Combining Formal Reasoning Techniques with CSP

Anne Liret, Pierre Roy, François Pachet.

(1) LIP6, 4 PLACE JUSSIEU, 75252 PARIS CEDEX 05, FRANCE.
e-mail: {liret, roy, pachet}@poleia.lip6.fr

20 octobre 1997

1 Introduction

Finite-domain constraint satisfaction programming (FDCSP) is a powerful paradigm for solving combinatorial problems. The success of FDCSP comes from the fact that lots of combinatorial problems fit naturally in this paradigm (e.g. scheduling, planning). These techniques are usually based on enumeration procedures augmented with various forms of pruning techniques, such as arc-consistency [15]. However, there are numerous examples in which theses techniques are clearly "misfit overkill". In particular, we are interested in studying a class of situations in which formal reasoning techniques can be substituted to pure enumeration to find solutions more easily or more quickly. Examples of such "favorable situations" are the following:

- The Pigeonholes problem consists in locating $(N+1)$ pigeons in N pigeonholes, such that a given hole accommodates at most one pigeon. This problem has no solution. This can be found by simply remarking that it is equivalent to finding an injection from a set into a smaller set. However, standard CSP techniques spend an exponential time to prove unsatisfiability.
- The equality $[4.M+3.N^2 = 34]$, where N and M are integer variables, illustrates a different kind of formal reasoning. A simple reasoning on parity shows that this equality has no solution : since 34 and $4.M$ are necessarily even, we deduce that $3.N^2$, and therefore $N2$, have to be even. $N2$ is even if and only if N is even. Finally, M and N have to be even numbers, thus $4.M+3.N^2$ is a multiple of 4. Since 34 is not a multiple of 4, there is no solution. Using standard CSP techniques, a combinatorial exploration of the domains is needed to obtain this result.
- If we now consider the following set of constraints, {(1) $X = Y$; (2) $Z = Y$; (3) $X \neq Z$ }, one can easily prove, by reasoning, that these constraints conflict because (1) and (2) imply $[X = Z]$, which obviously contradicts (3). A classical strategy based on arc-consistency would find this result using huge combinatorial exploration.

(•] In real world problems such as loan management, straightforward CSP formulation involves non linear numeric constraints, thus leading to hard, or even intractable, combinatorial problems. Frequently, formal reasoning can yield analytic expressions of solutions.

Moreover, these favorable configurations may appear during the resolution of a problem. As variables get instantiated, the constraint set changes, and such favorable situations may appear as a result of these partial instantiations.

In this paper, we are interested in identifying precisely such situations, and elaborating ways of combining some sort of "formal reasoning" on constraints, with classical enumeration techniques. This idea is not new and was the core of the Alice system [9]. Alice proposed a scheme for combining formal reasoning and constraint satisfaction to solve combinatorial problems. Alice was validated on several examples [10], and shown to be sometimes more efficient than specialized procedures. However, the Alice experience was difficult to share, because Alice was basically a black box. The performance relied on a judicious choice of complex heuristics ; too few information was accessible about the resolution strategy, and the trace facility was scarce. Strangely enough, while constraint satisfaction has received many attention in the last twenty years, Alice has somehow become a "mythical" system with no direct lineage.

We claim however, that Alice contains still unexploited good ideas for solving difficult combinatorial problems. More precisely, we consider it as a good starting point for studying precisely these "favorable situations", in which a fine tuning between formal reasoning (smart thinking) and brute force can save substantial amounts of time. Consequently, we started a project to reconstruct Alice using recent CSP technology. The aim of this system - called AliceTalks - is, eventually, to gain insights on the nature of favorable situations, to somehow master the beast.

This paper describes the implemented system, and sketches our experiments in progress.

2 The Original Alice System

Alice is a complex system, whose complete description may be found in chapter 8 of [11]. The system was also reconstructed by J. Pitrat using his declarative language Maciste (see chapter 12 of [14]), this reconstruction clarified the inner mechanism of propagation. However, to our knowledge, no systematic study of Alice, and especially the formal reasoning capabilities, has been conducted.

2.1 Overview

Alice is at the same time a language for stating combinatorial numerical problems and a system for solving them (Alice stands for A Language for Intelligent Combinatorial Exploration). The language is close to mathematical notations. A problem in Alice is stated using mathematical expressions and concepts from set and function theories.

The overall Alice system can be divided up in several modules: a parser, a graph, a set of constraints and the resolution algorithm. All these modules are responsible for particular tasks: the parser is used to generate a problem from a mathematical description. The graph maintains a representation of the variables and of the domains of the problem, and it is responsible for constraint propagation. The constraint set maintains the list of constraints and is responsible for formal reasoning. The resolution algorithm links together these different modules, and is controlled by several heuristics.

