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 (ATP) and constraint programming (CP). 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 paradigm of declarative programming is becoming very attractive for solving different classes of problems. In particular, constraint modelling and programming has emerged as a winning approach for, among the others, combinatorial, scheduling, planning, and resource allocation problems, since the high declarativeness of problem specifications may coexist with the increasing efficiency of state-of-the-art solvers. To this end, a constraint problem specification may definitively be viewed as a *program*, which is given in a highly declarative language, and in which users state *what* conditions a solution must satisfy, rather than *how* it can be computed. Ideally, the latter issue is left to the system.

However, the great emphasis that the declarative paradigm reserves to the modelling task collides with the limits of current systems for contraint solving, e.g., AMPL [13], OPL [31], GAMS [7], DLV [20], SMODELS [26], and NP-SPEC [6], where the efficiency of computation can be highly affected in several ways. To this end, several choices have to be made by users in order handle instances of realistic size, and this is one of the major obstacles to reach full declarativeness.

The user choices that greatly influence performances are many-fold. A key role is played by the model chosen for the given problem. In fact, different but equivalent formulations for the same problem usually exist, and adopting one of them can make the difference. To this end, many techniques have been proposed, in order to modify (or reformulate) the given constraint problem into an equivalent one, with the goal of reducing the solving time. Good examples are the addition of new constraints, e.g., symmetry-breaking (cf., e.g., [28, 8, 12, 22]), or implied constraints (cf., e.g., [30]), in order to, respectively, reduce the size of the search space, and to be able to perform better propagation, or the opposite strategy of deleting (abstracting) some constraints (cf., e.g., [17, 4]) in order to obtain a simplified problem whose solutions can be used to compute, in an efficient way (in some cases without further search), valid solutions of the original one. All of these approaches have been widely studied, and have led to different and widely independent (even if correlated) research fields.

A second user choice that can greatly affect system performance concerns the search strategy to be applied, and the heuristic to be used to order variables and domain values to branch on. To this end, even if some systems, e.g., OPL [31], use a default strategy, there are many situations in which a smart reasoning on the problem specification is needed for inferring a good one. 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. If some of the predicates are recognized to be dependent on the others (cf. forthcoming Section 4), the declarative constraint program provided by the user can be transformed by synthesizing a search strategy that exploits such dependencies, e.g., by avoiding branches on those predicates (cf. [5]).

Although much research has been done on these aspects, almost 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: it is the designer that chooses, based on her experience, the best formulation for a problem together with a suitable search strategy, for the solver used. 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. Good examples are the use of packages such as NAUTY [25] or CGRASS [14] for finding symmetries and useful implied constraints on Constraint Satisfaction Problems (CSPs), and the development of SAT algorithms that deal with dependent variables added during the clausification of non-CNF formulae [16], and with the so-called "equivalence clauses" [21].

On the other hand, current systems for CP exhibit a neat separation between problem *specifications* and *instances*, usually adopting a two-level architecture for finding solutions: the specification is instantiated (or grounded) against the instance, and then an appropriate solver is invoked (cf. Figure 1). Moreover, in many cases properties amenable to be optimized derive from the structure of the problem, and not from the specific instance considered. 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, at least for two reasons: *(i)* For designing systems that automatically detect structural properties, and autonomously proceed to their optimization; *(ii)* For designing computer-aided modelling tools that assist the user during this difficult but crucial phase, validating the correctness of her choices, by, e.g., confirming that an expected symmetry really holds, and that a given constraint correctly breaks it.

Although relations between constraint satisfaction and deduction have been observed since several years (cf., e.g., the early work [2], and [18] for an up-to-date report), not much work has been done on reasoning at the *specification* level. Current systems do not perform any kind of reasoning on the problem specification. An exception is given by OPL, which checks (syntactically) if the input specification contains only linear constraints and objective function. In this case, the (typically very efficient)

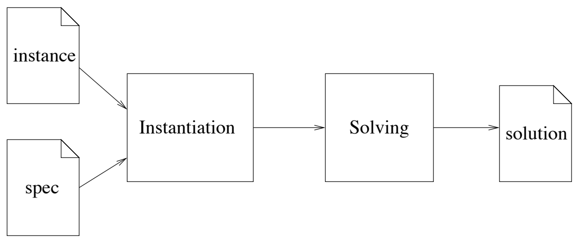

Fig. 1. Two-level architecture of current problem solving systems.

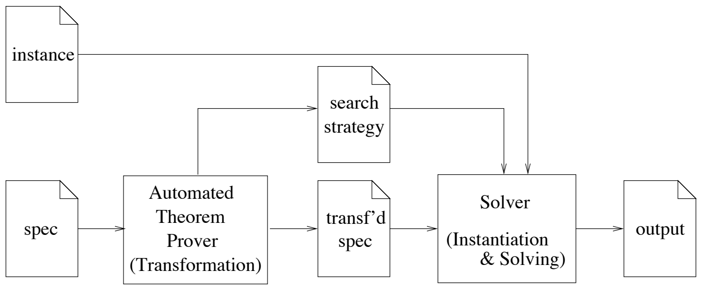
instance
search strategy
spec
Automated Theorem Prover (Transformation)
transf'd spec
Solver (Instantiation & Solving)
output

Fig. 2. Architecture of the problem solving system.

integer linear programming solver CPLEX is invoked; otherwise, the backtracking-based constraint programming solver SOLVER is used.

