Extending RISC-CLP($\mathcal{R}eal$) to Handle Symbolic Functions

Olga Caprotti*

Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria
e-mail: ocaprott@risc.uni-linz.ac.at

Abstract. In the previous version of the constraint logic programming language RISC-CLP($\mathcal{R}eal$), the domain $\mathcal{R}eal$ of real numbers was the intended domain of computation. In this paper, we extend it to the domain $\mathcal{T}rees\mathcal{R}eal$ of finite symbolic trees with real numbers as leaves, that is the integration of the domain of reals with the domain of finite Herbrand trees. In the extended language, a system of constraints over the new domain is decided by first decomposing equations into a tree-solved form produced by an adapted unification algorithm. Then, polynomial real constraints are decided by the partial cylindrical algebraic decomposition method and a solution to the original system is constructed.

1 Introduction

In this paper we describe how the language RISC-CLP($\mathcal{R}eal$) [12] has been extended by the introduction of symbolic function symbols. This language is an instance of the Constraint Logic Programming (CLP) scheme that was defined by Jaffar and Lassez in [16] and describes a class of logic programming languages combining logic programming with constraint solving. The CLP scheme relaxes the restriction on the Herbrand universe by allowing a variety of domains of computation, such as trees, booleans and reals, and it relaxes the restriction on unification by employing domain-specific decision algorithms. Therefore, there are two key implementation issues:

1. a clear definition of what is the intended domain of computation,
2. the existence of a decision procedure to solve constraints over such a domain.

We dealt with both these issues while extending the language RISC-CLP($\mathcal{R}eal$).

Currently, among the main CLP representatives are the languages: Prolog III, CLP($\mathcal{R}$), and CAL. Colmerauer's Prolog III [7, 6] combines the domains of rational numbers, lists and booleans into the domain of infinite rational trees. It supports construction, boolean, arithmetical operations and binary relations

* In the frame of the ACCLAIM project sponsored by
European Community ESPRIT BRA 7195
Austrian Science Foundation P9374 – PHY.

such as $\{\neq, >, <, \geq, \leq\}$ together with unary relations that declare the domain to which a variable belongs.

Jaffar and his group's CLP($\mathcal{R}$) [9, 14] computes on the domain of trees over reals and supports decision of linear constraints among arithmetical terms [15, 19]. An analysis of the main differences among CLP($\mathcal{R}$) and RISC-CLP is presented in [12].

Sakai and Aiba's CAL (Contrainte avec Logique) [22] offers nonlinear constraints over the complex numbers by employing Buchberger algorithm as decision algorithm [2, 3]. RISC-CLP($\mathcal{R}eal$) incorporates Gröbner bases method to check unsolvability before resorting to the more expensive cylindrical algebraic decomposition (CAD) method. Monfroy in [20] also describes a CLP language that deals with nonlinearity over the complex numbers using Gröbner bases method. Examples of other CLP languages are CHIP over finite domains [1], CLP(Σ^*) over regular sets [24], Echidna over discrete sets [8], BNR-Prolog over real intervals [21].

Whereas most of the well known implementation of the CLP scheme are concerned with finite domains or only with linear constraints, RISC-CLP($\mathcal{R}eal$) is able to deal with nonlinearity over the real numbers by employing partial cylindrical algebraic decomposition as quantifier elimination procedure for real closed fields [4, 10]. On the other hand, RISC-CLP($\mathcal{R}eal$) offered so far no possibility of data structuring via non arithmetical terms because symbolic functions and lists were not supported. We extended the domain of computation of RISC-CLP($\mathcal{R}eal$) by adding symbolic function identifiers to the set of arithmetical function symbols, $\{+, *, -\}$, and we modified the solver module in order to decide the new type of constraints arising. The new version of RISC-CLP($\mathcal{R}eal$) has as domain of computation the domain of *finite symbolic trees with real numbers* which is the usual Herbrand universe equipped with real numbers. Because the decision procedure on the Herbrand universe is unification, it is natural that a unification algorithm, inspired by the one described in [17] and presented as a rule based algorithm like in [18], is the major change we have integrated into the solver.

The paper is structured as follows. Section 2 describes the domain of symbolic trees with real numbers as a combination of two well-known domains: real numbers and finite trees. Section 3 shows how a system of constraints over $\mathcal{T}rees\mathcal{R}eal$ is decided by the solver that combines a decision algorithm for the real numbers with the "adapted unification" algorithm. For the details of how the solver decides nonlinear constraints over the reals, the interested reader is referred to [12, 10]. Section 4 shows an example program in the RISC-CLP language that can solve problems in chemical reaction systems.

