribution is a new formalism as compilation target language: the QCSP base. Our second contribution is a definition of an optimality property for QCSP bases in order to give a polytime answer to the next move choice problem [16] which raises the issue of whether one can change for another solution during the game. Our third contribution is a compilation algorithm embedded in a search algorithm which is proved to compile a QCSP in an optimal QCSP base.

This article is organized as follows: Section 2 establishes the necessary preliminaries, section 3 presents our framework and target language for the compilation of QCSP, section 4 specifies an algorithm to compile a QCSP in our target language, section 5 concludes with a discussion and some further works.

2 Preliminaries

Symbol \exists stands for existential quantifier and symbol \forall stands for universal quantifier. Symbol \wedge stands for logical conjunction, symbol \top stands for what is always true and symbol \perp stands for what is always false. A QCSP is a tuple $(\mathbf{V}, \text{rank}, \text{quant}, \mathbf{D}, \mathbf{C})$: \mathbf{V} is a set of n variables, rank is a bijection from \mathbf{V} to $[1..n]$, quant is a mapping from \mathbf{V} to $\{\exists, \forall\}$ (quant(v) is the quantifier associated to the variable v), \mathbf{D} is a mapping from \mathbf{V} to a set of domains $\{D(v_1), \dots, D(v_n)\}$ where, for every variable $v_i \in \mathbf{V}$, $D(v_i)$ is the finite domain of all the possible values ($D(v)$ is the domain associated to the variable v), \mathbf{C} is a set of constraints. If v_{j_1}, \dots, v_{j_m} are the variables of a constraint $c_j \in \mathbf{C}$ then the relation associated to c_j is a subset of the Cartesian product $D(v_{j_1}) \times \dots \times D(v_{j_m})$. In what follows, we denote for every $i \in [1..n]$, $q_i = \text{quant}(v_i)$ and $D_i = D(v_i)$. A QCSP $(\mathbf{V}, \text{rank}, \text{quant}, \mathbf{D}, \mathbf{C})$ on n variables will be denoted as follows to simplify notation:

$$q_1 v_1 \dots q_n v_n \bigwedge_{c_j \in \mathbf{C}} c_j$$

with $v_1 \in D_1, \dots, v_n \in D_n$, rank(v_i) = i , for every $i \in [1..n]$; $q_1 v_1 \dots q_n v_n$ is the binder.

The QCSP $(\{x, y, z, t\}, \text{rank}, \text{quant}, \{\{0, 1, 2\}\}, \mathbf{C})$ with

$$\begin{cases} \text{rank} = \{(1, x), (2, y), (3, z), (4, t)\}, \\ \text{quant} = \{(x, \exists), (y, \exists), (z, \forall), (t, \exists)\}, \\ D(x) = D(y) = D(z) = D(t) = \{0, 1, 2\} \text{ and} \\ \mathbf{C} = \{(x = (y * z) + t)\} \end{cases}$$

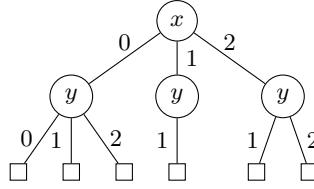
is, for example, denoted : $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ with $x, y, z, t \in \{0, 1, 2\}$.

In a binder, a maximal homogeneous sequence of quantifiers forms a bloc ; the first one (and also the outermost) is the leftmost.

The set $\mathcal{T}_i(Q)$ with $v_1 \in D_1, \dots, v_n \in D_n$, $Q = q_1 v_1 \dots q_n v_n$ for $1 \leq i \leq n$ is the set of trees such that

- every leaf node is labeled by the symbol \square and is at depth i ,
- every internal node at depth k , $0 \leq k < i - 1$ is labeled with the variable v_{k+1} ,
- every edge linking a node at depth k to one of its children's nodes is labeled with an element of D_k ,
- all the labels of the edges linking a node to its children nodes are different.

The following tree is, for example, an element of the set $\mathcal{T}_2(\exists x \exists y \forall z \exists t)$ with $x, y, z, t \in \{0, 1, 2\}$:

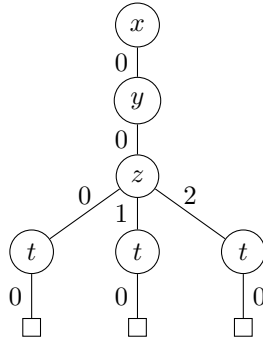


Let $(\mathbf{V}, \text{rank}, \text{quant}, \mathbf{D}, \mathbf{C})$ be a QCSP such that $\mathbf{V} = \{v_1, \dots, v_n\}$, with $v_1 \in D_1, \dots, v_n \in D_n$, then a scenario is the sequence of the labels val_1, \dots, val_n on the path $(v_1, val_1), \dots, (v_n, val_n)$, $val_i \in D_i$ for every i , $1 \leq i \leq n$, of a tree of $\mathcal{T}_n(q_1 v_1 \dots q_n v_n)$ and a strategy is a tree of $\mathcal{T}_n(q_1 v_1 \dots q_n v_n)$ such that

- every node labeled with an existentially quantified variable has a unique child node and
- every node labeled with a universally quantified variable whose associated domain is of size k admits k children nodes.