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 also observe that focusing on the specification does not rule out the possibility of additionally 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") [4], how to detect and break symmetries [3, 22], and how to recognize and exploit functional dependencies in a specification [5], showing how the original problem model can be appropriately reformulated. Experimental analysis shows that 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 that ATP technology can be effectively used to perform the required forms of reasoning.

The architecture of the system we envision is represented in Figure 2. Here, the output of the transformation phase is a reformulated constraint program and, possibly, a search strategy.

We report the results on using theorem provers and finite model finders (OTTER [24], and MACE [23], respectively) 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 [11, 27]. This is a simplification with respect to the modelling languages offered by current systems. Anyway, it is important to observe that, from an abstract point of view, all of them are extensions of ESO over finite databases, where the existential second-order quantifiers and the first-order formula represent, respectively, the *guess* and *check* phases of the constraint modelling paradigm. In particular, even if all such languages have a richer syntax and more complex constructs, in all of them it is possible to embed ESO queries, and the other way around is also possible, as long as only finite domains are considered. Intuitively, the relationship between ESO and available modelling languages is similar to that holding between assembler and high-level programming languages. Additionally, reasoning tasks on problem specifications written in ESO reduce to check semantic properties of logic formulae, which are, in many cases, first-order.

Formally, an ESO specification describing a search problem π is a formula

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

where:

- The set of predicates $\boldsymbol{\mathcal{R}} = \{R_1^{ar(R_1)}, \ldots, R_k^{ar(R_k)}\}$ is the input relational schema, i.e., a fixed set of relations of given arities denoting the schema of all input instances for π. An instance $\boldsymbol{\mathcal{I}}$ of the problem is given, as it happens in current systems, as a relational database over the schema $\boldsymbol{\mathcal{R}}$, i.e., as a finite extension for all relations in $\boldsymbol{\mathcal{R}}$.
- Existentially quantified predicates in set $\boldsymbol{\mathcal{S}} = \{S_1^{ar(S_1)}, \ldots, S_n^{ar(S_n)}\}$ are called *guessed*, and their possible extensions, with tuples in the domain given by constants occurring in $\boldsymbol{\mathcal{I}}$ plus those occurring in ϕ, i.e., the so called Herbrand universe, encode points in the search space for problem π on instance $\boldsymbol{\mathcal{I}}$.
- ϕ is a closed first-order formula on the relational vocabulary $\boldsymbol{\mathcal{S}} \cup \boldsymbol{\mathcal{R}} \cup \{=\}$ ("=" is always interpreted as identity), that encodes the constraints an extension of predicates in $\boldsymbol{\mathcal{S}}$ must satisfy to be a solution of π on instance $\boldsymbol{\mathcal{I}}$.

Formula ψ_π correctly encodes problem π if, for every input instance $\boldsymbol{\mathcal{I}}$, a bijective mapping exists between solutions to π and extensions of predicates in $\boldsymbol{\mathcal{S}}$ which verify $\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{I}})$. More formally, the following must hold:

$$\text{For each instance } \boldsymbol{\mathcal{I}}\text{:} \quad \boldsymbol{\Sigma} \text{ is a solution to } \pi(\boldsymbol{\mathcal{I}}) \quad \Longleftrightarrow \quad \{\boldsymbol{\Sigma}, \boldsymbol{\mathcal{I}}\} \models \phi.$$

It is worthwhile to note that, when a specification is instantiated, a constraint satisfaction problem (CSP, in the sense of [10]) is obtained.

Example 1 (Graph 3-coloring [15]). 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 ψ on the input schema $\boldsymbol{\mathcal{R}} = \{edge(\cdot,\cdot)\}$:

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

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). □

Much syntactic sugar can be added to ESO in order to natively express typed relations, functions, as well as bounded integers and arithmetics. This eases expressions, and makes them very similar to their counterparts in state-of-the-art modelling languages, like OPL. Nonetheless, this issue is out of the scope of this paper, and will not be considered (cf., e.g., [22] for more details). In what follows, we refer to the basic ESO framework, presented above. Nonetheless, some examples using the syntax of the implemented language OPL are recalled in Section 4. We observe that these examples contain arithmetic 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 [3], 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 [3]: in particular, non-monadic guessed predicates can be transformed in monadic ones by unfolding and by exploiting the finiteness of the input database. We refer to [3] 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. A complete theoretical framework for handling all kind of symmetries has been recently proposed by ourselves [22]. However, it reduces to that described in Subsection 3.1 in case of uniform symmetries on values.

