On the Equivalence of Constraint Satisfaction Problems

Francesca Rossi[1]
Charles Petrie
MCC
3500 W. Balcones Cntr. Dr.
Austin, Texas 78759
rossi, petrie@mcc.com

Vasant Dhar[2]
Dept. of Information Systems
New York University
40 West 4th Street
NY, NY 10003
dhar@vx1.gba.nyu.edu

[1]On leave from the Computer Science Department of the University of Pisa, Pisa, Italy.
[2]Part of this work was done while this author was visiting MCC.

Abstract

A solution of a Constraint Satisfaction Problem (CSP) is an assignment of values to all its variables such that all its constraints are satisfied. Usually two CSPs are considered equivalent if they have the same solution set. We find this definition limiting, and develop a more general definition based on the concept of mutual reducibility. In this extended scheme it is reasonable to consider a pair of CSPs equivalent even if they have different solutions. The basic idea behind the extended scheme is that two CSPs can be considered equivalent whenever they contain the same "amount of information", i.e. whenever it is possible to obtain the solution of one of them from the solution of the other one, and viceversa. In this way, both constraint and variable redundancy are allowed in CSPs belonging to the same equivalence class.

As an example of the usefulness of this new notion of equivalence, we formally prove that binary and non-binary CSPs are equivalent (in the new sense). Such a proof is not possible with the usual notion of equivalence. Two different algorithms, currently used for transforming any non-binary CSP into an equivalent binary one, are described. It turns out that only one of them produces a binary CSP equivalent to the given non-binary problem, while the other one can achieve the transformation only at the cost of adding some new arbitrary information.

1 Introduction

A Constraint Satisfaction Problem (CSP) consists of a set of variables which can be instantiated on a finite domain and are subject to a set of constraints. In this paper we investigate the concept of equivalence of CSPs.

A first, naive definition of equivalence of CSPs could be:

two CSPs are equivalent if they have the same variables and the same constraints.

This approach to equivalence is very restrictive. For example, it does not consider the possibility of redundant information in the constraints. Such redundancy, in fact, would allow to obtain, starting with a given CSP, an infinite number of distinct CSPs by simply taking the given one and adding redundant information. Since all the resultant CSPs represent the same problem, they should be equivalent. However, the above given definition is not satisfied.

To overcome this problem, we could modify the definiton of equivalence to deal with constraint redundancy by adopting the following definition:

two CSPs are equivalent if they have the same solution

where the intended meaning of solution of a CSP is the set of all assignments of the variables on the domain such that all the constraints are satisfied. This is the usual definition of equivalence of CSPs. Here, all the redundant CSPs (with respect to the same given one) are considered equivalent. However, redundancy can occur not only in the constraints, but also in the number of variables, i.e., it is possible that two CSPs have different variables all behaving exactly in the same way (that is, only the names of the variables are different). Such problems, while syntacticly different, should be considered equivalent since their information content is essentially identical. In other words, it is only the information content of a CSP that should matter. Accordingly, we propose the following new definition of equivalence of CSPs:

two CSPs are equivalent if they are mutually reducible where we say that P_1 is reducible to P_2 if it is possible to obtain the solution of P_1 from the solution of P_2, by mapping the variable and the values in one problem to the variables and the values in the other problem. This definition allows both constraint and variable redundancy, and it can be proved to be more general than the previous definitions of equivalence.

In this paper we adopt this new definition of equivalence of CSPs and we use it to prove formally that binary and non-binary CSPs are equivalent in this sense. Such a result is useful not only from the theoretical point of view, but also because many efficient solution algorithms have been developed for binary CSP only. Thus it is important to be able to state the problem at hand as a non-binary one if necessary or more convenient, transform it into a binary CSP, solve it (the latter), and finally be able to go from that to the solution of the given problem. The proof that this is possible cannot be obtained without the extended notion of equivalence.

In the literature on CSPs, the issue of the equivalence of binary and non-binary CSPs has been addressed very vaguely. The first formal attempt to compare these two different representations of a problem appeared in [2], where there is an example of an n-ary constraint which cannot be represented in terms of binary constraints between pairs of those n variables. Thus, binary and non-binary constraints seemed not to be equivalent in general: more precisely, non-binary constraints seemed to be more powerful than binary ones. In reality, this is true only if we put restrictions on the number of variables to be used in the transformed problem, or on the shape of its representation.

This paper shows that binary and non-binary constraints are equivalent by considering two transformations, well known in the constraint satisfaction "folklore", to transform a non-binary CSP into an equivalent binary one.

The first approach is to represent any non-binary constraint connecting k variables with a star-like set of k binary constraints, each one connecting a new variable and one of the original k variables. The role of the new variable is to maintain the relationship, previously represented by the non-binary constraint, among the original k variables. The two problems look very similar. However, not only are they not equivalent in the usual sense, but also, using the new definition of equivalence of CSPs described above, it is possible to show that the non-binary CSP is reducible to the binary one, but that the opposite does not hold. Thus the two problems are not even equivalent in the new sense. The intuitive reason is that the binary CSP is built from the given non-binary one by adding a variable, values for its domain, and some constraints. While the binary constraint definition can automatically be generated by the definition of the original k-ary constraint, the variable and the domain are generated by arbitrary choices which cannot be inverted to yield the original problem.

Thus, we describe a second approach which can be proved to produce an equivalent binary CSP from a given non-binary one. This approach takes inspiration from Dechter and Pearl ([1]). While trying to represent any CSP in an acyclic form (hoping that this is easier to solve), they state informally that any non-binary CSP can be represented as a binary one by considering its dual constraint graph. The idea is that while a CSP is usually represented as a graph whose nodes represent variables and whose hyperarcs represent constraints, in the dual graph nodes are constraints and arcs represent shared variables. We prove that this new graph can always be seen as the representation of a binary CSP which is equivalent to the original one in accordance with our new definition: the given non-binary CSP and the obtained binary one are mutually reducible.

As mentioned above, the formal proof of the equivalence of binary and non-binary CSPs is not possible with the usual notion of equivalence. However, it has been formally proved, in the field of philosophic logic ([4]), that binary and non-binary predicates have the same power (by showing that the approach which adds variables to produce a binary CSP is logically equivalent to the original representation) . Thus, one approach could be the transformation of each constraint into a predicate, which is possible, and the use of logic tools to obtain a formal proof. This however would mean that the CSP theory is incomplete in an important sense: the equivalence of CSPs could not be generally expressed within the theory. The definition of CSP equivalence proposed in this paper provides an elegant way to eliminate this incompleteness.

In summary, the new definition of equivalence of CSPs provides two different fundamental contributions to the constraint satisfaction theory:

- it allows the presence of variable redundancy (besides constraint redundancy, which was already allowed by the usual notion) in CSPs belonging to the same equivalence class, thus giving a more intuitive understanding of the notion of pairs of CSPs with the same amount of information. In fact, there are only two different ways to store information into a CSP, as a variable and as a constraint, and our definition permits redundancy in both of them.

