Gentra4cp: a Generic Trace Format for Constraint Programming*

Ludovic Langevine[1] and The OADymPPaC RNTL Project Team**

SICS – Uppsala – Sweden, langevin@sics.se

Introduction. Several debugging tools have been designed for constraint programming (CP). There is no ultimate tool, that satisfies all the needs, but rather a set of complementary tools. Most of them are dynamic tools. They collect data from the execution and produce abstract views of this execution, for instance a search-tree, the evolution of some domains, or an application-specific display.

So far, there are two issues concerning CP debugging tools. Firstly, each tool is dedicated to a given platform: there is no sharing of tools among the CP platforms, whereas most of solvers are based on the same techniques. Secondly, the extraction of execution data requires the solver to be instrumented. Such instrumentation is tedious and needs to access the solver code. We propose to address those two issues by means of a generic trace format which allows the execution to be described as a sequence of elementary events reflecting the behavior of the search procedure and the propagation process. The tools can then pick in the trace the data they need.

The genericity of the trace format is twofold. It is generic with respect to the tools since the trace contains all the data they need. It is also generic with respect to the solvers, as the very same set of events can reflect the executions of many solvers. The difficult and tedious work of instrumenting the solver is made only once. The efficiency of this scheme is ensured by an adaptation of the trace to the needs of a specific tool. There exist four tracers implementing Gentra4cp, namely for CHIP, Choco, PaLM and GNU-Prolog. Another tracer is currently under implementation within SICStus Prolog. Several tools are freely available.

A Semantics to Specify the Trace. The trace is based on an abstraction of the solver state, including the domains and the constraint store. This abstract state specifies what in the execution state can be observed by a tool. The trace format defines a set of possible events that are elementary modifications of this state. Each one of these events is specified by a state transition rule [1]. This semantics helps interpret the trace.

A Trace of Search and Propagation. The execution of a constraint program is often viewed as the overlapping of two levels: search and propagation. The trace format follows the same distinction.

* This work has been partly supported by the OADymPPaC project.

** Pierre Deransart, François Fages (Inria), Jean-Daniel Fekete, Mohammad Ghoniem, Narendra Jussien (EMN), Mireille Ducassé, Erwan Jahier (IRISA), Alexandre Tessier, Willy Lesaint, Gerard Ferrand, Ali Ed-Dbali (LIFO).

The search level is described by 9 different events. Firstly, three events deal with the creation of entities (creation of a variable, of a constraint, adding of a constraint into the store). Three specific events are used to trace solutions, failures and choice-points. Most of search-strategies can be modeled as the traversal of a search-tree. The *jump-to* event aims at tracing the restoration of a former choice-point. The latter four events (solution, failure, choice-point and jump-to) can trace any tree-based search strategy, such as chronological backtracking associated to depth-first search, branch-and-bound, or dynamic backjumping. Some search strategies cannot be described as a search-tree, for instance repair techniques such as MAC-DBT. Two additional events, *relax* and *restore* allow tracing the relaxation of a constraint and an elementary restoration of a domain. It is thus possible to trace a large variety of search strategies [1].

Different solvers exhibit different propagation strategies (e.g. variable- or constraint-oriented propagation queues, and some priorities). The trace format models the common behaviors while allowing solver specific extensions. Six different events have been defined to trace the propagation process. They capture the common concepts of the solvers we studied. Most of the differences of these solvers are reflected by the order in which those events occur. The *reduce* event traces an elementary domain reduction. Four events describe the propagation loop: the awakening of a constraint, its suspension, the detection of its entailment or of its unsatisfiability. Those five events are generic: they can be found in many constraint solvers, whatever the exact propagation strategy is. The sixth event, *schedule*, is used to trace solver-specific aspects of the propagation.

Easy development of tools thanks to XML. The trace format is an XML dialect. Since XML is a widely-used standard, an interested developer can choose among dozens of libraries to parse the trace. XML answers the needs of trace structuring thanks to attributes and nested elements: an event is an XML tag that encloses all its attached data. WBXML, a binary representation using a table of symbols, copes with the verbosity of XML and speeds up the parsing of the trace.

Trace Content Negotiation. The trace format makes for a potentially very detailed description of the execution. This potential trace is not meant to be exhaustively generated. The format provides a protocol between the debugging tool and the tracer. This protocol is used to set the actual level of details. This level of details can even be modified during the execution. This protocol is flexible enough to cope with the versatility of the tools and the evolution of their needs.

Evolution of the trace format. The OADymPPaC project is now finished, but the Gentra4cp format is still under development by instrumentation of new solvers, (e.g. SICStus Prolog). In order to take advantage of these experiences, a new SourceForge project has been set up (see http://tra4cp.sf.net).

References

1. The OADymPPaC Project. Generic trace format for constraint programming. http://contraintes.inria.fr/OADymPPaC/Public/Trace/indexl.html, May 2004.