The resolution is based on a classical backtracking procedure in which the problem is simplified at each cycle using two strategies: 1) constraint propagation as in classical CSP to reduce the domains and 2) formal reasoning to generate simpler constraints. Domain reduction for individual constraints in Alice may be compared to constraints filtering, as defined in [2]. The global propagation scheme is similar to the forward-checking algorithm, described in [6]. The formal reasoning part of Alice is unique in the field of constraint satisfaction, and is described in the next section.

2.2 Formal Reasoning in Alice

Formal reasoning in Alice takes two forms: normalization of constraints, and combination rules. Normalization consists in rewriting all constraints in a unique form to allow the system to handle them in a systematic fashion. This process already allows to infer non trivial instantiations or failures. For instance, the normalized form of $[X+2.Y=X]$ is $[0=Y]$. Similarly, the normalized form of $[X=X+1]$ is $[false]$.

Constraint combination aims at creating new constraints which are simpler to handle. A particularly interesting case is when constraint combination leads to deducing a value, as in the problem $[X+Y=10$ and $X-Y=0]$. In other cases, the mere introduction of a redundant constraint may dramatically improve the resolution as illustrated by [3] for scheduling problems. Of course, constraint combination is an expensive process, and cannot be applied systematically. Deciding when to combine constraints is a tricky part of Alice and is precisely what makes it difficult to master.

3 AliceTalks: Revisiting Alice in an Object-Oriented Context

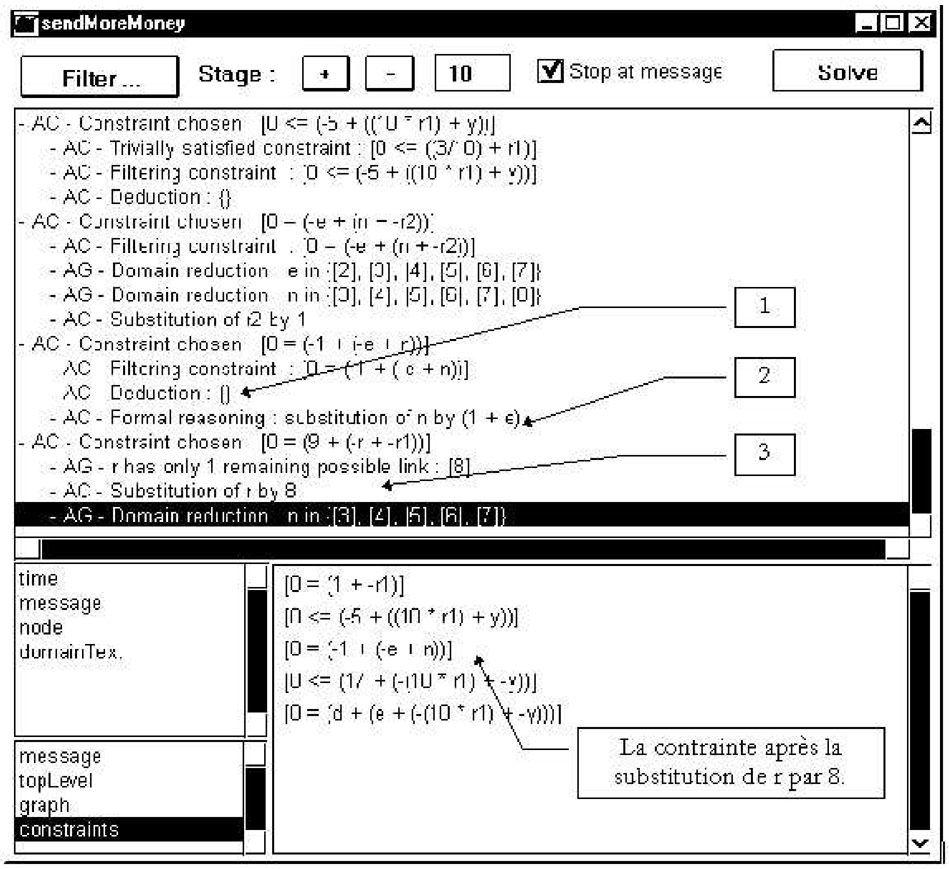
AliceTalks was born from the will to turn Alice into a more open and adaptable system. This section reports on this experience, describes the outcoming system and relates what of clarity and modularity we gained in the system design and use.

