Constraint Logic Programming — a Framework for Qualitative Reasoning

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*. The use of CLP has two advantages: (i) CLP gives a well defined and understood logical framework for the problem specification, and (ii) CLP is not only a logical framework, it is also a family of languages specially developed for solving classes of CSP problems. Thus we obtain a class of powerful implementation languages for rapid prototyping.

To illustrate the steps of specification and implementation we describe in detail the core of the QSIM algorithm (Kuipers 1994), namely the filtering of the state transitions in the CLP framework. We show how the basic constraints are specified in this framework and describe the technical aspects of an implementation. We want to demonstrate the advantages of CLP through an example for a large and complex qualitative reasoning algorithm.

Motivation

In the last fifteen years many frameworks were developed in the Qualitative Reasoning (QR) community to describe different qualitative reasoning problems. Some of these frameworks contain, or are special subproblems of the general class of *constraint satisfaction problems* (CSP) like qualitative simulation (QSIM), (Kuipers 1994), qualitative spatial reasoning (Hernández 1994) or qualitative temporal reasoning (van Beek 1992) to mention some important branches. However, as the problems are formulated in different areas, the specification is also done in different frameworks. This makes the understanding and comparison of the methods rather laborious. On the other hand, the implementation of these algorithms requires a substantial amount of work due to the specialized algorithms developed for every class of problems: one classical example is the *c-filter* of the QSIM algorithm in (Kuipers 1994).

We propose in this paper a common specification framework for the constraint satisfaction problems, namely the *Constraint Logic Programming* (CLP) (Jaffar & Maher 1994). We claim that by using CLP there are two advantages:

- CLP gives a well defined and 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.

To present the CLP scheme and the implementation aspects, we describe the QSIM algorithm by Kuipers 1994. This example was chosen precisely because the algorithm is well known in the QR community and it is a non-trivial problem that took many years to implement. Thus, we hope to give a large and complex example of an algorithm that allows us to describe many details and to present different paths of realizations and improvements.

All the ideas presented in this paper were implemented in ECL^{i}PS^e (User Manual 1995; Extension User Manual 1995), a CLP platform developed at the European Computer-Industrie Research Center (ECRC). Certainly, in almost every case, a specialized algorithm will give a better performance, but ECL^{i}PS^e also give very good results. 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 *Introduction*, we present the core of the QSIM algorithm and the CLP scheme. In Section *QSIM in CLP* we specify in every detail the constraints of QSIM in our

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

logical framework. In Section *Technical Aspects*, the implementation steps to be done by the user, the work done by the CLP system and different methods to improve the performance are described. We close with conclusions and further works in Section *Conclusions and Further Works*.

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.

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.

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 `qvals` must be consistent with the state's constraints *corresponding values*. The `qdirs` 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*, a correspondent to the domain of the function. Important elements in the landmark list are `minf`, `zero`, `inf` the correspondents to $-\infty$, 0 and ∞.

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.

A crucial task of QSIM is to filter in every time step from the result of the variable transitions 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.

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 *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).

The FD system alone will rarely be used alone to solve a problem since, in general, there remain com-

binations 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 & 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 $V(L) = \{(a, b), (b, c), (a, c), a, b, c\}$.

QSIM in CLP

We present the formal description of the filtering of the state transitions of the QSIM algorithm in 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.

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 `qval` $= \langle$`qmag`, `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:

$$\mathcal{D}_{x_i} = \{\langle v, dir \rangle | \; v \in V(L_{x_i}),\ dir \in \{\texttt{std}, \texttt{inc}, \texttt{dec}, \texttt{unknown}\}\}$$

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:

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

Signs

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

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

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. $befor(x, l_g, l_i, L_x)$ is *true* if l_g is before the element l_i in the list L; $after(x, l_g, l_i, L_x)$ is the contrary to $befor$.

