CLP(Intervals) Revisited

F. Benhamon[1]
D. Mc Allester[2]
P. Van Hentenryck[3]

Technical Report No. CS-94-18

April, 1994

[1]Universite Aix-Marselle II 163, Av. de Luminy, 13288 Marselle, France
[2]MIT, 421 Technology Square Cambridge, MA
[3]Brown University, COmputer Science Department Box 1910, Providence, RI 02912

CLP(Intervals) Revisited [1]

F. Benhamou	D. McAllester	P. Van Hentenryck
Université Aix-Marseille II	MIT	Brown University
163, Av. de Luminy	Technology Square, 421	Box 1910
13288 Marseille, France	Cambridge (USA)	Providence, RI 02912 (USA)
`benham@gia.univ-mrs.fr`	dam@ai.mit.edu	pvh@cs.brown.edu

Abstract

The design and implementation of constraint logic programming (CLP) languages over intervals is revisited. Instead of decomposing complex constraints in terms of simple primitive constraints as in `CLP(BNR)`, complex constraints are manipulated as a whole, enabling more sophisticated narrowing procedures to be applied in the solver. This idea is embodied in a new CLP language `Newton` whose operational semantics is based on the notion of box-consistency, an approximation of arc-consistency, and whose implementation uses Newton interval method. Experimental results indicate that `Newton` outperforms existing languages by an order of magnitude and is competitive with some state-of-the-art tools on some standard benchmarks. Limitations of our current implementation and directions for further work are also identified.

1 Introduction

The introduction of a relational form of interval arithmetic in logic programming has been proposed by Cleary in [3]. These ideas have been developed and made popular by the CLP system `BNR`-Prolog [18] and generalized to constraint solving over discrete quantities in its successor `CLP(BNR)` [17, 2]. Many other systems (e.g [11, 20]) have been developed on similar principles. The key idea behind CLP(Intervals) languages is to let users state arbitrary constraints over reals and to narrow down the set of possible values for the variables using various approximations of arc-consistency [12, 13], a notion well-known in artificial intelligence[2]. In addition, combining the constraint solver with splitting operations allows these systems to isolate narrow regions which may contain solutions to sets of constraints.

Traditionally, CLP(Intervals) languages have been designed in terms of simple primitive constraints (e.g. *add(a,y,z), mult(x,y,z), cos(x,z), ...*) on which they apply approximations of arc-consistency. Complex constraints are simply rewritten into a set of primitive constraints. The advantage of this methodology is the resulting elegant operational semantics of the language. The inconvenient is that convergence of the solver may be slow and the pruning relatively weak due to the decomposition process.

This situation is best contrasted with the interval community which has focused, among other topics, on "efficient" algorithms to bound solutions to systems of nonlinear equations. In general, these algorithms use ideas behind Newton root finding method, exploit properties such as

[1] Part of this research was carried out while Pascal Van Hentenryck was visiting MIT and the university of Marseille. The research was partly supported by the Office of Naval Research under grant N00014-91-J-4052 ARPA order 8225, the National Science Foundation under grant numbers CCR-9357704, a NSF National Young Investigator Award, and European Esprit Basic Research project ACCLAIM no 7195.

[2] Arc-consistency itself cannot be computed exactly due to machine limitation.

differentiability, and define various pruning operators, many of them extending the seminal work of Krawczyk [10]. These algorithms can often be viewed as an iteration of two steps, constraint propagation and splitting, although they are rarely presented this way and it is not always clear what the constraint propagation step computes.

The goal of our research is to reconcile as best as possible the conflicting goals of simplicity and efficiency in CLP(Intervals) languages. This paper is a step in this direction and presents the design and implementation of `Newton`, a new CLP(Intervals) language manipulating complex constraints as a whole. The key conceptual idea behind `Newton` is the notion of box-consistency. From a programming language standpoint, box-consistency is a new approximation of arc-consistency which is essentially equivalent to traditional approximations on simple constraints. As a result, box-consistency leads to an operational semantics in the same style and spirit as those of traditional CLP(Intervals) languages. From an algorithmic standpoint, box-consistency is applied to complex constraints, allowing effective pruning techniques to be applied. In particular, `Newton` enforces box-consistency by finding the left- and right-most "zeros" of an interval function using a special-purpose interval Newton method.

`Newton` has been implemented as a complete CLP system (subsuming Prolog) and tested on some standard benchmarks from the interval community. Our preliminary results indicate that `Newton` can produce exponential speed-ups over traditional implementations and compares well with some established and recent methods in interval arithmetics. The limitations of `Newton` have also been identified and are discussed in the paper.

The rest of the paper is organized as follows. Section 2 describes the constraint systems while section 3 contains the preliminaries on interval arithmetic. Sections 4, 5, and 6 describe respectively the operational semantics, the narrowing operator in `Newton` and some implementation details. Section 7 contains the experimental results. Section 8 discusses the related work, the limitations of `Newton`, and directions for future developments. Section 9 concludes the paper.

2 Constraint Systems

We consider a CLP(Intervals) language in which the constraint system is defined as follows. Let Σ be a structure $< \Re, \mathcal{O}, \mathcal{R} >$ with $\mathcal{O} = \{+, -, \times, \div, \cos, \sin, \arccos, \log, \ldots\}$ a set of operations over the reals and $\mathcal{R} = \{=, \geq\}$ and let $V = \{v_1, \ldots, v_n, \ldots\}$ be an infinite countable set of variables taking their values over $\Re$. Terms in the constraint language are syntactic expressions built in the usual way from constants, variables, and operations. Constraints are expressions of the form $E \diamond 0$, where E is a term and $\diamond$ a relational symbol from $\{=, \geq\}$. A *constraint system* in the structure Σ is a pair $< S, \langle D_1, \ldots, D_n \rangle >$ where S is a finite set of constraints from Σ over the variables $\{v_1, \ldots, v_n\}$ and $D_1, \ldots, D_n$ are subsets of $\Re$ called the *domains* of the variables $v_1, \ldots, v_n$. In the following, we note in the same way operation and relation symbols over the reals and their interpretations.