AliceTalks is designed with four distinct, independent, modules, corresponding to the original decomposition of Laurière's system. The parser improves the original Alice parser in two directions: it offers a more natural syntax and is more flexible since it is generated using a parser generator. The graph module is designed to be completely independent of the other modules (this was not the case in original Alice). The constraint set includes an extended formal reasoning mechanisms, as described in the next section. Moreover we added a sophisticated tracing module, described in Section 3.2, and a heuristic browser, which allows a fine-tuning of the resolution strategy. We will now focus on the formal reasoning part of AliceTalks.

3.1 Formal Reasoning in AliceTalks

As in Alice, the symbolic reasoning in AliceTalks is based on two procedures : 1) normalization, and 2) combination rules. Each of these two procedures has been extended in various ways.

Normalization of constraints is now seen under the light of standard rewriting systems [7], [5]. The theory of rewriting which originates from algebra and theorem proving, has quickly found a direct application in the development of formal calculus system, such as Macsyma [13]. It has been recently applied to the conception of prototyping systems [8] and proofs assistant based on types theory [4]. Rewriting consists in replacing subterms in a given expression by equal terms with the aim to obtain a normal form. This is ensured when the rule base is convergent; that is all sequences of rewriting steps are finite (termination property) and end with an irreducible unique expression (confluence property. The current rewriting rule base of AliceTalks contains the so-called set of arithmetic and logical expressions [1], augmented with functional terms. For instance, a constraint such as $[z+y+(y+t-z+x)=2.f(x)+(2.y)+(-2.f(x+0))]$ is rewritten as $[0=t+x]$. The confluence of the rule base of AliceTalks has been checked using a Knuth-Bendix standard completion program developed at LIP6 [12].



```
- AC - Constraint chosen   [0 <= (-5 + ((10 * r1) + y))]
    - AC - Trivially satisfied constraint : [0 <= (37 0) + r1)]
    - AC - Filtering constraint  : [0 <= (-5 + ((10 * r1) + y))]
    - AC - Deduction : {}
- AC - Constraint chosen   [0 = (-e + (n + -r2))]
    - AC - Filtering constraint  : [0 = (-e + (n + -r2))]
    - AG - Domain reduction   e in {[2], [3], [4], [5], [6], [7]}
    - AG - Domain reduction   n in {[3], [4], [5], [6], [7], [8]}
    - AC - Substitution of r2 by 1
- AC - Constraint chosen   [0 = (-1 + (-e + n))]
    - AC - Filtering constraint  : [0 = (-1 + (-e + n))]
    - AC - Deduction : {}
    - AC - Formal reasoning : substitution of n by (1 + e)
- AC - Constraint chosen   [0 = (9 + (-r + -r1))]
    - AG - r has only 1 remaining possible link : [8]
    - AC - Substitution of r by 8
    - AG - Domain reduction   n in {[3], [4], [5], [6], [7]}
```

```
[0 = (1 + -r1)]
[0 <= (-5 + ((10 * r1) + y))]
[0 = (-1 + (-e + n))]
[0 <= (17 + (-(10 * r1) + -y))]
[0 = (d + (e + (-(10 * r1) + -y)))]
```

FIG. 1 – *Excerpt of the resolution track of the send + more = money problem. Emphasis is set on an "intelligent" phase of the resolution.*

The combination rules can also be seen, from a theoretical standpoint, as yet another rewriting system. This rewriting system applies to a set of constraints (instead of applying to an individual constraint), to produce an "augmented" constraint set, consisting of the initial constraint set and a set of redundant constraints. Of course, this rewriting process is interesting only if the redundant constraints are interesting, from the perspective of constraint satisfaction. Currently, we implement this rewriting system with production rules, applied globally on the constraint set. For instance, consider the two following constraints, which appear during the resolution of the $send+more = money$ problem: $[N = 1+E]$ and $[10+E = R+R1+N]$. Note that in this situation, the domain of R1 is $\{0,1\}$ and the domains of N, R and E have been reduced to $\{1 ... 8\}$. An interesting combination of these two constraints in this case consists simply in "adding" the two constraints to create the constraint $[N + 10 + E = 1 + E + R + R1 + N]$. This constraint, once normalized is rewritten in $[9 = R + R1]$. This constraint is, of course, redundant with the two original constraints. However, in the current context of instantiation, a simple filtering of this constraint produces the instantiations $R1 = 1$ and $R = 8$. (see Figure 1). Note that without this constraint combination, filtering applied to the original constraint set would not lead to any domain reduction, let alone variable instantiation!

3.2 The Tracing Mechanism of AliceTalks

In the original Alice system, only few information were accessible about the resolution strategy. As a consequence, the user had no insights on the resolution, and was therefore unable to adapt the resolution strategy or heuristics to the resolution of a given problem.