2 Domain of Computation and Language

In the following Section, we recall the two domains: $\mathcal{R}eal$ of real numbers and $\mathcal{T}rees$ of finite trees, and we describe the combined domain: $\mathcal{T}rees\mathcal{R}eal$.

Real constant symbols:	fractions, e.g. `2/3, -1, 3.75`
Real function symbols:	arithmetical, i. e. $\{+, *, -\}$.
Real relation symbols:	arithmetical, i. e. $\{\neq, >, <, \geq, \leq\}$, real equality $=_R$.
Real-term:	is a real constant, or is a variable or has the form '$t_1 \;\; \alpha \;\; t_2$' where α is a real function symbol and t_1, t_2 are real-terms.
Real-constraint:	is of the form '$t_1 \;\; \rho \;\; t_2$' where t_1, t_2 are real-terms and ρ is a real relation symbol.

Fig. 1. Language and syntax of the domain $\mathcal{R}eal$.

Tree constant symbols:	symbolic.
Tree function symbols:	symbolic tree constructors.
Tree relation symbols:	tree equality $=_T$.
Tree-term:	is a tree constant, or is a variable or has the form: '$f(t_1, \ldots, t_n)$' where f is a tree function symbol of arity $n > 0$ and $t_1, \ldots, t_n$ are tree-terms.
Tree-constraint:	has the form '$t_1 =_T t_2$' where t_1, t_2 are tree-terms.

Fig. 2. Language and syntax of the domain $\mathcal{T}rees$.

Constant symbols:	real and tree constant symbols.
Function symbols:	real and tree function symbols.
Relation symbols:	arithmetical, i.e. $\{\neq, >, <, \geq, \leq\}$, equality $=$.
Term:	is a variable, a real-term, a tree-term or has the form: '$f(t_1, \ldots, t_n)$' where $t_1, \ldots, t_n$ are terms and f is a tree function symbol of arity $n > 0$.
Constraint:	is of the form '$t_1 \; \rho \; t_2$' where t_1, t_2 are terms and ρ is a relation symbol.

Fig. 3. Language and syntax of the domain $\mathcal{T}rees\mathcal{R}eal$.

The domain $\mathcal{R}eal$ is obviously the set of all real numbers, including all real algebraic and transcendental numbers. A careful reader might wonder how it is possible to compute on the domain $\mathcal{R}eal$ when such a domain is known to be uncountable. The answer is provided by Tarski's theorem [23] which states that the validity of any first order sentence over real numbers is the same as the validity over algebraic numbers, and these are indeed countable. Hence, it is convenient to assume that the domain $\mathcal{R}eal$ is the set of only real algebraic numbers. The language and the syntax of the domain of $\mathcal{R}eal$ are recalled in Figure 1. Real-terms and real-constraints are interpreted in the conventional way of arithmetic. In this domain, all functions are total and conjunction of real-constraints is decided by the quantifier elimination algorithm based on partial CAD [4, 10, 5].

The domain $\mathcal{T}rees$ consists in the set of finite trees labelled by identifiers. Language and syntax of the domain of $\mathcal{T}rees$ are recalled in Figure 2. In this domain, tree terms are interpreted as labelled n-ary trees. Like in the domain $\mathcal{R}eal$, all the functions are total. Syntactical unification is the well known decision procedure for solving a conjunction (set) of tree-constraints.

Now we describe the combined domain $\mathcal{T}rees\mathcal{R}eal$ that is similar to the domain of CLP($\mathcal{R}$). $\mathcal{T}rees\mathcal{R}eal$ is the set of finite symbolic trees having possibly real numbers as leaves. It can be recursively defined as follows.

- Leaves: Nodes labelled by real numbers in $\mathcal{R}eal$, and by identifiers in $\mathcal{T}rees$, are in $\mathcal{T}rees\mathcal{R}eal$.
- Recursion: For all tree function symbols f of arity $n > 0$, if $d_1, \ldots, d_n$ are in $\mathcal{T}rees\mathcal{R}eal$, then the tree, whose root is labelled by the identifier "f" and whose children are the trees $d_1, \ldots, d_n$, is in $\mathcal{T}rees\mathcal{R}eal$.

Its language consists in the merging of the languages of the two domains $\mathcal{R}eal$ and $\mathcal{T}rees$, and the syntax is as described in Figure 3. Note that we do not keep the equalities $=_R$ and $=_T$, but we have instead a unique symbol $=$. The (logical or declarative) semantics interprets arithmetical symbols as the usual operations and relations over $\mathcal{R}eal$, while tree functions are interpreted over $\mathcal{T}rees\mathcal{R}eal$.

