A General Implementation Framework for Tabled CLP

Pablo Chico de Guzmán[1] **Manuel Carro**[1,2]
Manuel V. Hermenegildo[1,2] **Peter Stuckey**[3,4]

[1]IMDEA Software Institute, Spain [2]School of Computer Science, UPM, Spain
[3]NICTA Laboratory, Australia [4]School of Computer Science, U. of Melbourne, Australia

Abstract: This is a summary of [PCHS12], where a framework to combine tabling evaluation [TS86, War92] and constraint logic programming [JM94] is described (TCLP). This combination has been studied previously from a theoretical point of view [Tom97, Cod95], where it is shown that the constraint domain needs to offer projection and entailment checking operations in order to ensure completeness w.r.t. the declarative semantics. However, existing TCLP frameworks and implementations lack a complete treatment of constraint projection and / or entailment. The novelty of our proposal is that *we present a complete implementation framework for TCLP, independent from the constraint solver, which can use either precise or approximate projection and entailment, possibly with optimizations.*

Keywords: Constraint Logic Programming, Tabling, Implementation, Performance.

Tabled evaluation is an execution strategy for logic programs that records calls and their answers in order to reuse them in future calls. The operational semantics of tabled LP differentiates the first call to a tabled predicate, a *generator*, from subsequent identical calls, the *consumers*. Generators resolve against program clauses and insert the answers they compute in a global table. Consumers read answers from the global table and suspend when no more answers are available (therefore breaking infinite loops) and wait for the generation of more answers by their generators. A generator is said to be *complete* when it is known not to be able to generate more (unseen) answers. In order to check this property, a fixpoint procedure is executed where all the consumers inside the generator execution subtree are reading their pending answers until no more answers are generated. Due to the fixpoint computation of the completion operation, tabled LP is useful in several scenarios. The following program computes lengths of paths in a graph:

```
:− table path/3.
path(X,Y, 1) :− edge(X,Y).
path(X,Y, L) :− edge(X,Z), path(Z,Y,L2), L is L2 + 1.
edge(a,a).
```

where edge/2 defines the graph connectivity, the query ?− path(a,*Y*,*L*). returns the length of the paths from node a to all its reachable nodes, even if the graph has cycles. The previous program can be rewritten using constraints in order to return only those paths whose length is less than a given number. To this end, the third line of the program turns into the line:

```
path(X,Y, L) :− edge(X,Z), L #= L2 + 1, path(Z,Y,L2).
```

M.M. Gallardo, M. Villaret, L. Iribarne (Eds.): PROLE'2012, pp. 137-138, ISBN:978-84-15487-27-2.
Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.

Now, the query ?− L#=< 10, path(a,Y,L). creates a generator whose first clause computes the answer $\{Y$=a, L=1$\}$. On backtracking, the second clause of path/3 calls path(a,Y,$L2$), where $L2$ is knows to be less or equal than 9. This call is more specific than the previous generator and it can be considered as a consumer; this is detected using entailment. A set of constraints C_1 is entailed by another set of constraints C_2 in the domain D if $D \models C_2 \rightarrow C_1$. In order to check for entailment, the constraints associated with variables in the store but not in the call (e.g., L) should be ignored. This is done via projection on the variables of the tabled call. The projection of constraint C onto variables V is a constraint C' over variables V such that $D \models \exists \bar{x}.C \leftrightarrow C'$ where $\bar{x} = vars(C) - V$. The projection and entailment operations ensure soundness and termination for "constraint-compact" domains of constraints under tabled evaluation.

Our tabled CLP framework builds on a usual tabling system, providing checking of Herbrand terms (to avoid repeated tabled calls/answers) and consumer suspension/resumption, and defines an interface to be implemented by the constraint solver. The operations of this interface are responsible for managing the different constraint sets for tabled calls/answers with identical Herbrand terms (up to variable renaming). Answer merging can be also implemented to optimize the memory consumption of answer memoing. Therefore, our tabled CLP is independent from the constraint solver. We have validated the flexibility and generality of our framework (with excellent performance results) by implementing two examples: difference constraints and disequality constraints. Among others, tabled CLP can be applied to constraint databases, model checking of timed automata and abstract interpretation.