We designed a tracing module that produces a detailed output of the resolution. In order to give useable information about the resolution, this output reflects the hierarchy of procedure calls (indentation). Furthermore, each trace event is typed according to 1) the module that produced it (AG for Alice graph, AC for Alice Constraint set, etc.), and 2) a label identifying the kind of trace generated. This typology can be used to filter displayed information (see Figure 1).

The 1 shows the tracing module during the resolution of the $send + more = money$ problem. At this particular step of the resolution, the variable s is instantiated to 9 and the carry $r2$ to 1. The tracing module allows to identify clearly three steps in the reasoning:

1. AliceTalks cannot perform any more domain reduction by constraint filtering.

2. AliceTalks chooses to perform a symbolic treatment on constraint $[0 = (-1 + (-e+n))]$. This results in trying to combine this constraint with other constraints of the constraint set. In practice, this amounts to substituting variable n by expression $(1 + e)$ in all constraints involving n. In this process, constraint $[0 = 10 + e - r - r1 - n]$ is rewritten in $[0 = 9 - r - r1]$.

3. Constraint $[0 = 9 - r - r1]$ is chosen for treatment. Simple filtering results in deducing that variable $r = 8$ and $r1 = 1$.

As noticed above, this instantiations have been obtained without any choice nor backtracking. Eventually, the resolution of the whole problem is achieved without any backtracking at all.

4 Conclusion, Experiments in Progress

In the range of combinatorial problem, we have identified a set of situations where formal reasoning is particularly interesting, because it allows to solve problems that classical CSP techniques cannot solve without any combinatorial enumeration. The Alice system was the first to exploit this idea. We have reconstructed Alice with the aim to shed lights on Laurière's ideas. In the outcoming system, AliceTalks, the rewriting rule base of Alice is complete and convergent, and the constraint filtering procedure use the recent works on CSP. One way of combining formal reasoning with classical constraint filtering, consists in handling constraints in their normal form and combining them with symbolic deduction rules. We plan to experiment AliceTalks on combinatorial problems in order to specify which situations are relevant to formal reasoning, and which heuristics can fine-tune the combination of these two strategies.

5 References

[1] L. Bachmair. Canonical Equational Proofs. Progress in Theoretical Computer Science. Birkhäuser, Romual V. Book, University of California, 1991.

[2] C. Bessière and J.Ch. Régin. Arc-consistency for General Constraint Networks: Preliminary Results. *Proceedings of IJCAI'97*, Nagoya, Japan, vol. 1, pp. 398-404, august 1997.

[3] Y. Caseau and F. Laburthe. Improved CLP Scheduling with Task Intervals. *Proceedings of the 11th International Conference on Logic Programming*, MIT Press, vol. 11, pp. 1994.

[4] C. Cornes, J. Courant, J-C. Filliatre, G. Huet, P. Manoury, C. Munoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi and B. Werner. The Coq Proof Assistant Reference Manual, version 5.10. INRIA, report 0177, 1995.

[5] N. Dershowitz and D.A. Plaisted. Logic Programming cum Applicative Programming. *IEEE Symposium on Logic Programming*, vol. 1, pp. 54-66, July 1985.

[6] R. Haralick and G. Elliot. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, vol. 14, pp. 263-313, 1980.

[7] J.P. Jouannaud and P. Lescanne. Rewrite systems, Handbook of Theoretical Computer Science.North-Holland,Van Leeuwen. vol. B, pp. 243-319, 1990.

[8] C. Kirchner, H. Kirchner and M.Vittek. Implementing Computational Systems with Constraints. *Proceedings of the first Workshop on Principles and Practice of Constraint Programming*, Providence, USA, Aris Kanellakis, Jean-Louis Lassez and Vijay Saraswat, vol. 1, pp. 166-175, 1993.

[9] J.L. Laurière. Un langage et un programme pour énoncer et résoudre des problèmes combinatoires. Ph. D. Thesis, University Pierre et Marie Curie, Paris, 1976.

[10] J.L. Laurière. A Language and a Program for Stating and Solving Combinatorial Problems Artificial Intelligence. *Artificial Intelligence*, vol. 10, pp. 29-127, 1978.

[11] J.L. Laurière. Intelligence artificielle, résolution de problèmes par l'homme et la machine. Eyrolles, Paris, 1986.

[12] P. Manoury, M. Simonot and J.L. Krivine. Des preuves de totalité de fonctions comme synthèse de programmes. Ph. D. thesis, Paris 7, Paris, France, 1992.

[13] MathLab. MACSYMA Reference Manual. The MathLab Group, Laboratory for Computer Science, report Cambridge, USA, January 1983.

[14] J. Pitrat. Penser autrement l'informatique. Hermès, Paris, 1993.

[15] P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, vol. 9, pp. 268-299, 1993. 1