DEPARTAMENTO DE LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

LCC ITI 98/16

INFORME TÉCNICO
INVESTIGACIÓN

A GENERIC EXECUTION MODEL CLP(X)$_<$

Antonio J. Fernández[1]
Lenguajes y Ciencias de la Computación,
Universidad de Málaga,
29071 Teatinos,
Málaga, Spain
e-mail: afdez@lcc.uma.es
Fax-number: +34-5-2131397

Patricia M. Hill
School of Computing,
University of Leeds,
Leeds, LS2 9JT,
England
e-mail: hill@comp.leeds.ac.uk

Abstract

It is well known that the expressive power and efficiency of Constraint Logic Programming (CLP) systems is reduced by the strong partitioning of the structures in which constraints can be expressed. This means constraints defined on a particular domain must be solved by a specific reasoning method with its associated solver that is not appropriate for constraints defined on other domains.

In this paper, we describe a common unified framework for processing constraints on any computation domain in which an ordering has been defined. This framework is based on a single generic constraint and unique inference rule. As usual on CLP our system can make use of built-in domains and constraints defined on them but also allows the definition of user-defined domains as well as of user-defined constraints on them. Moreover, the system is independent of the underlying type and cardinality of the domain which means a form of genericity for the constraint solving. This means a higher flexibility with respect to the existing CLP systems. We demonstrate, through examples, the expressiveness of our framework compared to traditional CLP systems.

keywords: finite domains, interval reasoning, domain reasoning, constraint solving, interval narrowing, solver, constraint store.

Contents

List of Figures

List of Tables

1 Introduction and Motivation

Constraint Logic Programming (CLP) systems [JAF94] have been applied to many different domains each with distinct algebraic structures. The type of the domain gives place to different instances of the CLP scheme such as CLP(FD) (CLP on finite ranges of integers) [COD96a], CLP($\Re$) (CLP on real domain) [REF96], CLP(Sets) (CLP on finite sets of elements) [GER97] or CLP(Bool) (CLP on boolean domain) [COD96b]. However, problems suitable for solving with constraints often have a natural formulation which use more than one domain or a domain other than the built-in domains. Existing constraint systems (with some exceptions such as the CHR language [FRÜ94]), only support specific domains and assume a strong partitioning of the constraints between these domains. Thus, current constraint systems lack expressiveness and the final code often bears no resemblance to the initial problem statement. Moreover, the constraints defined on a domain must be solved on very different solvers that have been implemented particularly for a specific domain (see left-hand diagram in Figure 1).

In current systems, many of the constraints have the control fixed by the system. These *black box* constraints provide very efficient tools for common constraint applications but lack adaptability when used for non-standard problems. To overcome this lack of flexibility, some constraint systems now provide *glass box* constraints [FRÜ94, COD96a]. These allow new constraints to be defined by the user, although, in most cases, these are still restricted to particular predefined domains such as the integers.

However there are two practical problems with the current approach. The constraint solving process is usually "hard-wired" using a low-level language so that it is difficult (sometimes, impossible) to build new solvers over new domains. Moreover, the cardinality of the domain determines the constraint solving procedure so that existing CLP systems have distinct constraint solving methods for the finite and the infinite domains.

This paper has two main contributions. First, it describes a CLP system with a generic constraint solver that can perform constraint propagation for any ordered domain, finite or infinite (as illustrated in the right-hand diagram of Figure 1). Note that this means that the system is valid as unified framework for the usual domains used in CLP including the integer, Boolean and real domains.

Secondly, it opens up the *glass box* approach given by the CLP finite domain constraint system [COD96a] to any ordered domain, finite or infinite, user-defined or system-defined. With this more generic approach, the user

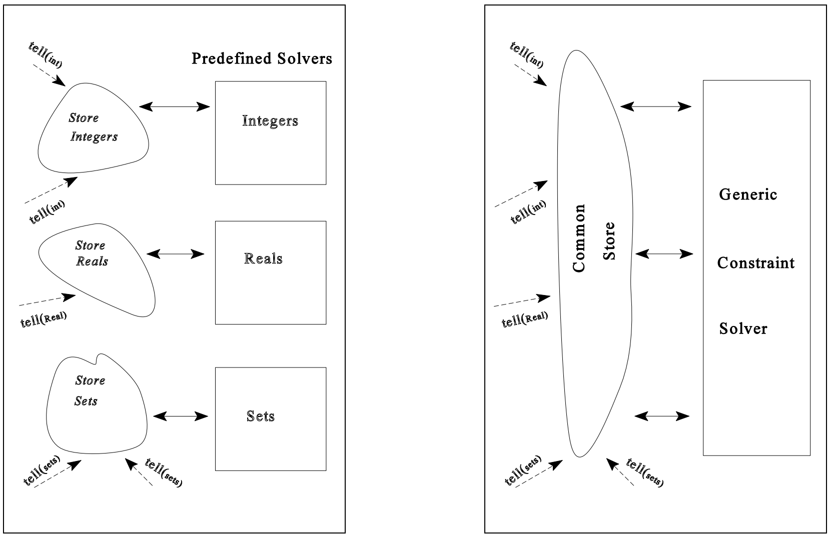

Figure 1: Actual state of CLP. A generic solver approach

can define new domains, new constraints on these domains and, for each constraint, the intended propagation behaviour of the domain values. Figure 2 shows how this new *glass box* approach grows in two different directions (vertically, by defining planes which means the user defines new domains, and/or horizontally on any plane, by defining new constraints on each domain).

We call our constraint system CLP(X)$_<$ since is valid for any domain X in which an ordering relation $<$ (not necessarily total) is defined. Since the approach is independent of the cardinality and base type of the domain and presents a form of generic constraint solving, definitively, this means that the users:

- can define constraints on built-in domains (the typical *glass box* approach);
- can define their own domains and constraints on them for particular applications (our new *glass box* approach) when necessary;
- can specialise the constraint solving for particular applications since they can formulate the problem in a more natural way.

The CLP(X)$_<$ constraint system

- has a single primitive constraint that can be used with any ordered domain;

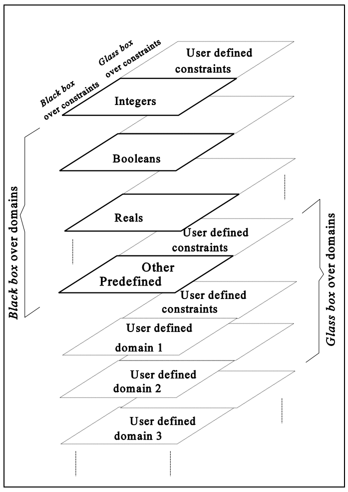

Figure 2: A two dimensional *glass box* approach: over constraints and over domains

- uses a novel propagation mechanism for this constraint;
- supports certain special extensions required by small domains such as the Booleans and the infinite domains;
- allows the user the option of controlling the precision of the results.

The primitive constraint is then used to define more complex constraints that are specialised for the particular domains of interest.

Of course, it is vital that any new constraint system is not only expressive but also have an efficiency comparable with existing constraint systems. It is known that although most of the solvers are based on extensions of the consistency techniques in [MAC77], the detailed reasoning varies for each specific domain. There are two basic forms of reasoning: domain reasoning and interval reasoning. Interval reasoning (which prunes just domain bounds) is more efficient that domain reasoning (which prunes the values in the domain). The constraint solver described here is based on a combination of two interval reasoning techniques: interval variations and indexicals ranges (similar to those used on finite domains [COD96a]) to update and (re)activate constraints in terms of other constraints.

The paper is structured as follows. Section 2 recalls algebraic concepts used in the paper. The computation domain and the primitive generic constraint are described in Section 3. Section 4 describes the unique inference rule used and defines the operational semantics of the system. In Section 5, three problems on very different domains are solved to show the expressiveness and flexibility of our system and Section 6 compares our system with related work. The paper ends with some considerations about further work and the conclusions.

2 Preliminaries

This section recalls the algebraic concepts [dBUR88] and defines notation used in this paper.

Definition 1 *Let C be a set. The binary relation $\preceq$ on C is a* weak order *if it is reflexive, antisymmetric and transitive. The binary relation $\prec$ on C is a* strict order *if it is transitive and irreflexive (hence necessarily asymmetric). Note that if equality and inequality are defined for the elements of a set C ordered by $\prec$ and $\preceq$, then*

$$c \preceq c' \wedge c \neq c' \Rightarrow c \prec c',$$
$$c \prec c' \vee c = c' \Rightarrow c \preceq c'.$$

Definition 2 *Let C be a set with elements c and c' and a (weak or strict) ordering relation R. Then c and c' are* comparable by R, *when either cRc' or $c'Rc$. We say a set C is* totally ordered *if every pair of elements of C is comparable. We say the ordering R in the set C is* partial, *if there are at least two elements which are not comparable.*

Evidently any *order* relation for a set C is an *order* for any subset of C. We write (C, R) to indicate C is a set *ordered* by the relation R.