- it removes a source of incompleteness from the already existing constraint satisfaction theory. This allows proof of an informally known result within the constraint framework.

The paper is organized as follows. Section 2 gives basic definitions about CSPs. Section 3 introduces our new definition of equivalence, proves some properties about it, and gives examples of pairs of CSPs which become equivalent by using this new definition. Section 4.1 and 4.2 describe respectively the first and the second approach to transforming a non-binary CSP into a binary one, giving also some examples and proving the relationship (equivalence or not) between the obtained problem and the original one. Finally section 5 summarizes the contributions of the paper.

2 Constraint Satisfaction Problems

In this section we will give the basic definitions of constraints and constraint satisfaction problems (CSPs). In doing that, we will basically use the same notation as in [6] and [5].

Definition 1 (k-ary constraint) *A k-ary constraint connecting variables $x_1, \ldots, x_k$ having domains $D_1, \ldots, D_k$ respectively is defined as a subset of the cartesian product $D_1 \times \ldots \times D_k$, and it is intended as the set of allowed k-tuples for these k variables. A 2-ary constraint is called binary. A 1-ary constraint is called unary.*■

Note that in this paper we will only consider CSPs where the variables have a finite domain, otherwise the above definition would not be reasonable, i.e. it wouldn't be possible to represent each constraint in an extensive fashion.

It is not restrictive to consider the same domain for all the variables. In fact, if the variables have different domains, then it is always possible to consider the union of all of them as a unique domain. Thus in the following we will always consider only one domain.

Definition 2 (CSP) *A constraint satisfaction problem (CSP) $P =< V, D, C, con, def >$ is:*

- *a set of variables $V = \{v_1, \ldots, v_n\}$;*
- *a domain D for the variables in V;*
- *a set of constraints $C = \{c_1, \ldots, c_m\}$. C is a ranked set, i.e. $C = \bigcup_k C_k$, such that $c \in C_k$ if c connects k variables;*
- *a connection function*
$$con : \bigcup_k (C_k \rightarrow V_k),$$
where $con(c) =< v_1, \ldots, v_k >$ is the tuple of variables connected by c;
- *a definition function*
$$def : \bigcup_k (C_k \rightarrow \wp(D^k)),$$
where $\wp(D^k)$ is the powerset of D^k, such that $def(c)$ is the set of k-tuples of values allowed for the variables connected by c.$\blacksquare$

Definition 3 (binary CSP) *A CSP is called binary if all its constraints are either unary or binary.*$\blacksquare$

Definition 4 (Solution of a CSP) *The solution of a CSP P, $Sol(P)$, is the set of all the tuples of values for all the variables of P such that all the constraints are satisfied, i.e such that, for each such tuple, the projection of the tuple on the subset of variables connected by each constraint belongs to the definition of that constraint.*$\blacksquare$

3 Extended Equivalence of CSPs

We first give the usual notion of equivalence and then we extend it by introducing the new extended equivalence, which relies on the concept of mutual reducibility.

Definition 5 (Usual equivalence of CSPs) *Two CSPs P_1 and P_2 are equivalent (in the usual sense) iff $Sol(P_1) = Sol(P_2)$. We will write $P_1 \equiv_u P_2$.*■

¿From the above definition, it is clear that two CSPs with different sets of variables cannot be equivalent in the usual sense, because their solutions are always different. The same observation holds also for two CSPs with different domain structure (if one domain contains values and the other one tuples of values, they are said to have a different domain structure). In fact, if the elements in the two domains have different "shape", obviously the solutions, being sets of assignments on the given domain, cannot coincide.

Thus, we will now extend the usual notion of equivalence, because it is possible for two problems to not have the same solution but still be representations of the same world.

Definition 6 (Extended equivalence of CSPs) *Two CSPs P_1 and P_2 are equivalent (in the extended sense) if P_1 is reducible to P_2 and P_2 is reducible to P_1. We will write $P_1 \equiv_e P_2$.*■

Let us now see what we mean by "reducibility". The idea is that a CSP, P_1, is reducible to another one, P_2, if there is way to go from the solution of P_2 to the solution of P_1, by mapping variables and values in P_2 into variables and values in P_1.

Definition 7 *Given a set S, we will write $t(S)$ for the set of elements e defined in the following way:*

- *$e \in t(S)$ for all $e \in S$;*
- *$< e_1, \ldots, e_n > \in t(S)$ for all $n \geq 1$ and for all $e_i \in t(S)$.*■

In other words, the set $t(S)$ is the set of all the tuples of elements of S and of all the tuples of tuples of elements of S, and so on with any level of tuple nesting. In the following definition of reducibility we will apply the above operation to sets of variables (usually named V_i for some i) and also to sets of parameters (PAR_i).

Definition 8 (reducibility of one CSP to another) *Given two CSPs P_1 and P_2, P_1 is reducible to P_2 if there exists a pair of functions $F = (f_1, f_2)$, such that*

- $f_1 : V_1 \rightarrow t(V_2)$, *and*
- $f_2 : (V_1 \times t(PAR_2)) \rightarrow t(PAR_1)$, *where* PAR_1 *is a set of parameters ranging over* D_1, *while* PAR_2 *contains parameters ranging over* D_2, *and*
- $f_2(x, p_2) = p_1$, *where* $p_1 \in t(PAR_1)$ *and* $p_2 \in t(PAR_2)$, *if and only if the value of* x *in each solution of* P_1 *is* p_1 *whenever the value of* $f_1(x)$ *in each solution of* P_2 *is* p_2.$\blacksquare$

The meaning of the two functions f_1 and f_2 is the following: f_1 maps variables of P_1 to tuples of variables of P_2, in a way that writing $f_1(x) = y$ means that to know the value of x in a solution of P_1 we have to look at the value of y in a solution of P_2. On the other hand, f_2 tells how those values are related. In fact, as stated in the definition, writing $f_2(x, p_2) = p_1$ means that x assumes the value p_1 whenever y (the variable corresponding to x according to f_1) assumes the value p_2. Note that p_1 and p_2 *are not values of the domains but parameters to be instantiated on the domain,* thus eliminating the possibility of a semantic relationship between them. In particular, it is not possible to write $f_2(x, p_1 + 1) = p_1$, because $p_1 + 1$ is not in $t(S)$ for any set S, since it is not a tuple. Also, it is not possible to write $f_2(x, < 3, 4 >) = p_1$ either, since 3 and 4 are not parameters but values of a domain. Let us clear the concept of reducibility with the following example.