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 $sign(x) : \mathbb{R} \to S'$ with $S' = \{+, -, 0, ?\}$ the set of extended signs. The three first elements of S' divide the $\mathbb{R}$ 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 $sign(x)_a = sign(x - a)$. If $a = 0$ we have the definition presented previously.

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

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

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. $sign(x,(a,c),[a,b,c,d])_{(b,d)} = \mathit{unknown}$ for the variable x with the landmark list $[a,b,c,d]$.

In the following we use $sign(x,l,L_x)$ for $sign(x,l,L_x)_{\mathit{zero}}$. The *sign* function is valid for the two symbols `inf` and `minf`.

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

The basic constraints of QSIM

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

$x =_+ y$ iff
$\langle x, y\rangle \in$
$\{\langle \mathit{pos}, \mathit{pos}\rangle, \langle \mathit{neg}, \mathit{neg}\rangle, \langle \mathit{zero}, \mathit{zero}\rangle,$
$\langle \mathit{unknown}, \mathit{pos}\rangle, \langle \mathit{unknown}, \mathit{neg}\rangle, \langle \mathit{unknown}, \mathit{zero}\rangle\}$

$x =_- y$ iff
$\langle x, y\rangle \in$
$\{\langle \mathit{pos}, \mathit{neg}\rangle, \langle \mathit{neg}, \mathit{pos}\rangle, \langle \mathit{zero}, \mathit{zero}\rangle,$
$\langle \mathit{unknown}, \mathit{pos}\rangle, \langle \mathit{unknown}, \mathit{neg}\rangle, \langle \mathit{unknown}, \mathit{zero}\rangle\}$

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 $\mathtt{qval} = \langle \mathtt{qmag}, \mathtt{qdir}\rangle$. The following two functions make the projections onto the two members of the tuple:

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

We now focus on the exact definition of the constraints:

M+: $(\mathtt{M+}(x,y), CV)$. 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. $\forall \langle l_{x_i}, l_{y_i}\rangle \in CV,\ sign(x, qmag(x_g), L_x)_{l_{x_i}} =_+ sign(y, qmag(y_g), L_y)_{l_{y_i}}$

M-: $(\mathtt{M-}(x,y), CV)$. 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. $\forall \langle l_{x_i}, l_{y_i}\rangle \in CV,\ sign(x, qmag(x_g), L_x)_{l_{x_i}} =_- sign(y, qmag(y_g), L_y)_{l_{y_i}}$

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

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

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

$R_{\mathtt{add}}$	*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).

[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 $y_g \in D_y$.

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:
   $sign(x, \texttt{zero}, L_x)sign(y, \texttt{inf}, L_y) = unknown$,
   $sign(x, \texttt{zero}, L_x)sign(y, \texttt{minf}, L_y) = unknown$,
   $sign(x, \texttt{inf}, L_x)sign(y, \texttt{minf}, L_y) = unknown$.
   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).

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: $\texttt{d/dt}(x, y)$. 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 $\texttt{constant}(x)$ or $\texttt{constant}(x, a)$. **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. $(sign(qdir(x_g)) =_+ zero)$

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

Technical Aspects

With the constraint specification in the FD scheme, the implementation is straightforward; this is one of the major gains if we use this logical framework. The difficult work of the constraint solving mechanism, namely the propagation of the domain reductions is done by the system. The user only needs to specify the single constraints and does not needs to solve the constraint network.

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.

General Aspects

The general procedure is presented in three steps. First, we present the implementation of the constraints. In a second subsection the propagation of the FD system is described. We finish with the presentation of the labeling procedure.

Implementation of Constraints The goal of the implementation of a constraint is to determine those elements of the variable domains that satisfy the constraint[2], as defined in the previous Section. This verification will lead to a domain reduction propagated later on by the FD solver.