Definition 3 *Given $(C, \preceq)$, an element s in C is a* lower (upper) bound *of a totally ordered subset E of C iff $s \preceq x$ ($s \succeq x$) for every x in E. If the set of lower (upper) bounds of C has a greatest (least) element, then that element is called the* infinum (supremum) *of C. The* infinum *of C is also called the 'greatest lower bound' of C (denoted by $glb(C)$); the* supremum *of C is also called the 'least upper bound' of C (denoted by $lub(C)$).*

Definition 4 *Suppose the set C is weak ordered by $\preceq$, $c, c' \in C$ and $c \preceq c'$. Then c is called a* predecessor *of c' and c' a* successor *of c. c is called the* immediate predecessor *of c' if either $c = c' = glb(C)$ or $c' \neq glb(C)$ and $c = lub\{c''|c'' \prec c\}$. Similarly, c' is called the* immediate successor *of c if either $c = c' = lub(C)$ or $c \neq lub(C)$ and $c' = glb\{c''|c'' \succ c'\}$.*

Definition 5 *Given* $(C, \preceq)$ *and any two comparable elements* a *and* b *in* C*, then each of the sets:*

$$\begin{aligned}[a,b] &= \{c \in C \mid a \preceq c \preceq b\},\\ [a,b) &= \{c \in C \mid a \preceq c \prec b\},\\ (a,b] &= \{c \in C \mid a \prec c \preceq b\},\\ (a,b) &= \{c \in C \mid a \prec c \prec b\}.\end{aligned}$$

is called an Interval with origin a *and* end b*. An* Interval *said to be consistent when is different from the empty set. Otherwise the interval is inconsistent. The notation* $\{a,b\}$ *denotes an interval of any of the above forms.*

For example, using the domain of integers, $[1,1],[1,2),(1,2],(1,3)$ are all consistent intervals, but $[2,1],(1,1],[1,1),(1,2)$ are inconsistent.

Our constraint system is built on top of a pure logic programming language and we base our notation on the Prolog syntax.

We use $\#C$ to denote the cardinality of a set C.

3 Computation Domain

This section presents the domain constraints on which our system is based. We explain how high-level constraints can be constructed from them.

3.1 Domains

A computation (or type) domain $D(\tau)$ interprets the set of values of a type τ. For example, integer, real, Booleans are common domains that are often provided by CLP systems. The user can also define domains by means of declarations:

$$\textbf{data } D(\tau) = \textit{ data constructor}_1 | \ldots | \textit{data constructor}_m.$$

For example:

$$\begin{aligned}&\textbf{data } \textit{Letter} = a \mid ... \mid z.\\ &\textbf{data } \textit{Natural} = \textit{zero} \mid \textit{suc}(\textit{Natural}).\end{aligned}$$

The constraint solving system described here is valid for any domain $D(\tau)$ in which a *strict order* relation $\prec$ and the (in)equality over elements of type τ has been defined. Thus, the type τ is identified with the computation domain $D(\tau)$ over which the constraints are solved.

Let τ be a domain partially ordered with the strict order relation $\prec$. We define $\bot_\tau$ to be $glb(\tau)$, if it exists, and a new constant not in τ, otherwise. Similarly, we define $\top_\tau$ to be $lub(\tau)$, if it exists, and a new constant not in τ, otherwise. From now on, $\bot_\tau$ and $\top_\tau$ are assumed to be elements of τ, that is, $\tau = \tau \cup \{\bot_\tau, \top_\tau\}$.

If $\top_\tau$ or $\bot_\tau$ were added to τ as new constants then any operation involving exactly one of them must return the same bound. For example, $\bot_\tau \; op_\tau \; C = \bot_\tau$ where C::τ and op_τ::$\tau \to \tau \to \tau$. The result when an operation involves both bounds is undefined. This situation can be changed by the user.

3.2 Interval constraints

A range $R(\tau)$ is a set of possible values in the domain $D(\tau)$ over which the constraint can be satisfied.

Definition 6 *The* range $R_X(\tau)$ for a variable X *is an interval* $\{a, b\}$*, where* $a, b :: \tau$ *are terms* (t_τ) *defined as:*

$$t_\tau ::= ct \mid I_\tau(Y) \mid t_\tau \;\; op_\tau \;\; t_\tau$$

where the constant $ct \in \tau$*,* Y *is a variable distinct from* X *but of the same type* τ*,* $I_\tau(Y)$ *is an indexical term, and* op_τ::$\tau \to \tau \to \tau$ *is a functional term called a* domain operator*. Note that* op_τ *can be predefined or user-defined. The* indexical term $I_\tau(Y)$ *is an expression of the form* $min(Y)$ *or* $max(Y)$ *where* $min, max :: \wp(\tau) \to \tau$ *are defined:*

$$min(Y) = glb(R_Y(\tau)),$$
$$max(Y) = lub(R_Y(\tau)).$$

Y *which is distinct from* X *is called the* indexed variable*.*

Note that an indexical term as defined here is similar to the indexicals provided by *CLP* finite domain languages [COD96a]).

An interval constraint assigns a constraint variable to a range.

Definition 7 *An* interval constraint *for a variable* X *of type* τ *is a unary constraint* $X \in R_X(\tau)$*. When the interval* $R_X(\tau)$ *is* $[\bot_\tau, \top_\tau]$ *it contains every element of type* τ*, it is called a* type constraint *and we use the notation* X::$'\tau$*.* X *is called the* constrained variable*. An interval constraint is* consistent *(resp.,* inconsistent*) if the range is consistent (resp., inconsistent). A* simple *interval constraint is an interval constraint that does not use indexicals.* $X = ct$ *denotes the interval constraint* $X \in [ct, ct]$*.*

To illustrate these definitions: $Y, X{::}'Integer$, $B{::}'Bool$, $W, T{::}'Real$, $C{::}'Letter$ and $N{::}'Natural$ are examples of type constraints; $Y \in [1, 4)$, $X \in [min(Y)\text{-}+ 3, max(Y) + 3]$, $B \in [True, True]$, $T \in (1.21 * min(W), 4.56)$, $C \in [a, z)$ and $N \in [zero, suc(suc(suc(suc(zero))))]$ are examples of interval constraints. where $+, *{::}\tau \to \tau \to \tau$ are the domain operators for the $Integer$ and $Real$ domains.

Definition 8 *A* constraint store *is a finite set of interval constraints [JAF94]. The constraint store is in a* stable form *if, for each constrained variable, there is at most one interval constraint which doesn't use indexicals.*

3.3 High-level constraints

High-level constraints are predicates defined solely in terms of interval constraints. For example, generic disequality constraints $<', \leq'$for any domain τ can be defined as follows.

$$
\begin{array}{l}
<', \leq' :: \tau \to \tau \to Bool. \\
X <' Y \;:-\; X \in [\bot_\tau, max(Y)),\; Y \in (min(X), \top_\tau]. \\
X \leq' Y \;:-\; X \in [\bot_\tau, max(Y)],\; Y \in [min(X), \top_\tau].
\end{array}
$$

A generic (in)equality predicate $=', \neq'$can be also defined:

$$
\begin{array}{l}
=', \neq' :: \tau \to \tau \to Bool. \\
X =' Y :- X \in [min(Y), max(Y)], Y \in [min(X), max(X)]. \\
X \neq' Y :- X \in [\bot_\tau, max(Y)), Y \in (min(X), \top_\tau]; \\
\qquad\qquad X \in (min(Y), \top_\tau], Y \in [\bot_\tau, max(X)).
\end{array}
$$

As usual, ; indicates a choice point. The apostrophe at the end of an operator symbol indicates that the operator is a high level constraint. A high level constraint can be defined as a function of the basic domain operators. For example, a generic sum constraint $+'$a domain τ can be defined as follows:

$$
\begin{array}{l}
+' :: \tau \to \tau \to \tau. \\
+'(X, Y, Z) :- X \in [min(Z) - min(Y), max(Z) - max(Y)], \\
\qquad\qquad\qquad Y \in [min(Z) - min(X), max(Z) - max(X)], \\
\qquad\qquad\qquad Z \in [min(X) + min(Y), max(X) + max(Y)].
\end{array}
$$

where $+, - :: \tau \to \tau \to \tau$ are the basic arithmetic domain operators. For simplicity, we use an infix notation for high level constraints that are also operators. For example, $+'(X, Y, Z)$ will be denoted as $X +' Y$ and an intermediate constraint variable Z for the value of the sum will be created, that is, $Z \in [min(X)+min(Y), max(X)+max(Y)]$.

Any finite set of constraints is called a *system of constraints*. For example, $\{X \preceq' Y, X \in [1.4, 3.6]\}$ is a system of constraints.

3.4 Meaning of a program

A program is a triple *(L,T,R)* where L defines the computation domains, T defines the high level constraints and R is a set of rules. L, T as well as R may consist of user and system components. Each rule has the form

$$p(t_1, \ldots, t_n) :- a_1 \ldots a_n, S.$$

where p is a predicate symbol (distinct from those used in T to define high level constraints), the t_i are terms, $a_1, \ldots, a_m$ are atoms, and S is a possibly empty system of constraints. The meaning of such a rule is as follows: "provided S is satisfied, the atom $p(t_1, \ldots, t_n)$ is admissible if, for each $i = 1, \ldots, n$, a_i is an admissible atom.

4 Execution model

A goal or body of a clause consists of a sequence atoms and a set of constraints. The atoms are resolved by the inference machine of the logic language whereas the constraints are processed by the constraint solver. This is a single generic solver that is independent of the constraint domains. Interval constraints of the form $n \in R(\tau)$ where $n :: \tau$ and $R(\tau)$ does not contain indexicals are considered as tests that result directly in failure or success. Interval constraints of the form $n \in R(\tau)$ where $n :: \tau$ and $R(\tau)$ contains indexicals are delayed to test and not sent to the solver. Note that if $n :: \tau$, $min(n) = n$ and $max(n) = n$. For example, in the integer domain a call such as $1 \leq' Y$ only generates an interval constraint for the solver: $Y \in [1, \top_{Int}]$[2] since constraint $1 \in [\bot_{Int}, max(Y)]$ is not sent to the solver.