Example 1 *: Consider a CSP* $P_1 =< \{x, y, z\}, \{a, b, c\}, \{c\}, con, def >$, *where* $con(c) =< x, y, z >$ *and* $def(c) = \{< a, b, c >\}$. *Consider also a CSP* $P_2 =< \{x', y', z', v\}$, $\{a, b, c, d\}$, $\{c_1, c_2, c_3\}$, con, def $>$, *where* $con(c_1) =< x', v >$ *and* $def(c_1) = \{< a, d >\}$, *and* $con(c_2) =< y', v >$ *and* $def(c_2) = \{< b, d >\}$, *and* $con(c_3) =< z', v >$ *and* $def(c_3) = \{< c, d >\}$. P_1 *and* P_2 *are represented in the following figure.*

In this figure and in all the following ones, variables are inside circles, binary constraints are represented by arcs (straight lines between two variables) and non-binary constraints are squared objects connected to all the involved variables through lines. Sometimes, besides the name of the constraints, also their definition will be written near the line (or the box) representing the constraint.

P_1 *is reducible to* P_2 *because of the existence of the following two functions:*

- $f_1 : \{x, y, z\} \rightarrow \{x', y', z', v\}^+$, *such that:*
  - $f_1(x) = x'$,
  - $f_1(y) = y'$,
  - $f_1(z) = z'$.
- $f_2 : (\{x, y, z\} \times \{p_2\}) \rightarrow \{p_1\}$, *such that:*
  - $f_2(x, p) = p$,
  - $f_2(y, p) = p$,
  - $f_2(z, p) = p$.

The meaning of the above function definitions is that, in all the solutions of P_1, x (resp. y, and z) assumes the same value as x' (resp. y', and z') does in P_2. Or, equivalently, that the solution of P_1 can be obtained from the solution of P_2 by dropping the value of the variable v.$\blacksquare$

If P_1 is reducible to P_2, then P_2 has at least as much information as P_1, if not more. In fact, only in this case could we obtain the solution of P_1 from that of P_2. Thus, if P_1 and P_2 are mutually reducible, then it is fair to conclude that they contain the same amount of information. That's why they should be considered equivalent.

Note that the two CSPs of the previous example are not equivalent in the extended sense, because there is no way to reduce P_2 to P_1, i.e. given any function f_1 that maps the variables of P_2 to variables or tuples of variables of P_1, there is no way to find a function f_2 that respects Definition 8.

Note also that we allow the correspondence between a variable of a problem and a tuple of variables of another problem. This is, among other reasons, because we admit the possibility of structured domains, i.e. domains whose elements are tuples of values instead of, as usually, values. Let us consider the following example.

Example 2 *: Let $P_1 =< \{x, y, z, v\}, \{a, b, c, d\}, \{c_1, c_2\}, con, def >$, where $con(c_1) =< x, y, z >$ and $def(c_1) = \{< a, b, c >\}$, and $con(c_2) =< y, v >$ and $def(c_2) =< c, d >$.*

Also, let $P_2 =< \{w_1, w_2\}, \{a, b, c, d\}, \{c\}, con, def >$*, where* $con(c) =< w_1, w_2 >$ *and* $def(c) =$
$\{<< a, b, c >, < c, d >>\}$. P_1 *and* P_2 *are represented in the following figure.*

P_1 *can be reduced to* P_2 *as follows:*

- $f_1 : \{x, y, z, v\} \rightarrow \{w_1, w_2\}^+$*, such that:*
  - $f_1(x) = w_1$,
  - $f_1(y) = w_1$,
  - $f_1(z) = w_1$,
  - $f_1(v) = w_2$.
- $f_2 : (\{x, y, z, v\} \times \{p_2\} \rightarrow \{p_1\}$*, such that:*
  - $f_2(x, < par_1, par_2, par_3 >) = par_1$,
  - $f_2(y, < par_1, par_2, par_3 >) = par_2$,
  - $f_2(z, < par_1, par_2, par_3 >) = par_3$,
  - $f_2(v, < par_1, par_2 >) = par_2$.

Also, P_2 *can be reduced to* P_1 *as follows:*

- $f_1 : \{w_1, w_2\} \rightarrow \{x, y, z, v\}^+$*, such that:*
  - $f_1(w_1) =< x, y, z >$,
  - $f_1(w_2) =< y, v >$.
- $f_2 : (\{w_1, w_2\} \times \{p_2\} \rightarrow \{p_1\}$*, such that:*
  - $f_2(w_1, < par_1, par_2, par_3 >) =< par_1, par_2, par_3 >$,
  - $f_2(w_2, < par_1, par_2 >) =< par_1, par_2 >$.

Thus P_1 *and* P_2 *are equivalent in the new sense of Definition 6.*

Note that the construction of f_1 *and* f_2 *such that* P_1 *is reducible to* P_2 *is not unique. In fact, we could have* $f_1(y) = w_2$ *(instead of* $f_1(y) = w_1$*), and then* $f_2(y, < par_1, par_2 >) = par_1$. *The reduction would have been different, but it would have been correct as well.*■

To prove that our new definition really describes an equivalence relation, we have to show that it is reflexive, symmetric, and transitive.

Lemma 1 (reflexivity) *The extended equivalence is reflexive, i.e.* $P \equiv_e P$ *for all CSPs* P.

Proof: Given P, we define f_1 and f_2 as follows:

- $f_1(x) = x$ for all x in P;
- $f_2(x, p) = p$ for all x in P.

The functions f_1 and f_2 so defined respect Definition 8, thus P and P are equivalent.■

Lemma 2 (symmetry) *The extended equivalence is symmetric, i.e.* $P_1 \equiv_e P_2$ *if and only if* $P_2 \equiv_e P_1$*, for all CSPs* P_1 *and* P_2.

Proof: Obvious from Definition 6.■

Lemma 3 (transitivity) *The extended equivalence is transitive, i.e.* $P_1 \equiv_e P_2$ *and* $P_2 \equiv_e P_3$ *implies* $P_1 \equiv_e P_3$*, for all CSPs* P_1, P_2, P_3.

Proof: If $P_1 \equiv_e P_2$, then there exists a function $f_1 : V_1 \rightarrow t(V_2)$. Also, because of the equivalence of P_2 and P_3, there exists a function $f_1' : V_2 \rightarrow t(V_3)$. Consider now the composition of such functions, i.e. $f_1 \circ f_1'$. By definition of function composition, we have that $f_1 \circ f_1' : V_1 \rightarrow t(t(V_2))$. But by Definition 7 it is easy to see that $t(t(S)) = t(S)$ for any set S, so $f_1 \circ f_1' : V_1 \rightarrow t(V_2)$. Also, let us consider f_2 and f_2' similarly. Because of the totality of V_1 and V_2 respectively, it is easy to see that $f_2 \circ f_2' : (V_1 \times \{p_3\}) \rightarrow \{p_1\}$, where p_1 ranges over D_1 and p_3 over $t(D_3)$. Also, by looking at the definitions of f_2 and f_2', we can deduce that $f_2 \circ f_2'(x, p_3) = p_1$ if and only if the value of x in each solution of P_1 is p_1 whenever the value of $f_1 \circ f_1'(x)$ in each solution of P_3 is p_3. Thus we can set $F = (f_1 \circ f_1', f_2 \circ f_2')$, and we have that P_1 can be reduced to P_3. Similar reasoning can lead to the construction of two functions such that P_3 is reducible to P_1. Thus $P_1 \equiv_e P_3$.■

Thus the following theorem obviously holds.