Then, in Subsection 3.2 we 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 $\boldsymbol{\mathcal{S}} = \{S_1, \dots S_n\}$, S_i monadic for every $i \in [1, n]$, and input schema $\boldsymbol{\mathcal{R}}$, a* uniform value transformation (UVT) *for ψ 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 domain 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 $\boldsymbol{\mathcal{S}} = \{S_1, \dots S_n\}$, S_i monadic for every $i \in [1, n]$, and input schema $\boldsymbol{\mathcal{R}}$, and let σ be a UVT for ψ. Transformation σ is a* uniform value symmetry (UVS) *for ψ if every extension for $\boldsymbol{\mathcal{S}}$ which satisfies ϕ satisfies also ϕ^σ, defined as $\phi[S_1/\sigma(S_1), \dots, S_n/\sigma(S_n)]$ 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 σ has at least the corresponding uniform value symmetry.

In [3], it is shown that checking whether a UVT is a UVS reduces to checking equivalence of two first-order formulae:

Proposition 1. *Let ψ be a problem specification of the kind (1), with only monadic guessed predicates, and σ a UVT for ψ. Transformation σ is a UVS for ψ 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. They 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 σ is a closed (except for $\boldsymbol{\mathcal{S}}$) formula $\beta(\boldsymbol{\mathcal{S}})$ such that the following two conditions hold:*

1. *Transformation σ is no longer a symmetry for $\exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}})$:*

$$(\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}})) \not\equiv (\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}}))^\sigma\,; \tag{9}$$

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 σ:*

$$\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \models \bigvee_{\boldsymbol{\sigma} \in \sigma^*} (\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}}))^{\boldsymbol{\sigma}}; \tag{10}$$

where $\boldsymbol{\sigma}$ is a sequence (of finite length ≥ 0) over σ, 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., σ 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 σ, this actually reduces to sequences of length at most $|\boldsymbol{\mathcal{S}}|!$, since this is the maximum number of successive applications of σ that can lead to all different permutations. Finally, we note that the inverse logical implication in formula (10) always holds, because σ 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 ψ of the kind (1) can in principle be performed in the following way:

1. Selecting a UVT σ, i.e., a permutation of guessed predicates in ψ (if ψ has n guessed predicates, there are $n!$ such UVTs; n is usually very small);
2. Checking whether σ 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 [3] that this problem is not decidable, we show that 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 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 non-equivalence holds, and we are done. The approach of using finite model finders for looking for small counter-examples has already been successfully adopted in other contexts, e.g., software verification, and systems exhibiting such a feature do exist (cf., e.g., Alloy[1]).

In our research we used the first-order refutation theorem prover Otter [24], and the finite model finder Mace [23], both in full-automatic mode.

[1] `http://alloy.mit.edu`

Detecting symmetries The examples we worked on are the following.

Example 2 (Graph 3-coloring: Example 1 continued). The mapping $\sigma^{R,G} : \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}}$ 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 ϕ. 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. □

Example 3 (Not-all-equal Sat [15]). 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.

We assume that the input formula is encoded by the following relations:

– $inclause(\cdot,\cdot)$; tuple $\langle l, c\rangle$ is in *inclause* iff literal l is in clause c;
– $l^+(\cdot,\cdot)$; a tuple $\langle l, v\rangle$ is in l^+ iff l is the positive literal relative to the propositional variable v, i.e., v itself;
– $l^-(\cdot,\cdot)$; a tuple $\langle l, v\rangle$ is in l^- iff l is the negative literal relative to the propositional variable v, i.e., $\neg v$;
– $var(\cdot)$, containing the set of propositional variables occurring in the formula;
– $clause(\cdot)$, containing the set of clauses of the formula.

A specification for this problem is as follows (T and F represent the set of variables whose truth value is true and false, respectively):

$$\exists TF\ \forall X\ var(X) \leftrightarrow T(X) \vee F(X)\ \wedge \tag{11}$$
$$\forall X\ \neg\left(T(X) \wedge F(X)\right)\ \wedge \tag{12}$$
$$\forall C\ clause(C) \to \left[\exists L\ inclause(L,C) \wedge \forall V\ \left(l^+(L,V) \to T(V)\right) \wedge \left(l^-(L,V) \to F(V)\right)\right]\ \wedge \tag{13}$$
$$\forall C\ clause(C) \to \left[\exists L\ inclause(L,C) \wedge \forall V\ \left(l^+(L,V) \to F(V)\right) \wedge \left(l^-(L,V) \to T(V)\right)\right]. \tag{14}$$

Constraints (11–12) force every variable to be assigned exactly one truth value; moreover, (13) forces the assignment to be a model of the formula, while (14) requires at least one literal whose truth value is false in every clause.

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 ϕ. □

Example 4 (Social golfer (prob. 10 at `www.csplib.org`)). Given a set of players, a set of groups, and a set of weeks, encoded in monadic relations $player(\cdot)$, $group(\cdot)$, and $week(\cdot)$ respectively, this problem amounts to decide whether there is a way to arrange a scheduling for all weeks in relation *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 $|player|/|group|$, i.e., the group size, integral) is the following ($Play(P, W, G)$ states that player P plays in group G on week W):

$$\exists Play\ \forall P, W, G\ Play(P, W, G)\ \to\ player(P) \wedge week(W) \wedge group(G)\ \wedge \tag{15}$$
$$\forall P, W\ player(P) \wedge week(W)\ \to\ \exists G\ Play(P, W, G)\ \wedge \tag{16}$$

