# Using a theorem prover for reasoning on constraint problems

Marco Cadoli and Toni Mancini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
cadoli|tmancini@dis.uniroma1.it

**Abstract.** Specifications of constraint problems can be considered logical formulae. As a consequence, it is possible to infer their properties by means of automated reasoning tools, with the goal of automatically synthesizing transformations that can make the solving process more efficient. The purpose of this paper is to link two important technologies: automated theorem proving and constraint programming. We report the results on using ATP technology for checking existence of symmetries, checking whether a given formula breaks a symmetry, and checking existence of functional dependencies in a specification. The output of the reasoning phase is a transformed constraint program, consisting in a reformulated specification and, possibly a search strategy. We show our techniques on problems such as Graph coloring, Sailco inventory and Protein folding.

## 1 Introduction

The efficiency of current systems for constraint programming (CP), e.g., OPL [18], GAMS [5], DLV [10], and NP-SPEC [4], depends on the choices that users have to make in order handle instances of realistic size. This may degrade the declarativeness of the CP approach. The main user choices that greatly influence performances are about:

– The model for the given problem. In fact, different, but equivalent formulations for the same problem usually exist, and choosing one of them can make the difference. To this end, many techniques have been proposed, in order to modify the given constraint problem into an equivalent one, with the goal of reducing the solving time (e.g., symmetry-breaking, addition of implied constraints, or the opposite strategy of constraint abstraction).
– The search strategy to be applied, and the heuristic to be used to order variables and domain values. To this end, even if some systems, e.g., OPL [18], use a default strategy if the user-defined one is missing, there are many situations in which a smart reasoning on the problem specification is needed in order to infer good ones. An example is given by specifications in which functionally dependent predicates exist, either because of a precise modelling

choice (e.g., redundant modelling), or because of intrinsic properties of the modelled problem. In [3] (and in forthcoming Section 4) we show how a suitable search strategy that exploits dependencies can be synthesized.

Although much research has been done on these aspects, all techniques proposed in the literature in order to optimize the solving process either apply at the instance level, or are reformulations of a specific constraint problem, obtained as the output of a human process. In particular, the use of automated tools for preprocessing and reformulating arbitrary constraint problems has been limited, to the best of our knowledge, to the *instance* level (cf., e.g, [15, 9, 11]).

On the other hand, current systems for CP exhibit a neat separation between problem *specifications* and *instances*, and, in many cases, the structure of the problem, and not the instance at hand, exhibits properties amenable to be optimized. Hence, reasoning at the symbolic level can be more natural and effective than making these "structural" aspects emerge after instantiation, when the structure of the problem has been hidden. This makes specification-level reasoning very attractive from a methodological point of view, also for *validation purposes*, since the presence or lack of a property (e.g., a dependence) may reveal a bug in the problem model.

Our research explicitly focuses on the specification level, and aims to transform the constraint model given by the user into an equivalent one (possibly integrated with additional information about the search strategy to be used), which is more efficiently evaluable by the solver at hand. We note that focusing on the specification does not rule out the possibility of applying all existing optimization techniques at the instance level, to handle also those aspects that arise from the instance considered.

In previous work, we studied how to highlight constraints that can be ignored in a first step (the so called "safe-delay constraints") [2], how to detect and break symmetries [1, 12], and how to recognize and exploit functional dependencies in a specification [3], showing how the original problem model can be appropriately reformulated. Experimental analysis shows how these approaches are effective in practice, for different classes of solvers. In this paper, we tackle the following question: *is it possible to check the above mentioned properties, and possibly other ones, automatically?* The main result is that, even if the underlying problems have been proven to be not decidable (cf. previously cited works), in practical circumstances the answer is often positive. In particular, we show how automated theorem proving (ATP) technology can be effectively used to perform the required forms of reasoning.

We report the results on using theorem provers and finite model finders for reasoning on specifications of constraint problems, represented as existential second order logic (ESO) formulae. We focus on two forms of reasoning:

– Checking existence of *value symmetries*; on top of that, we check whether a given formula breaks such symmetries or not;
– Checking existence of *functional dependencies*, i.e., properties that force values of some guessed predicates to depend on the value of other ones.

The rest of the paper is organized as follows: in Section 2 we give some preliminaries on modelling combinatorial problems as formulae in ESO. Sections 3 and 4 are devoted to the description of experiments in checking symmetries and dependencies, respectively. In Section 5 we conclude the paper, and present current research.