Theorem 1 *The extended equivalence, as defined in Definition 6, is an equivalence relation.*

Proof: Follows from Lemmas 1, 2, and 3.$\blacksquare$

Note also that any pair of CSPs which are equivalent in the old definition are still equivalent in the new definition, so no information is lost. To show that, it's enough to propose a pair of functions f_1 and f_2 which respect Definition 6 but map one solution to a concident one. The definition that we propose is as follows:

- $f_1(x) = x'$, for all x in P_1 and x' in P_2;
- $f_2(x,p) = p$, for all x in P_1 and any parameter p.

This means that for every variable x in P_1 there is a variable x' in P_2 which assumes the same value in all the solutions. Thus such solutions of P_1 and P_2 coincide. Thus the following corollary obviously holds.

Corollary 1 ($\equiv_e$ extends $\equiv_u$) *Given two CSPs P_1 and P_2, if $P_1 \equiv_u P_2$, then $P_1 \equiv_e P_2$.*$\blacksquare$

Furthermore, the new equivalence is a strict extension fo the old one, i.e there are pairs of CSPs which are not equivalent according to the usual definition, but they are now with the new one, as shown by the above example. However, the new definition does not seem too loose, because only the pairs of CSPs which intuitively describe the same situation, at least in a syntactic sense, turn out to be equivalent in the extended sense. The following example shows a pair of CSPs which are not equivalent according to our definition.

Example 3 *: Let $P_1 =< \{x\}, \{a,b\}, \{c\}, con, def >$, where $con(c) =< x >$, and $def(c) = \{< a >, < b >\}$. Also, let $P_2 =< \{v_1, v_2\}, \{a,b\}, \{c\}, con, def >$, where $con(c) =< v_1, v_2 >$ and $def(c) = \{< a, b >\}$ (note that con, def and all the constraint and variable names are local to each CSP). P_1 and P_2 are represented in the following figure.*

Now, even if the solutions of the two CSPs contain the same set of values, i.e. $\{a,b\}$, there is no way to define f_1 and f_2 so to respect Definition 8, because of the parameter restcriction on f_2. In fact, if we set $f_1(x) = v_1$ (resp. $f_1(x) = v_2$), then f_2 cannot bring *the value b (resp. a) from v_1 (resp. v_2) to x. If we set $f_1(x) =< v_1, v_2 >$, f_2, being deterministic and parametric, cannot cannot produce both values a and b as different alternatives for x.*■

As mentioned above, the original idea behind the extension of the usual notion of equivalence of CSPs was that we wanted to allow variable redundancy (besides constraint redundancy) in the same equivalence class. It is easy to see that our new definition of equivalence succeeds in this goal, i.e. P_1 is equivalent to any other problem P_2 if P_2 is P_1 plus some redundant variables (or constraint, of course, since $\equiv_e$ subsumes $\equiv_u$). The following simple example should clarify the notion of variable redundancy.

Example 4 *Consider $P_1 =< \{x\}, \{a\}, \{c\}, con, def >$, where $con(c) =< x >$ and $def(c) = \{< a >\}$. Also, $P_2 =< \{y, z\}, \{a\}, \{c_1, c_2\}, con.def >$, where $con(c_1) =< y >$, $con(c_2) =< z >$, $def(c_1) = \{< a >\}$, and $def(c_2) = \{< a >\}$. P_1 and P_2 can be seen in the following figure.*

Now, P_1 and P_2 are equivalent since P_1 can be reduced to P_2 (we can define f_1 and f_2 in the following way: $f_1(x) = y$, $f_2(x, p) = p$) and P_2 can be reduced to P_1 ($f_1(y) = x$, $f_2(y, p) = p$).

Note that the redundant part of the problem does not need be disconnected from the rest. In fact, consider $P_3 =< \{v, w\}, \{a\}, \{c\}, con, def >$, where $con(c) =< v, w >$ and $def(c) = \{< a, a >\}$ (see following figure).

P_3 is obviously equivalent to P_2, and thus also to P_1 by transitive property.■

Note, however, that the two CSPs P_1 and P_2 of Example 1 are not equivalent. In fact, even if P_2 seems to be P_1 with the addition of a variable v, this variable is not redundant because its information content (the value d) is not embedded in the other variables.

However, such situations can be seen from a different point of view by giving a new definition of solution of a CSP, which allows the possibility of *hidden variables.* More precisely, a solution of a CSP P can be defined as the instantiation of *some* of the variables such that *all* the contraints are satisfied (see [6, 3] for a formal definition). The *hidden* variables are of course the variables not appearing in the solution. Our extended definition of equivalence of CSPs can straightforwardly be restated for CSPs with this new definition of solution. Informally, the idea would be to say that two problems are equivalent if they contain the same information in the non-hidden variables. Thus, problems P_1 and P_2 of Example 1 could be equivalent if the variable v in P is hidden.

A deeper understanding of our new notion of equivalence of CSPs can be gained in the next section, where we will use it to prove (or disprove) the equivalence between the non-binary and the binary CSP which are the input and output respectively of two known transformation algorithms.

4 Extended Equivalence of Binary and Non-binary CSPs

We will now use the new notion of extended equivalence of CSPs proposed in the previous section to show that binary and non-binary CSPs have the same expressive power. The idea is to build a binary CSP P_2 from a given non-binary one P_1, and then to prove, if possible, that $P_1 \equiv_e P_2$. In the following two subsections we will describe and compare two different well-known approaches to the implementation of this idea.

Note that the proof of the equivalence of binary and non-binary CSPs does not imply that one of those two formalisms has to be abandoned. In fact, many problems are better expressible in one of the formalisms and trying to expresss them in the other one would be very unnatural. Also, even though many efficient solution algorithms have been developed for binary CSPs in the past (thus suggesting that the best way is to always translate a non-binary CSP into a binary one before any processing), there has also been a successful attempt to develop a theory of efficient relaxation and solution algorithms for non-binary CSPs ([6, 3]).

4.1 First Approach: Addition of Variables

In this section, we show that given a k-ary constraint, it is possible to obtain a set of binary constraints which expresses at least the same information contained in the original constraint. Consider a CSP $P =< V, D, C, con, def >$ such that

- $V = \{v_1, \ldots, v_k\}$;
- $C = \{c\}$;
- $con(c) =< v_1, \ldots, v_k >$.