- Terms: Real constants are interpreted as nodes labelled by real numbers in $\mathcal{R}eal$.
  Tree constants are interpreted as nodes labelled by identifiers in $\mathcal{T}rees$.
  Real-terms are interpreted as nodes labelled by the real numbers that are their interpretation over $\mathcal{R}eal$.
  A term of the form '$f(t_1, \ldots, t_n)$' is the tree in $\mathcal{T}rees\mathcal{R}eal$ whose root is labelled by the identifier "f" and whose children are the the interpretations of $t_1, \ldots, t_n$ over $\mathcal{T}rees\mathcal{R}eal$.
- Constraints: '$t_1 \; \rho \; t_2$', where ρ is among $\{\neq, >, <, \geq, \leq\}$ is true iff it is true when t_1 and t_2 are interpreted over $\mathcal{R}eal$.
  '$t_1 = t_2$' is true if the trees in $\mathcal{T}rees\mathcal{R}eal$ corresponding to t_1 and to t_2 are the same, more precisely:

nodes labelled by real numbers are equal as in $\mathcal{R}eal$,
nodes labelled by identifiers are equal as in $\mathcal{T}rees$,
and trees corresponding to '$f(t_1, \ldots, t_n)$' and to '$g(s_1, \ldots, s_m)$' are equal iff the identifier "f" is equal to "g" and m is equal to n.

We now recall the notions of solution of a system of constraints and fix some notation for the rest of the paper.

Let V be the set of all variables, let W be the subset $\{X_0, \ldots, X_n\} \subset V$, and let t and s be terms. A *system of constraints* S is a finite set of constraints. An *assignment* α *of* W *on a domain* D is a set having the form: $\{X_0 := d_0, \ldots, X_n := d_n\}$ in which $d_0, \ldots, d_n$ are in D. An assignment of W on the domain D satisfies a system of constraints S in variables W iff, interpreting the variables by the assigned element of D, each constraint in the system evaluates to true on D. Such an assignment is called a *solution* of S. We will not specify the domain unless it is needed to avoid confusion. If W' is a subset of the variables W occurring in a system of constraints S, then an assignment α' of W' is a solution of S on W' iff it is a subset of at least one solution of S.

Two constraints or two systems of constraints are *equivalent* iff the sets of their solutions are equal. A constraint or a system of constraints admitting at least one solution is called *solvable*.

3 The Solver

A solver for a domain is an algorithm which decides the solvability of a given system of constraints over the domain and produces a "simplified" form in the positive case.

For instance, the well-known unification is a solver for the domain $\mathcal{T}rees$. Also the partial cylindrical algebraic decomposition method (used in RISC-CLP($\mathcal{R}eal$)) is a solver for the domain $\mathcal{R}eal$.

In this section, we describe a solver for the combined domain $\mathcal{T}rees\mathcal{R}eal$. The solver is essentially an integration of the unification and the partial cylindrical algebraic decomposition method.

It first applies an "adapted unification" on the given system of constraints which decomposes the system of equations into an equivalent but simpler system of equations called "a tree-solved form". Then, it passes some part of the tree-solved form and the other constraints to the partial cylindrical algebraic decomposition method. In the following we describe the details.

First, we begin by defining the notion of "tree-solved form". We define the *depth* of a term t, $\delta(t)$, as a function from the terms in the language of $\mathcal{T}rees\mathcal{R}eal$ to the natural numbers as follows.

$$\delta(t) := \begin{cases} 1 & \text{if } t \text{ is a variable, a constant or a real-term} \\ 1 + max\{\delta(t_1), \ldots, \delta(t_n)\} & \text{if } t \text{ is of the form } f(t_1, \ldots, t_n) \end{cases}$$

The definition is extended to equations and to finite systems of equations.

$$\delta_e(t = s) := min\{\ \delta(t),\ \delta(s)\ \},$$

$$\delta_s(\{\, e_1, \ldots,\ e_n \,\}) := max\{\, \delta_e(e_i) \mid 1 \leq i \leq n \,\}.$$

Definition 1. A finite system S of equations in variables W is in *tree-solved form* iff there exists a subset of variables $\{X_1, \ldots, X_h\} \subseteq W$ such that S has the form:

$$S = \{X_1 = t_1, \ldots, X_h = t_h,\ s_{h+1} = t_{h+1}, \ldots, s_n = t_n\}$$

and the following holds :
(i) S has depth 1 and contains no identity.
(ii) X_i does not occur in t_i nor anywhere else in S, $1 \leq i \leq h \leq n$.