The execution model of the constraint solver is based on narrowing the constraint intervals for the constrained variables by removing values from the domain that do not satisfy the constraint. The techniques we use are independent of the particular domain. The idea consists of first constraining each

[2] Note that $min(1) = 1$.

variable X of a type τ to have values in the range $[\bot_\tau, \top_\tau]$ and, secondly, removing values from this range that can never be part of any feasible solution.

The *indexicals* in an interval constraint specify how the constrained variable's interval must be updated when the domains of other variables changes. A constraint on a variable X using an indexical on another variable Y must be checked each time the domain of Y is updated. These constraints remain in the constraint store to propagate future modifications on the *indexed variables.* Any change in an *indexed variable* activates the constraint narrowing procedure for the constrained variable. This process is called *constraint propagation.*

4.1 Inference rule

There is one inference rule called *interval narrowing* ($\cap_\tau$) which takes two interval constraints which share the same *constrained variable* and returns a range of possible values. The rule is formulated as follows:

$$X \in \{_1 a, b\}_1,\ X \in \{_2 c, d\}_2\ \vdash\ X \in \{_3 maximum(a,c), minimum(b,d)\}_3 \qquad \cap_\tau$$

Given the ordering ($<$ [$<$) $<$] for the brackets, the following tables give the values for $\{_3$ and $\}_3$ in the inference rule $\cap_\tau$ (for example, $\{_3 \equiv \{_2$ iff $a \prec c$).

	$a \prec c$	$c \prec a$	$a = c$	$a \not\prec c$
$\{_3$	$\{_2$	$\{_1$	minimum of $(\{_1, \{_2)$	[

	$b \prec d$	$d \prec b$	$b = d$	$b \not\prec d$
$\}_3$	$\}_1$	$\}_2$	minimum of $(\}_1, \}_2)$	]

where, $a \not\prec b$ means that neither $a \prec b$ nor $b \preceq a$ is defined.

The functions *minimum* and *maximum* are the core of the interval narrowing mechanism. They are defined as follows:

$$\begin{aligned}
&minimum, maximum :: \tau \to \tau \to \tau.\\
&minimum(a,b) = \ a\ if\ a \prec b,\\
&\qquad\qquad\qquad\quad b\ if\ b \preceq a,\\
&\qquad\qquad\qquad\quad a\ op_\tau\ b, otherwise\\
&maximum(a,b) = \ b\ if\ a \prec b,\\
&\qquad\qquad\qquad\quad a\ if\ b \preceq a,\\
&\qquad\qquad\qquad\quad a\ op'_\tau\ b, otherwise
\end{aligned}$$

where $op_\tau, op'_\tau :: \tau \to \tau \to \tau$ are user-defined functions. They are only applied when the ordering on τ is partial and thus neither $a \prec b$ nor $b \preceq a$ is defined. This allows the user control of the propagation mechanism. The following conditions are imposed on the user-defined functions $op_\mathcal{T}$ and $op'_\mathcal{T}$:

$$(A\ op_\mathcal{T}\ B) \preceq A\ and\ (A\ op_\mathcal{T}\ B) \preceq B \qquad \mathbf{c1}$$
$$(A\ op'_\mathcal{T}\ B) \succeq A\ and\ (A\ op'_\mathcal{T}\ B) \succeq\ B \qquad \mathbf{c2}$$

An example of their use is shown in Subsection 5.4

4.1.1 Properties of the inference rule

Let $X \in d_1,\ X \in d_2\ \vdash\ X \in d_3$, where $d_1 = \{_1 a, b\}_1, d_2 = \{_2 c, d\}_2$ and $d_3 = \{_3 e, f\}_3$ be consistent intervals so that

(1) $a \preceq b, c \preceq d$ and $e \preceq f$ and
(2) $\{_3 \preceq \{_1,\ \{_3 \preceq \{_2,\ \}_3 \preceq \}_1$ and $\}_3 \preceq \}_2$.

The main properties of the inference rule $\cap_\mathcal{T}$ are:

(1) *Contractance*: d_3 is a subset of d_1 and d_2.

Proof: By $\cap_\mathcal{T}$, $e = maximum(a, c)$ and $f = minimum(b, d)$. Definitions of *maximum* and *minimum* and properties $c1$ and $c2$ guarantee that $a \preceq e$, $c \preceq e$, $f \preceq b$ and $f \preceq d$. By (1) this infers that $a \preceq e \preceq f \preceq b$ (that is, $[e, f] \subseteq [a, b]$) and $c \preceq e \preceq f \preceq d$ ($[e, f] \subseteq [c, d]$). Then, by (2) it is inferred that $\{_3 e, f\}_3 \subseteq \{_1 a, b\}_1$ and $\{_3 e, f\}_3 \subseteq \{_2 c, d\}_2$

(2) *Correctness*: Only values which do not belong to $d_1 \cap d_2$, and therefore can't be part of any feasible solution, are removed.
If $s \in d_1$ and $s \in d_2$, then $s \in d_3$.

Proof: Let s be a solution, then $s \in d_1$ and $s \in d_2$. By the *minimum* and *maximum* definitions and properties $c1$ and $c2$ $s \in \{_3 maximum(a, c), \preceq minimum(b, d)\}_3$ which means $s \in d_3$.

(3) *Idempotence*: The final range d_3 has to be computed once.

$$X \in d_1,\ X \in d_3\ \vdash\ X \in d_3;$$
$$X \in d_3,\ X \in d_2\ \vdash\ X \in d_3.$$

Proof: Trivial.

(4) *Monotonicity*: The inclusion is preserved. Let $X \in d_4,\ X \in d_5 \vdash X \in d_6$ where $d_4 = \{_4 a', b'\}_4, d_5 = \{_5 c', d'\}_5$ and $d_6 = \{_6 e', f'\}_6$ are consistent intervals; if $d_1 \cap d_2 \subseteq d_4 \cap d_5$ then $d_3 \subseteq d_6$.

Proof: Clearly, $d_1 \cap d_2 = \{_3 maximum(a,c), minimum(b,d)\}_3$ and $d_4 \cap d_5 = \{_6 maximum(a',c'), minimum(b',d')\}_6$. Thus, $d_1 \cap d_2 \subseteq d_4 \cap d_5 \Rightarrow (maximum(a',c') \preceq maximum(a,c) \preceq minimum(b,d) \preceq minimum(b',d'), \{_3 \preceq \{_6, \ \}_3 \preceq \}_6)$ which means $d_3 \subseteq d_6$.

4.2 Operational semantics

The interval narrowing rule $\cap_\tau$ is defined for a pair of interval constraints. The operational semantics described by the algorithm later in this section refines and infers the consistency of a system of constraints.

The initial domain of a constraint variable with type constraint $X ::' \tau$ is defined by the corresponding constraint $X \in [\bot_\tau, \top_\tau]$. This is reduced as a consequence of its interaction with any other interval constraints defined on X.

Let I be the constraint system to be solved. Each constraint $c \in I$ is represented by a triple $(n_c, \vec{s}, \vec{d_s})$ where n_c identifies the constraint, $\vec{s}$ is the set of constrained variables[3] over which the constraint c is defined and $\vec{d_s}$ is the set of interval constraints with respect to $\vec{s}$. For example, a constraint $S' \leq' S$ is represented as the triple $(\leq', \{S', S\}, \{(S' \in [\bot_\tau, max(S)], S \in [min(S'), \top_\tau]\})$. Interval and type constraints are identified by $\in$ and $::'$ respectively. Let C denote the set of triples corresponding to I. In this section, these triples are called "constraints".

The *constraint store* (recall that this is a set of interval constraints) is denoted by S. This is partitioned into: S_d which contains all the simple interval constraints in S and S_I which contains the non-simple interval constraints (that is, interval constraints that contain indexicals). If X is a constraint variable, we write d_X to denote a constraint for X in S_d and $d_X^{I(Y)}$ for a constraint for X in S_I. For simplicity, only one indexed variable Y is considered. The algorithm, with minor changes, can be extended to work with indexicals $d_X^{I(\vec{v})}$ over a set of indexed variables $\vec{v}$. A *state* is a triple $\langle C, S_d, S_I \rangle$ where, in S_d, there is at most one simple interval constraint for each variable.

The operational semantics is based on the non-deterministic transition rule defined in the procedure below which is called **constraint solving procedure**. It takes as input a state $\langle C, S_d, S_I \rangle$ and a selected constraint $(n_c, \vec{s}, \vec{d_s})$ in C and determines if constraints in $\vec{d_s}$ are locally consistent wrt the constraints in S_d. Constraints with indexicals are added to S_I and remain in S_I to produce further propagation.

The initial state of the transition system is specified by $\langle C, \emptyset, \emptyset \rangle$ where

[3] Note thus each variable s is constrained by a interval constraint d_s.

all the constraints in C need to be checked. The final state is either $fail$ or $\langle \emptyset, S'_d, S'_I \rangle$ where S'_d is in a *stable form* and no more constraint propagation can be done. S'_d contains the solution intervals.