A scenario val_1, \dots, val_n for a QCSP $(\mathbf{V}, \text{rank}, \text{quant}, \mathbf{D}, \mathbf{C})$ such that $\mathbf{V} = \{v_1, \dots, v_n\}$ is a winning scenario if $(\bigwedge_{1 \leq i \leq n} v_i = val_i) \wedge (\bigwedge_{c_j \in \mathbf{C}} c_j)$ is true ; such a scenario corresponds to the complete instantiation $[v_1 \leftarrow val_1], \dots, [v_n \leftarrow val_n]$; it is a winning scenario if the instantiation satisfies all the constraints. A strategy is a winning strategy if all the scenarios are winning scenarios. If there is no quantifier, the \square strategy is always a winning strategy.

The scenario 0, 0, 2, 0, which corresponds to the complete instantiation $[x \leftarrow 0], [y \leftarrow 0], [z \leftarrow 2]$ and $[t \leftarrow 0]$, is a winning scenario, since $0 = (0 * 2) + 0$. The following strategy is a winning strategy



for the QCSP $\exists x \exists y \forall z \exists t (x = (y * z) + t), x, y, z, t \in \{0, 1, 2\}$ since $(0 = (0 * 0) + 0)$, $(0 = (0 * 1) + 0)$ and $(0 = (0 * 2) + 0)$.

We can give a more intuitive and recursive decision semantics for QCSP as follows: A QCSP $\forall x \forall y \exists z \exists t (x = (y * z) + t)$ with $x \in D$ admits a winning strategy if and only if, for every $val \in D$, $Q(C \wedge (x = val))$ admits a winning strategy and a QCSP $\exists x \forall y \exists z \exists t (x = (y * z) + t)$ with $x \in D$ admits a winning strategy if and only if, for at least one $val \in D$, $Q(C \wedge (x = val))$ admits a winning strategy.

3 Base for QCSP

From a complexity point of view and under some classical assumptions, winning strategies are exponential in space in worst case w.r.t. the number of variables of the QCSP [10]. But the number of winning strategies may also be exponential in worst case. A naive way to compile a QCSP would be to store all the winning strategies in a set but this approach is intractable in practice. For example, the QCSP $\forall x \forall y \exists z \exists t (x = (y * z) + t), x, y, z, t \in \{0, 1, 2\}$ admits 324 winning strategies. Another way is to store a tree which contains only the scenarios present in the winning strategies. This approach is not very useful too from the knowledge representation point of view since there is no direct access to the possibilities of an existentially quantified variable except for those of the first bloc.

We define in this section our formalism as a target language for QCSP compilation: the *QCSP base*. We also define the semantics of QCSP bases in terms of QCSP. We introduce a property of optimality for QCSP bases and prove a very interesting result about optimal QCSP.

3.1 Definitions for QCSP bases

Intuitively, a QCSP base is a set of strategies organized according to a mechanism of guards for every existentially quantified variable and every value of the domain. Such a guard is a pair of a value and a tree which is the expression of what have already been played by both opponents.

Definition 1 (QCSP base). A QCSP base is either

- the symbol *bl_top*
- the symbol *bl_bottom*
- a pair $\langle Q \mid G \rangle$ with $n > 0$, $Q = q_1 v_1 \dots q_n v_n$ and $G = [G_{e_1}, \dots, G_{e_m}]$ a list such that
 - e_1, \dots, e_m is the set of indexes of the existentially quantified variables¹ ;
 - every G_{e_k} , $1 \leq k \leq m$ is a function with non-empty graph $\{(val_1 \mapsto T_1), \dots, (val_{j_k} \mapsto T_{j_k})\}$, $val_1, \dots, val_{j_k} \in D(v_{e_k})$ and $T_1, \dots, T_{j_k} \in \mathcal{T}_{e_k}(Q)$.

¹ u_1, \dots, u_p is the set of indexes of the universally quantified variables, $\{e_1, \dots, e_m\} \cup \{u_1, \dots, u_p\} = [1..n]$, $\{e_1, \dots, e_m\} \cap \{u_1, \dots, u_p\} = \emptyset$, $\text{quant}(v_{e_i}) = \exists$, for every i , $1 \leq i \leq n$, $\text{quant}(v_{u_i}) = \forall$, for every i , $1 \leq i \leq p$.

A pair $(val, T) \in G_{e_k}$ is a guard for the existentially quantified variable v_{e_k} .

In what follows, the QCSP bases bl_top and bl_bottom are semantically interpreted as respectively what is always true and what is always false and algorithmically as respectively what admits every strategy as a winning strategy and what admits no winning strategy at all.

