Embedding Qualitative Reasoning into Constraint Logic Programming

László Teleki*
Institut für Photogrammetrie
Universität Bonn
Nussallee 15, 53115 Bonn
laszlo@ipb.uni-bonn.de

Abstract

We propose to use *Constraint Logic Programming* (CLP) for the specification and implementation of *Qualitative Reasoning* (QR) problems that are specialized *Constraint Satisfaction Problems*.

Although it has long been recognized that many frameworks, like the QSIM algorithm (Kuipers 1994) can be viewed as a set of constraint satisfaction problems (CSP), the area of QR is relatively unknown in the CLP literature. In this paper we would like to present, through the language of QSIM, one set of questions and problems analyzed in the area of QR and to present the CLP specification of the key algorithm of QSIM, the c-filter. We show how the basic constraints of QSIM are specified and describe the technical aspects of our implementation.

1 Motivation

Qualitative Reasoning (QR), a branch of Artificial Intelligence, works on the frontiers between common sense reasoning, mathematical foundations and task-level reasoning. QR tries to predict the behavior of physical systems and processes in the context of incomplete knowledge.

The incompleteness of the knowledge is given on one hand by the absence of quantitative data. An example of this is the *Starling*-equilibrium, a problem involving a slightly atypical case of a kidney disorder presented in (Kuipers 1994) on page 136. On the other hand to model large problems, like the flight of a space-shuttle, into the last detail is also an impossible task.

In such cases it is more helpful to build *qualitative models*, that capture essential aspects that make an important qualitative difference and ignore others. (A good survey about QR is given in (MQ&D 1995)).

A very important contribution to this field is the QSIM algorithm by Kuipers (1994). Although it has long been recognized that QSIM can be viewed as a set of constraint satisfaction problems (CSP), the area of QR is relatively unknown in the CLP literature. In this paper we would like to present, through the language of QSIM, a set of questions and problems analyzed in the area of QR and to present the CLP specification of the key algorithm of QSIM, the c-filter.

We claim that there are two advantages to using CLP:

- CLP gives a well-defined and well-understood logical framework for the problem specification;
- CLP is not only a logical framework, it is also a family of languages specifically developed for solving classes of CSP problems. Thus we obtain a class of powerful implementation languages for rapid prototyping.

This example was chosen precisely because the algorithm is widely used in the the area of QR, and it is a non-trivial problem that took many years to implement.

All the ideas presented in this paper were implemented in ECL^{i}PS^e (Aggoun et al. 1995; Brisset et al. 1995), a CLP platform developed at the European Computer-Industry Research Center (ECRC). Certainly, in almost every case, a specialized algorithm will give a better performance. But ECL^{i}PS^e also gives very good results in a fraction of the developement time

*This work is funded by the *Deutsche Forschungsgemeinschaft* through the *Sonderforschungsbreich 350* project.

of the specialized algorithm. This gain in implementation time can be used to determine at an early stage conceptual problems or limits of the specification.

This paper is structured as follows: in Section 2, we present the core of the QSIM algorithm and the CLP scheme. In Section 3 we specify in every detail the constraints of QSIM in our logical framework. In Section 4, the implementation steps and different methods to improve the performance are described. We close with conclusions and further works in Section 5.

2 Introduction

In the next two subsections we present the two areas that we want to connect: the QSIM algorithm and the CLP framework. We concentrate only on the relevant aspects necessary to understand the functioning of QSIM and CLP.

2.1 QSIM

The QSIM algorithm developed by Benjamin Kuipers, detailed in (Kuipers 1994), is a major tool used by many researchers for describing physical models in QR. The model specification is done by *Qualitative Differential Equations* (QDEs), a symbolic correspondent to ordinary differential equations. QDEs describe the development of processes over time in the context of incomplete knowledge about the process itself, the boundary and initial value problems. The whole specification of the model is done on a symbolic level.

To give an idea about the qualitative modeling we give a small example. We analyze the simple physical system of a ball that is thrown upward in a gravitational field. If we ignore quantitative values we have the following relations: (1) The ball has an initial velocity v^* that is positive and lies in the interval $(0, \infty)$. The acceleration is constant and negative. Some basic physical rules are also known, like $v = dy/dt$ and $a = dv/dt$ with a the acceleration and y the trajectory of the ball. With these general relations QSIM is capable of determining that the ball will fly to a not-exactly-specified hight $y^* \in (0, \infty)$ and then fall back to the ground.