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.

$$\forall P, W, G, G' \; \mathit{Play}(P, W, G) \wedge \mathit{Play}(P, W, G') \;\rightarrow\; G = G' \;\wedge \tag{17}$$

$$\begin{aligned} &\forall P, P', W, W', G, G' \\ &\quad (P \neq P' \wedge W \neq W' \wedge \mathit{Play}(P, W, G) \wedge \mathit{Play}(P', W, G)) \rightarrow \\ &\qquad \neg\, [\mathit{Play}(P, W', G') \wedge \mathit{Play}(P', W', G')] \;\wedge \end{aligned} \tag{18}$$

$$\begin{aligned} &\forall G, G', W, W' \; \mathit{group}(G) \wedge \mathit{group}(G') \wedge \mathit{week}(W) \wedge \mathit{week}(W') \;\rightarrow \\ &\quad |\{P : \mathit{Play}(P, W, G)\}| = |\{P : \mathit{Play}(P, W', G')\}|. \end{aligned} \tag{19}$$

Constraints (15–17) force *Play* to be a total function assigning a group to each player on each week; moreover, (18) is the *meet only once* constraint, while (19) forces groups to be of the same size (this constraint can be written in ESO using standard techniques).

In order to highlight UVSs according to Definition 2, we need to substitute the ternary guessed predicate *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 [3], and has not to be performed in practice. □

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 $\mathit{clause}(\cdot, \cdot, \cdot)$, 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 did not terminate in one hour for a larger number of weeks or groups, and terminated without a proof for the original specification the with ternary guessed predicate $\mathit{Play}(\cdot, \cdot, \cdot)$. We are currently investigating the use of other theorem provers, 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. We wrote a first-order logic formula encoding condition of Proposition 1, and gave its negation to Otter in order to find

```
set(auto).
formula_list(usable).

% Definition of phi
covering <-> (all x (R(x) | G(x) | B(x))).
disjRG   <-> (all x (R(x) -> - G(x))).
disjRB   <-> (all x (R(x) -> - B(x))).
disjBG   <-> (all x (B(x) -> - G(x))).
goodColR <-> (all x y (x!= y & R(x) & R(y) -> - Edge(x,y))).
goodColG <-> (all x y (x!= y & G(x) & G(y) -> - Edge(x,y))).
goodColB <-> (all x y (x!= y & B(x) & B(y) -> - Edge(x,y))).

3col   <-> (covering & disjRG & disjRB & disjBG & goodColR & goodColG & goodColB).

% Definition of phi^(sigma)
covering_sigma <-> (all x (G(x) | R(x) | B(x))).
disjGR_sigma   <-> (all x (G(x) -> - R(x))).
disjGB_sigma   <-> (all x (G(x) -> - B(x))).
disjBR_sigma   <-> (all x (B(x) -> - R(x))).
goodColG_sigma <-> (all x y (x!= y & G(x) & G(y) -> - Edge(x,y))).
goodColR_sigma <-> (all x y (x!= y & R(x) & R(y) -> - Edge(x,y))).
goodColB_sigma <-> (all x y (x!= y & B(x) & B(y) -> - Edge(x,y))).

3col_sigma   <-> (covering_sigma & disjGR_sigma & disjGB_sigma & disjBR_sigma &
                  goodColG_sigma & goodColR_sigma & goodColB_sigma).

% Conjecture: Is 3col equivalent to 3col_sigma?
-(3col <-> 3col_sigma).

end_of_list.
```

Fig. 3. OTTER input file to prove that $\sigma^{R,G}$ is a UVS for the 3-coloring problem specification.

a refutation. Figure 3 shows the whole OTTER code used to check that $\sigma^{R,G}$ is a UVS for the 3-coloring specification.

As for proving that some UVTs are not 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 ϕ: 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 formula:

$$\beta_{SA}^{R,G}(R,G,B) \doteq R(\overline{v}) \vee B(\overline{v}), \tag{20}$$

that forces a selected node, say $\overline{v}$, not to be colored in green (*selective assignment* of colors for node $\overline{v}$). It is worth noting that the simpler formula $R(\overline{v})$ is not a symmetry-breaking formula for $\sigma^{R,G}$, since it does not match condition (10) of Definition 3. To give the intuition for this fact, it suffices to observe that the symmetric assignment would color $\overline{v}$ in green ($R(\overline{v})^{\sigma^{R,G}}$ is $G(\overline{v})$), so loosing all solutions that color $\overline{v}$ in blue. Actually, the stronger formula $R(\overline{v})$ breaks two different symmetries, $\sigma^{R,G}$ and $\sigma^{R,B}$, and can be obtained as the logical "and" of (20) and $R(\overline{v}) \vee G(\overline{v})$.

As described in [3, 22], a UVS can be broken in several ways, and with different effectiveness. A second (and more useful) formula to break $\sigma^{R,G}$ is the following one:

$$\beta_{LI}^{R,G}(R,G,B) \ \doteq\ \forall w\ G(w) \rightarrow \exists v\ R(v)\ \wedge\ v < w, \tag{21}$$

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).