Example 1. The following guard sets G_x , G_y and G_z are guard sets of the QCSP base $B = \langle \exists x \exists y \forall z \exists t \mid [G_x, G_y, G_t] \rangle$ with $x, y, z, t \in \{0, 1, 2\}$.

$$G_x = [(0, \square), (1, \square), (2, \square)],$$

$$G_y = [(0, \begin{array}{c} x \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \square \quad \square \quad \square \end{array}), (1, \begin{array}{c} x \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \square \quad \square \quad \square \end{array}), (2, \begin{array}{c} x \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \square \quad \square \quad \square \end{array})]$$

$$G_t = [(0, \begin{array}{c} x \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \begin{array}{c} y \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \begin{array}{c} z \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \square \quad \square \quad \square \end{array} \end{array} \quad \begin{array}{c} y \\ \downarrow \\ 1 \\ \begin{array}{c} z \\ \downarrow \\ 0 \\ \square \end{array} \end{array} \quad \begin{array}{c} y \\ \swarrow \quad \downarrow \quad \searrow \\ 1 \quad 2 \\ \begin{array}{c} z \\ \downarrow \\ 2 \\ \square \end{array} \quad \begin{array}{c} z \\ \downarrow \\ 1 \\ \square \end{array} \end{array} \end{array}),$$

$$(1, \begin{array}{c} x \\ \swarrow \quad \downarrow \quad \searrow \\ 1 \quad 2 \\ \begin{array}{c} y \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \begin{array}{c} z \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \square \quad \square \quad \square \end{array} \quad \begin{array}{c} z \\ \downarrow \\ 0 \\ \square \end{array} \quad \begin{array}{c} z \\ \downarrow \\ 1 \\ \square \end{array} \end{array} \quad \begin{array}{c} y \\ \downarrow \\ 2 \\ \begin{array}{c} z \\ \downarrow \\ 0 \\ \square \end{array} \end{array} \end{array}), (2, \begin{array}{c} x \\ \downarrow \\ 2 \\ \begin{array}{c} y \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \begin{array}{c} z \\ \swarrow \quad \downarrow \quad \searrow \\ 0 \quad 1 \quad 2 \\ \square \quad \square \quad \square \end{array} \quad \begin{array}{c} z \\ \downarrow \\ 0 \\ \square \end{array} \quad \begin{array}{c} z \\ \downarrow \\ 0 \\ \square \end{array} \end{array})$$

3.2 Interpretation

The semantics of a QCSP base is expressed by an interpretation to the QCSP. First of all, we interpret the trees of the guards of the QCSP bases as tuples of values.

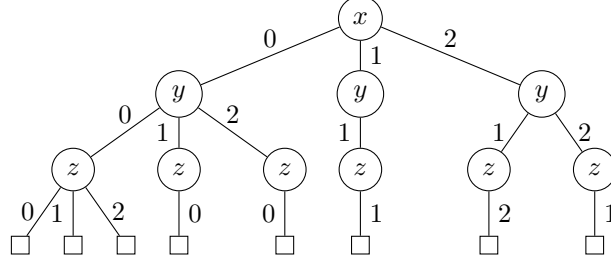
Definition 2 (interpretation of a tree). *The interpretation of a tree T of a guard (val, T) according to a value val is the set*

$$I^{val}(T) = \{(val, e_1, \dots, e_n) \mid e_1 \dots e_n \text{ a branch of a tree } T\}.$$

In particular, $I^{val}(\Box) = \{val\}$. The interpretation of a set G of guards (value, tree) is by extension :

$$I(G) = \bigcup_{(val, T) \in G} I^{val}(T).$$

Example 2. (Example 1 continued.) The interpretation of the tree T extracted from the set of guards G_t :



according to the value 0 is the set

$$I^0(T) = \{(0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 0, 2), \\ (0, 0, 1, 0), (0, 0, 2, 0), (0, 1, 1, 1), \\ (0, 2, 1, 2), (0, 2, 2, 1)\}.$$

and

$$I(G_t) = \{(0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 0, 2), \\ (0, 0, 1, 0), (0, 0, 2, 0), (0, 1, 1, 1), (0, 2, 1, 2), \\ (0, 2, 2, 1), (1, 1, 0, 0), (1, 1, 0, 1), (1, 1, 0, 2), \\ (1, 1, 1, 0), (1, 1, 2, 0), (1, 2, 1, 1), (2, 2, 0, 0), \\ (2, 2, 0, 1), (2, 2, 0, 2), (2, 2, 1, 0), (2, 2, 2, 0)\}$$

One can remark that for all $(val_t, val_x, val_y, val_z) \in I(G_t)$, the instantiation $[t \leftarrow val_t][x \leftarrow val_x][y \leftarrow val_y][z \leftarrow val_z]$ satisfies the constraint $(x = (y * z) + t)$.

We now define the interpretation of a QCSP base.

Definition 3 (interpretation of a QCSP base). The interpretation function $(\cdot)^*$ of a QCSP base to a QCSP is defined as follows ($Q = q_1 v_1 \dots q_n v_n$) :

$$\begin{aligned} (bl_top)^* &= \top \\ (bl_bottom)^* &= \perp \\ (\langle Q \mid [G_{e_1}, \dots, G_{e_m}] \rangle)^* &= \\ Q \bigwedge_{e_i \in [e_1, \dots, e_m]} ((v_{e_k}, v_1, \dots, v_{e_k-1}) &\in I(G_{e_k})) \end{aligned}$$

The interpretation of a QCSP base is a QCSP but only on table constraints.