Constraint solving procedure

Initialise S_d and S_I to the empty set $\emptyset$
while C is not empty **do**
  select and remove a constraint $(n_c, \vec{s}, \vec{d_s})$ from C
  for each constrained variable X in $\vec{s}$ **do**
    case of
    (1) $d_X^{I(Y)} \in \vec{d_s}$, **then** (* interval constraint with indexicals *)
      $S_I := S_I \cup d_X^{I(Y)}$
      if any $d_Y \in S_d$ **then** (* constraint propagation *)
        apply values $min(Y), max(Y)$ of d_y on $d_X^{I(Y)}$ which returns d'''_X
        if $d'''_X \notin S_d$ **then** $C := C \cup \{(\in, \{X\}, \{d'''_X\})\}$ **endif**
      endif
    (2) $d_X \in \vec{d_s}$, **then** (* simple interval constraint *)
      if $\nexists d'_X \in S_d$ **then** $S_d := S_d \cup d_X$; $d''_X := d_X$;
      else
        select and remove d'_X from S_d
        apply inference rule $\cap_\tau$ on d_X and d'_X which returns d''_X
        if d''_X is inconsistent **then** exit with failure **endif**
          (this will cause backtracking)
        $S_d := S_d \cup d''_X$
      endif
      if $(\nexists d'_X \in S_d)$ **or** $(d'_X \neq d''_X)$ **then** (* constraint propagation *)
        for each $d_Y^{I(X)} \in S_I$ **do**
          apply values $min(X), max(X)$ of d''_X on $d_Y^{I(X)}$ which returns d_Y
          if $d_Y \notin S_d$ **then** $C := C \cup \{(\in, \{Y\}, \{d_Y\})\}$ **endif**
        endfor
      endif
    endcase
  endfor
endwhile

The **constraint solving procedure** is adapted from the arc consistency algorithms in [MAC77] although with significant differences. The main difference is in the constraint propagation which is based on the use of indexicals and in the inference rule applied.

Termination. As will be shown in Theorem 1, the **constraint solving procedure** terminates when the computation domain is finite since these domains cannot be refined indefinitely. However, this is not necessarily the case if the domain is infinite since constraint propagation can add new constraints to C causing other constraints to propagate and infinite domains could be refined indefinitely. Note that this is not a problem in practice with well known domains such as the real domain since the precision of any real number is bound so that the domain is effectively finite. This is the argument given in [CLE87, LEE93] and [BEN97]. Our system allows (possibly infinite) user-defined domains. For this reason, we incorporate a generic predicate $Precision :: \tau \rightarrow \tau \rightarrow Bool$ which limits domain refinement. Predicate *Precision* defines the condition that the *glb* and the *lub* of a range must hold to get the refinement required. When this condition holds the range is not refined. Examples of definition of this predicate are shown below:

(1) On integer domain: $Precision(A, B) :- B–A \leq 0$
(2) On real domain: $Precision(A, B) :- B–A \leq 0.00000001$
(3) On $\Re^2$: $Precision(A, B) :- \ldots maximum\ distance\ between\ A\ and\ B \ldots.$

The only condition required by this predicate is that there must be at least elements $x, y :: \tau$ satisfying $x \leq y$ and $Precision(x, y)$.

Let $X \in \{c, d\}$ be a simple interval constraint defined on a domain D. If $Precision(c, d)$ holds, then the constraint is not added to C. Note that the predicate *Precision* allows the user direct control over the accuracy of the results[4]. To use the *Precision* predicate, the **constraint solving procedure** needs to be extended with an extra test. Line 20 in the procedure is thus reformulated as follows:

20 **if** (**Not** $Precision(glb(d_X), lub(d_X))$) **and** (($\nexists d'_X \in S_d$) **or** ($d'_X \neq d''_X$)) **then**

We call the new procedure obtained by replacing line 20 of the **constraint solving procedure** with the above line the **constraint solving algorithm** since the following theorem proves termination.

Theorem 1 *The* **constraint solving algorithm** *always terminates. The resulting store S_d is either inconsistent or stable.*

We present here just the proof outline.

Proof Outline: The algorithm terminates when an inconsistency is detected or when $\#C$ is zero. In the former case (see line 17) S_d is inconsistent and the algorithm clearly terminates.

If no inconsistency is detected, in each step of the **constraint solving algorithm** a constraint is removed from C (line 3). Then,

(a) If constraint propagation is never executed then, in $\#C$ steps, C will be empty and thus the algorithm will terminate. Note that S_d will remain in

[4][SID92] provides a similar approach but only over reals.

a stable form ($\cap_\tau$ and lines 13-19 assure that at most only one simple domain constraint on a variable remains in S_d).

(b) If constraint propagation is executed (lines 8-11 and 20-25), new triples (containing only simple interval constraints) are added to C. But constraint propagation cannot be executed indefinitely since the ranges of these simple domain constraints are refined (*contractance* property) when $\cap_\tau$ is applied (line 16) and the distance (measured in number of elements) between the bounds (*glb* and *lub*) of these ranges decreases. This assures that the condition defined by map $Precision/2$ can only succeed a finite number of times. Thus the condition in the reformulated line 20 will eventually succeed and hence, C will become empty after a finite number of steps. Thus the algorithm terminate.

Theorem 2 *If a solution exists, it can be derived (completely or incompletely)*[5] *from the* **constraint solving algorithm**.

Proof. This follows directly from the *correctness* property.

4.3 Some considerations on finite domains

A Constraint Satisfaction Problem (CSP) involving *constrained* variables on finite domains isn't always solvable by using the process described in preceding sections. For example, consider the Boolean domain, an ordering relation defined by $false < true$ and the constraint $X \in [false, true)$. This constraint provides enough information to know the value of X must be *false*, but this is not computed. For finite domains, this problem can be solved using an enumeration strategy[6] as is usually done for the existing finite domain constraint languages. However, suppose the domain is discrete and that the *immediate predecessor* $pre(K)$ and *immediate successor* $suc(K)$ of every value K in the domain can be computed. Then we can eliminate the $(,)$ brackets in favour of the $[,]$ ones using the two *range rules*:

$$\{a, K) \equiv \{a, pre(K)] \qquad \textbf{rleft}$$

$$(K, a\} \equiv [suc(K), A\} \qquad \textbf{rright}$$

which would be applied only when no more constraint propagation is possible. For the *glb* and *lub* elements, we define: (1) $pre(\bot_\tau) \equiv \bot_\tau$ and (2) $suc(\top_\tau) \equiv \top_\tau$. For our example on the Boolean domain, we have that $suc(false) = true$ and $pre(true) = false$ and by applying **rleft**, the constraint $X \in [false, true)$

[5] A solution is complete if all the values within the ranges of the *constrained* variables which are in the store S_d are valid solutions. An incomplete solution has potential solutions within the ranges.

[6] That is, each possible value is assigned in turn to the constrained variables in order to solve constraints.

is transformed to $X \in [false, false]$. Note that an inconsistency is detected in a range (a, b) when $pred(b) = a$ or $suc(a) = b$. An example is shown in Subsection 5.3.

5 Examples on different domains. Applications

In this section, by solving three examples over very different domains we demonstrate the expressiveness and flexibility of the system described.

5.1 Notation used in the examples

We follow the notation shown in the basic **constraint solving procedure** in Subsection 4.2:

- Triples: c_i.
- Simple interval constraints: d_X.
- Interval constraints using indexicals: d_X^Y.
- Constraint propagation: $d_X^Y \rightarrow c_i$.
  This means that the simple interval constraint d_Y belonging to the store S_d is propagated by d_X^Y in c_i (see lines 9-10 and 22-23 in the **constraint solving procedure**).
- Interval narrowing: $d_X^i, d_X^j \vdash d_X^h$.
  This means that d_X^i and d_X^j are reduced to d_X^h by applying the inference rule $\cap_\tau$.
- Step without change: NC.
  This means that a calculation without consequence was done. For example, $d_X^Y \rightarrow NC$ denotes that d_X^Y is propagated in a triple c_i without affecting C and S_d.

We discuss three problems in turn. For each one, the system of constraints for the goal is first translated to a system of triples (the starting point of the basic algorithm). For each problem there are two tables. The first table associates these triples and the interval constraints used in the algorithm with a label. In the second table, we show step by step, how the constraint solving process is executed (by following the basic algorithm shown in Subsection 4.2). The columns labelled as C,S_d and S_I show the current value of the state $\langle C, S_d, S_I \rangle$ (see Subsection 4.2). The first row indicates the initial state. In subsequent rows, an integer in column 1 corresponds to a complete inference step and means that a constraint has been selected and removed from C. A non-integer in column 1 indicates that the **for** loop of the **constraint solving algorithm** is being executed. The column labelled as $Ch.P.$ marks the steps in which a choice point is introduced by some disjunctive constraint. The column labelled as *Remarks* describes the process that was followed.

5.2 A problem on Reals

The first example illustrates the basic **constraint solving algorithm** in Subsection 4.2. Consider the constraint system $\{S{::}'Real, S \in [0,4], 1 \preceq' S, 3.75 \preceq' S\}$. This is represented by the triples: $\{c1, c2, c3, c4\}$ as defined in Table 1. Table 1 also gives the interval constraints used by Table 2 to show the constraint solving process.

