Qualitative Simulation with CLP

Aleksander Bandelj, Ivan Bratko and Dorian Šuc
Faculty of Computer and Information Science
University of Ljubljana, Slovenia
aleksander.bandelj@eranova.si, {dorian.suc, ivan.bratko}@fri.uni-lj.si.

Abstract

In this paper we explore advantages of implementing the QSIM qualitative simulation algorithm in CLP(FD) (Constraint Logic Programming over Finite Domains). In comparison with usual implementations of QSIM, eg. in Lisp, CLP enables elegant treatment of partially instantiated qualitative states and qualitative behaviors, and automatic separation of the variables of interest from independent or loosely related components of the model. This may lead to significant, sometimes exponential improvements in efficiency.

Introduction

Qualitative simulation (Forbus 1984; Kuipers 1986) proved useful for systems where accurate numerical information about the system is unavailable, or when a precise, numerical answer is not required or is overly elaborate. Kuipers' qualitative simulation algorithm QSIM (Kuipers 1986; 1994) predicts possible behaviors of a dynamic system modelled by qualitative constraints in the form of Qualitative Differential Equations (QDEs). A QDE model of the system and the smoothness assumption whereby the variables in the system may only change smoothly, constrain possible behaviors of the model. Qualitative simulation is naturally viewed as a set of Constraint Satisfaction Problems (CSP). An approach to solving CS problems is Constraint Logic Programming (CLP) (Jaffar & Maher 1994). CLP merges two declarative paradigms: constraint solving and logic programming. In this paper we investigate the benefits of implementing QSIM-like qualitative simulation in CLP as one approach to CSP.

Formulation of qualitative simulation as CSP has been explored by other authors. For example, Clancy and Kuipers (Clancy & Kuipers 1998) studied the qualitative model decomposition in the CS framework. Teleki (Teleki 1996) implemented QSIM in ECLiPSe, a CLP system. Some advantages of the CSP approach to qualitative simulation have been reported in comparison with the traditional QSIM algorithm and its Lisp implementation (Kuipers 1986; 1994). Experiments with our CLP implementation of qualitative simulation, called ASIM, show some considerable further advantages of CLP. The advantages are of various kinds as follows:

- computational efficiency, specially in cases when the modelled system exhibits a structure with subcomponents
- flexibility in answering various types of questions about the modelled system
- flexibility in the modelling language provided by CLP as an extension of Prolog with a constraint solver
- ease of programming

These advantages come automatically from the CLP programming platform at no extra programming cost. For example, there is no need to explicitly formulate a decomposition structure of the system into sub-components.

In the rest of this paper we give a short introduction to CLP as an extension of Prolog, outline the principles of qualitative simulation in CLP(FD) (CLP over finite domains), experimentally compare ASIM and QSIM, and analyse the results.

Constraint Logic Programming and Qualitative Simulation

Constraint Logic Programming over Finite Domains, CLP(FD)

CLP (Constraint Logic Programming) is an extension of Logic Programming with constraints. One usual way of implementing CLP is to add a constraint solver to the logic programming language Prolog. Such a constraint solver is specialized to solving types of constraints over particular domains, for example real numbers which is denoted by CLP(R). For the purpose of qualitative simulation, constraints over finite domains are of particular interest, and the corresponding programming paradigm is denoted by CLP(FD). Constraints can then be used as goals in Prolog programs that are processed by the corresponding built-in constraint solver.

Satisfiability problems in finite domains are typically NP-hard, so incremental constraint solving is required. Incremental constraint solver operates over a set of constraints, known as constraint store. The conjunction of constraints in the store determines values of set of variables. This is also known as projection of constraint store on set of variables.

When a new constraint is posted to the store, constraint propagation is started. Constraint propagation adds new information to the store by inferring new constraints from the constraints already in store. New constraints must be consistent with the information already in the store. Propagation stops once no such new constraints can be added. From the viewpoint of the variables in constraint store, constraint propagation incrementally narrows the domains of the variables.

As an example, consider the finite domain of non-negative integers, and the constraint store which contains only constraint $X > 0$. If we post a new constraint $X < Y$, constraint propagation is able to infer and add new constraint $Y > 1$. If we then post $Y < 3$, constraint propagation is able to infer $Y = 2$, $X = 1$. The conjunction of constraints in the store now fully determines values of X and Y, that is X and Y are *instantiated*.