Example 3. (Examples 1 and 2 continued.)

$$(B)^* = \exists x \exists y \forall z \exists t ((x \in I(G_x)) \wedge ((y, x) \in I(G_y)) \wedge ((t, x, y, z) \in I(G_t)))$$

with $x, y, z, t \in \{0, 1, 2\}$, $I(G_x) = \{0, 1, 2\}$ and $I(G_y) = \{0, 1, 2\}^2$.

3.3 Properties of QCSP bases

To a given QCSP, many different QCSP bases may be such that their interpretations have exactly the same set of winning strategies as that QCSP. We define this property as the *compatibility property*.

Definition 4 (compatibility of a QCSP base). *A QCSP base is compatible with a QCSP if its interpretation has exactly the same winning strategy.*

In what follows, we will see that the set of compatible QCSP bases may be seen as a good candidate as a target for a compilation language.

Example 4. (Examples 1, 2 and 3 continued.) The QCSP base B is compatible with the QCSP $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ with $x, y, z, t \in \{0, 1, 2\}$. If, for example, the pair $(2, \square)$ of G_x is discarded then the resulting QCSP base is no more compatible with the QCSP since two of the four winning strategies are lost.

The following theorem establishes immediately the completeness of the QCSP base formalism w.r.t. QCSP.

Theorem 1 (completeness). *For every QCSP there exists a compatible base.*

When a QCSP represents a finite two-player game, one of the most important issues for the existential player, at each turn during the game, is the following: “What do I have to play to be certain to win the game?” If a winning strategy has been already computed before the game begins, the player has only to follow it. But if the uncertainty was not completely known and if the current winning strategy can not be applied anymore, the existential player has to compute again a new strategy and has to pay also the complete algorithmic price.

Definition 5 (next move choice problem). *The next move choice problem is defined as follows.*

- *Instance :* A QCSP $q_1 v_1 \dots q_n v_n \bigwedge_{c \in C} c$ with $v_1 \in D_1, \dots, v_n \in D_n$ and a sequence of instantiations $[v_1 \leftarrow val_1], \dots, [v_i \leftarrow val_i]$ obtained from a winning strategy for a QCSP with $\text{quant}(v_i) = \exists$ and $val_1 \in D_1, \dots, val_i \in D_i$.
- *Query :* Is there any winning strategy for a QCSP

$$q_{i+1} v_{i+1} \dots q_n v_n \bigwedge_{c \in C} c \wedge (v_1 = val_1) \wedge (v_{i-1} = val_{i-1}) \wedge (v_i = val'_i)$$

with $v_{i+1} \in D_{i+1}, \dots, v_n \in D_n, val'_i \in D_i, val'_i \neq val_i$?

Clearly enough the next move choice problem is still a PSPACE-complete problem since $q_{i+1} v_{i+1} \dots q_n v_n \bigwedge_{c \in C} c \wedge (v_1 = val_1) \wedge (v_{i-1} = val_{i-1}) \wedge (v_i = val'_i)$ with $v_{i+1} \in D_{i+1}, \dots, v_n \in D_n, val'_i \in D_i, val'_i \neq val_i$ is a QCSP.

We introduce a new property for a QCSP base which guarantees that the next move choice problem is no more PSPACE-complete but polytime w.r.t. the size of the QCSP base. A QCSP base is *optimal* if all the guards associated to the moves played by the existential player are verified then this player is sure to follow a winning strategy.

Definition 6 (optimality). Let $B = \langle q_1 v_1 \dots q_n v_n \mid [G_{e_1}, \dots, G_{e_m}] \rangle$ be a QCSP base and $(B)^* = q_1 v_1 \dots q_n v_n C$ with $v_1 \in D_1, \dots, v_n \in D_n$. This base is optimal if the following property is verified. For every $i, i \in [1 \dots m]$, let C_i be the set of constraints $\{(v_{e_k} = val_{e_k}) \mid 1 \leq k < i\}$ such that $(val_{e_k}, val_{e_1}, \dots, val_{e_{k-1}}) \in I^{val_{e_k}}(a_{e_k}), (val_{e_k}, a_{e_k}) \in G_{e_k}$.

Then for every guard $(val, a) \in G_{e_i}, (val, val_{e_1}, \dots, val_{e_{i-1}}) \in I^{val}(a)$ if and only if $q_{e_i+1} v_{e_i+1} \dots q_n v_n (C \wedge (v_{e_i} = val) \wedge \bigwedge_{c \in C_i} c)$ admits a winning strategy.

The underlying order of this notion of optimality is the number of winning scenarios which are not a branch of any winning strategy. In case of the interpretation of an optimal base this number is zero.