Finally, a third, and more complex, symmetry-breaking formula for $\sigma^{R,G}$ is:

$$\beta_{SB}^{R,G}(R,G,B) \ \doteq\ |R| \leq |G|, \tag{22}$$

that forces green nodes to be at least as many as the red ones (*size-based ordering* on red and green nodes). It is easy to prove that formulae (20), (21) and (22) 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 (22), some considerations are in order, since this example highlights some difficulties that can arise when using first-order ATPs. In fact, formula (22) 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. In the following we show an example of first-order definable steps that imply that condition (9) of Definition 3 holds, for the case of a formula $\beta(\boldsymbol{\mathcal{S}})$ expressed in ESO, i.e., of the form:

$$\beta(\boldsymbol{\mathcal{S}}) \ \doteq\ \exists \boldsymbol{\mathcal{T}}\ \gamma(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{T}}), \tag{23}$$

where $\gamma(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{T}})$ is a (possibly quantified) first-order formula open only with respect to $\boldsymbol{\mathcal{S}}$, i.e., the set of guessed predicates of the whole specification, and $\boldsymbol{\mathcal{T}}$, i.e., the set of existentially quantified predicates in $\beta(\boldsymbol{\mathcal{S}})$ (cf. forthcoming Example 6).

For such a formula, since $\boldsymbol{\mathcal{T}}$ occurs neither in $\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}})$ nor in $\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}})^\sigma$, condition (9) of Definition 3 becomes the following:

$$\exists \boldsymbol{\mathcal{T}}\ \left(\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}})\right)\ \not\equiv\ \exists \boldsymbol{\mathcal{T}}\ \left(\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}})\right)^\sigma . \tag{24}$$

We immediately note that testing whether $(\phi \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}}))\ \not\equiv\ (\phi \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}}))^\sigma$ does not suffice. However, we can proceed as follows: our goal is to find an extension $\langle \overline{\boldsymbol{S}}, \overline{\boldsymbol{R}}\rangle$ for predicates in $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{R}}$ which is a model of the left hand side of (24) and not of the right hand side (or vice versa). Thus, we are interested in an interpretation $\langle \overline{\boldsymbol{S}}, \overline{\boldsymbol{R}}\rangle$ for which an extension for predicates in $\boldsymbol{\mathcal{T}}$ that satisfies $\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}})$ does exist, but an extension for predicates in $\boldsymbol{\mathcal{T}}$ that satisfies $(\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}}))^\sigma$ does not (or vice versa).

To this end, we can use a first-order finite model finder to get an extension $\langle \overline{\boldsymbol{S}}, \overline{\boldsymbol{R}}, \overline{\boldsymbol{T}}\rangle$ for $\boldsymbol{\mathcal{S}}$, $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{T}}$ such that:

$$\langle \overline{\boldsymbol{S}}, \overline{\boldsymbol{R}}, \overline{\boldsymbol{T}}\rangle\ \models\ \phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}}).$$

$\langle \overline{\boldsymbol{S}}, \overline{\boldsymbol{R}}\rangle$ consists of an instance for the problem $\psi \doteq \exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}})$ plus a solution for that instance. Moreover, this solution satisfies the symmetry-breaking formula $\beta(\boldsymbol{\mathcal{S}}) \doteq \exists \boldsymbol{\mathcal{T}}\ \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}})$.

Of course we are not yet guaranteed that for $\langle \overline{\boldsymbol{S}}, \overline{\boldsymbol{R}}\rangle$ an extension $\overline{\overline{\boldsymbol{T}}}$ such that $\left(\phi(\overline{\boldsymbol{S}},\overline{\boldsymbol{R}}) \wedge \gamma(\overline{\boldsymbol{S}},\overline{\overline{\boldsymbol{T}}})\right)^\sigma$ evaluates to true does not exist. But, if $\gamma(\overline{\boldsymbol{S}},\boldsymbol{\mathcal{T}})^\sigma$ is a contradiction, this happens, and so we have found an instance $\overline{\boldsymbol{R}}$ and an extension $\overline{\boldsymbol{S}}$ for $\boldsymbol{\mathcal{S}}$ that is a solution for $\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}})$, but not for $(\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{S}}))^\sigma$. This implies that condition (9) holds.

It is worthwhile noting that, in general, not every model found in the first step satisfies the additional condition that $\gamma(\overline{\boldsymbol{S}},\boldsymbol{\mathcal{T}})^\sigma$ is a contradiction, so we may need to *backtrack* in order to look for another model. To reduce the number of such backtracks, a simple heuristic is to look, in the first step, for a model of $\phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}}) \wedge \neg\gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}})^{\boldsymbol{\sigma}}$.

Summing up, the suggested procedure to check whether the ESO formula $\beta(\boldsymbol{\mathcal{S}}) \doteq \exists \boldsymbol{\mathcal{T}}\ \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}})$ (i.e., of the form (23)) satisfies (9) for UVS σ is the following:

1. Find $\langle \overline{\boldsymbol{S}}, \overline{\boldsymbol{R}}, \overline{\boldsymbol{T}}\rangle$ such that:
$$\langle \overline{\boldsymbol{S}}, \overline{\boldsymbol{R}}, \overline{\boldsymbol{T}}\rangle\ \models\ \phi(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{R}}) \wedge \gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}}) \wedge \neg\gamma(\boldsymbol{\mathcal{S}},\boldsymbol{\mathcal{T}})^{\boldsymbol{\sigma}};$$
2. If $\gamma(\overline{\boldsymbol{S}},\boldsymbol{\mathcal{T}})^\sigma\ \models\ \bot$, then condition (9) of Definition 3 holds; otherwise return to Step 1.