Constraint propagation is not sufficient to fully solve an arbitrary constraint satisfaction problem, since, in general, it cannot prove global consistency of constraint store or generate all consistent values of variables. A simple example of the first case is the conjunction of constraints $X \neq Y$, $X \neq Z$, $Y \neq Z$, where the domain of all X, Y and Z is [0,1]. Constraints in store are not consistent, but the CLP solver cannot see this by constraint propagation. However, once we assign a value for one of X, Y or Z, all the variables become instantiated and the inconsistency of constraint store is detected. Assigning concrete values to variables is called *labelling*.

Constraint propagation is therefore combined with labelling, that is enumeration of values of variables. Labelling enumerates values of yet uninstantiated variables from possible remaining values in their domains. If constraint store becomes inconsistent after an instantiation of a variable, backtracking to alternative instantiation occurs. The combination of constraint propagation and labelling yields a complete solution method for constraint satisfaction problems. As opposed to constraint propagation, labelling has to be explicitly requested in a CLP(FD) program.

Labelling is much more computationally expensive than propagation. Time complexity of labelling grows exponentially with the number of variables and the sizes of their domains. In contrast, constraint propagation can be performed in polynomial or even linear time. In general, labelling on a set of variables is required to determine *all* solutions to this set of variables. However, if we only want to know if the value of a given variable is consistent with all the constraints, labelling on all problem variables is not necessary, as we only need to find a single solution. In this case, major part of determining global consistency of the constraint store is achieved by constraint propagation. Labelling is then only performed on uninstantiated variables. The number of such variables is typically small. By reducing labelling, we can significantly reduce the computational complexity, and often achieve exponential improvement. No labelling is necessary when constraint propagation alone can detect inconsistency of constraint store. In such cases, it is sufficient to request labelling only on the variables we are interested in.

Application of this property of CLP(FD) to qualitative reasoning follows from the common scenario of observing a complex system with many variables, where most of the variables are only indirectly relevant to the part of the system or behavior we are interested in. For instance, some variables might be "auxiliary" or "internal" to the system. We say a variable is auxiliary if it appears in model definition, but it does not appear in a solution. For example, consider the Prolog clause $p(X, Y) \vdash q(X, Z), r(Z, Y)$. Here variable Z is auxiliary. Concrete values of auxiliary variables in a solution are of no interest. Labelling over such auxiliary variables may not be necessary.

In ASIM, however, we do not rely on constraint propagation alone ensuring the existence of solution. Rather we perform limited labelling on auxiliary variables to find a first solution only. Thus the global consistency of solutions for selected variables is guaranteed. We call this technique *partial labelling*.

As shown in experiments with ASIM, savings in computational complexity due to careful labelling are considerable. The reason for performance gain is that it is sufficient to find a *single* solution of a CSP for all the auxiliary variables to prove the consistency of labelled variables with the constraints.

Implementing Qualitative Simulation in CLP(FD)

To describe qualitative simulation as constraint satisfaction problem in CLP(FD) scheme, we represent qualitative variables with FD variables and qualitative constraints with FD constraints. In our implementation in ASIM, a finite domain is an ordered set of nonnegative integers. A qualitative variable V is translated to a pair of FD variables, one for its qualitative magnitude $M = qmag(V)$ and the other for its direction of change $D = qdir(V)$. The range of M is also known as quantity space (Kuipers 1986) and is a finite, totally ordered set of landmarks. Value of M can be a landmark or an interval between two adjacent landmarks.

We map a quantity space to a finite domain of even integers $\{0, 2, 4, ..., 2n\}$, so that 0 represents minus infinity ($minf$), 2 represents the first landmark after $minf$, and $2n$ the last landmark. If value of M is a landmark with zero-based index i in quantity space, then M is represented with even integer $2i$. If value of M is an interval between two adjacent landmarks with indices i and $i + 1$ in quantity space, then M is represented by odd integer $2i + 1$. The range of D is ordered set of qualitative directions $\{dec, std, inc\}$, which are mapped to integers $\{0,1,2\}$. A similar representation of qualitative variables is used in CQ (Dvorak 1992), a highly optimized implementation of the QSIM algorithm in the C language.

It can be seen that CS problems in QDE simulation typically have small domains of variables, due to small number of landmarks and directions of change. Labelling is therefore efficient, but it is still much more costly than constraint propagation.

State-based constraints (d/dt, M^+, add, ...) involve reasoning over relative order of qualitative magnitudes and directions of change. Relative order can be defined with the $qsign$ function, which maps two qualitative magnitudes or directions to their relative sign. Finite domain of signs is $\{0,1,2\}$, where 0 stands for minus, 1 for zero and 2 for plus.