The reason behind defining such form is that, once the equality constraints are in tree-solved form, we can easily decide the whole system of constraints by using a decision procedure for the domain $\mathcal{R}eal$, as will be shown now.

Assuming the availability of an algorithm for transforming equality constraints into the tree-solved form, we give a solver for the domain $\mathcal{T}rees\mathcal{R}eal$ in Figure 4.

Let C be solvable system of constraints in variables $W' \subseteq W$, and let S be a new system of constraints in variables W. The solver decides if the system $C \cup S$ has a solution on W over $\mathcal{T}rees\mathcal{R}eal$. In case it does, then the simplified output is called a "tree-real partition" of $C \cup S$ of W. Let C be a solvable system of constraints over $\mathcal{T}rees\mathcal{R}eal$ in variables W. $C_e \uplus C_r$ is a *tree-real partition of C of W* if, for every solution σ on $\mathcal{R}eal$ of C on the variables W_r occurring in C_r, there exists a unique assignment α of $W \setminus W_r$ on $\mathcal{T}rees\mathcal{R}eal$ such that $\sigma \cup \alpha$ is a solution of C over $\mathcal{T}rees\mathcal{R}eal$.

We assume that the input to the solver is a tree-real partition of system $C = C_e \uplus C_r$ and a new system of constraints S. The algorithm is structured mainly in three parts: firstly, it checks whether an empty system of new constraints is given. Then, it constructs the equation part of the tree-real partition of $C \cup S$ from the tree-solved form of the system of equations $C_e \cup S_e$. If this construction succeeds without causing any type error, the solver decides, in the last step, whether the real part of the tree-real partition of $C \cup S$ has a solution over $\mathcal{R}eal$ on the variables therein. If it does, then we are done. If it does not, then the system $C \cup S$ is not solvable because the real-constraints are not.

The correctness is based on the observation that the system obtained from the tree-solved form C'_e in step (2.c) of the algorithm is a "solved system" in the usual sense of unification: each equation contains a variable that occurs nowhere else, in particular (after type checking) it does not occur in any arithmetical term. Therefore, it is possible to extract an associated substitution representing the required assignment α for the tree-real partition $C' = C'_e \uplus C'_r$ that combined with a solution σ of C'_r is a solution of C'.

As promised earlier, now we give in Figure 5 an algorithm for computing a tree solved form of a given system of equations. The algorithm is a unification algorithm adapted from [17, 18].

In the algorithm description, the following (standard) notion of "substitution" is used. A *substitution σ of W* is a set: $\{X_0 \leftarrow t_0, \ldots, X_n \leftarrow t_n\}$, *SOLVER* $(C, S;\ u, C')$

Input: $C = C_e \uplus C_r$ is a tree-real partition of C of $W' \subseteq W$.
S is a system of new constraints in variables W.

Output: $C' = C'_e \uplus C'_r$ is a tree-real partition of $C \cup S$ of W.
u is '`solvable`' if a partition of $C \cup S$ of W is found, '`unsolvable`' otherwise.

(1) [Trivial.] If the new set of constraints S is empty, set $C' \leftarrow C, u \leftarrow$ '`solvable`' and return.

(2) [Tree-solve.]
   (a) Set $S \leftarrow S_e \cup S_r$, where equations are in S_e, and constraints involving arithmetical relation symbols are in S_r.
   (b) Apply *TREE-SOLVE* $(S_e \cup C_e;\ C'_e)$. If the tree-solved form C'_e is `FALSE`, then $C \cup S$ is not solvable. Set $u \leftarrow$ '`unsolvable`' and return.
   (c) Set $C'_e \leftarrow C'_e \setminus S'_r$, where S'_r is the set of equations '$t = s$' in which at least one among t or s is a real-term or is a variable occurring in a real-term or in $C_r \cup S_r$.
   (d) Type-check each variable X appearing in equations '$X = t$' of C'_e where t is not a variable. If X occurs in $C_r \cup S_r \cup S'_r$, then there is a type-error and $C \cup S$ is not solvable. Set $u \leftarrow$ '`unsolvable`' and return.