Within a qualitative simulation, a physical process is represented by a succession of states starting from an initial state. The states alternate between *time-points* and *time-intervals*. A state is completely characterized by its variable description, the parameters of the mechanism. In each state, the variables are assigned *qualitative values* (`qval`). A `qval` is a pair $\texttt{qval} = \langle \texttt{qmag}, \texttt{qdir} \rangle$ of *qualitative magnitude* (`qmag`) and *qualitative direction* (`qdir`). A `qmag` is either a point value, called a *landmark*, or an open interval between two landmarks. The `qval`s must be consistent with the *corresponding values* of the state's constraints. The `qdir`s may be increasing (*`inc`*), decreasing (*`dec`*), steady (*`std`*) or unknown (*`unknown`*).

Variables represent on a symbolic level time-dependent functions of the physical process. Each variable has a *landmark list*, corresponding to the domain of the function. Important elements in the landmark list are `minf`, `zero`, `inf`, the correspondents to $-\infty$, 0 and ∞.

For our example the `qval` for the velocity v at t_0 is given by the `qval` v^* and the `qdir` is *`dec`*, as the function will decrease from the very beginning of the flight. The hight y is described by the `qval zero` and the `qdir` *`inc`*.

State transitions are described by changes in the values of the variables; the number of possible valid transitions is always restricted due to the mathematical background, e.g. continuity, mean value theorem etc. For example, a function cannot change from *`inc`* to *`dec`* without passing through a point where the derivative is `zero`. The possible set of values for a variable is always finite.

The constraints are symbolic equivalents to the simple mathematical notions of addition, multiplication, derivative, minus, constant functions, monotonic increasing and monotonic decreasing functions. In QSIM the notations are `add`, `mult`, `d/dt`, `MINUS`, `constant`, `M+` and `M-`. The constraints connect from one to three variables. For a detailed presentation see Section *QSIM in CLP*. As the values of the variables are symbolic, we need so-called *corresponding values lists*; these lists interconnect the symbolic definition domains of the variables.

In every time step, the algorithm will generate the possible successor values for all variables. A crucial task of QSIM is to filter from these possible values, those value combinations that fulfill the constraints defined in the QDE. In QSIM a specialized constraint satisfaction algorithm called *c-filter* is responsible for this part. For this special task we propose the CLP framework. Kuipers (1994) developed an optimized algorithm that fits only for the structure of the qualitative value `qval`. It is a combination of tuple-filtering, Waltz-algorithm and backtrack-search to determine all consistent states. This algorithm is certainly faster than the version specified in CLP. On the other hand it took a long time to develop and implement it and the difference between the runtimes is not essential in large problems.

In the case of our small example the steps in the simulation are described by the following states:

	t_0	(t_0, t_1)	t_1
v	$\langle v^*, \mathtt{dec} \rangle$	$\langle \mathtt{zero}, \mathtt{dec} \rangle$	$\langle v_1, \mathtt{dec} \rangle$
a	$\langle g, \mathtt{std} \rangle$	$\langle g, \mathtt{std} \rangle$	$\langle g, \mathtt{std} \rangle$
y	$\langle \mathtt{zero}, \mathtt{inc} \rangle$	$\langle y_1, \mathtt{std} \rangle$	$\langle \mathtt{zero}, \mathtt{dec} \rangle$

c-filter is used in every time step to determine the consistent successor state: the step $t_0 \rightarrow (t_0, t_1)$ and $(t_0, t_1) \rightarrow t_1$. c-filter is also used to determine the initial state at t_0. In more complex examples the number of consistent successor states is in general > 1. This will lead to so-called behavior trees, where every path from the root to a leaf is a possible behavior of the process.

2.2 Constraint Logic Programming with Finite Domains

Constraint logic programming (CLP) is a generalization of logic programming (LP) where unification, the basic operation of LP languages, is replaced by constraint handling in a constraint system (van Hentenryck 1991). In practice, this means the enhancement of PROLOG-like languages with constraint solving mechanisms. PROLOG-like languages have performance problems in solving Constraint Satisfaction Problems due to their simple computational rule, the depth-first search procedure, resulting in a *generate and test* procedure. The new paradigm allows a new computational rule that can be characterized as *constrain and generate* (Frühwirth et al. 1992).

Three constraint systems are widely used and implemented: Boolean Algebra, Linear Rational Arithmetic and Finite Domains. We propose *Finite Domains* (FD) for problems in Qualitative Reasoning.