We illustrate the procedure by the following example. The variable x has the landmark list $L_x = [\texttt{zero}, \texttt{full}, \texttt{inf}]$ and the variable y the landmark list $L_y = [\texttt{zero}, \texttt{inf}]$. The constraint has the form $(\texttt{M+}(x, y), \{\langle \texttt{zero}, \texttt{zero} \rangle, \langle \texttt{inf}, \texttt{inf} \rangle\})$. We suppose that the domains are:

$$D_x = [\langle \texttt{full}, \texttt{inc} \rangle, \langle (\texttt{zero}, \texttt{full}), \texttt{inc} \rangle]$$
$$D_y = [\langle (\texttt{zero}, \texttt{inf}), \texttt{inc} \rangle, \langle (\texttt{zero}, \texttt{inf}), \texttt{std} \rangle]$$

The **M+** constraint allows the following combinations of variable values:

$$\langle x, y \rangle \in \{\langle \langle \texttt{full}, \texttt{inc} \rangle, \langle (\texttt{zero}, \texttt{inf}), \texttt{inc} \rangle \rangle, \langle \langle (\texttt{zero}, \texttt{full}), \texttt{inc} \rangle, \langle (\texttt{zero}, \texttt{inf}), \texttt{inc} \rangle \rangle\}$$

The other combinations do not satisfy the constraint, e.g.

$$\langle \langle (\texttt{zero}, \texttt{full}), \texttt{inc} \rangle, \langle (\texttt{zero}, \texttt{inf}), \texttt{std} \rangle \rangle$$

as $sign(qdir(\langle (\texttt{zero}, \texttt{full}), pos \rangle)) = pos$, $sign(qdir(\langle (\texttt{zero}, \texttt{inf}), zero \rangle)) = zero$ and

[2] Those who are familiar with the details of QSIM will realize that there is no difference in the algorithm to determine the initial state or to generate new states. The specification of the initial value problem can also be regarded as a constraint.

$\langle pos, zero \rangle \notin =_+$ (the first constraint of M+). This means that after the verification of the combination of variable values the domain D_x remains unchanged and D_y is reduced to $D'_y = [(\texttt{zero}, \texttt{inf}), \texttt{inc}\rangle]$.

It is important to see that the algorithm which reduces the domain is not given. In QSIM we did not implement a special algorithm to decide which elements of the domains fulfill the constraints. Even in the case of constraints with three variables a simple backtracking is fast enough. In other problems, with domains of high cardinality or with computationally complex constraints, the decision about value combinations that satify the constraints may be slow. In these cases, a special algorithm can be implemented even for only one of the constraints. But all these implementation aspects will not at all influence the work done by the system. This is the reason why new constraints can always be added with minimal effort.

Propagation of domain reductions Now we are able to describe the link between QSIM and CLP in a straightforward manner. Reasoning is finding admissible state sequences. It is realized by using the FD solver of CLP. The FD solver propagates the domain reductions (and only those) over the constraint network and wakes up the constraints containing one or more variables with reduced domains. The wake up of the constraints is done until no domain change occurs in the whole network.

We illustrate the propagation and rescheduling on a network with two constraints.

$(\texttt{M+}(x, y), CV_1)$
$(\texttt{add}(x, y, z), CV_2)$

D_x, D_y and D_z are the domains of the variables. M+ will reduce D_x to D'_x and D_y to D'_y. The FD solver will take the new domains and wake up the add constraint. Let's say this constraint will reduce D'_x to D''_x and D_z to D'_z; D'_y will remain unchanged. The FD solver will realize that the domain of the variable x is changed and will wake up M+ again, now with the new D''_x and the old D'_y domain. If there is a change in D''_x or D'_y than FD solver will wake up again the add constraint. This scheduling of the constraints is continued until there is no change in the variable domains.

Labeling In the regular case of labeling the domains will not contain only one element. There will instead sets of elements from which we need to create all the states to continue with. To find solutions, the system will search by labeling a variable with a value in its domain. This choice (which may later prove as having been erroneous) allows further propagation in the same manner as presented in the previous Subsection. The reduction of the domains will continue until there is no change in the domains. Then the other variables are labeled recursively until a solution is found.