(3) [Decide.] Decide the truth of the sentence $(\exists \bar{W})(C_r \cup S_r \cup S'_r)$ by the solver over $\mathcal{R}eal$. If it is `FALSE`, then $C \cup S$ is not solvable. Set $u \leftarrow$ '`unsolvable`' and return. Otherwise, set $C' \leftarrow C'_e \uplus C'_r$, where C'_r is the simplified system returned by the solver over $\mathcal{R}eal$. Set $u \leftarrow$ '`solvable`' and return.

Fig. 4. Algorithm *SOLVER*.

where $t_0, \ldots, t_n$ are terms such that $X_i \neq X_j$, and X_i does not occur in t_i for $1 \leq i,\ j \leq n$. By $t\sigma$ we denote the term obtained by simultaneously replacing in t each occurrence of the variable X_i with the term t_i, for all $1 \leq i \leq n$. We denote by $c\sigma$ and by $S\sigma$ the natural extension of the application of a substitution to a constraint c and to a system of constraints S. By `FALSE` we mean that the system is not solvable.

To prove the correctness of *TREE-SOLVE* we must show that the algorithm terminates according to a control strategy and that it conforms to the specification. Let the control strategy choose always the rule that applies to an equation in S of greater depth. Termination is based on the fact that after each application of a rule, either S is `FALSE`, or S is "simpler" in the following sense:

- S contains an identity less than before;
- S contains less occurrences for the variable substituted by rule (SV), and the rule cannot be applied anymore;
- S has a depth less than it had before the decomposition performed by rule (ET).

TREE-SOLVE $(S; S')$

Input: S is a system of equations in variables W.

Output: S' is a tree-solved form equivalent to S or `FALSE`.

(1) Transform S by applying one of the following rules until no change is possible or S is `FALSE`.

-RI- [Remove Identities]
$S \cup \{s = s\} \Longrightarrow S$.

-ET- [Equate Terms]
$S \cup \{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)\} \Longrightarrow S \cup \{s_1 = t_1, \ldots, s_n = t_n\}$.
$S \cup \{f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)\} \Longrightarrow$ `FALSE`,
if "f" is different from "g"
or m is not equal to n.

-EC- [Equate Constants]
$S \cup \{m = n\} \Longrightarrow$ `FALSE`,
if m and n are not equivalent constants.

-SV- [Substitute Variables]
$S \cup \{X = s\} \Longrightarrow S\{X \leftarrow s\} \cup \{X = s\}$,
if X is a variable in W occurring in S,
s is not a variable and X does not occur in s.
$S \cup \{X = Y\} \Longrightarrow S\{X \leftarrow Y\} \cup \{X = Y\}$,
if X and Y are distinct variables in W and
both occur in S.

-TC- [Type Check]
$S \cup \{s = t\} \Longrightarrow$ `FALSE`,
if s is a real-term and t is not a real-term.

-OC- [Occur Check]
$S \cup \{X = s\} \Longrightarrow$ `FALSE`,
if X is a variable occurring in s and s is not a real-term.

(2) Assign S to S' and return.

Fig. 5. Unification algorithm:*TREE-SOLVE.*

Therefore, because the initial set is finite and infinite trees are not allowed, *TREE-SOLVE* terminates.

Next, we show that each rule preserves the set of solutions of S. Rules (RI), (ET), (EC), (SV) and (OC) are clear, rule (TC) is a consequence of having defined arithmetical functions partial.

Finally, it remains to show that, given a system of equations S in variables W, *TREE-SOLVE* produces an equivalent tree-solved form S' or `FALSE`. Let S' be different from `FALSE` and be obtained from S by application of the rules until no change is possible. S' does not contain any trivial identity nor it contains equations among constants, because they are removed by rules (RI) and (EC).

Rule (ET) decreases the depth of S. Because no more rules can be applied to S', it is clear that S' must have depth equal to 1. Let the equation '$X = s$' be in S'. If X is a variable not occurring in the term s, then by the application of rule (SV), X does not appear anywhere else in S'. This shows that S' is a tree-solved form.

4 Example: Chemical Systems

This last section is devoted to showing by an example a possible application for the extended RISC-CLP($\mathcal{R}eal$) language. We have decided to give one big example instead of many "toy" examples, because we believe that this will better demonstrate the new language and its additional possibilities.

The scenario of our example is that of systems of chemical reactions. In a container we place some substances, say H_2O, CO, $Cl, \ldots$, and so on. Now, these substances will interact with each other, new substances will be produced and some will be used up, until possibly an equilibrium state is reached among them. At equilibrium, reactions still continue in the system but the net amounts of each substance remain the same.