The FD consistency technique rules out many inconsistencies at a very early stage and thus, cuts short the search for consistent labeling. It works by *propagating* information about the variables via the mutual constraints with the goal of reducing the domains. Constraints that can not contribute to a given time but may contribute later to a domain reduction are delayed (or suspended) and kept in a *constraint store*. The *scheduler* will *wake up* those constraints from the constraint store that are affected from a domain reduction after the propagation. Propagation continues until no domain reductions can be extracted from the constraints. The FD solver implements the well-known *node* and *arc consistency* (Mackworth 1977) methods.

The FD system will rarely be used alone to solve a problem since, in general, there remain combinations of values in the resulting domains which are inconsistent. To find a solution to a problem, the system performs some search by labeling a variable with an element of its domain. This choice allows further propagation that will end in a set of solutions. This set of values can be empty if a choice is erroneous. The labeling can be done by a simple backtracking search, a computational rule already included in LP. We also describe some improvements in Section *Technical Aspects* to speed up this task.

The most general description of a finite domain problem is given by a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ with a finite domain D_{x_i} for each x_i and a finite set of constraints $C = \{c_1, c_2, \ldots, c_n\}$, where each c_j refers to some subset of the set of variables X. The goal is to find one (or all) of the solutions that satisfy the set of constraints C. Constraints are first order formulas. For a detailed presentation of the CLP paradigm consult (Jaffar and Maher 1994).

Notation We use the following notations: $\{x_1, x_2, \ldots\}$ denotes a set, $[x_1, x_2, \ldots]$ a list, (x_1, x_2) an interval, $\langle x_1, x_2 \rangle$ a tuple or a pair and $\langle x_1, x_2, x_3 \rangle$ a triple.

If we look at a list L as a domain of a function we can generate the set of all intervals of this domain $I(L)$. For example, if $L = [a, b, c]$, $I(L) = \{(a, b), (b, c), (a, c)\}$. Further on, we define the set of all possible values $V(L)$ generated from a list L by adding to $I(L)$ all the elements of the list. In our example $V(L) = \{(a, b), (b, c), (a, c), a, b, c\}$.

3 QSIM in CLP

We present the formal description of the filtering of the state transitions of the QSIM algorithm in the CLP framework. This filtering algorithm is used after every state transition of a simulation. We do not argue *why* the constraints have the presented forms; the proofs are given in (Kuipers 1994). Some of the constraints are not exactly defined as in (Kuipers 1994). We ignore some details to concentrate on the essential aspects.

The notation and specification is taken from (Teleki 1996).

3.1 Domains

A QDE (Qualitative Differential Equation) is defined as a finite set $X_{QDE} = \{\ldots, (x_i, L_{x_i}), \ldots\}$ of variables (x_i, L_{x_i}) with their landmark list and a set of constraints $C_{QDE}\{c_j\}$. A landmark list L_{x_i} is a list where the succession of the elements will determine an order over the domain of the variable x_i. In a process specification the landmark list contains at least two elements, the `zero` element and either the `minf` or `inf`. An initial value problem is described by a set of variables x_k ($x_k \in X_{QDE}$) with initial qualitative values

$\texttt{qval} = \langle \texttt{qmag}, \texttt{qdir} \rangle$. The requirement for the qval's is that either the qmag or the qdir is defined. The domain of the variable x_i with the full set of possible values is given by the following set for each x_i:

$$\begin{aligned}\mathcal{D}_{x_i} = &\{\langle v, dir\rangle | \\ &v \in V(L_{x_i}),\ dir \in \{\textit{std}, \textit{inc}, \textit{dec}, \textit{unknown}\}\}\end{aligned}$$

This means that we include in the domain $\mathcal{D}_{x_i}$ of a variable x_i every element of the landmark list L_{x_i} and every possible interval derived from the landmark list in the combination with the four possible directions of change. So, for example, if the variable x has the landmark list [zero, inf], the complete domain of the variable is:

$$\begin{aligned}\mathcal{D}_x = \{&\langle \texttt{zero}, \textit{dec}\rangle, \langle \texttt{inf}, \textit{dec}\rangle, \langle (\texttt{zero}, \texttt{inf}), \textit{dec}\rangle, \\ &\langle \texttt{zero}, \textit{inc}\rangle, \langle \texttt{inf}, \textit{inc}\rangle, \langle (\texttt{zero}, \texttt{inf}), \textit{inc}\rangle, \\ &\langle \texttt{zero}, \textit{std}\rangle, \langle \texttt{inf}, \textit{std}\rangle, \langle (\texttt{zero}, \texttt{inf}), \textit{std}\rangle, \\ &\langle \texttt{zero}, \textit{unknown}\rangle, \langle \texttt{inf}, \textit{unknown}\rangle, \\ &\langle (\texttt{zero}, \texttt{inf}), \textit{unknown}\rangle\}\end{aligned}$$