The $qsign$ function in CLP is then implemented as the predicate $qsign(Mag_1, Mag_2, Sign)$. A formal definition of $qsign$ can be found in (Kuipers 1994).

More complex constraints like M^+, add or $mult$ require the determination of relative signs of magnitudes and corresponding values, where tuples of signs are further constrained according to relation table. Obviously, $qsign$ is a heavily used operation and efficient implementation is highly desirable. An advantage of our representation in FD is that $qsign$ requires only comparison of two integers, not list traversal as in the Lisp implementation of QSIM (Kuipers 1986).

Implementation of qualitative constraints in Prolog is quite straightforward, for example:

$deriv(\_:\_/S1, D2:M2/\_) :-$
$zero\_fd(D2, Zero)$, % Zero element of domain D2
$qsign(M2, Zero, S)$, % Relative sign of M2 w.r.t. zero
$S1$ #= S. % Signs S1 and S equal

The Prolog structure $Domain : Magnitude/Direction$ denotes a qualitative state of a qualitative variable whose domain is $Domain$. The code above posts constraints that define the qualitative derivation relation between two qualitative variables. First, we determine FD value $Zero$ of landmark zero inside the qualitative space $D2$. We constrain S to be the sign of qualitative magnitude $M2$ of the second qualitative variable. Finally, we constrain S to be equal to the direction of change of the first qualitative variable.

Note that we do not have to implement constraint propagation (as in QSIM), since this is already built-into the underlying CLP(FD) system. From this point of view, experimenting with new qualitative constraints in CLP is easier. But it must also be noted that debugging of constraints in a CLP system might be significantly harder.

Temporal constraints constrain a sequence of states according to state transition table (Kuipers 1986), which specifies valid value transitions of qualitative variables over time. Our representation of qualitative variable in CLP(FD) also enables efficient implementation of state transitions. For instance, qualitative magnitude transition from landmark X to interval Y and from interval X to landmark Y is modelled by simple FD constraint Y #= $X + 1$. Determination whether qualitative magnitude is a landmark or an interval also consists of simple FD constraint. State transition table is then modelled with reified constraints over these simple constraints, so that each possible state transition is reified into its own FD variable. A reified constraint projects the consistency of a constraint C into a FD variable V, so that V is bound to 1 if C is found to be consistent and 0 if not.

A QDE model is represented in Prolog in a manner similar to that in (Bratko 2001). Prolog is used for specification, setting up constraint network and gathering and interpreting results. The algorithm of qualitative simulation in ASIM is actually a constraint satisfaction algorithm exploiting the underlying CLP(FD) solver.

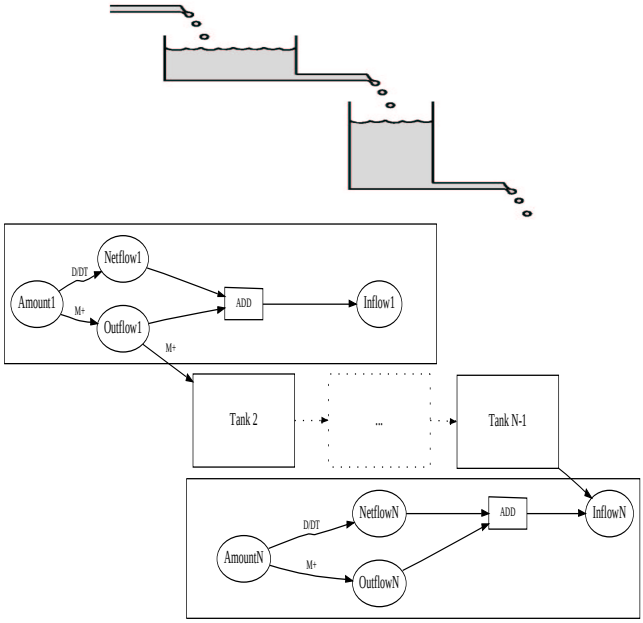

Figure 1: QDE model of *Cascade* model with *N* connected tanks.

Experiments

Here we empirically compare the performance of ASIM and traditional QSIM simulation. ASIM is implemented in Prolog with CLP(FD), specifically GNU Prolog (Diaz & Codognet 2000) and SICStus Prolog environments. Tests were run on GNU Prolog version 1.2.3, which is known as one of the fastest CLP(FD) solvers. The version of QSIM used for comparison was the latest published version 4.0 alpha 4 (Farquhar *et al.* 1994). It was running on CMUCL Lisp compiler version 18c (MacLachlan 1992), which is known as one of the fastest Lisp implementations. The PC used to run the experiments had 333 Mhz Celeron processor.