Table 1: Triples and interval constraints in the example of $\Re$

Triples	Simple Interval constraints
$(c1)$ $(::', \{S\}, \{d_S^1\})$	(d_S^1) $S \in [\perp_{Real}, \top_{Real}]$
$(c2)$ $(\in, \{S\}, \{d_S^2\})$	(d_S^2) $S \in [0,4]$
$(c3)$ $(\preceq', \{S\}, \{d_S^3\})$	(d_S^3) $S \in [1, \top_{Real}]$
$(c4)$ $(\preceq', \{S\}, \{d_S^4\})$	(d_S^4) $S \in [3.75, \top_{Real}]$
	(d_S^5) $S \in [1,4]$
	(d_S^6) $S \in [3.75,4]$

The values $\perp_{Real}$ and $\top_{Real}$ are added to the $Real$ domain to ensure that $Real$ has *glb* and *lub* elements. The process terminates at row 4 when $C = \emptyset$ and there is no more constraint propagation. Here, S_d has just the constraint d_S^6 on S. Thus it can infer that $S \in [3.75, 4]$.

Table 2: An example of constraint solving on $\Re$

	C	S_d	S_I	Remarks
(0)	$\{c1, c2, c3, c4\}$	$\emptyset$	$\emptyset$	
(1)	$\{c2, c3, c4\}$	$\{d_S^1\}$	$\emptyset$	
(2)	$\{c3, c4\}$	$\{d_S^2\}$	$\emptyset$	$d_S^1, d_S^2 \vdash d_S^2$
(3)	$\{c4\}$	$\{d_S^5\}$	$\emptyset$	$d_S^2, d_S^3 \vdash d_S^5$
(4)	$\emptyset$	$\{d_S^6\}$	$\emptyset$	$d_S^5, d_S^4 \vdash d_S^6$

5.3 A colouring problem

This example illustrates the use of the immediate predecessor and successor functions introduced in Subsection 4.3.

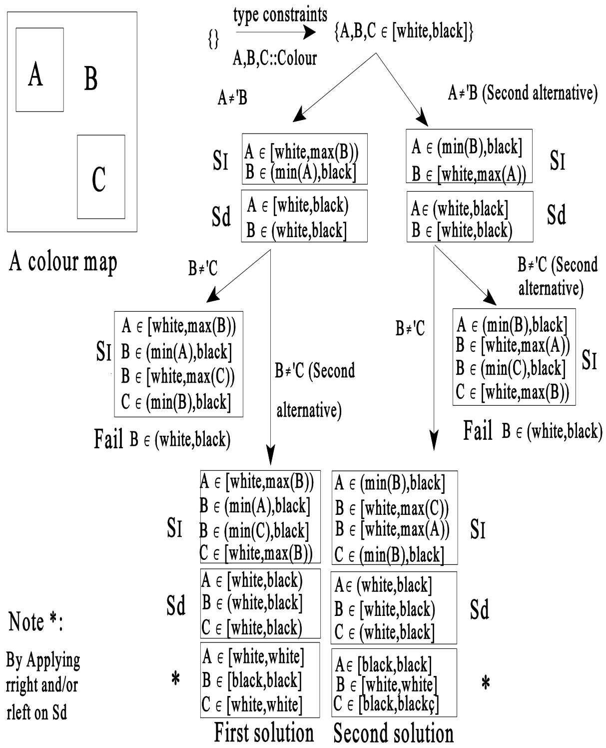

Figure 3: A colour map and its solving tree

Consider the problem of colouring the map shown in Figure 3 so that no two adjoining regions have the same colour. We consider the finite domain $Colours$ declared as **data** $Colour = white \mid black$. The problem consists in solving the following system of constraints: $\{A, B, C ::' Colour,\ A \neq' B,\ B \neq' C\}$. We assume the definition of $\neq'$ given in Subsection 3.3.

Let $\prec$ be defined by the order of the elements in the declaration of $Colour$ so that, $white \prec black$. Assume also that $suc(white) = black$ and $pred(black) = white$. By definition, $\perp_{Colour} = white$ and $\top_{Colour} = black$. The Tree in Figure 3 illustrates the constraint solving process. Each node shows the state of the stores S_I and S_d. When no more propagation is possible, rules $rleft$ and $rright$ (see Subsection 4.3) are applied on the interval constraints that belong to S_d (in the figure this is marked with $*$) and two solutions are found. Note that two branches detected an inconsistency when $B \in (white, black)$ since $suc(white) = black$ (see Subsection 4.3).

The initial system of constraints is represented by the following set of triples: $\{c1, c2, c3, c4; c4', c5; c5'\}$ defined in Table 3. The choice points are introduced by the definition of the propagator $\neq'$. Table 3 also gives the

Table 3: Triples and interval constraints in the colouring example

Triples	Simple Interval constraints	Constraints with indexicals
$(c1)$ $(::', \{A\}, \{d_A^1\})$	(d_A^1) $A \in [white, black]$	(d_A^B) $A \in [white, max(B))$
$(c2)$ $(::', \{B\}, \{d_B^1\})$	(d_B^1) $B \in [white, black]$	(d_B^A) $B \in (min(A), black]$
$(c3)$ $(::', \{C\}, \{d_C^1\})$	(d_C^1) $C \in [white, black]$	(d_B^C) $B \in [white, max(C))$
$(c4)$ $(\neq', \{A, B\}, \{d_A^B, d_B^A\})$	(d_A^2) $A \in [white, black)$	(d_C^B) $C \in (min(B), black]$
$(c4')$ $(\neq', \{A, B\}, \{d_A^{'B}, d_B^{'A}\})$	(d_B^2) $B \in (white, black]$	$(d_A^{'B})$ $A \in (min(B), black]$
$(c5)$ $(\neq', \{B, C\}, \{d_B^C, d_C^B\})$	(d_C^2) $C \in (white, black]$	$(d_B^{'A})$ $B \in [white, max(A))$
$(c5')$ $(\neq', \{B, C\}, \{d_B^{'C}, d_C^{'B}\})$	(d_A^3) $A \in (white, black]$	$(d_B^{'C})$ $B \in (min(C), black]$
$(c6)$ $(\in, \{A\}, \{d_A^2\})$	(d_B^3) $B \in [white, black)$	$(d_C^{'B})$ $C \in [white, max(B))$
$(c7)$ $(\in, \{B\}, \{d_B^2\})$	(d_C^3) $C \in [white, black)$	
$(c8)$ $(\in, \{B\}, \{d_B^3\})$		
$(c9)$ $(\in, \{C\}, \{d_C^2\})$		
$(c10)$ $(\in, \{C\}, \{d_C^3\})$		
$(c11)$ $(\in, \{A\}, \{d_A^3\})$		

interval constraints used by Table 4 to show the constraint solving process.

Three choice points are added by the inference machine and backtracking is executed when an inconsistency or a solution is found by the generic solver. When no more constraint propagation is possible, rules $rright$ and $rleft$ are applied (see Subsection 4.3). Two solutions are found: $(1)\{d_A^{wh}, d_B^{bl}, d_C^{wh}\}$ and $(2)\{d_A^{bl}, d_B^{wh}, d_C^{bl}\}$. In the solutions, d_X^{wh} denotes $X \in [white, white]$ and d_X^{bl} denotes $X \in [black, black]$ for $X = A, B$ or C.

5.4 Interval propagation to reason about sets

This final example illustrates how *minimum* and *maximum* are defined when the ordering is partial (see Subsection 4.1).

[GER97] describes a system for solving sets constraints. [GER97] provides the three inference rules shown below to describe the cases when two distinct set domains (ranges) are applied to a single set variable or when the set domain (range) of a set variable is reduced to one value or is inconsistent :

$$\textbf{I1} \quad \frac{a=b}{S_i \in [a,b] \vdash S_i = a} \qquad \textbf{I2} \quad \frac{b \subset a}{S_i \in [a,b] \vdash fail} \qquad \textbf{I3} \quad \frac{a'=a\cup c, b'=d\cap b}{S\in[a,b], S\in[c,d] \vdash S\in[a',b']}$$

where a, b, c, d, a' and b' are sets and S_i and S set variables. This can be easily modelled by our system. First, the domain Set is defined as follows:

$$data\ Set\ a \ = \ \{\} \mid \{a, Set\ a\}.$$

With the ordering $\preceq$ defined by set inclusion, rules I1 and I2 are implicitly defined by our system. However, this ordering is a partial ordering. $minimum/3$ and $maximum/3$ are defined:

$$
\begin{array}{ll}
minimum, maximum :: Set\ a \rightarrow Set\ a \rightarrow Set\ a & \\
minimum(A,B) = A\ if\ A \prec B, & maximum(A,B) = B\ if\ A \prec B, \\
\quad B\ if\ B \prec A, & A\ if\ B \prec A, \\
\quad A \cap B\ otherwise & A \cup B\ otherwise
\end{array}
$$

where $\cap$ and $\cup$ are defined respectively as the intersection and union on sets. Note that $\bot_{Set\ a} = \{\}$ whereas $\top_{Set\ a}$ is a new value added to the domain $Set\ a$. Cases for $\top_{Set\ a}$ can be explicitly defined as indicated in Subsection 3.1 (for example, $\top_{Set\ a} \cap A = A$). Note that $\cap$ satisfies condition **c1** and $\cup$ satisfies condition **c2** (see Subsection 4.1). Thus, rule $I3$ is defined by the inference rule $\cap_\tau$.
As an example of a high level constraint $\subseteq'$ is easily defined as follows:

$$
\begin{array}{l}
\subseteq' :: Set\ a \rightarrow Set\ a \rightarrow Bool \\
S \subseteq' S' :- S \in [\bot_{Set\ a}, max(S')], S' \in [min(S), \top_{Set\ a}].
\end{array}
$$

Constraint systems involving sets can be solved. For example, the system of constraints $\{S, S1 ::' Set\ Int, S \in [\{1\}, \{1,2,3,4\}], S1 \in [\{3\}, \{1,2,3\}],\ S \subseteq' S1\}$ produces the refined domains $S \in [\{1\}, \{1,2,3\}]$ and $S1 \in [\{1,3\}, \{1,2,3\}]$.