Example 5. The following guard sets G_x^{opt}, G_y^{opt} and G_t^{opt} are guard sets for the QCSP base $B^{opt} = \langle \exists x \exists y \forall z \exists t \mid [G_x^{opt}, G_y^{opt}, G_t^{opt}] \rangle$ which is optimal and compatible with the QCSP : $\exists x \exists y \forall z \exists t (x = (y * z) + t)$ with $x, y, z, t \in \{0, 1, 2\}$.

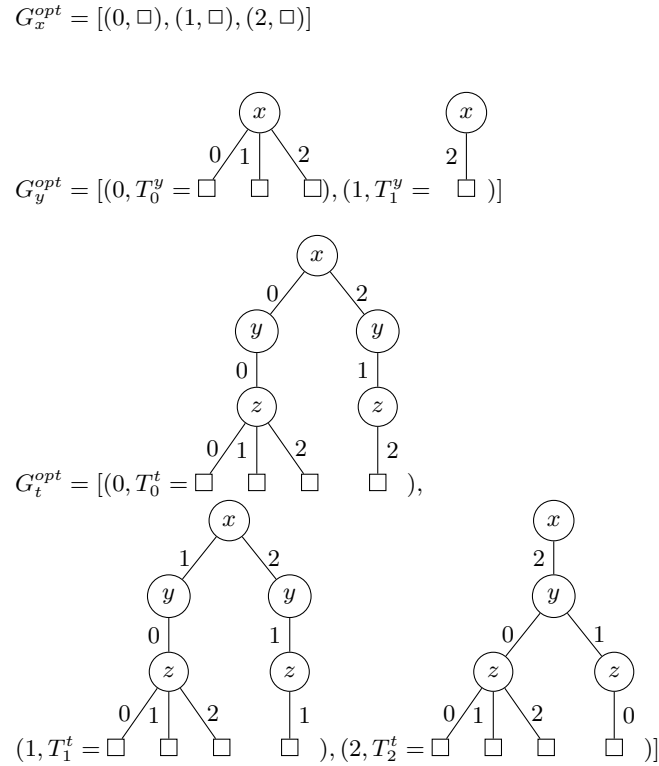


Fig. 1. Guard sets for an optimal QCSP base.

We explicit hereafter the optimality of the QCSP base but we use the QCSP constraint $(x = (y * z) + t)$ instead of the table constraints in order to simplify.

- $i = 1$ (i.e. $v_{e_i} = x$) then $C_i = \emptyset$ and for every $K \in \{0, 1, 2\}$, $K \in I^K(\square) = \{K\}$ and $\exists y \forall z \exists t (x = (y * z) + t) \wedge (x = K)$, with $y, z, t \in \{0, 1, 2\}$, admits a winning strategy.
- $i = 2$ (i.e. $v_{e_i} = y$) then
 - for every $K \in \{0, 1, 2\}$ $(K, \square) \in G_x^{opt}$, $I^0(T_0^y) = \{(0, 0), (0, 1), (0, 2)\}$ and for every $K \in \{0, 1, 2\}$, $(0, K) \in I^0(T_0^y)$ and $\forall z \exists t (x = (y * z) + t) \wedge (y = 0) \wedge (x = K)$, with $z, t \in \{0, 1, 2\}$, admits a winning strategy ;
 - $(1, \square) \in G_x^{opt}$, $I^1(T_1^y) = \{(1, 1)\}$ and $(1, 1) \in I^1(T_1^y)$ and $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 1)$, with $z, t \in \{0, 1, 2\}$, admits winning strategy ; $(1, 0) \notin I^1(T_1^y)$ and $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 0)$, with $z, t \in \{0, 1, 2\}$, does not admit a winning strategy ; $(1, 2) \notin I^1(T_1^y)$ and $\forall z \exists t (x = (y * z) + t) \wedge (y = 1) \wedge (x = 2)$, with $z, t \in \{0, 1, 2\}$, does not admit a winning strategy ;
 - for $y = 2$, there is no pair $(2, T_2^y) \in G_y$ and $\exists x \forall z \exists t (x = (y * z) + t) \wedge (y = 2)$, with $x, z, t \in \{0, 1, 2\}$, does not admit a winning strategy.
- $i = 3$ (i.e. $v_{e_i} = t$) then (we only treat the case $t = 0$, the others are similar)
 - $(2, \square) \in G_x^{opt}$, $(1, T_1^y) \in G_y^{opt}$ with $(1, 2) \in I^1(T_1^y)$ and $(0, 2, 1, 2) \in I^0(T_0^t)$ and $(x = (y * z) + t) \wedge (x = 2) \wedge (y = 1) \wedge (z = 2) \wedge (t = 0)$ admits a winning strategy ; it is similar for $(0, 0, 0, 0)$, $(0, 0, 0, 1)$ and $(0, 0, 0, 2)$;
 - for all the other cases $(0, val_x, val_y, val_z) \notin I^0(T_0^t)$ and $(x = (y * z) + t) \wedge (x = val_x) \wedge (y = val_y) \wedge (z = val_z) \wedge (t = 0)$ does not admit a winning strategy.