In the usual model of a chemical system of reactions, we can denote the substances by $A_i, A_j, \ldots$ and their concentrations (atoms per unit of volume, measured in $moles \cdot liter^{-1}$) by $[A_i], [A_j], \ldots$, respectively. Each system is also described by reactions laws among substances that tell us what is the kinetics of the system. As an example, consider the system of reversible reactions:

$$\begin{aligned} A_i + A_j &\xrightarrow{k_1} A_k \\ A_k &\xrightarrow{k_{-1}} A_i + A_j \end{aligned}$$

where k_1 and k_{-1} are the reaction constants (in sec^{-1}) measuring the "rate" by which the substances on the left transform into substances on the right. For each particular system, we can associate a system of differential equations that tells us about the dynamic of the reactions. In case of the system above, the differential equations are as follows:

$$\begin{aligned} \frac{d[A_i]}{dt} = \frac{d[A_j]}{dt} &= -k_1[A_i][A_j] + k_{-1}[A_k] \\ \frac{d[A_k]}{dt} &= k_1[A_i][A_j] - k_{-1}[A_k] \end{aligned}$$

Note that each equation is not a "stand alone" description of each single reaction, but takes into account the entire system. The system described above reaches an equilibrium state that is characterized by the equations:

$$\begin{aligned} 0 &= -k_1[A_i][A_j] + k_{-1}[A_k] \\ 0 &= k_1[A_i][A_j] - k_{-1}[A_k] \end{aligned}$$

One can ask various questions about this system, for instance about the relative concentrations of the substances that made it possible to reach the equilibrium state.

The modelling process can be translated into a RISC-CLP($\mathcal{R}eal$) program by using the added capabilities provided by the extended language. In Figure 6, we show a RISC-CLP($\mathcal{R}eal$) example program that captures the scenario of reaction chains in its generality.

We now show how the model works by some examples, beginning with the description of water:

$$H_2O \xrightarrow{k_1} H^+ + OH^-$$
$$H^+ + OH^- \xrightarrow{k_2} H_2O$$

At equilibrium, we can ask RISC-CLP($\mathcal{R}eal$) what is the system of equations describing water by the following query:

```
?-  system([h2o,h,oh], [ [ [[h2o],[h,oh]],[H2O],K1],
                         [ [[h,oh],[h2o]],[H,OH],K2]],
          [0,0,0]).
```

The answer of RISC-CLP($\mathcal{R}eal$) is the general equilibrium state equation for a reversible system, like that given above.

```
K1 > 0
H2O >= 0
K2 > 0
H >= 0
H OH K2 - H2O K1 = 0
```

Suppose we want to compute the pH (defined as $-log_{10}[H^+]$), given that the concentration of pure water is known to be $55.5 \cdot moles \cdot liter^{-1}$, the equilibrium constant is $K = k_2/k_1 = 55.5 \cdot 10^{14}$ and the concentrations of H^+ and of OH^- are the same because there is no other source outside of water. The query would look as follows. Note that we are able to specify, in curly braces, which variables should appear in the answer.

```
?- H2O = 55.5, K1 * 55.5 * 10^14 = K2, H = OH,
   system([h2o,h,oh], [ [ [[h2o],[h,oh]],[H2O],K1],
                        [ [[h,oh],[h2o]], [H,OH],K2]],
         [0,0,0]) {H}.
```

The answer produced by RISC-CLP($\mathcal{R}eal$) is correct, the pH of water is 7 ($[H^+] = 10^{-7}$):

```
H = 1e-07
```

```
%========================
% chemical reactions laws
%========================

% predicate system(A,L,R): A is a list of names of substances, L is a
% list of reactions among substances and R is a list of rates Rate such
% that d[Ar]/dt = Rate, for each name Ar in list A, wrt system L.
%
system([],L,[]).
system([Ar|T],L,[Rate|Tail]) :-
                             rate(Ar,L,Rate),
                             system(T,L,Tail).
%
% predicate rate(Ar,L,Rate):  Ar is a substance name,  L  is a list of
% reactions among substances and Rate is the rate d[Ar]/dt wrt system L.
% Each element of L is of the form [ T,C,K ]: T is the reaction template
% list made of two lists, reactants names and products names; C is the
% list of the concentrations of reactants and K is the reaction constant.
%
rate(Ar,[],0).

% part of the rate of Ar, when Ar is among the reactants of a reaction.
%
rate(Ar,[ [ [R,P],[R1|Rt],K] |L ],Rate) :-
                             K > 0, R1 >= 0,
                             Rate = -K*R1*Rest + Tr,
                             member(Ar,R),
                             product(Rt,Rest),
                             rate(Ar,L,Tr).
%
% part of the rate of Ar, when Ar is among the products of a reaction.
%
rate(Ar,[ [ [R,P],[R1|Rt],K ]|L ],Rate) :-
                             K > 0, P1 >= 0,
                             Rate = K*R1*Rest + Tr,
                             member(Ar,P),
                             product(Rt,Rest),
                             rate(Ar,L,Tr).
%
% part of the rate of Ar, when Ar does not appear in the reaction.
%
rate(Ar,[ [ L1,Rs,K ]|L ],Rate) :-
                             rate(Ar,L,Rate).

% utility predicates
%
product([],1).
product([H|T],H*Pt) :- product(T,Pt).

member(E,[E|T]).
member(E,[H,T]) :- member(E,T).
```