Experimental evaluation on a set of qualitative models in (Platzner 1996) shows that checking whether a qualitative state satisfies state-based constraints consumes majority of time during simulation. This bottleneck of qualitative simulation with QSIM is most clearly pronounced in generation of all consistent states from partially specified state, so we have chosen this stage of QSIM for performance comparison. We call this stage generation of all *QDE-consistent completions* of a qualitative state. This stage first generates all possible completions of a qualitative state and then eliminates the qualitative states that do not respect the QDE constraints.

There is another reason for comparing the efficiency in generation of QDE-consistent *completions* of a qualitative state. In complex systems often only a part of a qualitative state is specified. Moreover, we are often interested in the qualitative behavior of some state variables only.

ASIM and QSIM were tested on a set of "extendible" models similar to those in (Clancy & Kuipers 1998). An "extendible" model is a model composed of a sequence of identical components thus enabling the incremental exten-

Cascade N	QSIM time	ASIM time	no. of sol.	ASIM* time	no. of sol.
2	10	1	3	1	3
3	90	4	23	8	23
4	570	30	215	38	62
5	5020	254	2067	66	62
6	55030	2312	19919	97	62

Loop N	QSIM time	ASIM time	no. of sol.	ASIM* time	no. of sol.
2	1160	19	121	35	121
3	11370	110	933	134	363
4	108470	950	8257	348	562
5	nc	8910	77421	600	562
6	nc	86500	739417	854	562

Table 1: Execution times in milliseconds for N tanks in the *cascade* (upper table) and the *loop* model (lower table). The first column gives the number of tanks in the model. The second and the third column give execution times of both QSIM and ASIM in milliseconds. The fourth column gives number of solutions, i.e. different QDE-consistent completions of the initial qualitative state found by both algorithms. The fifth and the sixth column (ASIM*) give the time and the number of solutions found by ASIM when the labelling is limited. *nc* means that resource limitations prevented completion.

sion of the model to facilitate the complexity evaluation of the asymptotic behavior of the algorithm. We used two versions of the cascaded tanks model consisting of $N = 2, 3, 4,$ etc. tanks where outflow of each tank is connected with inflow of the next tank with the M^+ constraint. In the first version of the model, called *cascade*, the outflow of the last tank and the inflow of the first tank are not connected. Figure 1 shows a QDE model of N tanks connected in the *cascade*. In the second version, called *loop* the outflow of the last tank is connected with inflow of the first tank with the M^+ constraint.

We measured the times of both algorithms required to generate all QDE-consistent completions of the initial qualitative state. With the *cascade* model the initial qualitative state has constant inflow in the first tank and constant zero outflow of the first tank, no other qualitative values are specified. In the *loop* model,there were no constraints on the initial qualitative state. Accordingly, the *loop* model has more solutions. Table 1 gives the results for $N = 2, 3, 4, 5, 6$ tanks in the *cascade* (upper table) and in the *loop* model (lower table).

In Figures 2 and 3 and in Table 1, ASIM* denotes the simulation scenario when we are interested only in small part of the system, in this case only the first and last tank. The variables belonging to the intermediate tanks are now auxiliary variables, not included in a solution. The number of possible qualitative variables, states and behaviors that interest us is now much smaller than the number of states and behaviors of the whole system. With ASIM, this leads to reduced computation. ASIM only performs partial labelling (complete labelling only on the variables of interest). QSIM, on

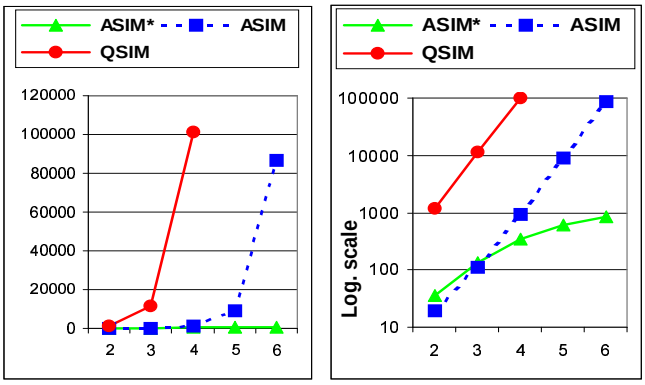

Figure 2: Execution times for N tanks (on x-axis) in the *loop* model. Both graphs give the results from Table 1, but the right graph uses logarithmic scale.