The most important property of optimal QCSP base is that the next move choice problem for the interpretation of compatible optimal base with a QCSP is no more PSPACE-complete but polytime.

Theorem 2 (next move choice problem). *The next move decision problem for the interpretation of an optimal base is polytime in the size of the base.*

4 Compilation of a QCSP to an optimal QCSP base

We present in this section an algorithm based on a search algorithm and establish that the result of the application of this algorithm is an optimal QCSP base compatible with the initial QCSP. Algorithm 1 *rec.comp* computes a compatible QCSP base from a QCSP following the inductive definition of the semantics of the QCSP. This algorithm first computes a fix-point for the set of constraints and returns *bl.bottom* if a contradiction is detected. If it is not the case and the binder is not empty then *bl.top* is returned. Otherwise for every value *val* of the domain of the outermost variable *x* of the binder, the constraint $(x = val)$ is added to the constraint store and the algorithm is recursively called. If the variable is universally quantified and at least one subproblem returns *bl.bottom* then *bl.bottom* is returned. If the variable is existentially quantified and all the subproblems return *bl.bottom* then *bl.bottom* is returned. In any other cases, operators \oplus_\exists or \oplus_\forall are called to combine the resulting QCSP bases together.

Algorithm 1 *rec_comp*

In: Q : a binder of a QCSP
In: C : a set of constraints of a QCSP
Out: a QCSP base or *bl_top* or *bl_bottom*

```
if reach_fixpoint( $C$ ) = failure then
  return bl_bottom
end if
if empty( $Q$ ) then return bl_top end if
 $qx_D \leftarrow \text{head}(Q)$ ;  $listValBase \leftarrow []$ ;  $d \leftarrow D$ 
while !empty( $d$ ) do
   $val \leftarrow \text{head}(d)$ ;  $d \leftarrow \text{tail}(d)$ 
   $base \leftarrow \text{rec\_comp}(\text{tail}(Q), C \cup \{x = val\})$ 
  if  $base = bl\_bottom \wedge q = \forall$  then
    return bl_bottom
  end if
   $listValBase \leftarrow [(val, base) | listValBase]$ 
end while
if empty( $listValBase$ ) then
  return bl_bottom
end if
if  $q = \exists$  then
  return  $\oplus_{\exists}(x_D, Q, listValBase)$ 
else
  return  $\oplus_{\forall}(x_D, Q, listValBase)$ 
end if
```

Operators \oplus_{\forall} and \oplus_{\exists} specified respectively by the algorithms 2 and 3 work as follows. First we describe the \oplus_{\forall} operator. Function *constants*(l) checks if the list of pairs as argument l does not contain only *bl_top* or *bl_bottom* for second element. If the check *constants*(l) is verified, it is necessarily only a list of *bl_top* associated with all the values of the domain of the variable in case of an innermost bloc of universal quantifiers and then *bl_top* is returned. Otherwise, the operator \oplus defined hereafter is applied and the result is returned since the universally quantified variables are not associated to guards. Now we describe the \oplus_{\exists} operator. If the check *constants*(l) is verified, it is necessarily an innermost existential quantifier and a QCSP base containing only the values associated to the *bl_top* is built thanks to the function *base_case* defined by $base_case(l) = \{(val, \square) | (val, bl_top) \in l\}$. Otherwise, the returned QCSP base is built by adding to the result of the \oplus operator the list of the values associated to each of the QCSP bases thanks to the function *first_values* defined by $first_values(l) = \{(val, \square) | (val, a) \in l\}$.

The \oplus operator works as follows. The *decompose* function extracts from the list *lvb* of pairs (value, QCSP bases), for the outermost existentially quantified variable y of the binder, a pair constituted of a list of pairs (val, list of guards)) and a list of pairs (val, remaining of the guards). The *compose* function builds for y its set of guards by distributing the trees for the different values. Functions

Algorithm 2 \oplus_{\exists}

In: x_D : a variable and its domain
In: Q : a binder
In: l : a list of pairs (value, QCSP base)
Out: a QCSP base
 if $constants(l)$ **then**
 return $\langle \exists x_D Q \mid case_base(l) \rangle$
 else
 return $\langle \exists x_D Q \mid [first_values(l) \mid \oplus (x, l)] \rangle$
 end if

Algorithm 3 \oplus_{\forall}

In: x_D : a variable and its domain
In: Q : a binder
In: l : a list of pairs (value, QCSP base)
Out: a QCSP base or bl_top
 if $constants(l)$ **then**
 return bl_top
 else
 return $\langle \forall x_D Q \mid \oplus (x, l) \rangle$
 end if

first and *second* give access to respectively the first and the second position of a pair.

Algorithm 4 \oplus

In: x : a variable
In: lvb : a list of pairs (value, QCSP base)
Out: a list of guards
 $lg \leftarrow []$
 $lvg \leftarrow extract_guards(lvb)$
 while $empty(lvg)$ **do**
 $dec_y \leftarrow decompose(lvg)$
 $lg \leftarrow [compose(x, first(dec_y)) \mid lg]$
 $lvg \leftarrow second(dec_y)$
 end while
 return lg

The following example shows how the *rec_comp* algorithm works.