Acknowledgements: Work partially funded by EU projects IST-215483 *S-Cube* and FET IST-231620 *HATS*, MICINN projects TIN-2008-05624 *DOVES*, and CAM project S2009TIC-1465 *PROMETIDOS*. Pablo Chico was also funded by a MICINN FPU scholarship.

Bibliography

[Cod95] P. Codognet. A Tabulation Method for Constraint Logic Programming. In *INAP'95*. Tokyo, Japan, Oct. 1995.

[JM94] J. Jaffar, M. Maher. Constraint LP: A Survey. *JLP* 19/20:503–581, 1994.

[PCHS12] P. Chico de Guzmán, M. Carro, M. Hermenegildo, P. Stuckey. A General Implementation Framework for Tabled CLP. In Schrijvers and Thiemann (eds.), *FLOPS'12*. LNCS 7294, pp. 104–119. Springer Verlag, May 2012.

[Tom97] D. Toman. Memoing Evaluation for Constraint Extensions of Datalog. *Constraints* 2(3/4):337–359, 1997.

[TS86] H. Tamaki, M. Sato. OLD Resol. with Tabulation. In *ICLP*. Pp. 84–98. LNCS, 1986.

[War92] D. S. Warren. Memoing for Logic Programs. *CACM* 35(3):93–111, 1992.

Abstract Diagnosis for Timed Concurrent Constraint programs - Abstract

M. Comini[1] **L. Titolo**[1] **A. Villanueva**[2*]

Dipartimento di Matematica e Informatica, U. di Udine[1]
DSIC, Universitat Politècnica de València[2]

Abstract:

This short paper is a summary of the published paper [CTV11] where a general framework for the debugging of *tccp* programs is defined. To this end, a new compact, bottom-up semantics for the language that is well suited for debugging and verification purposes in the context of reactive systems was presented. In order to effectively implement the technique, we also provided an abstract semantics.

Keywords: concurrent constraint paradigm, denotational semantics, abstract diagnosis, abstract interpretation

1 Abstract Diagnosis for *tccp*

Finding program bugs is a long-standing problem in software construction. In the concurrent paradigms, the problem is even worse and the traditional tracing techniques are almost useless. There has been a lot of work on algorithmic debugging for declarative languages, which could be a valid proposal for concurrent paradigms, but little effort has been done for the particular case of the concurrent constraint paradigm (*ccp* in short; [Sar93]). The *ccp* paradigm is different from other programming paradigms mainly due to the notion of store-as-constraint that replaces the classical store-as-valuation model. In this way, the languages from this paradigm can easily deal with partial information: an underlying constraint system handles constraints on system variables. Within this family, [BGM00] introduced the *Timed Concurrent Constraint Language* (*tccp* in short) by adding to the original *ccp* model the notion of time and the ability to capture the absence of information. With these features, it is possible to specify behaviors typical of reactive systems such as *timeouts* or *preemption* actions, but they also make the language non-monotonic.

We develop an abstract diagnosis method for *tccp* using the ideas of the abstract diagnosis framework for logic programming [CLMV99]. This framework, parametric w.r.t. an abstract program property, is based on the use of an abstract immediate consequence operator to identify bugs in logic programs. The intuition of the approach is that, given an abstract specification of the expected behavior of the program, one automatically detects the errors in the program. The framework does not require the determination of symptoms in advance. In order to achieve an effective method, abstract interpretation is used to approximate the semantics, thus results may be less precise than those obtained by using the concrete semantics.

* This work has been partially supported by the EU (FEDER), the Spanish MICINN under grant TIN2010-21062-C02-02 and by Generalitat Valenciana, ref. PROMETEO2011/052.

M.M. Gallardo, M. Villaret, L. Iribarne (Eds.): PROLE'2012, pp. 139-140, ISBN:978-84-15487-27-2.
Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.