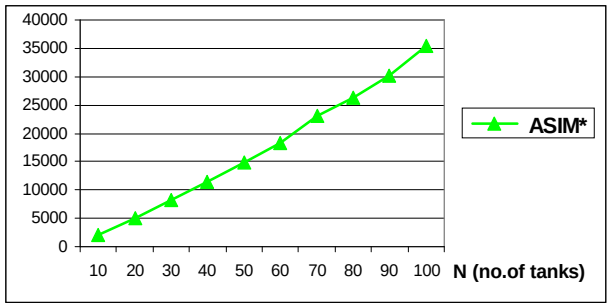

Figure 3: Execution times for N=10,20,..,100 tanks (on x-axis) in the *loop* model with ASIM*.

the other hand, searches the space of possible values of all variables in the system which would correspond to complete labelling of the variables.

Execution times for ASIM* show linear growth with increasing number of tanks once the number of possible solutions for first and last tank reaches a fixed point. This shows that global consistency can be determined in linear time by finding a single consistent solution of remaining tanks by partial labelling.

The results of ASIM indicate the advantage of using the CLP technology in solving CS problems of qualitative simulation. ASIM's results when observing only variables involved in the first and last tank are particulary interesting, as they show the ability of CLP to scale to models with large numbers of variables and constraints. CLP's constraint solver was in this case automatically able to exploit the fact that the modelled system's structure contains loosely connected sub-components.

Discussion and Related Work

Let us summarize the main results of this paper and the results of the experiments:

- CLP technology is appropriate for implementation of qualitative reasoning and can compete very well with state-of-the art implementations in terms of efficiency.

- In particular, QDE simulation in CLP is compared with the one in QSIM. CLP allows for more flexible labelling strategies that may lead to exponential improvements.
- The idea of complete labelling on a subset of qualitative variables only is somewhat related to the idea of system decomposition in DecSIM (Clancy & Kuipers 1998). Efficiency improvements in DecSIM compared to QSIM are comparable to these due to partial labelling. Partial labelling in ASIM guarantees that ASIM's results for selected variables are globally consistent and thus equivalent to QSIM's whereas DecSIM implementation reported in (Clancy & Kuipers 1998) does not offer this guarantee.

It should be noted that (Teleki 1996) also implemented QSIM's constraint filter in CLP(FD) solver ECLiPSe. He compared his implementation with the Lisp implementation of QSIM, but did not report any improvement in efficiency. This difference in his findings compared to ours is possibly due to different implementation of qualitative states and constraints, and different labelling strategy.

Acknowledgements

The work reported in this paper was partially supported by the European Fifth Framework project Clockwork and the Slovenian Ministry of Education, Science and Sport.

References

Bratko, I. 2001. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 3rd edition. Chapter 20: Qualitative reasoning.

Clancy, D., and Kuipers, B. 1998. Qualitative simulation as a temporally-extended constraint satisfaction problem. In *Proceedings of the 15th National Conference on Artificial Intelligence*. AAAI/MIT Press.

Diaz, D., and Codognet, P. 2000. The GNU prolog system and its implementation. In *SAC (2)*, 728–732.

Dvorak, D. 1992. A c-language implementation of qsim. Technical report, Department of Computer Science, University of Texas at Austin.

Farquhar, A.; Kuipers, B.; Rickel, J.; and Throop, D. 1994. Qsim, the program and its use. Part of qsim distribution.

Forbus, K. 1984. Qualitative process theory. *Artificial Intelligence* 24:85–168.

Jaffar, J., and Maher, M. 1994. Constraint logic programming: A survey. *Journal of Logic Programming* 20:503–581.

Kuipers, B. 1986. Qualitative simulation. *Artificial Intelligence* 29:289–338.

Kuipers, B. 1994. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Cambridge, Massachusetts: MIT Press.

MacLachlan, R. 1992. CMU Common Lisp user's manual. Research paper CMU-CS-92-161, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

Platzner, M. 1996. *Design, Implementation, and Experimental Evaluation of Coprocessor Architectures for Fast Qualitative Simulation*. Ph.D. Dissertation, Graz University of Technology.

Teleki, L. 1996. Embedding qualitative reasoning into constraint logic programming. In *Proceedings of CP'96 Workshop on Constraint Programming Applications*. Cambridge, MA.