We observe that a *hybrid* strategy, that uses both a finite model finder and a theorem prover at the same time, can be effective in practice to perform Step 2.

Example 6 (Graph 3-coloring: Example 2 continued). We refer again to the 3-coloring problem specification, the UVS $\sigma^{R,G}$, and the symmetry-breaking formula $\beta^{R,G}_{SB}(R,G,B)$ (22). A translation of $\beta^{R,G}_{SB}$ in ESO is as follows:

$$\begin{aligned}
\beta^{R,G}_{SB}(R,G,B)\ \doteq\ \exists T\quad & \forall XY\quad T(X,Y) \rightarrow R(X) \wedge G(Y)\ \wedge & (25)\\
& \forall X \exists Y\ \ R(X) \rightarrow T(X,Y)\ \wedge & (26)\\
& \forall XYY'\ T(X,Y) \wedge T(X,Y') \rightarrow Y = Y'\ \wedge & (27)
\end{aligned}$$

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

Table 2. Performance of MACE and 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 "?" means that OTTER terminated without a proof.

$$\forall X X' Y \ T(X,Y) \wedge T(X',Y) \rightarrow X = X' \tag{28}$$

where $\gamma(R,G,B,T)$ is its first-order part. Such formula evaluates to true if and only if a mapping T from tuples in R to tuples in G (25) exists, such that it is total (26), mono-valued (i.e., a function) (27), and injective (28). Since such a function exists if and only if the size of its domain is less than or equal to the size of its codomain, $\beta_{SB}^{R,G}(R,G,B)$ evaluates to true if and only if $|R| \leq |G|$.

By applying the procedure described above, in order to check whether such a formula is symmetry-breaking for $\sigma^{R,G}$, we first look for an interpretation $\langle \overline{R}, \overline{G}, \overline{B}, \overline{edge}, \overline{T} \rangle$ such that $\overline{R}, \overline{G}, \overline{B}$ define a good coloring of the graph encoded by relation $\overline{edge}$, and $\overline{T}$ is a total and injective function from tuples in $\overline{R}$ to those in $\overline{G}$, but not vice-versa, cf. Step 1 of the above procedure. One of the smallest examples of such an extension is the following:

$$\overline{R} = \{\}; \qquad \overline{G} = \{v\}; \qquad \overline{B} = \{\} \qquad \overline{edge} = \{\}; \qquad \overline{T} = \{\}.$$

Now, consider formula $\gamma(\overline{R}, \overline{G}, \overline{B}, T)^{\sigma^{R,G}}$, with $\overline{R}, \overline{G}, \overline{B}$ as before. This formula is satisfiable if and only if a total and injective function T from tuples in $\overline{G}$ to those in $\overline{R}$ exists, that is, if and only if $|\overline{R}| \geq |\overline{G}|$. By construction this is false (in Step 1 we defined $\langle \overline{R}, \overline{G}, \overline{B} \rangle$ such that $|\overline{R}| < |\overline{G}|$).

Hence, we have found an interpretation for predicates in $\boldsymbol{\mathcal{S}}$ and in $\boldsymbol{\mathcal{R}}$, namely $\langle \overline{R}, \overline{G}, \overline{B}, \overline{edge} \rangle$ which is a model of the left part of condition (24) and not of its right part. Thus, the second-order non-equivalence stated in (24) holds. □

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 time needed to perform all steps in order to check whether the ESO formula (22) satisfies the first condition of Definition 3, cf. the aforementioned procedure). As a final remark, we note that it is often possible to have candidate symmetry-breaking formulae, e.g., generalizations of (21), or (22), that respect the second condition of Definition 3 *by design* (cf. [22] for a detailed discussion). 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 [5] we showed that dependent predicates arise very often in declarative problem specifications, especially when auxiliary information or partial computations must be maintained, and that their recognition may significantly affect the efficiency of backtracking solvers, permitting the synthesis of suitable search strategies that, e.g., avoid branches on dependent variables.

Subsection 4.1 recalls the formal definition of dependent predicates in a specification, taken from [5], as well as some results. Then, Subsection 4.2 shows how a first-order theorem prover can be effectively used to automatically recognize functional dependencies.

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 and vice versa.

The problem of checking functional dependencies reduces to semantic properties of a first-order formula, as the following result of [5] 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:*

$$[\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{R}}) \wedge \phi(\boldsymbol{S'}, \boldsymbol{P'}, \boldsymbol{\mathcal{R}}) \wedge \neg(\boldsymbol{\mathcal{S}}\boldsymbol{\mathcal{P}} \equiv \boldsymbol{S'}\boldsymbol{P'})] \rightarrow \neg(\boldsymbol{\mathcal{S}} \equiv \boldsymbol{S'}). \tag{29}$$

To simplify notation, 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 $\bigwedge_{T \in \boldsymbol{T}} \forall \boldsymbol{X}\ T(\boldsymbol{X}) \leftrightarrow T'(\boldsymbol{X})$.