3.2 Signs

We will need two functions to reason over the order of the landmarks:

$$\begin{aligned}&\textit{befor}(x, l_g, l_i, L_x) = \textit{true} \text{ iff } L_x = [\ldots l_g, \ldots, l_i, \ldots] \\ &\textit{after}(x, l_g, l_i, L_x) = \textit{true} \text{ iff } L_x = [\ldots l_i, \ldots, l_g, \ldots]\end{aligned}$$

The two functions determine the position of the element l_g relative to the element l_i in the landmark list L_x of the variable x. $\textit{befor}(x, l_g, l_i, L_x)$ is *true* if l_g is before the element l_i in the list L; $\textit{after}(x, l_g, l_i, L_x)$ is the opposite of $\textit{befor}$.

To reason with the constraints and the values we need the definition of signs. In mathematics the *sign* function relative to 0 is defined as $\textit{sign}(x) : \mathbb{R} \to S'$ with $S' = \{+1, -1, 0, ?\}$ the set of extended signs. The three first elements of S' divide $\mathbb{R}$ into three intervals $(0, \infty)$, $(-\infty, 0)$ and $(0, 0)$. The sign ? is used as the ambiguous sign and denotes the interval $(-\infty, \infty)$. The general form of the sign function is $\textit{sign}(x)_a = \textit{sign}(x - a)$. If $a = 0$ we have the definition presented previously. Is the reference ∞, we define a new sign function:

$$\textit{sign}(x)_\infty = \begin{cases} +1 & \text{if} \quad x = \infty \\ 0 & \text{if} \quad x \text{ finite} \\ -1 & \text{if} \quad x = -\infty \end{cases}$$

We have to introduce a new sign function for the reference value ∞ as we want to operate with the signs in the same way as with reals. If we only use the classical sign definition we would have the following inconsistency: $\infty + 1 = \infty$ but $\textit{sign}(\infty)_\infty + \textit{sign}(1)_\infty = 0 + -1 = -1 \neq \textit{sign}(\infty)_\infty = 0$. The new sign definition will behave correctly: $\textit{sign}(\infty)_\infty + \textit{sign}(1)_\infty = +1 + 0 = +1 = \textit{sign}(\infty)_\infty$.

Now we have to determine the *sign* function in the context of symbolic values. Therefore we define the $\textit{sign}(x, l_g, L_x)_{l_i}$ over the domain $S' = \{\textit{pos}, \textit{neg}, \textit{zero}, \textit{unknown}\}$ as follows:

$$\textit{sign}(x, l_g, L_x)_{l_i} = \begin{cases} \textit{pos} & \text{if} \quad \textit{after}(x, l_g, l_i, L_x) = \textit{true} \\ \textit{zero} & \text{if} \quad l_g = l_i \\ \textit{neg} & \text{if} \quad \textit{befor}(x, l_g, l_i, L_x) = \textit{true} \end{cases}$$

and

$$\textit{sign}(x, l_g, L_x)_{\texttt{inf}} = \begin{cases} \textit{pos} & \text{if} \quad l_g = \texttt{inf} \\ \textit{zero} & \text{if} \quad \textit{befor}(x, l_g, \texttt{inf}, L_x) = \textit{true} \\ \textit{neg} & \text{if} \quad l_g = \texttt{minf} \end{cases}$$

with L_x the landmark list of the variable x and with the assumption that $l_g, l_i \in L_x$. As we see, we need in the function the landmark list as an argument, as the order is given by the succession of the elements of L_x.

The sign function can be extended in a straightforward way to intervals. The symbol *unknown* will be used as the ambiguous sign, e.g. $\textit{sign}(x, (a, c), [a, b, c, d])_{(b,d)} = \textit{unknown}$ for the variable x with the landmark list $[a, b, c, d]$.

In the following we use $\textit{sign}(x, l, L_x)$ for $\textit{sign}(x, l, L_x)_{\textit{zero}}$.

We also define the *sign* function for the qualitative directions: $\textit{sign}(\textit{inc}) = \textit{pos}$, $\textit{sign}(\textit{std}) = \textit{zero}$, $\texttt{sign}(\textit{dec}) = \textit{neg}$ and $\textit{sign}(\textit{unknown}) = \textit{unknown}$. This definitions follow directly from the definition of the derivative.