Fig. 6. An example program in the RISC-CLP($\mathcal{R}eal$) language.

A similar system of acid-base reactions involves the "Acetate buffer", described by the following equilibrium.

$$HOAc \xrightarrow{k_1} H^+ + OAc^-$$
$$H^+ + OAc^- \xrightarrow{k_2} HOAc$$

Buffer solutions are used to maintain constant pH in a system: they contain a base that reacts if acids are added and, conversely, an acid that reacts if bases are added.

The "buffering" effect that characterizes a buffer solution can be illustrated with the following example. First we compute the pH in a buffered solution for the initial concentrations of $[HOAc]_0 = [OAc]_0 = 1\ moles \cdot liter^{-1}$ with the addition of no extra acid ($X = 0$).

```
?-  HOAc = HOAc0 + X, OAc = OAc0 - X, K2 * 18 = K1 * 10^6,
    X = 0, HOAc0 = 1, OAc0 = 1,
    system([hoac,h,oac], [ [ [[hoac],[h,oac]],[HOAc],K1],
                           [ [[h,oac],[hoac]], [H,OAc],K2]],
           [0,0,0]) {H}.
```

We obtain a pH of 4.75:

```
H = 1.8e-05
```

Normally, the addition of a large amount of acid from an outside source would change the pH dramatically. But in a buffered solution, even the addition of a relatively large amount of strong acid, changes the pH only slightly. Below we add .1 $moles \cdot liter^{-1}$ to the solution and ask RISC-CLP($\mathcal{R}eal$) about the new value of the pH.

```
?- HOAc = HOAc0 + X, OAc = OAc0 - X, K2 * 18 = K1 * 10^6,
   X = 1/10,  HOAc0 = 1, OAc0 = 1,
   system([hoac,h,oac], [ [ [[hoac],[h,oac]],[HOAc],K1],
                          [ [[h,oac],[hoac]], [H,OAc],K2]],
          [0,0,0]) {H}.
```

Surprisingly, we see that the pH changes only to 4.65:

```
H = 2.2e-05
```

We can also ask the reverse question: how much acid should be added to decrease the pH to 4.

```
?- HOAc = HOAc0 + X, OAc = OAc0 - X, K2 * 18 = K1 * 10^6,
   H = 1/10^4, HOAc0 = 1, OAc0 = 1,
   system([hoac,h,oac], [ [ [[hoac],[h,oac]],[HOAc],K1],
                          [ [[h,oac],[hoac]], [H,OAc],K2]],
          [0,0,0]).
```

The answer is striking, we must add a lot of acid: .69 $moles \cdot liter^{-1}$.

```
X = 0.694915
```

We have tried to solve all the above queries by using CLP($\mathcal{R}$), but because they require solving non-linear equations, CLP($\mathcal{R}$) cannot give the complete answer. Non-linearity arises very naturally here, in the context of chemical systems, but also in the majority of "real world" problems.

5 Conclusions

In this paper we have presented the work that we have been doing in extending the domain of computation of RISC-CLP from $\mathcal{R}eal$ to $\mathcal{T}rees\mathcal{R}eal$. The motivation for supporting the more general domain of trees with real numbers is that the possibility of integrating various distinct decidable domains enhances declarativeness.

Solving systems of constraints over the reals is a very difficult task and a lot of work has still to be done in order to attain efficiency. The number of free variables is a critical measure of the complexity of deciding the polynomial system of equations [11] and therefore, it should be minimized. One examples are the variables generated by the renaming mechanism because they do not occur, in general, in the set of already solved constraints as remarked in [17]. This suggests that one could try to eliminate them without resorting to the expensive solver. A more detailed study of the problem of elimination of variables from polynomial systems of equations and inequalities has yet to be done. A first step in this direction is represented by [13].