Unfortunately, in [5], 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 (29) 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 (29), 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, described next.

Example 7 (The Sailco inventory problem [31, 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 each period is known and, in addition, an inventory `inv` of already produced boats is available initially. In each period `t`, Sailco is able to produce boats at a given unitary cost, up to a certain limit. Additional boats can be produced, but at higher cost. Finally, in any period, boats which exceed the demand may be stored in the inventory, and sold later. Also storing boats in the inventory has a cost.

The search space is made of 3 arrays of variables: `regulBoat[t]`, `extraBoat[t]`, and `inv[t]` that guess, respectively, how many regular and extra boats must be produced, and how many boats are stored in the inventory in each period `t`. The complete OPL specification can be found in [5]. Here we note that the following relationship among the variables holds: `inv[t] = regulBoat[t] + extraBoat[t] - demand[t] + inv[t-1]`, i.e., the number of boats stored in the inventory in period `t` is given by that of boats stored in period `t-1` plus the total number of boats produced in period `t`, minus those sold in the same period. 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 point 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)))).
```

Example 8 (The HP 2D-Protein folding problem [19]). This specification models a simplified version of one of the most important problems in computational biology. It consists in finding the spatial conformation of a protein (i.e., a sequence of amino-acids) with minimal energy. The simplifications are twofold: *(i)* The amino-acids alphabet is reduced to just H (*hydrophobic*) and P (*polar*), and *(ii)* The protein is forced to fold in a 2D discrete space (i.e., a grid). However, the simplified problem is known to be NP-complete [9].

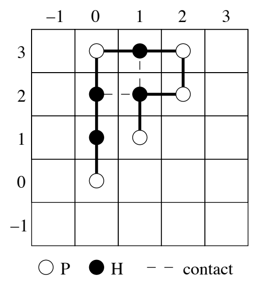

Fig. 4. HP 2D-Protein folding problem: A possible conformation for protein "PHHPHPPHP" with two contacts, and overall energy -2.

Given the sequence of amino-acids of the protein, i.e., a string over {H,P} of length n, the problem aims to find a connected shape for it on a 2D grid (with coordinates in $[-(n-1), (n-1)]$, starting at $(0, 0)$), which is non-crossing, and such that the number of "contacts", i.e., the number of non-sequential pairs of Hs for which the Euclidean distance of the positions is 1 is maximized (the overall energy is the opposite of the number of contacts). Figure 4 shows a possible conformation of the protein "PHHPHPPHP", with overall energy -2.

Various alternatives for the search space exist: as an example, we can guess the position on the grid of each amino-acid, and then force the shape to be connected, non-crossing, and with minimal energy. However, a preferred approach that reduces the size of the search space (4^n points versus $(2n)^{2n}$) is to guess the shape of the protein as a connected path starting at $(0, 0)$, by guessing, for each index $i \in [1, n-1]$, the direction (binary predicate *Move*) that the $(i+1)$-th amino-acid assumes wrt the i-th one (directions can only be North, South, East, West). The extension of *Move* for the shape in Figure 4 is:

$$\{\langle 1, N\rangle,\ \langle 2, N\rangle,\ \langle 3, N\rangle,\ \langle 4, E\rangle,\ \langle 5, E\rangle,\ \langle 6, S\rangle,\ \langle 7, W\rangle,\ \langle 8, S\rangle\}\,.$$

However, the latter model is not completely satisfactory: to express the non-crossing constraint, and to compute the number of contacts, absolute coordinates of each amino-acid must be computed and maintained. This can be done by introducing two additional binary guessed predicates, X and Y, which are functionally dependent on *Move*: hence, the non-crossing constraint may be expressed in terms of X and Y, stating that there are no pairs of distinct amino-acids whose absolute positions coincide. We implemented the last constraint by introducing an additional ternary guessed predicate *Hits* that maintains, for every position on the grid, the number of amino-acids of the protein that are placed on it at each point during the construction of the shape. This number cannot be greater than one, which implies that the string does not cross. Also the *Hits* predicate depends on *Move*.

Finally, analogously to the Sailco inventory problem, additional constraints have been added to the OTTER formula in order to correctly handle arithmetic constraints. In [5] an OPL specification of the HP 2D-Protein folding problem can be found, while [22] shows an ESO one.

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

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 $\boldsymbol{\mathcal{S}}$.

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). 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 [5] 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 other problems.

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 encodings of an inventory problem and of 2D HP-Protein folding. The latter examples contain arithmetic constraints, 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 [29]. We note that the wide availability of constraint problem specifications, both in computer languages, cf., e.g., [31], and in natural language, cf., e.g., [15], the CSP-Library[2], and the OR-Library[3], offers a brand new set of benchmarks for ATP systems, which is not represented in large repositories, such as TPTP[4].