In other words, P has k variables and only one k-ary constraint connecting all of them. We want now to obtain from P an equivalent binary CSP, let us call it $B(P)$.

Let us define $B(P) =< V_b, D_b, C_b, con, def >$ such that:

- $V_b = V$;
- $D_b = D$;
- $C_b = \{c_{ij}, 1 \leq i, j \leq k, i \neq j\}$;
- $con(c_{ij}) =< v_i, v_j >$ for all $c_{ij} \in C_b$;
- $def(c_{ij})$ is the projection of $def(c)$ on the variables v_i and v_j, for all $c_{ij} \in C_b$.

That is, $B(P)$ is obtained from P by projecting the constraint in P on each pair of the original k variables.

Example 5 *Consider a CSP P as defined above where $k = 4$, i.e., P has four variables, all connected by the constraint c. In this case, both P and $B(P)$ can be seen in the following figure.*

In this figure, the definition of each constraint c_{ij} of $B(P)$ is the projection of the definition of c onto the variables v_i and v_j.$\blacksquare$

We will now try to find some relationship between the solutions of P and $B(P)$.

Theorem 2 *Given P as described above, we have that $Sol(P) = def(c)$.*

Proof: Straightforward from Definition 4 (for the solution of P) and Definition 2 (for $def(c)$).$\blacksquare$

If $Sol(P) = Sol(B(P))$, then we have found an equivalent (in the usual sense) binary representation for the given k-ary constraint c of P. However, it has been shown in [2] that in general $Sol(P) \neq Sol(B(P))$. Now, the problem really is not the fact that the solutions are different, but that it is always $Sol(B(P)) \subseteq Sol(P)$, which means that $B(P)$ contains less information than P.

The idea for solving this problem is to build a new binary CSP, say $B'(P)$, by relaxing the assumption that the binary CSP has to have the same number of variables as the non-binary one. The aim is to be able to encode in $B'(P)$ at least the same amount of information that P contains. Note that $B'(P)$ cannot be equivalent to P in the usual sense because of the added variables. The question is now if P and $B'(P)$ are equivalent even in the extended case.

As mentioned above, this transformation (from P to $B'(P)$) is already known in the CSP "folklore", together with the informal (and wrong) idea that P and $B'(P)$ are equivalent. The following figure illustrates the key idea of this new transformation.

The new variable, say v_{k+1}, has as many elements in its domain as the number of tuples defining the original constraint c. The new problem has k binary constraints, $c_1, \ldots, c_k$, each one (say c_i) connecting v_i and v_{k+1}. More precisely, if n tuples define constraint c, say $t_1, \ldots, t_n$, then we introduce $d_1, \ldots, d_n$ as the domain of v_{k+1}. For each tuple $t_j =< t_1j, \ldots, t_kj >$ in the definition of c, we have $< t_1j, d_j >$ in the definition of c_1, $\ldots$, $< t_kj, d_j >$ in the definition of c_k. In this way, the old variables are linked together indirectly by the values of the new variable and are constrained to assume just those values allowed by $Sol(P)$.

We will now formally describe the binary CSP $B'(P)$. Given the CSP $P =< V, D, C, con, def >$ as described before, suppose that $def(c) = \{t_1, \ldots, t_n\}$, i.e. that the constraint c is defined by n k-tuples of values of D.

We define $B'(P)$ as follows: $B'(P) =< V_+, D_+, C_+, con_+, def_+ >$ such that:

- $V_+ = V \bigcup \{v_{k+1}\}$;
- $D_+ = D \bigcup \{d_1, \ldots, d_n\}$, where $d_1, \ldots, d_n$ are any n distinct values corresponding to the distinct tuples $t_1, \ldots, t_n$;
- $C_+ = \{c_1, \ldots, c_k\}$;

- $con_+(c_i) =< v_i, v_{k+1} >$;
- $def_+(c_i) = \{(t_{ji}, d_j)$, where $j = 1, \ldots, n$ and t_{ji} is the projection of k-tuple t_j on variable v_i (from $def(c)$)$\}$.

We will now prove that P is reducible to $B'(P)$, which means that $B'(P)$ has at least as much knowledge as P.

Theorem 3 *P is reducible to $B'(P)$.*

Proof: Given $P =< V, D, C, con, def >$ and $B'(P) =< V_+, D_+, C_+, con_+, def_+ >$, we can define the two functions f_1 and f_2 in the following way:

- $f_1 : V \to V_+$, with $f_1(x) = x$ for all x in V (note that $V_+ \in t(V_+)$, so our definition of f_1 respects Definition 8);
- $f_2 : (V \times \{p\}) \to \{p\}$, with $f_2(x, p) = p$ for all x in V.

Thus, by Definition 8, P is reducible to $B'(P)$.$\blacksquare$

occurs in

The following theorem shows that the opposite is not true, i.e. that $B'(P)$ is not reducible to P. This means that $B'(P)$ contains strictly more information than P. Thus it is not correct to consider these two problems as equivalent, not even in the extended sense.

Theorem 4 *$B'(P)$ is not reducible to P.*

Proof: There is no way to define f_1 and f_2 according to Definition 8. Specific counterexamples can easily be given to show that in not all cases can any one parametric transformation associate the proper values between the two CSPs. In particular, as mentioned before in an instance of this general situation (see Example 1), there is no way to define f_2 such that the new values assigned to the variable v_{k+1} in $B'(P)$ can be syntacticaly transformed into P. This restriction is due to the parametric definition of f_2.$\blacksquare$

If now we consider a general CSP with both binary and non-binary constraints, we can represent it in a binary way by applying the transformation described above to each one of its non-binary constraints. More precisely, given a CSP P with n variables, b binary constraints and nb non-binary constraints (each connecting m_i variables, for $i = 1, \ldots, nb$), let us call $B'(P, i)$, for $i = 1, \ldots, nb$ the binary CSP corresponding to the ith non-binary constraint as described above, and let us also call $B"(P)$ the CSP obtained putting together $B'(P, 1), \ldots, B'(P, nb)$ and the binary constraints of P.

More precisely, we have $P =< V, D, C, con, def >$, where

- $V = \{v_1, \ldots, v_n\}$,
- $C = \{c_1, \ldots, c_b, c_{b+1}, \ldots, c_{b+nb}\}$, where $\mid con(c_i) \mid = 2$ for $i = 1, \ldots, b$ and $\mid con(c_i) \mid = m_i$ for $i = b + 1, \ldots, b + nb$.

Also, we have $B'(P, i) =< V_i, D, C_i, con_i, def_i >$ (obtained from the ith constraint as described before) and $B"(P) =< V", D, C", con", def" >$, where

- $V" = \bigcup_{i=1,\ldots,nb} V_i$,
- $C" = (\bigcup_{i=1,\ldots,nb} C_i) \bigcup \{c_1, \ldots, c_b\}$.