In the constraint solving process followed by our system, the system of constraints is translated to the set of triples: $\{c1, c2, c3, c4, c5\}$. Table 5 defines these triples and labels for the interval constraints used in Table 6.

The process terminates when the solution $\{d^4_S, d^4_{S1}\}$ is found. User definitions for *maximum* and *minimum* are used in the narrowing of the interval constraints (that is, when $\cap_\tau$ is applied).

6 Related work

Accounts of conventional interval arithmetic can be found in [MOO66]. [CLE87] and [SID92] describe several proposals to incorporate arithmetic into logic programming. Benhamou and Older [BEN97] and Lee and Van Emden [LEE93] apply interval reasoning to numerical computations on the real domain. [MAJ97] presents a survey of the existing work in the area of interval-based performance analysis of computing systems. Relative to the finite domains, we have built on the works of Gervets [GER97] on sets and Diaz and Codognet [COD96b, COD96a] on integers.

We have built on the work of of Codognet and Diaz [COD96a]. In particular, we have developed the *glass box* approach over constraints. Also, we have used a primitive constraint similar to the indexical in CLP(FD) over which high level constraints can be defined.

7 Conclusions

By combining interval reasoning with the use of indexicals we have built a unified framework in which to process constraints in any ordered domain independently of its length. The range of application and the flexibility of the system is wider than the usual CLP system since our framework allows the users to define their own domains and constraints, to control the constraint propagation mechanism and to modify the precision of the required results. The system provides a unified framework for the different and well-known instances of CLP such as CLP(FD), CLP($\Re$), CLP(Bool) and CLP(Set) among other.

An implementation of the system to test the feasibility of our proposal is currently under development.

Since CLP(X)$_<$ can mix different domains (declared as types), we plan to study the possibility of combining our system with a CLP type system [FRA97] in order to incorporate domain checking (type checking) and domain inference (type inference).

Acknowledgments

Specially, Antonio Fernández thanks Pablo López Olivas his nice talks (during the lunch time) and his useful comments on the development of a prototype implementation of the system proposed in this paper.

References

[ALL83] ALLEN J.F., Maintaining Knowledge About Temporal Intervals. In *Communications of the ACM*, 26 (11), 1983.

[BEN97] BENHAMOU F., and OLDER W.J., Applying Interval Arithmetic to Real, Integer and Boolean Constraints. In the *Journal of Logic Programming*, vol(32), Number 1, pps:1-24, July 1997.

[CLE87] CLEARY J.G., Logical Arithmetic. In Future computing Systems 2(2):125-149, 1987.

[COD96a] CODOGNET P. and DIAZ D., Compiling Constraints in *clp(FD)*. In *The Journal of Logic Programming*, 27:185-226, 1996.

[COD96b] CODOGNET P. and DIAZ D., Local Propagation Methods for Solving Boolean Constraints in Constraint Logic Programming. In *The Journal of Automated Reasoning*, vol. 17, n.1, 1996.

[dBUR88] de BURGOS J., Curso de Algebra y Geometría, Editorial Alhambra S.A., 1988.

[FRA97] FAGES F. and PALTRINIERI M., Statically Typing CLP Programs. In *COMPULOG Workshop*, October 1997.

[FRÜ94] FRÜHWIRTH T., Constraint handling rules. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, pp:90-107, LNCS 910, May, 1994.

[GER97] GERVET C., Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. In *Constraints: An International Journal*, Vol(1), Number 3, pps: 191-244, March 1997.

[JAF94] JAFFAR J. and MAHER M.J., Constraint Logic Programming: A survey. In *Journal of Logic Programming*, 19/20, pps:503-581, 1994.

[LEE93] LEE J.H.M. and Van EMDEN, Interval Computation as deduction in CHIP. In the *Journal of Logic Programming, Special Issue:Constraint Logic Programming*, Vol.16, Numbers 3&4, 1993.

[MAC77] MACKWORTH A.K., Consistency in Networks of Relations. In *Artificial Intelligence* vol(8), pps:99-118, 1997.

[MAJ97] MAJUMDAR S., Application of Relational Interval Arithmetic to Computer Performance Analysis: a Survey. In *Constraints: An International Journal*, Vol(2), Number 2, pps: 215-235, October 1997.

[MOO66] MOORE R.E., Interval Analysis. *Prentice hall*, Englewood Cliffs, NJ, 1996.

[REF96] REFALO P. and Van HENTENRYCK P., CLP($\Re_{lin}$) Revised. In Proceedings of Joint International Conference and Symposium on Logic Programming, JICSLP'96, ed.M. Maher, pps:22-36, Bonn, Germany, September 1996.

[SID92] SIDEBOTTOM G. and HAVENS, W.S., Hierarchical Arc Consistency for Disjoint Real Intervals in Constraint logic programming. In Computational Intelligence 8(4), 1992.

8 Appendix. A schedule problem on Reals

Consider the *Real* domain and the program [SID92] shown below that schedules tasks using some of the relations on temporal intervals described in [ALL83].

$in(task(S1, D1), task(S2, D2)) : -$
$\quad\quad S2 \leq' S1,$
$\quad\quad S1 +' D1 \leq' S2 +' D2.$
$noOverlap(\_, []).$
$noOverlap(task(S1, D1), [task(S2, D2) \ \ | \ Tasks]) : -$
$\quad S1 +' D1 \leq' S2 \vee S2 +' D2 \leq' S1,$
$\quad noOverlap(task(S1, D1), Tasks).$
$schedule([], \_).$
$schedule([Task \mid Tasks], Supertask) : -$
$\quad in(Task, Supertask),$
$\quad noOverlap(Task, Tasks),$
$\quad schedule(Tasks, SuperTask).$

A task is represented by a term $Task(S, D)$ where S is the start time of the task and D is the duration of the task. The predicate $in(Task, SuperTask)$ is true is the interval for $SuperTask$ contains the interval for $Task$. $noOverlap(Task, Tasks)$ is true if $Task$ overlaps with none of the tasks in the list $Tasks$. It uses a disjunctive inequality constraint to make sure $Task$ is either before or after all the tasks in $Tasks$. $Schedule(Tasks, Supertask)$ is true if all the tasks in the list $Tasks$ are in $SuperTask$ but no pair in $Tasks$ overlap. Consider the following goal:

$?{-}S{::}'Real, S \in [0, 4], schedule([task(0, 1), task(2.75, 1), task(S, 0.875)], task(0, 4.875)).$

Its resolution in the logical inference machine leads to solve the following system of constraints:

$\{S ::' Real, S \in [0, 4], 1 \leq' S \vee S +' 0.875 \leq' 0, 3.75 \leq' S \vee S +' 0.875 \leq' 2.75, 0 \leq' S, S +' 0.875 \leq 4.875\}$

Suppose arithmetical operators, the ordering relation $\prec$ and the equality are (pre)defined on *Reals* in the usual way. Figure 4 shows the solving tree. Each node contains the domain constraints actually in S_d which is initially empty. Each arrow is labelled by the constraint to consider. Note that disjunctive constraints introduce choice points. Propagators $\leq'$ and $+'$ were defined in

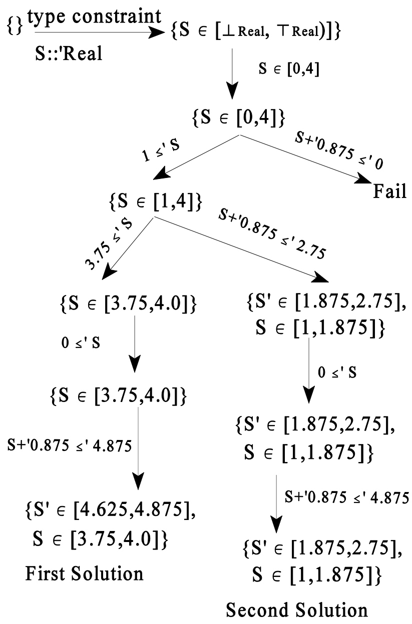

Figure 4: Tree solving for the scheduling problem on real domain

Subsection 3.3. The propagator $+'$ introduces an additional variable[7] (by clarity named S' in each branch) and two interval constraints in the form of $S' \in [min(S) + 0.875, max(S) + 0.875]$ and $S \in [min(S') - 0.875, max(S') - 0.875]$ which will remain in the store S_I (although by space restrictions, they aren't shown in the figure). Note $\perp_{Real}$ and $\top_{Real}$ are ficticial bounds which were added to the domain. The store becomes inconsistent in one branch and stable in another two ones. The solution set is $S \in [1, 1.875] \cup [3.75, 4.0]$. Figure 5 represents the problem and their solution ranges graphically.