## 2 Preliminaries

In this paper, we use *existential second-order logic* (ESO) for the formal specification of problems, which allows to represent all search problems in the complexity class NP [7]. For motivations about the use of ESO as a modelling language, cf. our previous work [1, 3, 2]. Here, we recall that ESO can be regarded as the formal basis of virtually all languages for constraint modelling, as long as only finite domains are considered. Additionally, many reasoning tasks on problem specifications written in ESO reduce to check semantic properties of first-order formulae. Finally, our results are a basis for transforming specifications written in higher-level languages.

An ESO specification describing a search problem $\pi$ is a formula $\psi_\pi$:

$$\exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}), \tag{1}$$

where $\boldsymbol{\mathcal{R}} = \{R_1, \ldots, R_k\}$ is the *relational schema* for every input instance, and $\phi$ is a quantified first-order formula on the relational vocabulary $\boldsymbol{\mathcal{S}} \cup \boldsymbol{\mathcal{R}} \cup \{=\}$ ("=" is always interpreted as identity). An instance $\boldsymbol{\mathcal{I}}$ of the problem is given as a relational database (coherently with all state-of-the-art systems) over the schema $\boldsymbol{\mathcal{R}}$. Predicates (of given arities) in the set $\boldsymbol{\mathcal{S}} = \{S_1, \ldots, S_n\}$ are called *guessed*, and their possible extensions over the Herbrand universe encode points in the search space for problem $\pi$. Formula $\psi_\pi$ correctly encodes problem $\pi$ if, for every input instance $\boldsymbol{\mathcal{I}}$, a bijective mapping exists between solutions to $\pi$ and extensions of predicates in $\boldsymbol{\mathcal{S}}$ which verify $\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{I}})$. It is worthwhile to note that, when a specification is instantiated against an input database, a CSP in the sense of [6] is obtained.

*Example 1 (Graph 3-coloring [8]).* Given a graph, the question is whether it is possible to give each of its nodes one out of three colors (red, green, and blue), in such a way that adjacent nodes (not including self-loops) are never colored the same way. The question can be easily specified as an ESO formula $\psi$ on the input schema $\boldsymbol{\mathcal{R}} = \{edge(\cdot, \cdot)\}$:

$$\begin{aligned}
\exists RGB \quad & \forall X \quad R(X) \vee G(X) \vee B(X) \ \wedge && (2)\\
& \forall X \quad R(X) \rightarrow \neg G(X) \ \wedge && (3)\\
& \forall X \quad R(X) \rightarrow \neg B(X) \ \wedge && (4)\\
& \forall X \quad B(X) \rightarrow \neg G(X) \ \wedge && (5)\\
& \forall XY \ X \neq Y \wedge R(X) \wedge R(Y) \rightarrow \neg edge(X,Y) \wedge && (6)\\
& \forall XY \ X \neq Y \wedge G(X) \wedge G(Y) \rightarrow \neg edge(X,Y) \wedge && (7)\\
& \forall XY \ X \neq Y \wedge B(X) \wedge B(Y) \rightarrow \neg edge(X,Y). && (8)
\end{aligned}$$

Clauses (2) and (3-5) force every node to be assigned exactly one color (covering and disjointness constraints), while (6-8) force nodes linked by an edge to be assigned different colors (good coloring constraints). □

## 3 Detecting and breaking uniform value symmetries

In this section we face the problem of automatically detecting and breaking some symmetries in problem specifications. In Subsection 3.1 we give preliminary definitions of problem transformation and symmetry taken from [1], and show how the symmetry-detection problem can be reduced to checking semantic properties of first-order formulae. We limit our attention to specifications with monadic guessed predicates only, and to transformations and symmetries on values. Motivations for these limitations are given in [1]; here, we just recall that non-monadic guessed predicates can be transformed in monadic ones by unfolding and by exploiting the finiteness of the input database. We refer to [1] also for considerations on benefits of the technique on the efficiency of problem solving, in particular on the Graph coloring, Not-all-equal Sat, and Social golfer problems. In Subsection 3.2 we then show how a theorem prover can be used to automatically detect and break symmetries.

### 3.1 Definitions

**Definition 1 (Uniform value transformation (UVT) of a specification).** *Given a problem specification* $\psi \doteq \exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})$*, with* $\boldsymbol{\mathcal{S}} = \{S_1, \ldots S_n\}$*,* $S_i$ *monadic for every* $i \in [1, n]$*, and input schema* $\boldsymbol{\mathcal{R}}$*, a* uniform value transformation (UVT) *for* $\psi$ *is a mapping* $\sigma : \boldsymbol{\mathcal{S}} \rightarrow \boldsymbol{\mathcal{S}}$*, which is total and onto, i.e., defines a permutation of guessed predicates in* $\boldsymbol{\mathcal{S}}$*.*

The term "uniform value" transformation in Definition 1 is used because swapping monadic guessed predicates is conceptually the same as uniformly exchanging domain values in a CSP. Referring to Example 1, the domains values are the colors, i.e., red, green, and blue. We now define when a UVT is a symmetry for a given specification.