Note that $B"(P)$ has exactly $n + nb$ variables and $b + \Sigma_{i=1,\ldots,nb} m_i$ binary constraints. As in the simple case of only one non-binary constraint, in the general case we can prove the same results about reducibility between P and $B"(P)$. I.e., we can prove that P is reducible to $B"(P)$ but $B"(P)$ is not reducible to P.

Theorem 5 *Given P and $B"(P)$ as described above, then P is reducible to $B"(P)$ but $B"(P)$ is not reducible to P.*

Proof: Straightforward from Theorems 3 and 4.∎

In other words, the above theorem shows that the approach of adding variables to transform a non-binary CSP into a binary one is erroneously believed to obtain an equivalent CSP. In fact, it produces neither a CSP equivalent to the given one in the usual sense, nor a CSP equivalent in the extended sense.

However, as mentioned in Section 3, the approach in which it is possible to partition the variables of a CSP into *hidden* and *non-hidden* variables could be useful to solve such discrepancy, since the additional variable v_{k+1} could be considered a *hidden* variable.

The combination of the *hidden variables* approach with ours could be of great interest. In fact, in such framework it would be possible to use our approach to allow constraint and variable redundancy, and the other approach to allow a kind of *user-specified* redundancy which wouldn't be automatically recognizible.

4.2 Second Approach: Dual Representation

In this section we describe the second approach to transform a non-binary CSP to a binary one. Unlike for the approach described in the previous section, this one produces a binary CSP equivalent to the non-binary one in the extended sense.

Usually, a CSP is graphically represented by a labelled (hyper)graph, which is referred to in the literature as the primal constraint graph.

Definition 9 (Labelled Graph) *A labelled graph is a triple $< N, A, l >$, where*

- *N is a set of nodes,*
- *A is a set of arcs (i.e. subsets of N), and*
- *l is a function labelling the arcs.*$\blacksquare$

Definition 10 (Primal Constraint Graph) *Given a CSP $P =< V, D, C, con, def >$, the primal constraint graph associated with it is a labelled graph $PG(P) =< N, A, l >$, such that there exist two bijections pnode and parc such that*

- *$pnode : V \rightarrow N$,*
- *$parc : C \rightarrow A$, and*
- $\forall c \in C, l(parc(c)) = def(c)$.$\blacksquare$

In other words, $| N |=| V |$ and $| A |=| C |$, and if arc a of $PG(P)$ corresponds to constraint c of P (i.e., $a = parc(c)$), then a connects the nodes corresponding to the variables connected by c, and it is labelled by the definition of c.

The idea now is to represent a CSP not by its primal constraint graph, but by using a graph that is dual to the primal constraint graph. The dual constraint graph is again a labelled graph, where nodes represent arcs of the primal, and arcs are labelled by variables shared by the constraints represented by the connected nodes.

Example 6 *Consider a CSP P with four variables, v_1, v_2, v_3, v_4, and two 3-ary constraints c_1 and c_2, c_1 connecting v_1, v_2, v_3, and c_2 connecting v_2, v_3, v_4. The dual constraint graph contains two nodes, n_1 and n_2, each corresponding to one of the constraints of P, and an arc, a, which connects them and is labelled by the shared variables (in this case v_2 and v_3). The following figure shows both the primal and the dual constraint graph for P.*

■

More formally, we have:

Definition 11 (Dual Constraint Graph) *Given a CSP $P =< V, D, C, con, def >$, the dual constraint graph is a labelled graph $DG(P) =< N, A, l >$. Let us consider a bijection dnode such that $dnode : C \rightarrow N$. Then we have*

- *$N = dnode(C)$,*
- *$A = \{\{dnode(c_1), dnode(c_2)\}$ for each $c_1, c_2 \in C$ such that $con(c_1) \cap con(c_2) \neq \Phi\}$, and*
- *$l(a) = con(c_1) \cap con(c_2)$ for each $a \in A$ such that $a = \{dnode(c_1), dnode(c_2)\}$.■*

This implicitly defines a bijection darc between the set of constraint pairs $\{(c_1, c_2) \mid con(c_1) \cap con(c_2) \neq \Phi\}$ and A.■

It is easy to see from the above definition that, given an arbitrary CSP, while its primal constraint graph can contain in general non-binary arcs, its dual constraint graph is always a binary graph, no matter the arity of the constraints in the given CSP. Note, however, that a dual constraint graph is a labelled graph and not a CSP, thus we have to make another step in our process of going from a non-binary CSP to an equivalent binary CSP.

Since both primal and dual constraint graphs are labelled graphs, given a binary graph G which is the dual constraint graph $DG(P)$ of some CSP P, G can always be seen also as a primal constraint graph of some binary CSP, say P' (i.e. $G = PG(P')$). And because functions *dnode* and *darc* are invertible, we can establish a one-to-one mapping between the variables of P and P'.

To find P' given G, we must consider the inverse of the transformation described in Definition 10, and apply it to the given graph G. Given $G = DG(P) =< N, A, l >$ with $P =< V, D, C, con, def >$, we can define a new binary CSP $P' =< V', D', C', con', def' >$ as follows:

- $V' = \{v'_i$, where $v'_i = con(dnode^{-1}(n_i))$ and $D_{v'_i} = def(dnode^{-1}(n_i))\}$;
- $C' = \{c'(a), a \in A\}$, where
  - $def'(c'(a)) = def(c_i) \cap def(c_j)$ projected over $l(a)$ for $c_i, c_j = darc^{-1}(a)$, and
  - $con'(c'(a)) = \{v'_i, v'_j\}$ where $v'_i, v'_j \in V'$ and a connects nodes $n_i, n_j \in N$.

In other words, the variables of P' correspond to the nodes of G (which in turn correspond to the constraints of P) and the constraints of P' (all binary) correspond to the arcs in G (which in turn correspond to the shared variables of P). Since $G = DG(P)$, each variable v'_i in P' is in reality a tuple of variables: those connected by the constraint c of P corresponding (via $dnode^{-1}$) to the node n_i of $G = DG(P)$). This is the crucial transformation: each variable in the binary CSP P' represents k variables connected by some k-ary constraint in the non-binary CSP P.

The domain of v'_i is the set of all the tuples defining the non-binary constraint c in P. Also, each constraint c' of P', connecting variables v'_1 and v'_2, is an equality constraint among the variables labelling arc a of G, which connects the nodes corresponding to v'_1 and v'_2. In order to define each c', we must essentially find the pairwise solutions between two constraints which share variables in the original CSP.

