CPTEST: A framework for the automatic fault detection, localization and correction of constraint programs

Nadjib Lazaar
INRIA Rennes Bretagne Atlantique Campus Beaulieu 35042 Rennes, France
nadjib.lazaar@inria.fr

***Abstract*—Constraint programs, such as those written in high-level constraint modeling languages, e.g., OPL (Optimization Programming Language), are more and more used in business-critical programs. As any other critical programs, they require to be thoroughly tested and corrected to prevent catastrophic loss of money. This paper is a demonstrations tool of CPTEST, a first testing tool for constraint programs. In particular, the paper presents the design of CPTEST and the implementation of our approaches enabling so automatic detection, localization and correction of faults in OPL programs.**

I. Introduction

Constraint Programming emerged since 1960's for solving difficult combinatorial problems [7] and evolved through the development of high-level modeling languages such as OPL (Optimization Programming Language) [8]. In these languages, statements are replaced by constraints and any constraint program execution yields solutions to a constraint system instead of returning values. A few years ago, constraint programs started to be used in critical applications. In particular, constraint programs were developed for e-commerce in business-critical software to solve combinatorial auctions [3]. Other critical software sectors also started to be concerned such as air-traffic control and management [1] and software verification and certification [2]. As any other critical programs, constraint programs must be thoroughly tested and corrected before being used on real-size instances of problems. Typical faults in critical constraint programs include bad formulation or refinement of a specific constraint and addition of erroneous constraints.

CPTEST includes a complete OPL parser and a backend process that produces dedicated OPL programs as output that must be solved to detect, localize and correct faults. The underlying constraint solver of CPTEST is based on ILOG CP Optimizer 2.1.

Figure 1 shows a snapshot of CPTEST, acting on a classical constraint program that solves the well-known N-queens problem.

Experiments with our CPTEST tool are promising and provide a first validation of the proposed approach on testing, localization and correction. CPTEST and some results on well-known problems are available online[1].

[1]www.irisa.fr/celtique/lazaar/CPTEST

II. CPTEST, a testing tool

Constraint program developers usually start with an initial declarative model of the problem, which faithfully translates the problem specification, without granting interest to its performances. As this model cannot handle large-sized instances of the problem, they exploit several refinement techniques to build an improved model. For example, usual refinement techniques include the use of dedicated data structures, constraint reformulation, *global constraints* addition, redundant and surrogate constraint addition, as well as constraints which break symmetries. The refinement process, carried out by the developer, is error-prone and most of the faults are introduced during this step.

In [5], we introduced CPTEST as a Software Testing framework for constraint programs written in OPL. In this framework, given a combinatorial problem to solve, a first highly declarative constraint model (called M for Model-Oracle) is taken as a reference to detect non-conformities within a refined and optimized constraint program solving the same problem (called CPUT for Constraint Program Under Test). As proving conformity on any instance of a problem is undecidable in the general case, we have proposed in [5] *one_negated* algorithm based on constraint negation and aiming at detecting non-conformities.

CPTEST takes into account the constraint solving problems (one or all solutions sought) and optimization problems (with an objective function). For instance, if we seek one solution, a non-conformity is either a solution of the CPUT that is not a solution of the model-oracle M, or CPUT is reduced to fail. Systematic non-conformities detection can be performed by combining the negation of a constraint of the model-oracle with the constraints of CPUT (i.e., $CPUT \wedge \neg C_i$ where $C_i \in M$). The idea here is to isolate a non-conformity by looking independently at each constraint of the model-oracle.

CPTEST can negate most of the constraints that can be expressed in OPL. The global constraints can be represented as an aggregation of primitive constraints and then computing their negation becomes trivial.

The **Test** part in Figure 1 shows the non-conformity reported and the negated constraint.