**Definition 2 (Uniform value symmetry (UVS) of a specification).** *Let* $\psi \doteq \exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})$*, be a specification, with* $\boldsymbol{\mathcal{S}} = \{S_1, \ldots S_n\}$*,* $S_i$ *monadic for every* $i \in [1, n]$*, and input schema* $\boldsymbol{\mathcal{R}}$*, and let* $\sigma$ *be a UVT for* $\psi$*. Transformation* $\sigma$ *is a* uniform value symmetry (UVS) *for* $\psi$ *if every extension for* $\boldsymbol{\mathcal{S}}$ *which satisfies* $\phi$*, satisfies also* $\phi^\sigma$*, defined as* $\phi[S_1/\sigma(S_1), \ldots, S_n/\sigma(S_n)]$ *and vice versa, regardless of the input instance, i.e., for every extension of the input schema* $\boldsymbol{\mathcal{R}}$*.*

Note that every CSP obtained by instantiating a specification with UVS $\sigma$ has at least the corresponding uniform value symmetry.

In [1], it is shown that checking whether a UVT is a UVS reduces to checking equivalence of two first-order formulae:

**Proposition 1.** *Let $\psi$ be a problem specification of the kind* (1)*, with only monadic guessed predicates, and $\sigma$ a UVT for $\psi$. Transformation $\sigma$ is a UVS for $\psi$ if and only if $\phi \equiv \phi^\sigma$.*

Once symmetries of a specification have been detected, additional constraints can be added in order to *break* them, i.e., to transform the specification in order to wipe out from the solution space (some of) the symmetrical points. These kind of constraints are called *symmetry-breaking formulae*:

**Definition 3 (Symmetry-breaking formula).** *A* symmetry-breaking formula *for $\psi \doteq \exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})$ with respect to UVS $\sigma$ is a closed (except for $\boldsymbol{\mathcal{S}}$) formula $\beta(\boldsymbol{\mathcal{S}})$ such that the following two conditions hold:*

1. *Transformation $\sigma$ is no longer a symmetry for $\exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}})$:*
$$(\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}})) \not\equiv (\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}}))^\sigma\ ; \qquad (9)$$
2. *Every model of $\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})$ can be obtained by those of $\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}})$ by applying symmetry $\sigma$:*
$$\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})\ \models \bigvee_{\boldsymbol{\sigma} \in \sigma^*} (\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}}))^{\boldsymbol{\sigma}}; \qquad (10)$$
*where $\boldsymbol{\sigma}$ is a sequence (of finite length $\geq 0$) over $\sigma$, and, given a first-order formula $\gamma(\boldsymbol{\mathcal{S}})$, $\gamma(\boldsymbol{\mathcal{S}})^{\boldsymbol{\sigma}}$ denotes $(\cdots(\gamma(\boldsymbol{\mathcal{S}})^\sigma)\cdots)^\sigma$, i.e., $\sigma$ is applied $|\boldsymbol{\sigma}|$ times (if $\boldsymbol{\sigma} = \langle\rangle$, then $\gamma(\boldsymbol{\mathcal{S}})^{\boldsymbol{\sigma}}$ is $\gamma(\boldsymbol{\mathcal{S}})$ itself).*

If $\beta(\boldsymbol{\mathcal{S}})$ matches the above definition, then we are entitled to solve the problem $\exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}})$ instead of the original one $\exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})$. It is worthwhile noting that, even if in formula (10) $\boldsymbol{\sigma}$ ranges over the (infinite) set of finite-length sequences of 0 or more applications of $\sigma$, this actually reduces to sequences of length at most $n!$, since this is the maximum number of successive applications of $\sigma$ that can lead to all different permutations. Finally, we note that the inverse logical implication in formula (10) always holds, because $\sigma$ is a UVS, and so $\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})^\sigma \equiv \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})$.

## 3.2 Experiments with the theorem prover

Proposition 1 suggests that the problem of detecting UVSs of a specification $\psi$ of the kind (1) can in principle be performed in the following way:

1. Selecting a UVT $\sigma$, i.e., a permutation of guessed predicates in $\psi$ (if $\psi$ has $n$ guessed predicates, there are $n!$ such UVTs –$n$ is usually very small);
2. Checking whether $\sigma$ is a UVS, i.e., deciding whether $\phi \equiv \phi^\sigma$.