3 Interval Arithmetic

The key idea behind interval arithmetic [14] is the approximation of real numbers by intervals to quantify the errors introduced by finite precision arithmetic. In addition, interval computations provide an appropriate framework to deal with uncertain data. For general results on interval arithmetic, see for instance [14, 1]. Although the theory of interval arithmetic has been developed in a broader context (including intervals of real numbers), our interests are very pragmatic and we consider solely the case of intervals whose bounds are floating point numbers. Results of arithmetic operations are "outward-rounded" to preserve correctness of the computations.

Intervals We consider $\Re \cup \{-\infty, +\infty\}$ the set of real numbers augmented with the two infinity symbols and the natural extension of the relation $\leq$ to this set. For every $a, b \in \Re \cup \{-\infty, +\infty\}$, the interval $[a, b]$ represents the set $\{x \in \Re \mid a \leq x \leq b\}$. For simplicity, only closed intervals are considered in this paper but the results generalize easily to other intervals [2, 3]. Given an interval $I = [a, b]$, the left (resp. right) bound of I, denoted by *left(I)* (resp. *right(I)*), is a (resp. b). The center of I, denoted by *center(I)*, is $\frac{a+b}{2}$.

F-Intervals Let $\mathcal{F}$ be a finite subset of $\Re \cup \{-\infty, +\infty\}$, with $\{-\infty, +\infty\} \subset \mathcal{F}$. Elements of $\mathcal{F}$ are called *F-numbers* and are, in practice, floating-point numbers. We call *F-interval* any element $[a, b]$ such that $a, b \in \mathcal{F}$ and we note $I(\mathcal{F})$ the set of F-intervals. Set inclusion is a partial ordering on F-intervals. When $a \in \Re$, we denote by a^+ the smallest element of $\mathcal{F}$ greater than a.

Notations Real numbers are denoted by a, b, intervals by I, domains (subsets of $\Re$) by D, real variables by x, y, interval variables by X, Y, functions over reals by f, functions over intervals by F, relations over reals by r and relations over intervals by R, all possibly subscripted or superscripted.

Approximation CLP(Intervals) languages are generally concerned with two different levels of approximations of real subsets. Intuitively, the first level consists in approximating sets of real numbers with finite unions of F-intervals in order to manipulate computable quantities, while the second level approximates these sets with continuous F-intervals to avoid combinatorial explosion. Let $U(\mathcal{F}) = \{D \subseteq \Re \mid \exists \langle I_1, \ldots, I_n \rangle \in I(\mathcal{F})^n : D = I_1 \cup \ldots \cup I_n\}$. Both $I(\mathcal{F})$ and $U(\mathcal{F})$ are closed under intersection.

Definition 1 [Approximation] Let r be a subset of $\Re$. The *approximation* of r, denoted by $appx(r)$, is the smallest (w.r.t. the inclusion) element of $U(\mathcal{F})$ containing r.

This definition generalizes both the notion of "outward rounding" and the approximation of real numbers by "machine intervals" [1, 14], when F-numbers are floating point numbers. In the following, we denote $appx(\{a\})$ by $\overrightarrow{a}$ for simplicity.

Definition 2 [Hull] Let r be a subset of $\Re$, The *F-interval hull* of r, denoted by $hull(r)$, is the smallest F-interval containing r.

appx and *hull* are monotone and idempotent, $appx(r) \subseteq hull(r)$, $hull(appx(r)) = appx(hull(r)) = hull(r)$, and $appx(\{a\}) = hull(\{a\})$.

Interval Extensions A key concept in interval reasoning is the notion of interval extension.

Definition 3 [Interval Extension] An interval function $F : I(\mathcal{F})^n \to I(\mathcal{F})$ is an interval extension of a function $f : \Re^n \to \Re$ iff $f(a_1, \ldots, a_n) \in F(\overrightarrow{a_1}, \ldots, \overrightarrow{a_n})$. A relation $R \subseteq I(\mathcal{F})^n$ is an interval extension of the relation $r \subseteq \Re^n$ iff $(a_1, \ldots, a_n) \in r \Rightarrow (\overrightarrow{a_1}, \ldots, \overrightarrow{a_n}) \in R$.

Definition 4 [Monotonicity] An interval function $F : I(\mathcal{F})^n \rightarrow I(\mathcal{F})$ is monotone iff $I_1 \subseteq I'_1, \ldots, I_n \subseteq I'_n$ implies $F(I_1, \ldots, I_n) \subseteq F(I'_1, \ldots, I'_n)$. An interval relation $R \subseteq I(\mathcal{F})^n$ is monotone iff $I_1 \subseteq I'_1, \ldots, I_n \subseteq I'_n$ implies $(I_1, \ldots, I_n) \in R \Rightarrow (I'_1, \ldots, I'_n) \in R$.

In the following, we restrict attention to monotone interval extensions. The fundamental theorem of interval arithmetics [14] can be stated as follows.

Theorem 5 [Fundamental Theorem of Interval Arithmetics] Let $F : I(\mathcal{F})^n \rightarrow I(\mathcal{F})$ be a monotone interval extension of $f : \Re^n \rightarrow \Re$. Then $a_1 \in I_1, \ldots, a_n \in I_n$ implies $f(a_1, \ldots, a_n) \in F(I_1, \ldots, I_n)$. Similarly, let $R \subseteq I(\mathcal{F})^n$ be a monotone interval extension of $r \subseteq \Re^n$. Then $a_1 \in I_1, \ldots, a_n \in I_n$ implies $(a_1, \ldots, a_n) \in r \Rightarrow (I_1, \ldots, I_n) \in R$.

Interval Extensions of Primitive Operations and Relations There are of course multiple possible interval extensions of an operation or relation. We use the following extensions for primitive operations and relations.