Example 6. We compute for all $val_x, val_y \in \{0, 1, 2\}$ the QCSP base $B_{val_x val_y}$ as the result of the following call :

$$rec_comp(\forall z \exists t, \{(x = (y * z) + t), (x = val_x), (y = val_y)\})$$

with $z, t \in \{0, 1, 2\}$.

We obtain the QCSP bases (according to $T_{val_t}^{val_x val_y}$) :

$$B_{00} = \langle \forall z \exists t \mid [[(0, T_0^{00} = \square \begin{array}{c} \text{0} \swarrow \text{1} \downarrow \text{2} \searrow \\ \square \quad \square \quad \square \end{array})]]] \rangle$$

$$B_{10} = \langle \forall z \exists t \mid [[(1, T_1^{10} = \square \begin{array}{c} \text{0} \swarrow \text{1} \downarrow \text{2} \searrow \\ \square \quad \square \quad \square \end{array})]]] \rangle$$

$$B_{20} = \langle \forall z \exists t \mid [[(2, T_2^{20} = \square \begin{array}{c} \text{0} \swarrow \text{1} \downarrow \text{2} \searrow \\ \square \quad \square \quad \square \end{array})]]] \rangle$$

$$B_{21} = \langle \forall z \exists t \mid [[(0, T_0^{21} = \begin{array}{c} \text{z} \\ \downarrow \text{2} \\ \square \end{array}), (1, T_1^{21} = \begin{array}{c} \text{z} \\ \downarrow \text{1} \\ \square \end{array}), (2, T_2^{21} = \begin{array}{c} \text{z} \\ \downarrow \text{0} \\ \square \end{array})]]] \rangle$$

of for any other combination, $B_{val_x val_y} = bl_bottom$.

The following example shows how the trees are shared by the \oplus operator and also the distribution of the trees.

Example 7. The operator \oplus_{\exists} is applied during the execution of the call

$$rec_comp(\exists y \forall z \exists t, \{(x = (y * z) + t), (x = 2)\})$$

with $y, z, t \in \{0, 1, 2\}$, to the QCSP bases B_{20} , B_{21} and B_{22} which represent compatible QCSP bases with QCSP, respectively, $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 0))$, $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 1))$, $\forall z \exists t((x = (y * z) + t) \wedge (x = 2) \wedge (y = 2))$.

$$\oplus_{\exists}(y, \forall z \exists t, [(0, B_{20}), (1, B_{21}), (2, B_{22})])$$

$$\begin{aligned} &= \langle \exists y \forall z \exists t \mid [[(0, \square), (1, \square)], [(0, \begin{array}{c} \text{y} \\ \downarrow \text{1} \\ T_0^{21} \end{array}), (1, \begin{array}{c} \text{y} \\ \downarrow \text{1} \\ T_1^{21} \end{array}), (2, \begin{array}{c} \text{y} \\ \swarrow \text{0} \quad \searrow \text{1} \\ T_2^{20} \quad T_2^{21} \end{array})]]] \rangle \\ &= B_2 \end{aligned}$$

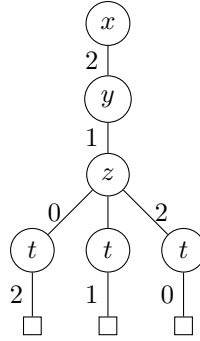
which is a compatible QCSP base with the QCSP $\exists y \forall z \exists t((x = (y * z) + t) \wedge (x = 2))$ with $y, z, t \in \{0, 1, 2\}$.

The following example shows how the existential player can play with an optimal QCSP base instead of only one winning strategy how the optimal QCSP base gives a direct access to the possibilities for a given existentially quantified variable.

Example 8. (Previous examples continued.) Let the following QCSP be the expression of a very simple two-player game:

$$\exists x \exists y \forall z \exists t (x = (y * z) + t), x, y, z, t \in \{0, 1, 2\}$$

Let us suppose that an existential player only knows the following winning strategy:



For his first move, he decides to play ($x = 2$). Now, following its winning strategy, he is supposed to play ($y = 1$). Let us suppose that it is no more possible because of an unexpected reason. He can not follow its winning strategy anymore. If he follows the compatible but not optimal base of Example 1 he will follow one of the two other choices thinking that he still has a chance. If he wants to be sure, he will have to pay the full computational price for the QCSP $\forall z \exists t ((x = (y * z) + t) \wedge (x = 2) \wedge (y = 0)), z, t \in \{0, 1, 2\}$ and $\forall z \exists t ((x = (y * z) + t) \wedge (x = 2) \wedge (y = 2)), z, t \in \{0, 1, 2\}$. If he follows its compatible and optimal QCSP base of Example 5, he knows that he has already lost.