The above procedure suggests that a first-order theorem prover can be used to perform automatically point 2. Even if we proved in [1] that this problem is not decidable, we show how a theorem prover usually performs well on this kind of formulae. As for the symmetry-breaking problem, from conditions of Definition 3 it follows that also the problem of checking whether a formula breaks a given UVS for a specification clearly reduces to semantic properties of logical formulae.

In this section we give some details about the experimentation done using automated tools. First of all we note that, obviously, all the above mentioned conditions can be checked by using a refutation theorem prover. It is interesting to note that, for some of them, we can use a finite model finder. In particular, we can use such a tool for checking statements (such as condition (9) of Definition 3 or the negation of the condition of Proposition 1) which are syntactically a non-equivalence. As a matter of facts, it is enough to look for a finite model of the negation of the statement, i.e., the equivalence. If we find such a model, then we are sure that the non-equivalence holds, and we are done. The tools we used are OTTER [14], and MACE [13], respectively, in full-automatic mode.

**Detecting symmetries** The examples we worked on are the following.

*Example 2 (Graph 3-coloring: Example 1 continued).* The mapping $\sigma^{R,G} : \boldsymbol{\mathcal{S}} \rightarrow \boldsymbol{\mathcal{S}}$ such that $\sigma^{R,G}(R) = G$, $\sigma^{R,G}(G) = R$, $\sigma^{R,G}(B) = B$ is a UVT for it. It is easy to observe that formula $\phi^{\sigma^{R,G}}$ is equivalent to $\phi$. This implies, by Proposition 1, that $\sigma^{R,G}$ is also a UVS for this problem. The same happens also for $\sigma^{R,B}$ and $\sigma^{G,B}$ that swap $B$ with, respectively, $R$ and $G$. $\square$

*Example 3 (Not-all-equal Sat [8]).* Given a propositional formula in CNF, the question is whether it is possible to assign a truth value to all the variables in such a way that the input formula is satisfied, and every clause contains at least one literal which is false. An ESO specification for this problem (cf. [1]) has two monadic guessed predicates $T(\cdot)$ and $F(\cdot)$, representing variables whose truth value is true and false, respectively. It is easy to prove that the UVT $\sigma^{T,F}$, defined as $\sigma^{T,F}(T) = F$ and $\sigma^{T,F}(F) = T$, is a UVS, since $\phi^{\sigma^{T,F}}$ is equivalent to $\phi$. $\square$

*Example 4 (Social golfer (cf. www.* `csplib.org`*)).* Given a set of players, a set of groups, and a set of weeks, encoded in monadic relations $\mathit{player}(\cdot)$, $\mathit{group}(\cdot)$, and $\mathit{week}(\cdot)$ respectively, this problem amounts to decide whether there is a way to arrange a scheduling for all weeks in relation $\mathit{week}$, such that: *(i)* For every week, players are divided into equal sized groups; and *(ii)* Two different players don't play in the same group more than once. A specification for this problem (assuming the ratio $|\mathit{player}|/|\mathit{group}|$, i.e., the group size, integral) has a single guessed predicate $\mathit{Play}(P, W, G)$ stating that player $P$ plays in group $G$ on week $W$ (cf. [1]). In order to highlight UVSs according to Definition 2, we need to substitute the ternary guessed predicate $\mathit{Play}$ by means of, e.g., $|\mathit{week}| \times |\mathit{group}|$ monadic predicates $\mathit{Play}_{W,G}(\cdot)$ (each one listing players playing in group $G$ on week $W$). UVTs $\sigma_W^{G,G'}$, swapping $\mathit{Play}_{W,G}$ and $\mathit{Play}_{W,G'}$, i.e., given a week $W$, and two groups $G$ and $G'$, assign to group $G'$ on week $W$ all players assigned to group $G$ on week $W$, and vice versa, are symmetries for the (unfolded) Social golfer problem (because group renamings have no effect). It is worth noting that unfolding non-monadic predicates is just a formal step in order to apply the definitions of [1], and has not to be performed in practice. $\square$

The results we obtained with OTTER are shown in Table 1. The third row refers to the version of the Not-all-equal Sat problem in which all clauses have three literals, the input is encoded using a ternary relation $clause(\cdot,\cdot,\cdot)$, and the specification varies accordingly. It is interesting to see that the time needed by OTTER is often very low. As for the fourth row, it refers to the unfolded specification of the Social golfer problem with 2 weeks and 2 groups of size 2. Unfortunately, OTTER does not terminate for a larger number of weeks or groups (cf. Section 5).