Definition 6 The interval extension of an operation $\mathtt{op} : \Re^n \rightarrow \Re$ is the function $\overrightarrow{\mathtt{op}} : I(\mathcal{F})^n \rightarrow I(\mathcal{F})$, defined as $\overrightarrow{\mathtt{op}}(X_1, \ldots, X_n) = \mathit{hull}(\{\mathtt{op}(x_1, \ldots, x_n) \mid x_i \in X_i \quad 1 \leq i \leq n \})$. The interval extension of a relation $\diamond \subset \Re^n$ is the relation

$$\overrightarrow{\diamond} = \{(X_1, \ldots, X_n) \in I(\mathcal{F})^n \mid \exists x_1 \in X_1, \ldots, \exists x_n \in Xn, (x_1, \ldots, x_n) \in \diamond\}.$$

It is easy to see that these definitions produce monotone interval extensions.

Interval Extensions of Terms and Constraints It is well-known that floating point arithmetic does not preserve most of the simplest properties of the real numbers (e.g. associativity and distributivity). As a consequence, the evaluation of an expression on a computer depends on the text of the expression and thus the syntax of constraints in CLP(Intervals) languages influences their operational semantics. In languages such as `CLP(BNR)`, this does not affect the semantics of primitive constraints which are extremely simple but it impacts the semantics of complex constraints which are rewritten into simple primitive constraints. In `Newton`, primitive constraints can be arbitrarily complex. To model their operational semantics precisely, it is necessary to manipulate their syntax and it is convenient to use simple forms of lambda expression to define their interval extensions. A *real lambda expression* is an expression of the form $\lambda x_1 \ldots \lambda x_n \; E$ where $x_1, \ldots, x_n$ are real variables and E is an expression constructed from real numbers, variables, and real operations. The above expression denotes a function whose evaluation for some real values $a_1, \ldots, a_n$ returns the value obtained by replacing the x_i by the a_i and applying the operations. We also use *interval lambda expressions* of the form $\lambda X_1 \ldots \lambda X_n \; E$ where $X_1, \ldots, X_n$ are variables ranging over $I(\mathcal{F})$ and E is an expression constructed from elements of $I(\mathcal{F})$, variables in E$\{X_1, \ldots, X_n\}$, and $I(\mathcal{F})$ operations. The textual function associated with a term can now be defined as follows.

Definition 7 [Textual Function] Let E be a term over variables $\{v_1, \ldots, v_n\}$. The textual function of E, denoted `TEXT`[E], is the real lambda expression $\lambda x_1 \ldots \lambda x_n \; T[E]$ where

$T[v_i] = x_i$; $T[a] = a$; $T[E_1 \; \mathtt{op} \; E_2] = T[E_1] \; \mathtt{op} \; T[E_2]$; $T[\mathtt{op}(E)] = \mathtt{op}(T[E])$.

We note $\phi[E] : \Re^n \rightarrow \Re$ the function denoted by `TEXT`[E].

Interval extensions of complex functions can now be defined as follows.

Definition 8 [Interval Extension of Real Lambda Expression] Let L be a real lambda expression $\lambda x_1 \dots \lambda x_n\ E$. The interval extension of L, denoted by $\overrightarrow{L}$, is an interval lambda expression $\lambda X_1 \dots \lambda X_n\ G[E]$ where

$G[x_i] = X_i$; $G[a] = \overrightarrow{a}$; $G[E_1 \texttt{ op } E_2] = G[E_1]\ \overrightarrow{\texttt{op}}\ G[E_2]$; $G[\texttt{op}(E)] = \overrightarrow{\texttt{op}}(G[E])$.

We note $\overrightarrow{\phi}[E] : I(\mathcal{F})^n \rightarrow I(\mathcal{F})$ the function denoted by $\overrightarrow{\texttt{TEXT}[E]}$.

Once again, it is easy to see that for every expression E, $\overrightarrow{\phi}[E]$ is a monotone interval extension of $\phi[E]$. We are now in position to define the interval extension of a constraint.

Definition 9 [Interval extension of constraints] Let C be a constraint of the form $E \diamond 0$ over the variables $\{v_1, \dots, v_n\}$. C denotes the relation $\rho[C] = \{\langle a_1, \dots, a_n\rangle \in \Re^n \mid \phi[E](a_1, \dots, a_n) \diamond 0\}$. The interval extension of C is the relation $\overrightarrow{\rho}[C] = \{\langle I_1, \dots, I_n\rangle \in I(\mathcal{F})^n \mid \overrightarrow{\phi}[E](I_1, \dots, I_n) \overrightarrow{\diamond} \overrightarrow{0}\}$.

4 Operational Semantics

This section presents the operational semantics of the solver of `Newton` and contrasts it with existing CLP(Intervals) languages. Section 4.1 defines the notion of projection constraint in term of which is expressed the operational semantics, as is traditional. CLP(Intervals) languages enforce a notion of local consistency on projection constraints and section 4.2 reviews the various notions proposed in the past and the notion of box-consistency used in `Newton`. In section 4.3 we define A-consistency and maximal A-consistency for the various consistency notions. Section 4.4 defines the operational semantics in a generic way. Section 4.5 shows how to extract the narrowing operators from A-consistency notions. Section 4.6 presents the fixpoint algorithm.

4.1 Projection Constraints

The fundamental idea behind CLP(intervals) consists in using projection constraints to narrow the range of variables.

Definition 10 [Projection Constraint] Let C be a constraint over $\{v_1, \dots, v_n\}$. The i-th projection constraint of C is the pair $< C, i >$ $(1 \leq i \leq n)$ defining the relation

$$\rho[< C, i >] = \{a_i \in \Re \mid \exists \langle a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n\rangle \in \Re^{n-1} : \langle a_1, \dots, a_n\rangle \in \rho(C)\}.$$

In the following, projection constraints are denoted by the letter P, possibly subscripted.

4.2 Local Consistency

CLP(Intervals) languages use projection constraints to approximate the notion of arc-consistency, well established in artificial intelligence [12, 13] especially for finite constraint satisfaction problems. In the rest of this section, we assume that constraints are defined over $\{v_1, \dots, v_n\}$ and that $\langle D_1, \dots, D_n\rangle$ is a sequence of domains, i.e. $D_i \subset \Re$, for all $i \in \{1, \dots, n\}$.