The crucial role of a notion of canonical form for a system non-linear constraints became also clear. Work has been pursued in these directions by developers of CLP($\mathcal{R}$), who have been mainly concerned about an efficient treatment of linear systems of constraints. All the above problems are still open for the more general setting of polynomial systems which we intend to pursue further.

Acknowledgments
I sincerely thank my advisor Hoon Hong for discussions, criticism and for his implementation of the partial CAD method, without which this work could not have been done.

References

1. A. Aggoun, M. Dincbas, A. Herold, H. Simonis, and P. Van Hentenryck. The chip system. Technical Report TR-LP-24, European Computer Industry Research Centre (ECRC), Munich, Germany, June 1987.
2. B. Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal.* PhD thesis, Universitat Innsbruck, Institut fur Mathematik, 1965. German.

3. B. Buchberger. Groebner Bases: An Algorithmic Method in Polynomial Ideal Theory. In N. K. Bose, editor, *Recent Trends in Multidimensional Systems Theory*, chapter 6. D. Riedel Publ. Comp., 1985.
4. G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Lecture Notes In Computer Science*, pages 134–183. Springer-Verlag, Berlin, 1975. Vol. 33.
5. G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, September 1991.
6. A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, July 1990.
7. Alain Colmerauer. Final specifications for PROLOG-III. Technical Report P1219(1106), ESPRIT, February 1988.
8. William Havens, Susan Sidebottom, Greg Sidebottom, John Jones, and Russel Ovans. Echidna: a constraint logic programming shell. In *Proceedings of the 1992 Pacific Rim International Conference on Artificial Intelligence*, volume I, pages 165–171, Seoul, Korea, 1992.
9. N. Heintze, J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. *The CLP(R) Programmer's Manual Version 1.1*, November 1991.
10. H. Hong. *Improvements in CAD–based Quantifier Elimination.* PhD thesis, The Ohio State University, 1990.
11. H. Hong. Comparison of Several Decision Algorithms for the Existential Theory of the Reals. Technical Report 91–41.0, Research Institute for Symbolic Computation, Johannes Kepler University A-4040 Linz, Austria, 1991. Submitted to *Journal of Symbolic Computation.*
12. H. Hong. Non-linear real Constraints in Constraint Logic Programming. In *International Conference on Algebraic and Logic Programming (LNCS 632)*, pages 201–212, 1992.
13. Hoon Hong. Quantifier Elimination for Formulas Constrainted by Quadratic Equations via Slope Resultants. *The Computer Journal, Special issue on Quantifier Elimination*, 1993.
14. J. Jaffar and S. Michaylov. Methodology and implementation of a CLP system. In J.-L. Lassez, editor, Proceedings 4^{th} ICLP, pages 196–218, Cambridge, MA, May 1987. The MIT Press.
15. J. Jaffar, S. Michaylov, and Roland H. C. Yap. A methodology for managing hard constraints in CLP systems. In *Proceedings of the ACM-SIGPLAN Conference on Programming Language Design and Implementation*, pages 306–316, Toronto, June 1991.
16. Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM, January 1987.
17. Joxan Jaffar, Spiro Michaylov, Peter Stuckey, and Roland H. C. Yap. The CLP(R) Language and System. Research Report RC 16292, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, November 1990.
18. Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in honour of Alan Robinson*, chapter 8, pages 257–321. MIT Press, Cambridge, (MA) USA, 1991.
19. D.A. Kohler. *Projection of Convex Polyhedral Sets.* PhD thesis, University of California, Berkeley, 1967.

20. E. Monfroy. Specification of geometrical constraints. Technical Report ECRC-92-31, ECRC, Munich, Germany, 1992. presented at AAGR 92, Linz, Austria.
21. William Older and Andrè Vellino. Constraint arithmetic on real intervals. In Frèdèric Benhamou and Alain Colmerauer, editors, *Constraint Logic Programming: Selected Research*. The MIT Press, Cambridge, 1992.
22. K. Sakai and A. Aiba. CAL: A Theoretical Background of Constraint Logic Programming and its Applications. *Journal of Symbolic Computation*, 8:589–603, 1989.
23. A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. Univ. of California Press, Berkeley, second edition, 1951.
24. Clifford Walinsky. Clp(Σ^*): Constraint logic programming with regular sets. In *Logic Programming: Proceedings* 6^{th} *International Conference, Jerusalem, Israel*, pages 181–196. MIT Press, June 1989.

This article was processed using the LaTeX macro package with LLNCS style