Example 7 *Consider the simple case of a CSP P with three variables v_1, v_2, and v_3 and a constraint c connecting them all which only allows the tuple $< a, a, a >$ as possible assignment to the three variables. The CSP P' which is derived from the dual constraint graph has only one variable, v', consisting of the 3-tuple $< v_1, v_2, v_3 >$ from P. The domain of v' is $< a, a, a >$, i.e., the definition of constraint c. The graphical representation of both P and P' can be seen in the following figure.*

■

Example 8 *Consider now the case in which P consists of four variables $\{v_1, v_2, v_3, v_4\}$ and three constraints $\{c_1, c_2, c_3\}$ defined as follows:*

- *c_1 connects the variables v_1, v_2, v_4, and its definition is the single tuple $< 1, 1, 0 >$ (the domain $D = \{0, 1\}$);*
- *c_2 connects the variables v_1, v_2, v_3, and its definition consists of the two tuples $< 1, 1, 1 >$ and $< 0, 1, 0 >$;*
- *c_3 connects the variables v_2, v_3, v_4, and its definition consists of the tuple $< 1, 0, 0 >$.*

By definition, $Sol(P)$ consists of the set of tuples of values of the variables $< v_1, v_2, v_3, v_4 >$ which satisfies all three constraints. In this particular case, such a set is empty.

The dual constraint graph $DG(P)$ has three nodes, n_1, n_2, n_3, corresponding to the three constraints c_1, c_2, c_3 of P. There is one arc between each pair of nodes, labelled by the shared variables. More precisely, we have the following constraints:

- *constraint a_1, connecting nodes n_1, n_2 and labelled by v_1, v_2;*
- *constraint a_2, connecting nodes n_1, n_3 and labelled by v_2, v_4;*
- *constraint a_3, connecting nodes n_2, n_3 and labelled by v_2, v_3.*

The corresponding P' has three variables: $v'_1 =< v_1, v_2, v_4 >$, $v'_2 =< v_1, v_2, v_3 >$, and $v'_3 =< v_2, v_3, v_4 >$, with domain the set of all possible triples of values on $\{0, 1\}$. Also, it has three binary constraints:

- c'_1, *connecting* v'_1 *and* v'_2, *and with definition* $\{<< 1, 1, 0 >, < 1, 1, 1 >>\}$ *(note that this definition is a singleton set containing a tuple with two elements, each element being a tuple of three values);*

- c'_2, *connecting* v'_2 *and* v'_3, *and with definition* $\{<< 1, 1, 0 >, < 1, 0, 0 >>\}$*;*

- c'_3, *connecting* v'_1 *and* v'_3, *and with definition* $\{<< 0, 1, 0 >, < 1, 0, 0 >>\}$*.*

Since there is no tuple of values for $< v'_1, v'_2, v'_3 >$ *which satisfies all three constraints, there is also no solution for* P'*. Both* P *and* P' *for this example can be seen in the following figure.*

■

Note that in the above example case P and P' have the same solution, i.e. the empty set of tuples. However, because of the different representation of P and P', they may have different sets of variables (in this case, P has four variables and P' has three), or different domain structure (here, the domain of the variables in P is a set of values, while the domains of the variables in P' are sets of triples of values). This means that in general they cannot have the same solution. That is, if the structure of the CSPs remains the same, then no matter how we change the values of the definitions of the constraints, the two CSPs can never be $\equiv_u$.

Example 9 *Suppose that* P *is the same as the previous example, but that the constraints have a different definition. More precisely, let*

- c_1 *and* c_2 *be defined as before;*

- c_3 *be defined by* $\{< 1, 1, 0 >\}$*.*

In this case, it is easy to see that the solution of P *(for the ordered set of variables* v_1, v_2, v_3, v_4*) is* $\{< 1, 1, 1, 0 >\}$*, while the solution of* P' *(for the ordered set of variables* v'_1, v'_2, v'_3*) is* $\{<< 1, 1, 0 >, < 1, 1, 1 >, < 1, 1, 0 >>\}$*.* ■

However, even if P and P' don't have the same solution, the intuition tells us that they have substantially the same meaning. We will formally prove this conjecture by showing that P and P' are mutually reducible, and thus equivalent in the extended sense of Definition 6.

Lemma 4 *Given any CSP P and the binary CSP P' as described above in this section, P is reducible to P'.*

Proof: We must define the functions f_1 and f_2 in such a way that Definition 8 is satisfied. Beginning with the variable mapping, we must define $f_1 : V \rightarrow t(V')$. We begin by noting that $\forall v \in V, \exists c \in C$ such that $v \in con(c)$. And $\forall c \in C, \exists v' \in V'$ such that $c = dnode^{-1}(pnode(v'))$. But if we naturally define $f_1(v) = v'$ if $v \in con(dnode^{-1}(pnode(v')))$, the result is not a function. Because any variable v in P may participate in more than one constraint, then there may be more than one v' in P' such that $v \in con(dnode^{-1}(pnode(v')))$. Although this ambiquity is not intuitively important, it must be removed formally. Therefore, let

- $f_1 : V \rightarrow t(V')$, such that $f_1(v) = v'_i$
  where i is the smallest index of V' such that $v \in con(dnode^{-1}(pnode(v'_i)))$;
- $f_2 : (V \times t(PAR')) \rightarrow t(PAR)$, such that, for all v in V, $f_2(v, < p_1, \ldots, p_k >= p_i$
  if $f_1(v) =< v_1, \ldots, v_k >$ and $v = v_i, 1 \leq i \leq k$.

Now we must show that every tuple $t =< t_1, \ldots, t_k >\in Sol(P)$ can be derived from these functions and $Sol(P')$. Suppose there was no tuple $t' \in Sol(P')$ such that t could be obtained by applying f_2 to t' for each $v_i \in V$. We can show that this supposition is false by building a t' which must be in $Sol(P')$ and will yield t by use of f_2.

Let $\mu(t, l)$ denote the lth element of tuple t; i.e, $\mu(< t_1, \ldots, t_k >, l) = t_l, 1 \leq l \leq k$. Let $S_i = \{v' \in V' \mid v_i \in con(dnode^{-1}(pnode(v'_i)))\}$. Now, for each element of S_i, v'_n, where $\mu(v'_n, l) = v_i$, construct t'_n by assigning $\mu(t'_n, l) = t_i$. Repeat this procedure for $1 \leq i \leq k$. Then we have defined a tuple t' of tuple values for each $v' \in V'$. So for a given $v_i \in V$, let $f_1(v_i)$ be v'_n, and let $t'_n \in t'$. If $\mu(v'_n, l) = v_i$, then $\mu(t'_n, l) = t_i$. So $f_2(v_i, t'_n) = t_i$. Thus the constructed t' yields t by use of f_2. And this t' must be in $Sol(P;)$.