A note on the encoding is in order. Initially, we gave the input to OTTER exactly in the format specified by Proposition 1, but the performance was quite poor: for 3-coloring the tool did not even succeed in transforming the formula in clausal form, and symmetry was proven only for very simplified versions of the problem, e.g., 2-coloring, omitting constraint (2). Results of Table 1 have been obtained by introducing new propositional variables defining single constraints. as an example, constraint (2) is represented as

```
covRGB <-> (all x (R(x) | G(x) | B(x))).,
```

where `covRGB` is a fresh propositional variable. Obviously, we wrote a first-order logic formula encoding condition of Proposition 1, and gave its negation to OTTER in order to find a refutation.

As for proving non-existence of symmetries, we used the following example.

*Example 5 (Graph 3-coloring with red self-loops).* We consider a modification of the problem of Example 1, and show that only one of the UVTs in Example 2 is indeed a UVS for the new problem. Here, the question is whether it is possible to 3-color the input graph in such a way that every self loop insists on a red node. In ESO, one more clause must be added to (2–8): $\forall X \;\; edge(X,X) \rightarrow R(X)$.

UVT $\sigma^{G,B}$ is a UVS also of the new problem, because of the same argument of Example 2. However, for what concerns $\sigma^{R,G}$, in this case $\phi^{\sigma^{R,G}}$ is not equivalent to $\phi$: as an example, for the input instance $edge = \{(v,v)\}$, the color assignment $\overline{R},\overline{G},\overline{B}$ such that $\overline{R} = \{v\}, \overline{G} = \overline{B} = \emptyset$ is a model for the original problem, i.e., $\overline{R},\overline{G},\overline{B} \models \phi(R,G,B,edge)$. It is however easy to observe that $\overline{R},\overline{G},\overline{B} \not\models \phi^{\sigma^{R,G}}(R,G,B,\ edge)$, because $\phi^{\sigma^{R,G}}$ is verified only by color assignments for which $\overline{G}(v)$ holds. This implies, by Proposition 1, that $\sigma^{R,G}$ is not a UVS. For the same reason, also $\sigma^{R,B}$ is not a UVS for the new problem. □

We wrote a first-order logic formula encoding condition of Proposition 1 for $\sigma^{R,G}$ on the above example and gave its negation to MACE in order to find a model of the non-equivalence. MACE was able to find the model described in Example 5 in few hundreds of a second.

**Breaking symmetries** We worked on the 3-coloring specification given in Example 1 and the UVS $\sigma^{R,G}$ defined in Example 2. This UVS can be broken by, e.g., the following formulae:

$$\beta^{R,G}_{sel}(R,G,B) \doteq R(\overline{v}) \vee B(\overline{v}), \tag{11}$$

that forces a selected node, say $\overline{v}$, not to be colored in green;

$$\beta^{R,G}_{least}(R,G,B) \doteq \forall w \; G(w) \rightarrow \exists v \; R(v) \; \wedge \; v < w, \tag{12}$$

where "$<$" is a (possibly pre-interpreted) total ordering on the graph nodes, hence forcing the least green node to be greater than the least red one (the so-called *lowest index ordering* on red and green nodes);

$$\beta^{R,G}_{card}(R,G,B) \doteq |R| \leq |G|, \tag{13}$$

that forces green nodes to be at least as many as the red ones. It is easy to prove that formulae (11), (12) and (13) respect both conditions of Definition 3. As for condition (10), $\boldsymbol{\sigma}$ can be limited to sequences of length at most 2, since $\sigma^{R,G}$ swaps only two guessed predicates.

For what concerns formula (13), some considerations are in order, since this example highlights some difficulties that can arise when using first-order ATPs. In fact, formula (13) can be written in ESO (it evaluates to true if and only if a total injective function from tuples in $R$ to those in $G$ exists). Therefore, conditions in Definition 3 are second-order (non-)equivalences, and the use of a first-order theorem prover may in general not suffice. However, in some circumstances, it is possible to write first-order conditions that can be used to infer the truth value of those of Definition 3. One such case is that of formulae in ESO (details omitted for lack of space).

We used MACE and OTTER in order to prove that the formulae described above are symmetry-breaking for 3-coloring with respect to $\sigma^{R,G}$, by checking conditions of Definition 3. As Table 2 shows, results are always good when checking the first condition, while may become worse when OTTER is used to prove the second one (the third row of Table 2 refers to the overall time needed to check whether formula (13) satisfies the first condition of Definition 3). However, these poor performances do not make the approach meaningless. In fact, it is often possible to have candidate symmetry-breaking formulae, e.g., (13) and its generalizations, that respect the second condition of Definition 3 *by design.* In these cases, checking the first condition with MACE suffices.