3.3 The basic constraints of QSIM

We define the relations $=_+$ and $=_-$:

$$\begin{aligned}&x =_+ y \text{ iff} \\ &\quad \langle x, y\rangle \in \\ &\quad\quad \{\langle \textit{pos}, \textit{pos}\rangle, \langle \textit{neg}, \textit{neg}\rangle, \langle \textit{zero}, \textit{zero}\rangle, \\ &\quad\quad \langle \textit{unknown}, \textit{pos}\rangle, \langle \textit{unknown}, \textit{neg}\rangle, \langle \textit{unknown}, \textit{zero}\rangle\} \\ &x =_- y \text{ iff} \\ &\quad \langle x, y\rangle \in \\ &\quad\quad \{\langle \textit{pos}, \textit{neg}\rangle, \langle \textit{neg}, \textit{pos}\rangle, \langle \textit{zero}, \textit{zero}\rangle, \\ &\quad\quad \langle \textit{unknown}, \textit{pos}\rangle, \langle \textit{unknown}, \textit{neg}\rangle, \langle \textit{unknown}, \textit{zero}\rangle\}\end{aligned}$$

In the following x, y, z will denote the variables from the QDE.

We already mentioned that in QSIM there is a limited set of possibilities for the variables' value transitions in a state transition. This means that the domain of a variable x will in general, after the state transition, have a subset D_x of the full possible value set $\mathcal{D}_x$. So the variables x, y, z will have in general the domains $D_x \subseteq \mathcal{D}_x$, $D_y \subseteq \mathcal{D}_y$, $D_z \subseteq \mathcal{D}_z$ of values. From the CLP point of view there is no difference if we use D_x or $\mathcal{D}_x$ as the domain of the variable x; it is the semantic of the QSIM algorithm that defines these restricted domains D_x.

A qualitative value of a variable `qval` is always a tuple of the form $\texttt{qval} = \langle \texttt{qmag}, \texttt{qdir} \rangle$. The following two functions make the projections onto the two members of the tuple:

$$qdir(\langle \texttt{qmag}, \texttt{qdir} \rangle) = \texttt{qdir}$$
$$qmag(\langle \texttt{qmag}, \texttt{qdir} \rangle) = \texttt{qmag}$$

We now focus on the exact definition of the constraints:

M+: $(\texttt{M+}(x, y), CV)$. CV, the set of *corresponding values* is a set $CV = \{\ldots \langle l_{x_i}, l_{y_i} \rangle \ldots\}$ of pairs $\langle l_{x_i}, l_{y_i} \rangle$ with $l_{x_i} \in L_x$ and $l_{y_i} \in L_y$. The constraint represents the assertion of a monotonic increasing function. The constraint is satisfied for a given pair $\langle x_g, y_g \rangle$[1] if the conjunction of the following constraints is satisfied.

1. $sign(qdir(x_g)) =_+ sign(qdir(y_g))$
2. $\forall \langle l_{x_i}, l_{y_i} \rangle \in CV, \quad sign(x, qmag(x_g), L_x)_{l_{x_i}} =_+ sign(y, qmag(y_g), L_y)_{l_{y_i}}$

M-: $(\texttt{M-}(x, y), CV)$. The set of corresponding values CV is again a set of pairs $\langle l_{x_i}, l_{y_i} \rangle$. The constraint represents the assertion of a monotonic decreasing function. The constraint is satisfied for a pair $\langle x_g, y_g \rangle$ if the conjunction of the following constraints is satisfied.

1. $sign(qdir(x_g)) =_- sign(qdir(y_g))$
2. $\forall \langle l_{x_i}, l_{y_i} \rangle \in CV, \quad sign(x, qmag(x_g), L_x)_{l_{x_i}} =_- sign(y, qmag(y_g), L_y)_{l_{y_i}}$

MINUS: $(\texttt{MINUS}(x, y), CV)$. The constraint is satisfied similarly to `M-` with the addition that the constraint has to be satisfied with the CV augmented with the set: $\{\langle \texttt{zero}, \texttt{zero} \rangle, \langle \texttt{inf}, \texttt{minf} \rangle, \langle \texttt{minf}, \texttt{inf} \rangle\}$. The constraints represent the relation $y(t) = -x(t)$.

add: $(\texttt{add}(x, y, z), CV)$. In this constraint $CV = \{\ldots, \langle l_{x_i}, l_{y_i}, l_{z_i} \rangle, \ldots\}$ is a set of triples $\langle l_{x_i}, l_{y_i}, l_{z_i} \rangle$ with $l_{x_i} \in L_x$, $l_{y_i} \in L_y$ and $l_{z_i} \in L_z$. The triple $\langle \texttt{zero}, \texttt{zero}, \texttt{zero} \rangle$ is always an element of the corresponding values set of the constraint. The constraint represents the relation $x(t) + y(t) = z(t)$. The constraint is satisfied for a given triple $\langle x_g, y_g, z_g \rangle$ if the conjunction of the following constraints is satisfied.