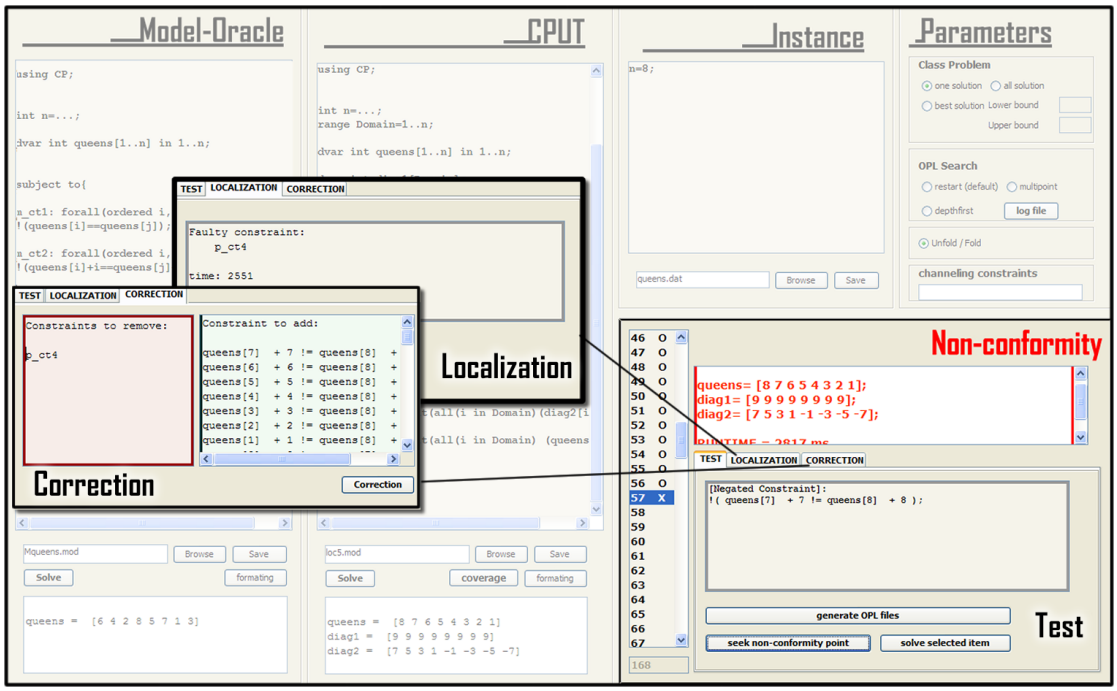

Fig. 1. CPTEST acting on n-queens problem.

III. CPTEST, A FAULT LOCALIZER TOOL

Once CPTEST detects a non-conformity, one faces the problem of localizing the faulty constraint. The major requirement on which our approach is based is the single fault hypothesis, i.e., there is only one constraint that is faulty in the CPUT. We proposed in [4] the *locate* algorithm and its implementation in CPTEST. The process of localization is based on relaxation to calculate the set of suspicious constraints that may contain the fault. A Constraint $C_i \in CPUT$ is suspicious to be a faulty constraint iff $M \wedge CPUT \backslash C_i$ is satisfiable.

The **localization** part in Figure 1 shows that there is only one constraint declared suspicious (i.e., `p_ct4`), in this case the reported constraint is the faulty one.

IV. CPTEST, A FAULT CORRECTION TOOL

Once a fault is localized in the CPUT, our approach to correction tries to reformulate the constraints of the *suspicious set*, returned by CPTEST, in order to restore the conformity with the model-oracle. In [6] we presented *correction* algorithm and its implementation for automatic correction based on the computation of a correction set from M which should be incorporated to the CPUT to correct it. The proposed algorithm aims at finding the set of constraints to revise non-conforming constraints within CPUT. It returns a set of pairs for which each pair is composed of two sets: a set of suspicious constraints R and a set A of constraints of the model-oracle. For each pair (R,A), removing or revising R and adding or reformulating A enables to correct automatically CPUT. For each suspicious set T_i, CPTEST computes the correcting constraints C_i of the model-oracle M for which $(CPUT \backslash T_i) \wedge \neg C_i$ does not fail. C_i should be in the correction set.

The **Correction** part in Figure 1 shows the pair of suspisious and correction set where the 4^{th} constraint of the CPUT must be replaced by the corresponding set of constraints to restore the conformity.

V. CONCLUSION

This tool demonstration paper has briefly presented the main features of CPTEST, namely testing, fault localization and correction. CPTEST implements our previous work and it is open for more extensions and features. We believe that tools that can automatically test and correct constraint programs will help to facilitate the adoption of constraint programs in critical applications.

REFERENCES

[1] P. Flener, J. Pearson, M. Agren, Garcia-Avello C., M. Celiktin, and S. Dissing. Air-traffic complexity resolution in multi-sector planning. *Journal of Air Transport Management*, 13(6):323 – 328, 2007.

[2] A. Gotlieb. Tcas software verification using constraint programming. *The Knowledge Engineering Review*, 2009.

[3] Alan Holland and Barry O'Sullivan. Robust solutions for combinatorial auctions. In *ACM Conference on Electronic Commerce (EC-2005)*, pages 183–192, 2005.

[4] N. Lazaar, A. Gotlieb, and Y. Lebbah. Fault localization in constraint programs. In *Proc. of the 2010 IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010*, Oct. 2010.

[5] N. Lazaar, A. Gotlieb, and Y. Lebbah. On testing constraint programs. In *Proc. of Principles of Constraint Programming, CP'2010*, Sept. 2010.

[6] N. Lazaar, A. Gotlieb, and Y. Lebbah. A framework for the automatic correction of constraint programs. In *Proc. of the 2011 IEEE International Conference on Software Testing, Verification and Validation, ICST 2011*, Mar. 2011.

[7] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[8] P. Van Hentenryck. *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA, 1999.