Definition 11 [Arc-Consistency] A projection constraint $< C, i >$ is arc-consistent wrt $\langle D_1, \ldots, D_n \rangle$ iff $D_i = D_i \cap \{a_i \in \Re \mid \exists a_1 \in D_1 \ldots \exists a_{i-1} \in D_{i-1} \exists a_{i+1} \in D_{i+1} \ldots a_n \in D_n : \langle a_1, \ldots, a_n \rangle \in \rho[C]\}$

Arc-consistency cannot be computed exactly on real numbers due to machine limitation since D_i may not be representable in a floating point system (an example is the constraint $X - arccos(0) = 0$). Hence, various CLP languages approximate this notion by other local consistency notions, achieving a tradeoff between the strength of the reduction, the complexity of the primitive constraints, and computational efficiency. We present these notions from the strongest to the weakest.

Definition 12 [Interval-Consistency] A projection constraint $< C, i >$ is interval-consistent wrt $\langle D_1, \ldots, D_n \rangle$ iff $D_i = appx(D_i \cap \{a_i \in \Re \mid \exists a_1 \in D_1 \ldots \exists a_{i-1} \in D_{i-1} \exists a_{i+1} \in D_{i+1} \ldots a_n \in D_n : \langle a_1, \ldots, a_n \rangle \in \rho[C]\})$

The key idea behind the notion of interval-consistency is to adapt arc-consistency to take into account machine precision. Real numbers are simply approximated by F-intervals. Interval-consistency is used for instance in the Echidna system [20]. The next notion is used in `CLP(BNR)`. Intuitively, the idea is to fill the gaps between the intervals to preserve only one F-interval. This is motivated by the fact that preserving multiple intervals may be computationally too expensive in many applications.

Definition 13 [Hull-Consistency] A projection constraint $< C, i >$ is hull-consistent wrt $\langle D_1, \ldots, D_n \rangle$ iff $D_i = hull(D_i \cap \{a_i \in \Re \mid \exists a_1 \in D_1 \ldots \exists a_{i-1} \in D_{i-1} \exists a_{i+1} \in D_{i+1} \ldots a_n \in D_n : \langle a_1, \ldots, a_n \rangle \in \rho[C]\})$

Both interval-consistency and hull-consistency are reasonable choices when primitive constraints are simple (e.g. they do not involve multiple occurrences of the same variable) as in `CLP(BNR)`. They are too demanding when primitive constraints are complex (e.g they allow multiple occurrences of the same variable), since they may require exploring multiple combinations of intervals for the variables appearing in the constraint. We illustrate this on an extremely simple example.

Example 14 Let C be the constraint $v_2 + v_1 - v_2 = 0$. $< C, 1 >$ is not hull-consistent with respect to $\langle [-1, 1], [0, 1] \rangle$. However, it is not possible in general to get hull-consistency without exploring combinations of intervals for v_1 and v_2. Of course here, simple symbolic manipulation removes the problem but this is not possible in general.

This motivates the notion of box-consistency used in `Newton`. The key idea is to remove the existential quantification and to replace variables by their domains using constraint extension.

Definition 15 [Box-Consistency] A projection constraint $< C, i >$ is box-consistent wrt $\langle D_1, \ldots, D_n \rangle$ iff $D_i = hull(D_i \cap \{a_i \in \Re \mid \langle D_1, \ldots, D_{i-1}, \overrightarrow{a_i}, D_{i+1}, \ldots, D_n \rangle \in \overrightarrow{\rho}[C]\})$.

There are several classes of constraints where hull-consistency and box-consistency are extremely close. This is the case for the primitive constraints of `CLP(BNR)`. The difference can become important when multiple occurrences of variables appear, since, for instance, the above example is box-consistent[3]. The definition also suggests the strategy of `Newton`: to rewrite n-ary constraints over reals to unary constraints over intervals.

[3] Of course, the same dependency problem appears in `CLP(BNR)` but at another level (i.e. in the fixpoint algorithm).

4.3 A-Consistency

The semantics is parametrized by the notion of consistency, i.e. we use A-consistency where A can be replaced by elements of the set {arc,interval,hull,box}.

Definition 16 Let $S = \{P_1, \ldots, P_m\}$ be a set of projection constraints over $\{v_1, \ldots, v_n\}$. S is A-consistent wrt $\langle D_1, \ldots, D_n \rangle$ iff P_i is A-consistent wrt $\langle D_1, \ldots, D_n \rangle$ $(1 \leq i \leq m)$.

Definition 17 [Maximal A-Consistency] $\langle D_1, \ldots, D_n \rangle$ are maximally A-consistent wrt $\{P_1, \ldots, P_m\}$ and $\langle D_1^0, \ldots, D_n^0 \rangle$ iff $\langle D_1, \ldots, D_n \rangle$ are the largest domains satisfying

1. $D_i \subseteq D_i^0$ $(1 \leq i \leq n)$;
2. $\{P_1, \ldots, P_m\}$ is A-consistent wrt $\langle D_1, \ldots, D_n \rangle$;

4.4 Operational Semantics

We now give the operational semantics of the solver in generic way. Here we restrict ourselves to {interval,hull,box}-consistency, since arc-consistency is not computable in the general case.

Definition 18 [Operational Semantics] Given A in $\{interval, hull, box\}$, a set of projection constraints $\{P_1, \ldots, P_m\}$ over $\{v_1, \ldots, v_n\}$ and a tuple of initial domains $\langle D_1^0, \ldots, D_n^0 \rangle$, the solver outputs a tuple of domains $\langle D_1, \ldots, D_n \rangle$ which is maximally A-consistent wrt $\{P_1, \ldots, P_m\}$ and $\langle D_1^0, \ldots, D_n^0 \rangle$.

4.5 Narrowing Operators

The solvers of CLP(Intervals) languages use narrowing operators to prune the domains. Narrowing operators are defined from the consistency notions.