## 4 Recognizing and exploiting dependent predicates

In this section we tackle the problem of recognizing guessed predicates that functionally depend on others in a given specification. This means that, for every solution of any instance, the extension of a dependent guessed predicate is determined by the extensions of the others.

In [3] we showed how dependent predicates arise very often in declarative problem specifications, especially when auxiliary information or partial computations must be maintained. We recall in Subsection 4.1 the formal definition of dependent predicate in a specification, taken from [3], as well as some results. Then, in Subsection 4.2 we show how a first-order theorem prover can be effectively used to automatically recognize functional dependencies.

| Spec | Symmetry | CPU time (sec) |
|---|---|---|
| 3-coloring | $\sigma^{R,G}$ | 0.27 |
| Not-all-equal Sat | $\sigma^{T,F}$ | 0.22 |
| Not-all-equal 3-Sat | $\sigma^{T,F}$ | 4.71 |
| Social golfer (unfolded) | $\sigma_W^{G,G'}$ | 0.96 |

**Table 1.** Performance of OTTER for proving that a UVT is a UVS.

| Spec | Symmetry | $\beta(\boldsymbol{\mathcal{S}})$ | CPU time (sec) | |
|---|---|---|---|---|
| | | | Cond. (9) (MACE) | Cond. (10) (OTTER) |
| 3-coloring | $\sigma^{R,G}$ | $\beta_{sel}^{R,G}(R,G,B)$ | 1.47 | 1.89 |
| | $\sigma^{R,G}$ | $\beta_{least}^{R,G}(R,G,B)$ | 2.67 | ? |
| | $\sigma^{R,G}$ | $\beta_{card}^{R,G}(R,G,B)$ | 0.25 | – |
| Social golfer | $\sigma_W^{G,G'}$ | $\beta_{least\ W}^{G,G'}(\cdots)$ | 0.04 | ? |

**Table 2.** Performance of OTTER for proving that a formula $\beta(\boldsymbol{\mathcal{S}})$ is symmetry-breaking for a given symmetry. "–" means that a timeout of 1 hour occurred, while "?" that OTTER terminated without a proof.

### 4.1 Definitions

**Definition 4 (Functional dependence of a set of predicates).** *Given a problem specification* $\psi \doteq \exists \boldsymbol{\mathcal{S}}\boldsymbol{\mathcal{P}}\ \phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{R}})$*, with input schema* $\boldsymbol{\mathcal{R}}$*, set* $\boldsymbol{\mathcal{P}}$ functionally depends *on set* $\boldsymbol{\mathcal{S}}$ *if, for each instance* $\boldsymbol{\mathcal{I}}$ *of* $\boldsymbol{\mathcal{R}}$ *and for each pair of interpretations* $M$, $N$ *of* $(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{P}})$ *it holds that, if* (i) $M \neq N$*,* (ii) $M,\boldsymbol{\mathcal{I}} \models \phi$*, and* (iii) $N,\boldsymbol{\mathcal{I}} \models \phi$*, then* $M_{|\boldsymbol{\mathcal{S}}} \neq N_{|\boldsymbol{\mathcal{S}}}$*, where* $\cdot_{|\boldsymbol{\mathcal{S}}}$ *denotes the restriction of an interpretation to predicates in* $\boldsymbol{\mathcal{S}}$*.*

As an example, in Graph 3-coloring (cf. Example 1), one of the three guessed predicates (e.g., $B$) is dependent on $R$ and $G$, since $\forall X\ B(X) \leftrightarrow \neg(R(X) \vee G(X))$. Also, in Not-all-equal Sat (cf. Example 3), predicate $T$ is dependent on $F$ or vice versa.

The problem of checking functional dependencies reduces to semantic properties of a first-order formula, as the following result of [3] shows:

**Proposition 2.** *Let* $\psi \doteq \exists \boldsymbol{\mathcal{S}}\boldsymbol{\mathcal{P}}\ \phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{R}})$ *be a problem specification with input schema* $\boldsymbol{\mathcal{R}}$*.* $\boldsymbol{\mathcal{P}}$ *functionally depends on* $\boldsymbol{\mathcal{S}}$ *iff the following formula is valid:*

$$[\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{P}},\boldsymbol{\mathcal{R}}) \wedge \phi(\boldsymbol{S'},\boldsymbol{P'},\boldsymbol{\mathcal{R}}) \wedge \neg(\boldsymbol{\mathcal{S}}\boldsymbol{\mathcal{P}} \equiv \boldsymbol{S'}\boldsymbol{P'})] \rightarrow \neg(\boldsymbol{\mathcal{S}} \equiv \boldsymbol{S'}). \qquad (14)$$

To simplify notations, given two sets of predicates $\boldsymbol{T}$ and $\boldsymbol{T'}$ of the same arities, we write $\boldsymbol{T} \equiv \boldsymbol{T'}$ as a shorthand for $\bigwedge_{T \in \boldsymbol{T}} \forall \boldsymbol{X}\ T(\boldsymbol{X}) \equiv T'(\boldsymbol{X})$.