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, because the OPL syntax for expressing constraints is similar to first-order logic. We plan to investigate this topic in future research.

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

References

1. *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, Austin, TX, USA, 2000. AAAI Press/The MIT Press.
2. W. Bibel. Constraint satisfaction from a deductive viewpoint. *Artificial Intelligence*, 35:401–413, 1988.
3. M. Cadoli and T. Mancini. Detecting and breaking symmetries on specifications. In *Proceedings of the Third International Workshop on Symmetry in Constraint Satisfaction Problems, in conjunction with the Ninth International Conference on Principles and Practice of Constraint Programming (CP 2003)*, pages 13–26, Kinsale, Ireland, 2003.
4. M. Cadoli and T. Mancini. Automated reformulation of specifications by safe delay of constraints. In *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 388–398, Whistler, BC, Canada, 2004. AAAI Press/The MIT Press.
5. M. Cadoli and T. Mancini. Exploiting functional dependencies in declarative problem specifications. In *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Artificial Intelligence*, pages 628–640, Lisbon, Portugal, 2004. Springer.
6. M. Cadoli and A. Schaerf. Compiling problem specifications into SAT. *Artificial Intelligence*, 162:89–120, 2005.
7. E. Castillo, A. J. Conejo, P. Pedregal, R. Garca, and N. Alguacil. *Building and Solving Mathematical Programming Models in Engineering and Science*. John Wiley & Sons, 2001.
8. J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 148–159, Cambridge, MA, USA, 1996. Morgan Kaufmann, Los Altos.
9. P. Crescenzi, D. Goldman, C. H. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3):423–466, 1998.
10. R. Dechter. Constraint networks (survey). In *Encyclopedia of Artificial Intelligence, 2nd edition*, pages 276–285. John Wiley & Sons, 1992.
11. R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation*, pages 43–74. American Mathematical Society, 1974.
12. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP 2002)*, volume 2470 of *Lecture Notes in Computer Science*, page 462 ff., Ithaca, NY, USA, 2002. Springer.
13. R. Fourer, D. M. Gay, and B. W. Kernigham. *AMPL: A Modeling Language for Mathematical Programming*. International Thomson Publishing, 1993.
14. A. Frisch, I. Miguel, and T. Walsh. CGRASS: A system for transforming constraint satisfaction problems. In *Proceedings of the Joint Workshop of the ERCIM Working Group on Constraints and the CologNet area on Constraint and Logic Programming on Constraint Solving and Constraint Logic Programming (ERCIM 2002)*, volume 2627 of *Lecture Notes in Artificial Intelligence*, pages 15–30, Cork, Ireland, 2002. Springer.
15. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, CA, USA, 1979.
16. E. Giunchiglia and R. Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In *Proceedings of the Sixth Conference of the Italian Association for Artificial Intelligence (AI*IA'99)*, volume 1792 of *Lecture Notes in Artificial Intelligence*, pages 84–94, Bologna, Italy, 2000. Springer.
17. F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57:323–389, 1992.
18. P. G. Kolaitis. Constraint satisfaction, databases, and logic. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 1587–1595, Acapulco, Mexico, 2003. Morgan Kaufmann, Los Altos.

19. K. F. Lau and K. A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986–3997, 1989.
20. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*. To appear.
21. C. M. Li. Integrating equivalency reasoning into Davis-Putnam procedure. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)* [1], pages 291–296.
22. T. Mancini and M. Cadoli. Detecting and breaking symmetries by reasoning on problem specifications. In *Proceedings of the Sixth International Symposium on Abstraction, Reformulation and Approximation (SARA 2005)*, volume 3607 of *Lecture Notes in Artificial Intelligence*, pages 165–181, Airth Castle, Scotland, UK, 2005. Springer.
23. W. McCune. MACE 2.0 reference manual and guide. Technical Report ANL/MCS-TM-249, Argonne National Laboratory, Mathematics and Computer Science Division, May 2001. Available at `http://www-unix.mcs.anl.gov/AR/mace/`.
24. W. McCune. Otter 3.3 reference manual. Technical Report ANL/MCS-TM-263, Argonne National Laboratory, Mathematics and Computer Science Division, August 2003. Available at `http://www-unix.mcs.anl.gov/AR/otter/`.
25. B. D. McKay. *Nauty* user's guide (version 2.2). Available at `http://cs.anu.edu.au/~bdm/nauty/nug.pdf`, 2003.
26. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
27. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Publishing Company, Reading, Massachussetts, Reading, MA, 1994.
28. J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In H. J. Komorowski and Z. W. Ras, editors, *Proceedings of the Seventh International Symposium on Methodologies for Intelligent Systems (ISMIS'93)*, volume 689 of *Lecture Notes in Computer Science*, pages 350–361, Trondheim, Norway, 1993. Springer.
29. A. Riazanov and A. Voronkov. Vampire. In *Proceedings of the Sixteenth International Conference on Automated Deduction (CADE'99)*, volume 1632 of *Lecture Notes in Computer Science*, pages 292–296, Trento, Italy, 1999. Springer.
30. B. M. Smith, K. Stergiou, and T. Walsh. Using auxiliary variables and implied constraints to model non-binary problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)* [1], pages 182–187.
31. P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.