Definition 19 The narrowing operator A-narrow is defined as follows. Let D_i' is the largest set included in D_i such that $< C, i >$ is A-consistent wrt $\langle D_1, \ldots, D_{i-1}, D_i', D_{i+1}, \ldots, D_n \rangle$.

A-narrow($< C, i >$,$\langle D_1, \ldots, D_n \rangle$) = D_i'

Narrowing operators are *contractant* (computed domains are smaller than the initial domains), *monotone* (inclusion is preserved by narrowing) and *idempotent* (no iteration is required). Furthermore, the following property establishes the soundness of the various approximations.

Property 4.1 [soundness] Let P be a projection constraint and $\overline{D} = \langle D_1, \ldots, D_n \rangle$ a tuple of domains. Then, arc-narrow(P,$\overline{D}$) $\subseteq$ interval-narrow(P,$\overline{D}$) $\subseteq$ hull-narrow(P,$\overline{D}$) $\subseteq$ box-narrow(P,$\overline{D}$).

4.6 The Fixpoint Algorithm

The fixpoint algorithm used by CLP(Intervals) languages is extremely simple and can be seen as a version of the algorithm `AC-3` [12]. We assume that all constraints are defined on subsets of $\{v_1, \ldots, v_n\}$. We note $var(< C, i >)$ the set of variables occurring in C.

```
fixpoint( in {P_1,...,P_m} ; inout {D_1,...,D_n} )
begin
      queue := {P_1,...,P_m};
      while queue ≠ ∅ do
            <C,i> := POP_QUEUE;
            D := A-NARROW(<C,i>,⟨D_1,...,D_n⟩);
            if D ≠ D_i then
                  D_i := D;
                  queue := queue ∪ {P_j | v_i ∈ var(P_j)} \ {<C,i>}
            endif
      endwhile
end;
```

5 Narrowing in `Newton`

This section describes the implementation of the narrowing operator `box-narrow` of `Newton`. The main idea behind the implementation consists in using parts of the interval Newton method, a very effective method to find what we call quasi-zeros of a function $f : \Re^n \to \Re$. Our implementation in general reduces the problem of enforcing box-consistency to the problem of finding the left- and right-most quasi-zeros of an interval function $F : I(\mathcal{F})^n \to I(\mathcal{F})$ (section 5.1). The search for a quasi-zero uses the main step of Newton Interval method, called here Newton reduction, which is based on derivatives (section 5.2). Newton Reduction method does not produce box-consistency however and we use a restricted internal splitting to achieve box consistency of an equation (section 5.3). Obtaining box consistency for inequalities is then relatively easy (section 5.4).

5.1 Problem Formulation

The goal of the narrowing operator of `Newton` is to enforce box-consistency of a projection constraint $< E \diamond 0, i >$ wrt intervals $\langle I_1, \ldots, I_n \rangle$. In fact, this problem is easily transformed into finding some quasi-zeros of an interval function $F : I(\mathcal{F}) \to I(\mathcal{F})$. We first define the notion of quasi-zero. Informally speaking, a quasi-zero of a real function is a zero or a value that cannot be distinguished from a zero due to machine precision.

Definition 20 [Quasi-Zero] A quasi-zero of a real function $f : \Re^n \to \Re$ wrt to one of its interval extensions $F : I(\mathcal{F})^n \to I(\mathcal{F})$ is an element of the set $\{\langle a_1, \ldots, a_n \rangle \in \Re^n \mid 0 \in F(\overrightarrow{a_1}, \ldots, \overrightarrow{a_n})\}$. A quasi-zero of an interval function $F : I(\mathcal{F})^n \to I(\mathcal{F})$ is an element of the set $\{\langle \overrightarrow{a_1}, \ldots, \overrightarrow{a_n} \rangle \mid \langle a_1, \ldots, a_n \rangle \in \Re^n \ \& \ 0 \in F(\overrightarrow{a_1}, \ldots, \overrightarrow{a_n})\}$.

We now give a definition to switch the parameter order of a lambda expression.

Definition 21 Let $\lambda x_1 \ldots \lambda x_n\ E$ be a lambda expression L. The switch of L wrt i, denoted by $S[L, i]$, is the lambda expression $\lambda x_1 \ldots \lambda x_{i-1} \lambda x_{i+1} \ldots \lambda x_n \lambda x_i\ E$.

Given P a projection constraint $< E \diamond 0, i >$ and some intervals $\langle I_1, \ldots, I_n \rangle$, we are interested in the function $\Phi[P, \langle I_1, \ldots, I_n \rangle] : I(\mathcal{F}) \to I(\mathcal{F})$ defined as follows:

$$\Phi[P, \langle I_1, \ldots, I_n \rangle] = \overrightarrow{S[\mathtt{TEXT}[E], i]}\ I_1 \ldots I_{i-1} I_{i+1} \ldots I_n.$$

Informally speaking, $\Phi[< E \ \diamond\ 0, i >, \langle I_1, \ldots, I_n \rangle]$ is the interval function obtained by replacing $X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n$ by their domains in the interval extension of $\mathtt{TEXT}[E]$.

To find the leftmost and rightmost quasi-zeros of $\Phi[P, \langle I_1, \ldots, I_n \rangle]$, `Newton` uses the interval function $\Phi'[P, \langle I_1, \ldots, I_n \rangle] : I(\mathcal{F}) \to I(\mathcal{F})$ defined as

$$\Phi'[P, \langle I_1, \ldots, I_n \rangle] = \overrightarrow{S[\mathtt{DERI}[E, i], i]}\ I_1 \ldots I_{i-1} I_{i+1} \ldots I_n.$$

where $\mathtt{DERI}[E, i]$ is the lambda expression denoting the derivative of wrt x_i of the function $\mathtt{TEXT}[E]$.

We thus focus on finding the quasi-zeros of an interval function $F : I(\mathcal{F}) \to I(\mathcal{F})$ given its "derivative" $F' : I(\mathcal{F}) \to I(\mathcal{F})$. The correctness of the transformation follows from the fundamental theorem of interval arithmetic.