Unfortunately, in [3], the problem of checking whether the set of predicates in $\boldsymbol{\mathcal{P}}$ is functionally dependent on the set $\boldsymbol{\mathcal{S}}$ is proven to be not decidable. Nonetheless, as shown in the next section, an ATP can perform very well in deciding whether formulae of the kind of (14) are valid or not.

## 4.2 Experiments with the theorem prover

Using Proposition 2 it is easy to write a first-order formula that is valid if and only if a given dependence holds. We used OTTER for proving the existence of dependencies among guessed predicates of different problem specifications:

- Graph 3-coloring (cf. Example 1), where one among the guessed predicates $R$, $G$, $B$ is dependent on the others.
- Not-all-equal Sat (cf. Example 3), where one between the guessed predicates $T$ and $F$ is dependent on the other.

For each of the above specifications, we wrote a first-order encoding of formula (14), and gave its negation to OTTER in order to find a refutation. For the purpose of testing effectiveness of the proposed technique in the context of more complex specifications, we considered also the *Sailco inventory*, and the *HP 2D-Protein folding* problems (description of the latter problem is omitted for lack of space, and can be found in [3]).

*Example 6 (The Sailco inventory problem [18, Section 9.4, Statement 9.17]).* This problem specification, part of the OPLSTUDIO distribution package (as file `sailco.mod`), models a simple inventory application, in which the question is to decide how many sailboats the Sailco company has to produce over a given number of time periods, in order to satisfy the demand and to minimize production costs. The `demand` for the periods is known and, in addition, an inventory `inv` of boats is available initially. In each period, Sailco can produce a maximum number of boats at a given unitary cost. Additional boats (`extraBoats`) can be produced, but at higher cost. Storing boats in the inventory also has a cost per period.

The following relationship among the variables holds: `inv[t] = regulBoat[t] + extraBoat[t] - demand[t] + inv[t-1]`. Of course, the same relationship holds in the equivalent ESO specification (omitted for brevity), making guessed predicate $inv(\cdot,\cdot)$ functionally dependent on $regulBoat(\cdot,\cdot)$ and $extraBoat(\cdot,\cdot)$. (The arity of such predicates is 2, since functions must be modelled in ESO as relations, plus additional constraints.)

We opted for an OTTER encoding that uses function symbols: as an example, the `inv[]` array is translated to a function symbol $inv(\cdot)$ rather than to a binary predicate. More precisely, according to Proposition 2, a pair of function symbols $inv(\cdot)$ and $inv'(\cdot)$ is introduced. The same happens for `regulBoat[]` and `extraBoat[]`. Moreover, we included in the OTTER formula additional constraints in order to make the system able to correctly handle expressions of interest (in particular, arithmetic constraints). Notably, the following formulae allow to infer $\forall t\ inv(t) = inv'(t)$ from equality of $inv$ and $inv'$ at the initial time period and equivalence of increments in all time intervals of length 1.

```
equalDiscrete <-> (inv(0) = inv_prime(0) &
                  (all t (t > 0 -> (inv(t) - inv(t-1)) =
                                   (inv_prime(t) - inv_prime(t-1))))).
induction <-> (equalDiscrete -> (all t (inv(t) = inv_prime(t)))).
```

| Spec | $\boldsymbol{\mathcal{S}}$ | $\boldsymbol{\mathcal{P}}$ | CPU time (sec) |
|---|---|---|---|
| 3-coloring | $R$, $G$ | $B$ | 0.25 |
| Not-all-equal 3-Sat | $T$ | $F$ | 0.38 |
| Sailco | $regulBoat$, $extraBoat$ | $inv$ | 0.21 |
| HP 2D-Prot. fold. (simpl.) | $Move$ | $X, Y$ | 569.55 |

**Table 3.** Performance of OTTER for proving that the set $\boldsymbol{\mathcal{P}}$ of guessed predicates is functionally dependent on the set $\boldsymbol{\mathcal{S}}$.

Results of the experiments to check that dependencies for all the aforementioned problems hold are shown in Table 3. As it can be observed, for almost all of them the time needed by OTTER is very small. Actually, as for HP 2D-Protein folding, OTTER was able to solve only a simplified version of the problem (in which dependence of guessed predicate *Hits* is not checked, cf. [3]). To this end, we are investigating the use of other theorem provers (cf. Section 5).