This t' has the property that for any $t'_x, t'_y \in t'$ such that $\exists v_l \in V$ such that $v_l = \mu(v'_x, m) = \mu(v'_y, n)$ for some m, n, then $t_l = \mu(t'_x, m) = \mu(t'_y, n)$. If this $t' \notin Sol(P')$, then

$\exists c' \in C'$ with $dom(c') = \{v'_x, v'_y\}$, such that $\{t'_x, t'_y\} \notin def(c')$. Then by the definition of P', $\exists m, n$ and $v_l \in V$ such that for $v_l = \mu(con(c_x), m) = \mu(con(c_y), n)$, $\mu(t'_x, m) = \mu(t'_y, n) \notin def(c_x) \cap def(c_y)$ projected on v_l and so $\mu(t, l) \notin Sol(P)$. $\rightarrow\leftarrow$ Therefore t' is in $Sol(P')$. And the functions f_1, f_2 can thus be used to derive any $t \in Sol(P)$ from $Sol(P')$.$\blacksquare$

Clearly P' contains as least as much information as P. Showing the inverse is easier with an intermediate lemma.

Lemma 5 *If tuple $t' \in Sol(P')$, then $\forall$ tuples t'_x and $t'_y \in t'$, for variables v'_x and v'_y respectively, if $\exists v \in V$ such that $v = \mu(v'_x, m) = \mu(v'_y, n)$, then $\mu(t'_x, m) = \mu(t'_y, n)$.*

Proof: By hypothesis, $\exists v \in V$, $c_x, c_y \in C$ such that $v = \mu(con(c_x), m) = \mu(con(c_y), n)$. By definition of P', since v is a shared variable between c_x and c_y, $\exists c' \in C'$ such that $con(c') = \{v'_x, v'_y\}$ and $def(c') = def(c_x) \cap def(c_y)$ projected over v. Since $t'_x, t'_y \in t' \in Sol(P')$, then $t'_x, t'_y \in def(c')$, and $\mu(t'_x, m) = \mu(t'_y, n)$.$\blacksquare$

Lemma 6 *Given any CSP P and the binary CSP P' as described above in this section, P' is reducible to P.*

Proof: For the reducibility of P' to P, the functions f_1 and f_2 are defined as follows:

- $f_1 : V' \rightarrow t(V)$, such that $f_1(v') =< v_1, \ldots, v_k >$
  if $con(dnode^{-1}(pnode(v'))) =< v_1, \ldots, v_k >$;
- $f_2 : (V' \times t(PAR)) \rightarrow t(PAR')$, such that, for all v' in V', where $f_1(v') =< v_1, \ldots, v_k >$, $f_2(v', < p_1, \ldots, p_k >) =< p_1, \ldots, p_k >$.

We need to show that any $t' \in Sol(P')$ can be obtained by applying these functions to some $t \in Sol(P)$. Actually, we can use the proof of lemma 4 to do so.

The proof of lemma 4 constructed a t' from a $t \in Sol(P)$ and showed that $t' \in Sol(P')$. This constructed t' had the property that $\forall$ tuples t'_x and $t'_y \in t'$, for variables v'_x and v'_y respectively, if $\exists v \in V$ such that $v = \mu(v'_x, m) = \mu(v'_y, n)$, then $\mu(t'_x, m) = \mu(t'_y, n)$. Lemma 5 assures that there is no $t' \in Sol(P')$ which does not have this property.

Thus, there is always some tuple of values t for V from which some $t' \in Sol(P')$ can be constructed. But, additionally, the proof of 4 showed that for all $t' \in Sol(P')$ which had this property, the associated $t \in Sol(P)$. So there is no $t' \in Sol(P')$ which cannot be constructed from a $t \in Sol(P)$. The defined functions above are simply that construction.■

Theorem 6 *Given any CSP P and the binary CSP P' as described above in this section, $P \equiv_e P'$.*

Proof: This theorem follows from lemmas 4 and 6.

We have thus shown that the second approach to the transformation of a non-binary CSP to a binary one yields an equivalent (in the extended sense) CSP. This result has been informally believed and used in the constraint literature before, but has never been formally proved, the only reason being that the tools for dealing with equivalence of CSPs were not appropriate.

5 Conclusions

Having found the usual definition of equivalence of CSPs unsatisfactory, we proposed a new definition, based on the concept of mutual reducibility of CSPs, where reducibility means information conservation. The key idea is that if it is possible to obtain the solution of one CSP from that of another one, and viceversa, by purely syntactic transformations between variables and values, then the two CSPs should be considered equivalent, because they obviously contain exactly the same non-redundant information. Semantic operations, which permit the active solving of one CSP, are precluded. (An extreme example of a semantic reduction of CSP $P2$ to CSP $P1$ would be to simply solve $P2$, even though the solution of $P1$ played no part in the problem solving.)

By using this new definition, we formally addressed the issue of the equivalence of binary and non-binary CSPs. We considered two known algorithms for transforming a non-binary CSP into a binary one and, even though they have always been believed to produce an "intuitively equivalent" binary CSP, we proved that in reality such belief was not correct. In fact, none of these two algorithms produces an equivalent CSP in the usual sense, while with the new definition it is possible to show that only one of them does.

This new definition of equivalence of CSPs makes a fundamental contribution to the existing constraint satisfaction theory, in that it formalizes our intuitions about equivalence, which were not captured by the traditional definition. The result is is a more appropriate tool to identify redundant information in CSPs, and the formal proof of an informally known result, not possible with previous definitions. This notion of equivalence may also be extensible to computational problems in general. The usual notion of equivalence in computer science is that of time and space complexity. This new equivalence concept is based on the information content expressed in the formal problem. While we have not attempted to generalize the equivalence of CSPs to other types of problems, this is a promising area for future research.

Acknowledgments

Francesca Rossi would like to thank the Italian Research Council for supporting her visit at MCC. MCC shareholders supporting this research are Bellcore, CDC, DEC, Harris, Kodak and NCR.

References

[1] Dechter R., Pearl J., "Tree-clustering schemes for constraint-processing", *Proc. AAAI 88*.

[2] Montanari U., "Networks of constraints: fundamental properties and application to picture processing", *Information Science 7*, 1974, pp. 95-132.

[3] Montanari U., Rossi F., "Constraint relaxation may be perfect", *Technical Report TR-21/89*, Computer Science Dept., University of Pisa, Pisa, Italy.

[4] Pierce, C. S., *Collected Papers, Vol. III*, C. Hartshorne and P. Weiss eds., Harward University Press, Cambridge, 1933.

[5] Rossi F., Montanari U., “Hypergraph grammars and networks of constraints versus logic programming and metaprogramming”, *Proc Workshop on Metaprogramming in Logic Programming*, Bristol, June 1988.

[6] Rossi F., Montanari U., “Exact solution of networks of constraints using perfect relaxation”, *Proc. 1st Int. Conf. on Principles of Knowledge Representation and Reasoning*, Toronto, May 1989.