As done in preceding examples, let's see the process in detail. The initial system of constraints is translated to the following set of triples:

$$\{c1, c2, c3; (ca, c3'), c4; (ca, c4'), c5, (ca, c6')\}$$

Table 7 shows the representation of each triple and each interval constraint used in the constraint solving process. Note that the triple ca is resulted from the propagator $+'$ (for example, constraint $S +' 0.875 \leq' 0$ is translated

[7]Note this variable corresponds to variable Z in the definition for $+'$ shown in Subsection 3.3. As told before, for simplicity we use a functional style instead of a relational style in order to get more expresiveness by operating with infix propagators. This means that $S' \equiv S +' 0.875$.

to $ca, c3'$). The whole process is detailed in Table 8. Two choice points are introduced as consequence of the disjunctive constraints ($\vee$). Two solutions are found: d_S^8 and d_S^7.

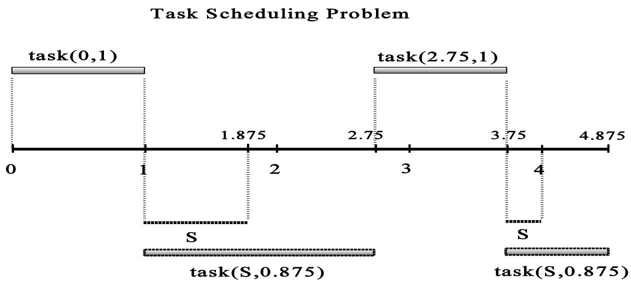

Figure 5: The task Scheduling problem graphically

Table 4: An example of constraint solving on the domain *Colour*

	Ch.P.	C	S_d	S_I	Remarks
(0)		$\{c1, c2, c3, c4; c4', c5; c5'\}$	$\emptyset$	$\emptyset$	
(1)		$\{c2, c3, c4; c4', c5; c5'\}$	$\{d_A^1\}$	$\emptyset$	
(2)		$\{c3, c4; c4', c5; c5'\}$	$\{d_A^1, d_B^1\}$	$\emptyset$	
(3)		$\{c4; c4', c5; c5'\}$	$\{d_A^1, d_B^1, d_C^1\}$	$\emptyset$	
(4)	1 ;c4'	$\{c5; c5'\}$	$\{d_A^1, d_B^1, d_C^1\}$	$\{d_A^B, d_B^A\}$	
(4.1)		$\{c6, c5; c5'\}$	$\{d_A^1, d_B^1, d_C^1\}$	$\{d_A^B, d_B^A\}$	$d_A^B \rightarrow c6$
(4.2)		$\{c7, c6, c5; c5'\}$	$\{d_A^1, d_B^1, d_C^1\}$	$\{d_A^B, d_B^A\}$	$d_B^A \rightarrow c7$
(5)		$\{c6, c5; c5'\}$	$\{d_A^1, d_B^2, d_C^1\}$	$\{d_A^B, d_B^A\}$	$d_B^1, d_B^2 \vdash d_B^2$
(6)		$\{c5; c5'\}$	$\{d_A^2, d_B^2, d_C^1\}$	$\{d_A^B, d_B^A\}$	$d_A^1, d_A^2 \vdash d_A^2$
(7)	2 ;c5'	$\emptyset$	$\{d_A^2, d_B^2, d_C^1\}$	$\{d_A^B, d_B^A, d_B^C, d_C^B\}$	
(7.1)		$\{c8\}$	$\{d_A^2, d_B^2, d_C^1\}$	$\{d_A^B, d_B^A, d_B^C, d_C^B\}$	$d_B^C \rightarrow c8$
(7.2)		$\{c9, c8\}$	$\{d_A^2, d_B^2, d_C^1\}$	$\{d_A^B, d_B^A, d_B^C, d_C^B\}$	$d_C^B \rightarrow c9$
(8)		$\{c8\}$	$\{d_A^2, d_B^2, d_C^2\}$	$\{d_A^B, d_B^A, d_B^C, d_C^B\}$	$d_C^1, d_C^2 \vdash d_C^2$
(9)		$\emptyset$	INCONSISTENCY	$\{d_A^B, d_B^A, d_B^C, d_C^B\}$	$d_B^2, d_B^3 \vdash B \in (white, black)$
					Backtrack Point 2 (c5')
		$\{c5'\}$	$\{d_A^2, d_B^2, d_C^1\}$	$\{d_A^B, d_B^A\}$	
(7)		$\emptyset$	$\{d_A^2, d_B^2, d_C^1\}$	$\{d_A^B, d_B^A, d_B^{'C}, d_C^{'B}\}$	
					$d_B^{'C} \rightarrow NC\ d_B^2 \in S_d$
(7.1)		$\{c10\}$	$\{d_A^2, d_B^2, d_C^1\}$	$\{d_A^B, d_B^A, d_B^{'C}, d_C^{'B}\}$	$d_C^{'B} \rightarrow c10$
(8)		$\emptyset$	$\{d_A^2, d_B^2, d_C^3\}$	$\{d_A^B, d_B^A, d_B^{'C}, d_C^{'B}\}$	$d_C^1, d_C^3 \vdash d_C^3$
					$d_B^{'C} \rightarrow NC\ d_B^2 \in S_d$
					$rleft + rright$ are applied
(8.1)		$\emptyset$	$\{d_A^{wh}, d_B^{bl}, d_C^{wh}\}$	$\{d_A^B, d_B^A, d_B^{'C}, d_C^{'B}\}$	FIRST SOLUTION
					Backtrack Point 1 (c4')
		$\{c4', c5; c5'\}$	$\{d_A^1, d_B^1, d_C^1\}$	$\emptyset$	
(4)		$\{c5; c5'\}$	$\{d_A^1, d_B^1, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}\}$	
(4.1)		$\{c11, c5; c5'\}$	$\{d_A^1, d_B^1, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}\}$	$d_A^{'B} \rightarrow c11$
(4.2)		$\{c8, c11, c5; c5'\}$	$\{d_A^1, d_B^1, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}\}$	$d_B^{'A} \rightarrow c8$
(5)		$\{c11, c5; c5'\}$	$\{d_A^1, d_B^3, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}\}$	$d_B^1, d_B^3 \vdash d_B^3$
					$d_A^{'B} \rightarrow c11NC\ (C = C \cup c11)$
(6)		$\{c5; c5'\}$	$\{d_A^3, d_B^3, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}\}$	$d_A^1, d_A^3 \vdash d_A^3$
					$d_B^{'A} \rightarrow NC\ d_B^3 \in S_d$
(7)	3 ;c5'	$\emptyset$	$\{d_A^3, d_B^3, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}, d_B^C, d_C^B\}$	
					$d_B^C \rightarrow NC\ d_B^3 \in S_d$
(7.1)		$\{c9\}$	$\{d_A^3, d_B^3, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}, d_B^C, d_C^B\}$	$d_C^{'B} \rightarrow c9$
(8)		$\emptyset$	$\{d_A^3, d_B^3, d_C^2\}$	$\{d_A^{'B}, d_B^{'A}, d_B^C, d_C^B\}$	$d_C^1, d_C^2 \vdash d_C^2$
					$d_B^C \rightarrow NC\ d_B^3 \in S_d$
					$rleft + rright$ applied
(8.1)		$\emptyset$	$\{d_A^{bl}, d_B^{wh}, d_C^{bl}\}$	$\{d_A^{'B}, d_B^{'A}, d_B^C, d_C^B\}$	SECOND SOLUTION
					Backtrack Point 3 (C5')
		$\{c5'\}$	$\{d_A^3, d_B^3, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}\}$	
(7)		$\emptyset$	$\{d_A^3, d_B^3, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}, d_B^{'C}, d_C^{'B}\}$	
(7.1)		$\{c7\}$	$\{d_A^3, d_B^3, d_C^1\}$	$\{d_A^{'B}, d_B^{'A}, d_B^{'C}, d_C^{'B}\}$	$d_B^{'C} \rightarrow c7$
					$d_B^{'C}$ NC. MAX(B) isn't solved
(8)		$\emptyset$	INCONSISTENCY		$d_B^3, d_B^2 \vdash B \in (white, black)$

Table 5: Triples and interval constraints in the example of the sets

Triples	Simple Interval constraints	Constraints with indexicals
$(c1)$ $(::', \{S\}, \{d_S^1\})$	(d_S^1) $S \in [\bot_{Set\ Int}, \top_{Set\ Int}]$	
$(c2)$ $(::', \{S1\}, \{d_{S1}^1\})$	(d_{S1}^1) $S1 \in [\bot_{Set\ Int}, \top_{Set\ Int}]$	
$(c3)$ $(\in, \{S\}, \{d_S^2\})$	(d_S^2) $S \in [\{1\}, \{1, 2, 3, 4\}]$	
$(c4)$ $(\in, \{S1\}, \{d_{S1}^2\})$	(d_{S1}^2) $S1 \in [\{3\}, \{1, 2, 3\}]$	
$(c5)$ $(\subseteq', \{S, S1\}, \{d_S^{S1}, d_{S1}^S\})$		(d_S^{S1}) $S \in [\bot_{Set\ Int}, max(S1)]$
		(d_{S1}^S) $S1 \in [min(S), \top_{Set\ Int}]$
$(c6)$ $(\in, \{S\}, \{d_S^3\})$	(d_S^3) $S \in [\bot_{Set\ Int}, \{1, 2, 3\}]$	
$(c7)$ $(\in, \{S1\}, \{d_{S1}^3\})$	(d_{S1}^3) $S1 \in [\{1\}, \top_{Set\ Int}]$	
	(d_S^4) $S \in [\{1\}, \{1, 2, 3\}]$	
	(d_{S1}^4) $S1 \in [\{1, 3\}, \{1, 2, 3\}]$	