When functional dependencies have been recognized in a problem specification, they can be exploited in several ways, in order to improve the solver efficiency. In [3] we show how simple search strategies can be automatically synthesized in order to avoid branches on dependent predicates. Despite their simplicity, such strategies are very effective in practice. As an example, speed-ups of about 99% have been observed for Protein folding and Blocks world [3].

## 5 Conclusions and current research

The use of automated tools for preprocessing CSPs has been limited, to the best of our knowledge, to the instance level. In this paper we proved that current ATP technology is able to perform significant forms of reasoning on specifications of constraint problems. We focused on two forms of reasoning: symmetry detection and breaking, and functional dependence checking. Reasoning has been done for various problems, including the ESO encodings of graph 3-coloring and Not-all-equal Sat, and the OPL encoding of an inventory problem, and 2D HP-Protein folding. In the latter examples, arithmetic constraints exist, and we have shown how they can be handled. In many cases, reasoning is done very efficiently by the ATP, although effectiveness depends on the format of the input, and auxiliary propositional variables seem to be necessary. There are indeed some tasks which OTTER –in the automatic mode– was unable to do. Hence, we plan to investigate other provers, e.g., VAMPIRE [17]. We note that the wide availability of constraint problem specifications, both in computer languages, cf., e.g., [18], and in natural language, cf., e.g., [8], the CSP-Library[1], and the OR-Library[2], offers a brand new set of benchmarks for ATP systems, which is not represented in large repositories, such as TPTP[3].

[1] `www.csplib.org`
[2] `www.ms.ic.ac.uk/info.html`
[3] `www.tptp.org`

Currently, the generation of formulae that are given to the theorem prover is performed by hand. Nonetheless, we claim that, as it can be observed from the various examples, this task can in principle be performed automatically, starting from an implemented language such as OPL. This is possible because the OPL syntax for expressing constraints is similar to first-order logic. We plan to investigate this topic in future research.

## References

1. M. Cadoli and T. Mancini. Detecting and breaking symmetries on specifications. In *Proc. of SymCon, in conj. with CP 2003*, 2003.
2. M. Cadoli and T. Mancini. Automated reformulation of specifications by safe delay of constraints. In *Proc. of KR 2004*, 2004. AAAI Press/The MIT Press.
3. M. Cadoli and T. Mancini. Exploiting functional dependencies in declarative problem specifications. In *Proc. of JELIA 2004*, 2004. Springer.
4. M. Cadoli and A. Schaerf. Compiling problem specifications into SAT. *Artif. Intell.*, 162:89–120, 2005.
5. E. Castillo, A. J. Conejo, P. Pedregal, R. Garcia, and N. Alguacil. *Building and Solving Mathematical Programming Models in Engineering and Science.* John Wiley & Sons, 2001.
6. R. Dechter. *Constraint Networks (Survey).* John Wiley & Sons, 1992.
7. R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, ed., *Complexity of Computation*, pages 43–74. Amer. Math. Soc., 1974.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.
9. E. Giunchiglia and R. Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In *Proc. of AI*IA'99*, 2000. Springer.
10. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. on Comp. Logic.* To appear.
11. C. M. Li. Integrating equivalency reasoning into Davis-Putnam procedure. In *Proc. of AAAI 2000*, 2000. AAAI Press/The MIT Press.
12. T. Mancini and M. Cadoli. Detecting and breaking symmetries by reasoning on problem specifications. In *Proc. of SARA 2005*, 2005. Springer.
13. W. McCune. MACE 2.0 reference manual and guide. Tech. Rep. ANL/MCS-TM-249, Argonne Nat. Lab., Math. and Comp. Sci. Div., 2001. Available at `http://www-unix.mcs.anl.gov/AR/mace/`.
14. W. McCune. Otter 3.3 reference manual. Tech. Rep. ANL/MCS-TM-263, Argonne Nat. Lab., Math. and Comp. Sci. Div., 2003. Available at `http://www-unix.mcs.anl.gov/AR/otter/`.
15. B. D. McKay. *Nauty* user's guide (version 2.2). Available at `http://cs.anu.edu.au/~bdm/nauty/nug.pdf`, 2003.
16. P. Meseguer and C. Torras. Solving strategies for highly symmetric CSPs. In *Proc. of IJCAI'99*, 1999. Morgan Kaufmann.
17. A. Riazanov and A. Voronkov. Vampire. In *Proc. of CADE'99*, 1999. Springer.
18. P. Van Hentenryck. *The OPL Optimization Programming Language.* The MIT Press, 1999.