1. $(sign(qdir(x_g)), sign(qdir(y_g)), sign(qdir(z_g))) \in R_{\texttt{add}}$
2. $\forall \langle l_{x_i}, l_{y_i}, l_{z_i} \rangle \in CV$
   $(sign(x, qmag(x_g), L_x)_{l_{x_i}},$
   $sign(y, qmag(y_g), L_y)_{l_{y_i}},$
   $sign(z, qmag(z_g), L_z)_{l_{z_i}}) \in R_{\texttt{add}}$

The addition table $R_{\texttt{add}}$ is given by:

$R_{\texttt{add}}$	*pos*	*zero*	*neg*
pos	*pos*	*pos*	*neg/zero/pos*
zero	*pos*	*zero*	*neg*
neg	*neg/zero/pos*	*neg*	*neg*

The addition is a relation and not a function to avoid the propagation of the ambiguous sign *unknown* (see the details in (Kuipers 1994) page 48-49).

mult: $(\texttt{mult}(x, y, z), CV)$. CV is again a list of triples $\langle l_{x_i}, l_{y_i}, l_{z_i} \rangle$. The constraint represents the relation $x(t)y(t) = z(t)$. The constraint is satisfied for a given triple $\langle x_g, y_g, z_g \rangle$ if the conjunction of the following constraints is satisfied.

1. $sign(x, qmag(x_g), L_x) sign(y, qmag(y_g), L_y) = sign(z, z_g, L_z)$ with the exceptions:
   $sign(x, \texttt{zero}, L_x) sign(y, \texttt{inf}, L_y) = \textit{unknown}$,
   $sign(x, \texttt{zero}, L_x) sign(y, \texttt{minf}, L_y) = \textit{unknown}$,
   $sign(x, \texttt{inf}, L_x) sign(y, \texttt{minf}, L_y) = \textit{unknown}$.
   The multiplication follows the rules given in $R_{\texttt{mult}}$.
2. $(sign(y, qmag(y_g), L_y) sign(qdir(x_g)),$
   $sign(x, qmag(x_g), L_x) sign(qdir(y_g)),$
   $sign(qdir(z_g))) \in R_{\texttt{add}}$.
   This constraint follows directly from $(x(t)y(t))' = x'(t)y(t) + x(t)y'(t)$. $x'(t)$ denotes dx/dt, the time derivative of $x(t)$.
   $sign(y, qmag(y_g), L_y) sign(qdir(x_g))$ and $sign(x, qmag(x_g), L_x) sign(qdir(y_g))$ can be determined directly from the $R_{\texttt{mult}}$ table.
3. The `mult` constraint has some other constraints where the corresponding values are used. Due to lack of space we do not present them here (see (Kuipers 1994) page 56).

[1] By a given pair $\langle x_g, y_g \rangle$, we mean a given pair of values where $x_g \in D_x$ and $y_g \in D_y$.

The multiplication table $R_{\texttt{mult}}$ is given by:

$R_{\texttt{mult}}$	*pos*	*zero*	*neg*
pos	*pos*	*zero*	*neg*
zero	*zero*	*zero*	*zero*
neg	*neg*	*zero*	*pos*

`d/dt`: `d/dt`(x, y). The constraint has no corresponding values. The constraint corresponds to $y(t) = dx(t)/dt$. `d/dt` is satisfied for a pair (x_g, y_g) if:

1. $sign(qdir(x_g)) =_+ sign(y, qmag(y_g), L_y)$

`constant`: The constraint has the form `constant`(x) or `constant`(x, a). `constant` has no corresponding values. The constraint represents the assertion that the variable x is constant. The constraint is satisfied for a given x_g if the conjunction of the following two constraints is satisfied.

1. $(sign(qdir(x_g)) =_+ \mathit{zero})$
2. $(sign(x, qmag(x_g), L_x)_a =_+ \mathit{zero})$ in the case where `constant`(x, a) is given.