Table 6: An example of constraint solving on sets of integers

	C	S_d	S_I	Remarks
(0)	$\{c1, c2, c3, c4, c5\}$	$\emptyset$	$\emptyset$	
(1)	$\{c2, c3, c4, c5\}$	$\{d_S^1\}$	$\emptyset$	
(2)	$\{c3, c4, c5\}$	$\{d_S^1, d_{S1}^1\}$	$\emptyset$	
(3)	$\{c4, c5\}$	$\{d_S^2, d_{S1}^1\}$	$\emptyset$	$d_S^1, d_S^2 \vdash d_S^2$
(4)	$\{c5\}$	$\{d_S^2, d_{S1}^2\}$	$\emptyset$	$d_{S1}^1, d_{S1}^2 \vdash d_{S1}^2$
(5)	$\emptyset$	$\{d_S^2, d_{S1}^2\}$	$\{d_S^{S1}, d_{S1}^S\}$	
(5.1)	$\{c6\}$	$\{d_S^2, d_{S1}^2\}$	$\{d_S^{S1}, d_{S1}^S\}$	$d_S^{S1} \rightarrow c6$
(5.2)	$\{c7, c6\}$	$\{d_S^2, d_{S1}^2\}$	$\{d_S^{S1}, d_{S1}^S\}$	$d_{S1}^S \rightarrow c7$
(6)	$\{c6\}$	$\{d_S^2, d_{S1}^4\}$	$\{d_S^{S1}, d_{S1}^S\}$	$d_{S1}^2, d_{S1}^3 \vdash d_{S1}^4$
(6.1)	$\{c6\}$	$\{d_S^2, d_{S1}^4\}$	$\{d_S^{S1}, d_{S1}^S\}$	$d_S^{S1} \rightarrow c6\ NC(C = C \cup c6)$
(7)	$\emptyset$	$\{d_S^4, d_{S1}^4\}$	$\{d_S^{S1}, d_{S1}^S\}$	$d_S^2, d_S^3 \vdash d_S^4$
(7.1)	$\{c7\}$	$\{d_S^4, d_{S1}^4\}$	$\{d_S^{S1}, d_{S1}^S\}$	$d_{S1}^S \rightarrow c7$
(8)	$\emptyset$	$\{d_S^4, d_{S1}^4\}$	$\{d_S^{S1}, d_{S1}^S\}$	$d_{S1}^4, d_{S1}^3 \vdash d_{S1}^4\ NC$
				FOUND A SOLUTION

Table 7: Triples and interval constraints for Scheduling example

Triples	Simple Interval constraints	Const. with indexicals
(ca) $(+', \{S, S'\}, \{d_S^{S'}, d_{S'}^{S}\})$		$(d_S^{S'})$ $S \in [min(S') - 0.875, max(S') - 0.875]$ $(d_{S'}^{S})$ $S' \in [min(S) + 0.875, max(S) + 0.875]$
$(c1)$ $(::', \{S\}, \{d_S^1\})$	(d_S^1) $S \in [\bot_{Real}, \top_{Real}]$	
$(c2)$ $(\in, \{S\}, \{d_S^2\})$	(d_S^2) $S \in [0, 4]$	
$(c3)$ $(\preceq', \{S\}, \{d_S^3\})$	(d_S^3) $S \in [1, \top_{Real}]$	
$(c3')$ $(\preceq', \{S'\}, \{d_{S'}^1\})$	$(d_{S'}^1)$ $S' \in [\bot_{Real}, 0]$	
$(c4)$ $(\preceq', \{S\}, \{d_S^4\})$	(d_S^4) $S \in [3.75, \top_{Real}]$	
$(c4')$ $(\preceq', \{S'\}, \{d_{S'}^2\})$	$(d_{S'}^2)$ $S' \in [\bot_{Real}, 2.75]$	
$(c5)$ $(\preceq', \{S\}, \{d_S^5\})$	(d_S^5) $S \in [0, \top_{Real}]$	
$(c6)$ $(\preceq', \{S'\}, \{d_{S'}^3\})$	$(d_{S'}^3)$ $S' \in [\bot_{Real}, 4.875]$	
$(c7)$ $(\in, \{S'\}, \{d_{S'}^4\})$	$(d_{S'}^4)$ $S' \in [4.625, 4.875]$	
$(c8)$ $(\in, \{S'\}, \{d_{S'}^5\})$	$(d_{S'}^5)$ $S' \in [1.875, 4.875]$	
	(d_S^6) $S \in [1, 4]$	
	(d_S^7) $S \in [3.75, 4]$	
	$(d_{S'}^6)$ $S' \in [1.875, 2.75]$	
$(c9)$ $(\in, \{S\}, \{d_S^8\})$	(d_S^8) $S \in [1, 1.875]$	
$(c10)$ $(\in, \{S'\}, \{d_{S'}^7\})$	$(d_{S'}^7)$ $S' \in [0.875, 4.875]$	

Table 8: Constraint solving process in the Scheduling problem on Reals

	Ch.P.	C	S_d	S_I	Remarks
(0)		$\{c1, c2, c3; (ca, c3'), c4; (ca, c4'), c5, (ca, c6)\}$	$\emptyset$	$\emptyset$	
(1)		$\{c2, c3; (ca, c3'), c4; (ca, c4'), c5, (ca, c6)\}$	$\{d_S^1\}$	$\emptyset$	
(2)		$\{c3; (ca, c3'), c4; (ca, c4'), c5, (ca, c6)\}$	$\{d_S^2\}$	$\emptyset$	$d_S^1, d_S^2 \vdash d_S^2$
(3)	$1 \,; (ca, c3')$	$\{c4; (ca, c4'), c5, (ca, c6)\}$	$\{d_S^6\}$	$\emptyset$	$d_S^2, d_S^3 \vdash d_S^6$
(4)	$2 \,; (ca, c4')$	$\{c5, (ca, c6)\}$	$\{d_S^7\}$	$\emptyset$	$d_S^6, d_S^4 \vdash d_S^7$
(5)		$\{ca, c6\}$	$\{d_S^7\}$	$\emptyset$	$d_S^7, d_S^5 \vdash d_S^7$ NC
(6)		$\{c6\}$	$\{d_S^7\}$	$\{d_S^{S'}, d_{S'}^S\}$	
(6.1)		$\{c7, c6\}$	$\{d_S^7\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_{S'}^S \to c7$
(7)		$\{c6\}$	$\{d_S^7, d_{S'}^4\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_S^{S'} \to NC(d_S^7 \in S_d)$.
(8)		$\emptyset$	$\{d_S^7, d_{S'}^4\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_{S'}^4, d_{S'}^3 \vdash d_{S'}^4$ NC
		FOUND FIRST SOLUTION (d_S^7)			
					Backtrack Point 2
		$\{(ca, c4'), c5, (ca, c6)\}$	$\{d_S^6\}$	$\emptyset$	
(4)		$\{c4', c5, (ca, c6)\}$	$\{d_S^6\}$	$\{d_S^{S'}, d_{S'}^S\}$	
(4.1)		$\{c8, c4', c5, (ca, c6)\}$	$\{d_S^6\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_{S'}^S \to c8$
(5)		$\{c4', c5, (ca, c6)\}$	$\{d_S^6, d_{S'}^5\}$	$\{d_S^{S'}, d_{S'}^S\}$	
					$d_S^{S'} \to NC(d_S^6 \in S_d)$
(6)		$\{c5, (ca, c6)\}$	$\{d_S^6, d_{S'}^6\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_{S'}^5, d_{S'}^2 \vdash d_{S'}^6$
(6.1)		$\{c9, c5, (ca, c6)\}$	$\{d_S^6, d_{S'}^6\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_S^{S'} \to c9$
(7)		$\{c5, (ca, c6)\}$	$\{d_S^8, d_{S'}^6\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_S^6, d_S^8 \vdash d_S^8$
					$d_{S'}^S \to NC(d_{S'}^6 \in S_d)$
(8)		$\{ca, c6\}$	$\{d_S^8, d_{S'}^6\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_S^8, d_S^5 \vdash d_{S'}^8$ NC
(9)		$\{c6\}$	$\{d_S^8, d_{S'}^6\}$	$\{d_S^{S'}, d_{S'}^S\}$	NC
(10)		$\emptyset$	$\{d_S^8, d_{S'}^6\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_{S'}^6, d_{S'}^3 \vdash d_{S'}^6$ NC
		FOUND SECOND SOLUTION (d_S^8)			
					Backtrack Point 1
		$\{(ca, c3'), c4; (ca, c4'), c5, (ca, c6)\}$	$\{d_S^2\}$	$\emptyset$	
(3)		$\{c3', c4; (ca, c4'), c5, (ca, c6)\}$	$\{d_S^2\}$	$\{d_S^{S'}, d_{S'}^S\}$	
(3.1)		$\{c10, c3', c4; (ca, c4'), c5, (ca, c6)\}$	$\{d_S^2\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_{S'}^S \to c10$
(4)		$\{c3', c4; (ca, c4'), c5, (ca, c6)\}$	$\{d_S^2, d_{S'}^7\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_{S'}^S \to NC(d_{S'}^7 \in S_d)$
(5)		$\{c4; (ca, c4'), c5, (ca, c6)\}$	$\{d_S^2, d_{S'}^7\}$	$\{d_S^{S'}, d_{S'}^S\}$	$d_{S'}^7, d_{S'}^1 \vdash Fail$
		INCONSISTENCY			