5.2 Newton Reduction Step

We now describe Newton reduction which can be derived from the mean value theorem as follows. Consider a function $f : \Re \to \Re$. The theorem states that $f(x) - f(y) = (x - y)f'(a)$ for some a between x and y. When y is a zero of f, we obtain $y = x - f(x)/f'(x)$. Given an interval I containing both x and y and thus a, the fundamental theorem of interval arithmetic gives $f'(a) \in F'(I)$, where F' is an interval extension of f'. Moreover, if F is an interval extension of f, then $y \in N(F, F', x, I)$ where the function N is the Newton reduction defined as $N(F, F', x, I) = \overrightarrow{x} \overrightarrow{-} F(\overrightarrow{x})/F'(\overrightarrow{I})$.

This suggests the following reduction method to bound the quasi-zeros of a function f given an initial interval I. y is a zero of f if $y \in N^*(F, F', I)$ where the function N^* is the iterated Newton reduction defined as

$N^*(F, F', I) = I_n \ \ (n \geq 1)$ `where`
$\quad I_0 = I$
$\quad I_{i+1} = N(F, F', \mathit{center}(I_i), I_i)$
$\quad I_n = I_{n-1}$

In fact, `Newton` makes use of a slightly more general procedure NR defined as follows:

$\mathit{NR}(F, F', I) = \langle s, I' \rangle$ `where`
$\quad I' = N^*(F, F', I)$
$\quad s = \mathit{success}$ `if` $I' = \{a\}$
$\quad s = \mathit{failure}$ `if` $I' = \emptyset$
$\quad s = \mathit{floundering}$ `otherwise.`

and applies it the functions defined previously.

5.3 Box Consistency of Equation

Given a projection constraint and its two associated functions F and F' as defined in section5.1 and the interval I, the first step of our narrowing operator consists in applying $\mathit{NR}(F, F', I)$ to obtain $\langle s, I' \rangle$. If $s \neq \mathit{floundering}$, box-consistency is guaranteed. Otherwise, more work may be necessary, since the left and right bounds of the intervals are not guaranteed to be quasi-zeros.

One way of proceeding is to enforce Newton reduction on the bounds of the intervals instead of on the center, i.e. the algorithm would compute

$$I_{i+1} = N(F, F', \mathit{left}(I_i), I_i) \cap N(F, F', \mathit{right}(I_i), I_i).$$

```
function ShrinkLeft( F: I(F) → I(F); F': I(F) → I(F); I: I(F)): I(F);
begin
   success := false;
   PUSH([center(I),right(I)]);
   PUSH([left(I),center(I)]);
   while ¬ EMPTY_STACK ∧ ¬ success do
      I := POP_STACK;
      ⟨ status, I ⟩ := NR(F,F',I);
      case status of
      success:
         success := true
      floundering:
         if LeftBounded(F,F',I) then
            success := true
         else
            PUSH([center(I),right(I)]);
            PUSH([left(I),center(I)]);
      endcase
   endwhile
   if ¬ success then I := ∅;
   return I
end
```

Figure 1: Function *ShrinkLeft*

after having applied N^*. This idea is in fact mentioned in [7]. Practical results however indicates that it behaves rather poorly (i.e. convergence is slow).

`Newton` uses another idea which consists in applying an internal splitting operation focusing on parts of the intervals to locate the leftmost and rightmost quasi-zeros. Informally, the idea can be described as follows for finding the leftmost quasi-zero. `Newton` splits the interval I into two intervals I_l and I_r covering I. If no quasi-zero can be found in the left interval I_l, this part can be pruned away and the algorithm can be restarted on I_r. To find a quasi-zero in a subinterval, the algorithm is applied recursively. `Newton` uses two functions *ShrinkLeft* and *ShrinkRight* and Figure 1 describes function `ShrinkLeft` in detail (*ShrinkRight* is symmetric)). *ShrinkLeft* makes use of a Boolean function *LeftBounded* defined as follows:

$$\mathit{LeftBounded}(F, F', [l, u]) \Leftrightarrow \begin{cases} 0 \in F[\overrightarrow{l}\,] \ \vee \\ 0 \in F([l, l^+]) \wedge 0 \in F'([l, l^+]) \ \vee \\ 0 \in F([l, l^+]) \wedge 0 \notin F'([l, l^+]) \wedge 0 \notin F([l^+, l^+]) \end{cases}$$

The key idea is to use a stack to explore the subintervals from left to right. The computation stops when the left bound cannot be improved.

Definition 22 [Box-Narrowing of Equations] Box-narrow($< E = 0, i >, \langle I_1, \ldots, I_n \rangle$) $= I$ where

$I = ShrinkLeft(F, F', ShrinkRight(F, F', NR(F, F', I_i)))$
$F = \Phi[< E = 0, i >, \langle I_1, \ldots, I_n \rangle]$
$F' = \Phi'[< E = 0, i >, \langle I_1, \ldots, I_n \rangle]$

5.4 Box Consistency of Inequalities

The handling of inequalities is only slightly more complex. The key idea is to test first whether the bounds satisfy the conditions, in which case no work is necessary. Otherwise, the leftmost (resp. rightmost) quasi-zero must be found. This is described as follows:

Definition 23 [Box-Narrowing of Inequalities] Box-narrow($< E \geq 0, i >, \langle I_1, \ldots, I_n \rangle$) = I" where

I' = **if** $right(F(\overrightarrow{left(I_i)})) \geq 0$ **then** I_i **else** $ShrinkLeft(F, F', I_i)$
I" = **if** $right(F(\overrightarrow{right(I')})) \geq 0$ **then** I' **else** $ShrinkRight(F, F', I')$
$F = \Phi[< E = 0, i >, \langle I_1, \ldots, I_n \rangle]$
$F' = \Phi'[< E = 0, i >, \langle I_1, \ldots, I_n \rangle]$

6 Implementation

`Newton` is a complete Prolog system extended with the constraint system and techniques described in this paper. It shares the Prolog and constraint engine (`AC-5` in particular) with `cc(FD)` [21]. Interval arithmetics is implemented with double precision IEEE floating point numbers by specifying the rounding direction. `Newton` also allows to specify the precision (i.e. the number of significant digits) required for the results. Derivatives are not computed symbolically in the current version but are obtained through automatic differentiation during the evaluation of the function [19]. This last decision may be reconsidered in a future implementation of the system, since additional accuracy may result from symbolic differentiation.