4 Technical Aspects

With the constraint specification in the FD scheme, the implementation is straightforward. The difficult work of the constraint solving mechanism, namely the propagation of the domain reductions is done by the system.

We present the technical aspects in two steps. The first is the presentation of the general idea of the solution of the constraint network in the FD system. In the second we sketch ideas to improve the performance.

4.1 Implementation of the constraints

The goal of the implementation of a constraint is to determine those elements of the variable domains that satisfy the constraints defined in the previous section. This verification will lead to a domain reduction propagated later on by the FD solver.

The general algorithm The algorithm for constraints is straightforward and well known from the CLP literature. We give as an example the code for `M+`:

```
1  proc mplus(X::D_x, Y::D_y)
2    begin
3      D_xtmp = {}; D_ytmp = {};
4      ∀x ∈ D_x, ∀y ∈ D_y
5      do if (Condition 1 ∧ Condition 2)
6           then
7                D_xtmp ← D_xtmp ∪ x;
8                D_ytmp ← D_ytmp ∪ y;
9      fi od
10     D_x ← D_xtmp;
11     D_y ← D_ytmp;
12   end.
```

With *Condition 1* and *Condition 2* we mean the conditions defined in Section 3.3 for this constraint. All two-valued constraints can be implemented in the same way by changing only the conditions in line 5 of the code. The three-valued constraints follow the same scheme with the difference that the verification also includes the third variable Z with its domain $\mathcal{D}_z$. The algorithm will work for the complete domain $\mathcal{D}_x$ of a variable x as well as for any reduced domain $D_x \subseteq \mathcal{D}_x$.

Labeling In the regular case of labeling, the domains will contain more than one value. In our problem the labeling procedure has to find all the consistent combinations of states; this is the requirement of the modeling procedure. In our implementation we are using the *first fail principle*. This means that it is more effective to use the variable with the smallest remaining domain for labeling: with fewer choices possible we will find out earlier if those were right or wrong.

4.2 Improving the performance

There are a few problem-specific aspects that can be used to improve the performance of the runtimes of the constraints.

Priorities It is not difficult to see that we have different classes of constraints w.r.t. the computational time. The most expensive are `add` and `mult`, then `M+`, `M-` and `MINUS`, followed by `d/dt` and finished with `constant`. `constant` has only one variable, so the domain reduction has to be done only once; there is no reason to wake it up again.

Due to these facts, we can give *priorities* to the different constraints. This means that the propagation should be done in different stages. The propagation should be kept as long in one class of constraints until no changes occur in the domains. It should then turn to the next lowest priority. If a domain reduction is realized by constraints of lower priority, the scheduler should, if possible, wake up again the constraints of higher priority. Through this strategy we achieve that the computationally expensive constraints are evaluated only when computationally cheaper constraints are not capable of reducing a domain.

For our constraints we determined the following priorities:

Priority	constraints
1	`constant`
2	`d/dt`
3	`M+`, `M-`, `MINUS`
4	`add`, `mult`

Delaying the computation In QSIM the generation of the initial state has a special characteristic. The problem is that the verification of the constraints with complete domains (and only in this case!) will leave the domains in the majority of cases unchanged; it will find a corresponding element in the other domains. This also means that the whole computation is of no effect in the majority of cases. What we propose is to wait with the domain reduction until the initial value problem is included. In other words, the initial value problem is regarded as a constraint with a priority higher than all of the other constraints. The initial values will certainly reduce the domains dramatically, if not to one element (if `qmag` and `qdir` are given). After these reductions, the propagation will wake up the different constraints and the verification of the constraints will then effectively reduce the domains.

4.3 Experimental Results

To obtain some realistic results for the efficiency of our implementation of c-filter, two different QSIM models have been taken: the *Starling* model with 17 variables and 18 constraints and the *bathtub* model with 6 variables and 6 constraints — both models are defined in (Kuipers 1994). The runtimes for the c-filter were measured with the internal timer of a Sun Sparc 10 workstation. To create similar conditions for the input of the c-filter in Lisp and in ECL^{i}PS^e the input for the C implementation of c-filter from Rinner (1995) is used.

We compare the runtimes of the compiled Lisp implementation of c-filter in QSIM on one hand with the untraceable version of c-filter in ECL^{i}PS^e on the other hand.

	Starling	bathtub
Lisp	2.40 [s]	0.02 [s]
ECL^{i}PS^e	3.21 [s]	0.31 [s]

We now compare the runtimes of the uncompiled Lisp implementation of c-filter in QSIM on one hand with the traceable version of c-filter in ECL^{i}PS^e on the other hand.