The following theorem establishes that the *rec_comp* algorithm not only computes a compatible QCSP base from a QCSP but also that this QCSP base is optimal.

Theorem 3. *Let QC be a QCSP. $\text{rec_comp}(Q, C)$ return an optimal base and compatible with the QCSP.*

5 Conclusion

We have proposed in this article a framework for the compilation of QCSP: the QCSP bases. We have defined an algorithm embedded in the state-of-the-art search algorithm of QCSP solver to compute a QCSP base from a QCSP.

We have shown that the obtained QCSP base is compatible with the initial QCSP and optimal in the sense that the construction of any winning strategy is polytime for the interpretation of a QCSP base.

We have implemented in Prolog the algorithms described in this article and the programs and examples are downloadable. at the following address <http://www.info.univ-angers.fr/pub/stephan/Research/Download.html>. We plan to integrate it in our QCSP solveur developed in the generic constraint development environment Gecode [15].

When a QCSP solver returns there is or there is no winning strategy, there is no way to check if the answer is correct while for CSP the associated result to the decision (a complete instantiation) is easy to check. A certificate for a QCSP which has a winning strategy is any piece of information that provides self-supporting evidence of the existence of a winning strategy for that QCSP. Due to the lack of space, we have not treated certificates for QCSP: our formalism includes QCSP certificates as a particular case. During the execution of the solver, a QCSP base is generated but only with strategies as guards. The interpretation of these trees (cf Definition 2) in tuples permits to verify such a QCSP certificate w.r.t. a QCSP by the resolution of a co-NP-complete problem. (The complexity is similar to the check of a winning strategy for QBF [4].)

We have proposed a recursive construction of the QCSP base but in practice it is often more efficient to consider a cooperation between a solver which emits a trace and a trace analyzer which builds the QCSP base. We have also develop this approach but due to the lack of space we only give here the main two important reasons for this cooperation between a solver and a trace analyzer: If the construction of the QCSP base is embedded into the solver, the memory management of the solver by means of a backtrack stack will have also to keep the state of the current QCSP base. We want to take into account modern architectures with multi-core multithreaded processors.

References

1. J. Amilhastre, H. Fargier, and P. Marquis. Consistency Restoration and Explanations in Dynamic CSPs - Application to Configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002.
2. S. Baba, Y. Joe, A. Iwasaki, and M. Yokoo. Real-Time Solving of Quantified CSPs Based on Monte-Carlo Game Tree Search. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 655–661, 2011.
3. F. Bacchus and K. Stergiou. Solution directed backjumping for QCSP. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, pages 148–163, 2007.
4. M. Benedetti. Extracting Certificates from Quantified Boolean Formulas. In *Proceedings of 9th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 47–53, 2005.
5. M. Benedetti, A. Lallouet, and J. Vautard. Reusing CSP propagators for QCSPs. In *Recent Advances in Constraints, 11th Annual ERCIM International, Workshop on Constraint Solving and Constraint Logic Programming (CSCLP'06)*, pages 63–77, 2006.

6. L. Bordeaux, M. Cadoli, and T. Mancini. CSP Properties for Quantified Constraints: Definitions and Complexity. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, pages 360–365, 2005.
7. L. Bordeaux and E. Monfroy. Beyond NP: Arc-Consistency for Quantified Constraints. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, pages 371–386, 2002.
8. S.C. Brailsford, C.N. Potts, and B.M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119:557–581, 1999.
9. H. Chen. The Computational Complexity of Quantified Constraint Satisfaction, PhD thesis, Cornell University, 2004.
10. S. Coste-Marquis, D. Le Berre, F. Letombe, and P. Marquis. Complexity Results for Quantified Boolean Formulae Based on Complete Propositional Languages. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:61–88, 2006.
11. H. Fargier and P. Marquis. On the Use of Partially Ordered Decision Graphs in Knowledge Compilation and Quantified Boolean Formulae. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI'06)*, 2006.
12. I. Gent, P. Nightingale, and K. Stergiou. QCSP-solve: A solver for quantified constraint satisfaction problems. In *Proceedings of 9th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 138–143, 2005.
13. I.P. Gent, P. Nightingale, and A. Rowley. Encoding Quantified CSPs as Quantified Boolean Formulae. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 176–180, 2004.
14. A.K. Mackworth. Consistency in networks of relations. In *Artificial Intelligence*, volume 8, pages 99–118, 1977.
15. C. Schulte and G. Tack. Views and Iterators for Generic Constraint Implementations. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, pages 817–821, 2005.
16. I. Stéphan and B. Da Mota. A unified framework for Certificate and Compilation for QBF. In *Proceedings of the 3rd Indian Conference on Logic and its Applications*, pages 210–223, 2009.
17. D. Stynes and K.N. Brown. Realtime Online Solving of Quantified CSPs. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP'09)*, pages 771–786, 2009.
18. G. Verger and C. Bessière. BlockSolve : une approche bottom-up des QCSP. In *Actes des Deuxièmes Journées Francophones de Programmation par Contraintes (JFPC'06)*, pages 337–345, 2006.