7 Experimental Results

We now describe some experimental results of `Newton`. The benchmarks were selected from textbooks and research papers on interval methods. All `Newton` programs were written in the traditional "constrain & choose" style. The first part of the program simply states the constraints while the second part applies variable splitting in a nondeterministic way. No special "intelligence" was used in the splitting phase: variables were included in a list and splitting around the center was applied on each of them in a round-robin manner until no more splitting was possible. Computation times are given on a `SUN SS-10/20` for results with an accuracy of 8 digits.

A Simple Example The first example [5] is extremely simple. It consists in finding the zeros of three closely related functions:

$$f_1(x) = x^4 - 12x^3 + 47x^2 - 60x$$
$$f_2(x) = x^4 - 12x^3 + 47x^2 - 60x + 24$$
$$f_3(x) = x^4 - 12x^3 + 47x^2 - 60x + 24.1$$

The functions highlight the virtues of interval arithmetic, since $0, 3, 4, 5$ are zeros of f_1, $0.888\ldots$ and 1 are zeros of f_2 and f_3 has no zeros. By constraint propagation alone, `Newton` returns the intervals $[0.0, 5.0]$ and $[0.88830577, 1.0]$ for f_1 and f_2 respectively and concludes that f_3 has no zero. The computing times are a couple of milliseconds.

	5	10	20	40	80	160
Times $[-1,1]$	1.160	8.810	25.920	61.650	127.750	264.610
Growth $[-1,1]$		7.59	2.94	2.38	2.07	2.07
Times $[-10^8,10^8]$	1.270	10.650	39.900	93.640	201.520	402.230
Growth $[-10^8,10^8]$		8.39	3.75	2.35	2.15	2.00

Table 1: Computation Results of `Newton` on the Broyden Banded function

	1	2	4	6	8	10	12
CLP(Intervals) Nodes	51	371	2961	20099	103171	517426	2537849
CLP(Intervals) Operations	500	10,265	150,019	1,163,439	9,099,305	50,085,952	257,631,995
CLP(Intervals) Growth		20.53	4.10	3.19	2.39	2.32	2.27
`Newton` Nodes	1	1	1	1	1	1	1
`Newton` Operations	2759	18,239	158,400	679,892	1,674,778	3,132,860	4,973,013
`Newton` Growth		6.61	2.53	2.17	1.47	1.33	1.24

Table 2: Comparison of `Newton` and traditional CLP(Intervals)

Broyden Banded Function This example was used as a benchmark in [6] and consists in finding the zeros of the functions

$$f_i(x_1,\ldots,x_n) = x_i(2+5x_i^2)+1-\textstyle\sum_{j\in J_i} x_j(1+x_j) \quad (1 \le i \le m)$$

where $J_i = \{j \mid j \neq i \ \& \ max(1,i-5) \le j \le min(m,i+1)\}$.

The results of `Newton` are shown in Table 1. In the table, we give computation times for the initial intervals in $[-1,1]$ (first line) and $[-10^8,10^8]$ (third line) as well as the growth of the times (i.e. *time(2i)/time(i)*). There are three important features of this `Newton` program. First, it is completely deterministic, i.e. box-consistency alone solves the problem and no splitting is necessary. Krawczyk method and its extensions do not have this property. Second, the `Newton` program is linear in the number of variables and we are not aware of any other algorithm with this property. Finally, the behaviour is essentially the same when the initial intervals are very large and the computation times increase by less than a factor 2.

It is also interesting to compare `Newton` to a traditional CLP(Intervals) implementation. Table 2 reports the number of floating-point operations and nodes in the search tree for both implementations to allow a computer-independent comparison. The results indicates that `Newton` is faster as soon as the number of variables n is greater than 4. It is already 50 times faster for $n = 12$. The growth of $CLP(intervals)$ is exponential, since it converges towards 2. `Newton` converges toward 1. Note also the large number of nodes in the traditional CLP(Intervals) implementation.

More-Cosnard Example This example comes from [15] and is also a standard benchmark for interval methods. It consists in finding the root of the functions $(1 \le k \le m)$:

$$f_k(x_1,\ldots,x_m) = x_k + 1/2[1-t_k)\textstyle\sum_{j=1}^{k} t_j(x_j+t_j+1)^3 + t_k \sum_{j=k+1}^{m}(1-t_j)(x_j+t_j+1)^2]$$

where $t_j = jh$ and $h = 1/(m+1)$. These functions come from a nonlinear integral equation. The variables x_i have initial domains $[-4,5]$ and the computation results are given in Table 3. Once again, it is interesting to note that, for $k \le 8$, box-consistency alone solves the problem. Note that there is no splitting either in Krawczyk methods.