	Starling	bathtub
Lisp	8.83 [s]	0.92 [s]
ECL^{i}PS^e	3.74 [s]	0.40 [s]

Multiple measurements of the same model will give deviations of only 1-2 milliseconds to the presented values.

As we can see there is no remarkable difference between the traceable and untraceable version of c-filter in ECL^{i}PS^e. This is due to the fact that ECL^{i}PS^e is already compiling the code even if it is traceable.

The ECL^{i}PS^e implementation is always faster if the Lisp code is not compiled; the Lisp implementation is faster only in the compiled form. Models with a few constraints and variables are considerably slower due to the overhead of the FD constraint solver. But this overhead pays off in large problems as we can see in the Starling model.

A major gain of the use of ECL^{i}PS^e for implementing c-filter is the implementation time: if the use of ECL^{i}PS^e and the specification of c-filter are known, the implementation will take about 2 - 4 weeks for one person.

5 Conclusions and Further Works

We have proposed a new application field, the Qualitative Reasoning for the finite domain solver of the CLP. To present the questions and problems of CSP in the area QR we chose the core filtering algorithm of the QSIM algorithm by Benjamin Kuipers. We gave the exact specification of the filtering algorithm in the logical framework in Section 3 and described technical details of the implementation in Section 4.

Further works concern more connections between Qualitative Reasoning and Constraint Logic Programming.

One interesting CSP in the area of QR is the *qualitative spatial reasoning* (Hernández 1994). The problem is not an arc-consistency but a *2-path-consistency* problem. We are convinced that the propagation of the finite domain library can be used efficiently to solve that problem. The solution would also have another advantage: if the algorithm solves the consistency problem in the qualitative spatial reasoning we would also have an algorithm for the *qualitative temporal reasoning* (Allen 1983).

Another problem is to evaluate other existing technical aspects of CLP for applications in QR. In ECL^{i}PS^e-5.2 **OR**-parallelism was implemented even in the finite domain library. We are very interested what increase in performance can be achieved through it.

Acknowledgments I wish to thank Wolfgang Förstner for his advice and support.

References

Aggoun, A., D. Chan, P. Dufresne, E. Falvey, H. Grant, A. Herold, G. Macartney, M. Meier, D. Miller, S. Mudambi, B. Perez, E. van Rossum, J. Schimpf, P. A. Tsahageas, and D. H. de Villeneuve (1995). ECL^{i}PS^e 3.5. User Manual. `http://www.ecrc.de/eclipse/eclipse.html`.

Allen, J. F. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM 26*(11), 832–843.

Brisset, P., T. Frühwirth, C. Gervet, P. Lim, M. Meier, T. L. Provost, J. Schimpf, and M. Wallace (1995). ECL^{i}PS^e 3.5. Extension User Manual. `http://www.ecrc.de/eclipse/eclipse.-html`.

Frühwirth, T., A. Herold, V. Küchenhoff, T. L. Provost, P. Lim, E. Monfroy, and M. Wallace (1992, September). Constraint Logic Programming — An Informal Introduction. In G. Comyn (Ed.), *Logic programming in action: second International Logic Programming Summer School*, Volume 636 of *LNCS*, pp. 3–35. Springer Verlag.

Hernández, D. (1994). *Qualitative Representation of Spatial Knowledge*. LNAI 804. Berlin: Springer.

Jaffar, J. and M. J. Maher (1994, May-July). Constraint Logic Programming: A Survey. *Journal of Logic Programming 20*, 503–581.

Kuipers, B. (1994). *Qualitative Simulation*. The MIT Press.

Mackworth, A. (1977). Consistency in networks of relations. *Artificial Intelligence 8*(1), 99–118.

MQ&D (1995, Sept./Dec.). Qualitative Reasoning: A Survey of Techniques and Applications. *AICOM 8*(3/4), 119–191. The survey was coordinated by Philippe Daque. MQ&D is the Frech 'Modélisation Qualitative et Désicion' group.

Rinner, B. (1995). Qsim kernel interface. Technical Report 95/02, Institute for Technical Informatics, Graz University of Technology, Austria.

Teleki, L. (1996, 21-24 may). Constraint Logic Programming: a Framework for Qualitative Reasoning. In *Proceedings of the Tenth International Workshop on Qualitative Reasoning*. AAAI.

van Hentenryck, P. (1991). Constraint Logic Programming. *The Knowledge Engineering Review 6*(3), 151–194.