Improving the performance

The last subsection describes the general ideas of domain reduction, propagation and labeling. If the number of variables and the cardinality of the domains is small there is no need for different improvements. However, in real case problems we will need a speedup.

The speedup can be achieved in many ways. There are so called *problem specific* improvements and *general improvements*. *Problem specific improvements* are defined by the specific constraints of QSIM. *General improvements* are independent from our problem.

Problem specific improvements 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.

In QSIM the generation of the initial state has a special characteristic. The problem is that the verification of the constraints with complete domains 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.

General improvements The order of labeling can also improve the effectiveness of the search procedure. In general, it is more effective to use the variable with

the smallest remaining domain for labeling. This principle is referred to as *first fail principle*, as with fewer choices possible we will find out earlier if those where right or wrong. Another technique is to choose the variable which occurs in most constraints.

When constraints have different priorities we can select variables that occur in constraints with a low computational cost. This act would achieve that when the scheduler reaches the constraints with low priority, the domains are already very small and only a small amount of values must be analyzed.

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 he 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.

Conclusions and Further Works

We have proposed a new framework, the finite domain solver of the constraint logic programming paradigm to describe different kinds of Constraint Satisfaction Problems used in the qualitative reasoning community. To present the details of the logical framework and the the flexibility of the implementation, we chose the core filtering algorithm of the QSIM by Benjamin Kuipers. We gave the exact specification of the filtering algorithm in the logical framework in Section *QSIM in CLP* and described technical details of the implementation in Section *Technical Aspects*.

Further work concerns two aspects: (i) how flexible the FD solver is and (ii) which other constraint solver can serve to specify and implement QR problems.

(i) To determine the flexibility of the FD solver we concentrate especially on the framework of Jézéquel & Zimmer (1995). The authors explicitly expressed the knowledge embedded in the different operators known from QSIM by introducing new constraints like inequalities ($\leq$, $<$), equality and constancy propagation, thus allowing transitive propagation between the parameters that leads to the elimination of spurious behavior. We are interested to see what changes needs to be made to integrate these constraints in the presented logical framework and the implementation.

(ii) Lastly, the CLP framework has not only the FD solver (van Hentenryck 1991). Frühwirth (1992) developed a flexible environment, the *Constraint Handling Rules* (CHR), to implement user defined constraints by introducing multi-head guarded rules allowing *propagation* and *simplification*. In (Frühwirth 1994) he describes the implementation of temporal reasoning presented in (Meiri 1991). We want to analyze how this framework can be used in constraint systems like QSIM with the aim of proposing other powerful tools to reduce the time of implementation development that will allow an early detection of conceptual problems or new direction of research.

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

References

1995. ECL^{i}PS^e 3.5 Extension User Manual. `http://www.ecrc.de/eclipse/eclipse.html`.

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

Frühwirth, T. 1992. Constraint simplification rules. Technical Report ECRC-92-18, ECRC (European Computer-Industry Research Centre).

Frühwirth, T. 1994. Temporal reasoning with constraint handling rules. Technical Report ECRC-94-05, ECRC.

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

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

Jézéquel, P., and Zimmer, L. 1995. Better expression of knowledge to reduce spurious behaviour in qualitative simulation. In *Working Papers of the Ninth International Workshop on Qualitative Reasoning, University of Amsterdam, Amsterdam*, 104–113.

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

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

Meiri, I. 1991. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *AAAI 91*, 260–267.

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

1995. ECL^{i}PS^e 3.5 User Manual. `http://www.ecrc.de/eclipse/eclipse.html`.

van Beek, P. 1992. Reasoning about Qualitative Temporal Information. In Faltings, B., and Struss, P., eds., *Recent Advances in Qualitative Physics.* The MIT Press. chapter 14.

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