	4	5	6	7	8
Times	0.67	1.74	3.43	6.20	11.60
Growth		2.60	1.97	1.81	1.87

Table 3: Computation Results of `Newton` on the More-Cosnard

Kinematics The last example we mention is the toughest of a recent report on interval methods [8]. It comes from robotics and, in particular, from the inverse kinematics of an elbow manipulator. It can be stated as follows:

$$
\begin{array}{l}
s_2 \times c_5 \times s_6 - s_3 \times c_5 \times s_6 - s_4 \times c_5 \times s_6 + c_2 \times c_6 + c_3 \times c_6 + c_4 \times c_6 = 0.4077 \\
c_1 \times c_2 \times s_5 + c_1 \times c_3 \times s_5 + c_1 \times c_4 \times s_5 + s_1 \times c_5 = 1.9115 \\
s_2 \times s_5 + s_3 \times s_5 + s_4 \times s_5 = 1.9791 \\
c_1 \times c_2 + c_1 \times c_3 + c_1 \times c_4 + c_1 \times c_2 + c_1 \times c_3 + c_1 \times c_2 = 4.0616 \\
s_1 \times c_2 + s_1 \times c_3 + s_1 \times c_4 + s_1 \times c_2 + s_1 \times c_3 + s_1 \times c_2 = 1.7172 \\
s_2 + s_3 + s_4 + s_2 + s_3 + s_2 = 3.9701 \\
s_i^2 + c_i^2 = 1 \ (1 \leq i \leq 6).
\end{array}
$$

The initial intervals are $[-1, 1]$. [8] mentions that the problem has an analytical solution but remains an interesting benchmark for interval methods. `Newton` finds the solution in 454 seconds after 864 splittings. [8] compares the implementations of two methods both implemented in $C++$ on a Silicon graphics MIPS 4000/4010, a faster machine. The method described in [7] takes 2395 seconds and 29,189 splittings. The method of [8] requires 117.10 seconds and 257 splittings. These results are rather encouraging for `Newton`, since this last work was developed independently and at about the same time as `Newton` (see the discussion).

8 Related Work and Discussion

Interval Methods As stated, box-consistency is enforced by using the interval Newton method to find the leftmost and rightmost quasi-zeros. Note however that `Newton` does not apply Newton interval method to the interval extension of a real function but rather to an interval function coming from the projection of an interval function. It is also interesting to look at all constraints as a whole and to compare box-consistency with Krawczyk's method and its successors [10, 7, 6]. On the one hand, Krawczyk's methods achieve more pruning since they manipulate all constraints as a whole, allowing a more precise evaluation of the interval functions. On the other hand, box-consistency provides additional pruning by obtaining tight bounds on the intervals. Both methods seem complementary and a combination of them should lead to even more efficient algorithms. This is also indicated by recent and independent work in [8] which proposes an extension of Krawczyk's method, called tightening operators, which has some similarity with box-consistency. A tightening operator is used to replace a tuple of intervals by a set of tuples obtained by applying a projection constraint on one of the intervals. [8] also proposes a tightening operator which uses Newton interval method to find all quasi-zeros of a projection constraint and use them to replace the initial interval. Box-consistency can in fact be viewed as a coarser tightening operator replacing a tuple by another tuple of intervals. It is interesting to observe that the work in [8] and ours started from very different perspectives to arrive at related solutions. Krawczyk's method generally requires that the Jacobian be diagonally dominant (i.e. the diagonal elements be large compared to other elements) to be effective. This is also the case for `Newton` (when viewing the constraints globally), except that, in `Newton`, it is sufficient that the system of constraints be diagonally dominant to a permutation of the columns of the Jacobian. The typical way around this limitation is to apply a preconditioning step to the Jacobian. Future work on `Newton` will aim at including the preconditioning step and the additional precision coming from a global view of the constraints while preserving an elegant operational semantics for the language. Note finally that W. Older mentioned the potential of Krawczyk's method for CLP languages [16].

CLP Systems Several CLP(Intervals) systems have been implemented in the past as mentioned in the introduction and `CLP(BNR)` is probably the most advanced system. As far as real constraints are concerned, its primitive constraints are of the form:

$t_1 = t_2 + t_3$, $t_1 = t_2 \times t3$, $t_1 = t_2 - t_3$, $t_1 = t_2/t_3$
$v \in [a_1, a_2]$, $t_1 \geq t_2$, $t_1 = cos(t_2)$, $t_1 = sin(t_2)$, ...

where t_i is a variable or a constant, v is a variable, and a_1, a_2 are constants. All variables appearing in a primitive constraint are distinct but the functions need not be differentiable (e.g. `max`). Complex constraints are rewritten into a set of primitive constraints. For instance, the constraints $v_1 = v_2 - v_2$ is rewritten[4] into the set $\{v_1 = v_2 - v_3, v_2 = v_3, v_3 \in [-\infty, \infty]\}$. `CLP(BNR)` enforces interval-consistency on its primitive constraints and the loss of precision due to the dependency problem only occurs in the fixpoint algorithm. On our simple example, `CLP(BNR)` does not produce any pruning. The precision of `Newton` and `CLP(BNR)` is not directly comparable. On the one hand, `CLP(BNR)` may be slightly more precise on simple constraints, since inverse functions are used removing one floating point operations. On the other hand, `Newton` by manipulating constraints as a whole can improve accuracy substantially compared to `CLP(BNR)` by reducing the dependency problem. However, precision was not the major goal of `Newton`. The main goal was to show that more advanced narrowing operators could be defined preserving an elegant compromise between efficiency and simplicity of the language. Note also that `CLP(BNR)` also contains Boolean and integer constraints over intervals. These constraints, as well as non-differentiable functions, can easily be integrated in `Newton` (since the underlying architecture of the two systems is the same) and are a main priority of our future work.

Other Systems There are many other systems based on intervals in other areas [4, 9]. Some mention the adequacy of working with specific forms of constraints (e.g. centered forms) but, to our knowledge, no system computes an approximation of arc-consistency using Newton interval method.

9 Conclusion

This paper revisited the design and implementation of CLP(Intervals). It presented the new CLP(Intervals) language `Newton` which enforces box-consistency, an approximation of arc-consistency, on complex constraints. The operational semantics has been presented in terms of box-consistency in the same style as previous CLP(Intervals) languages. `Newton` enforces box-consistency using a special-purpose interval Newton method on projection constraints. Preliminary experiments on

[4] This is the theory. Actually the `CLP(BNR)` system computes some simple algebraic transformations to achieve basic constraint simplifications.

some standard benchmarks indicate that `Newton` can produce dramatic improvements in efficiency over traditional implementations and compares well with some special-purpose tools. Limitations of our current implementation and directions for further work are also identified.

Acknowledgment

Discussions with Alain Colmerauer are gratefully acknowledged.

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, 1983.

[2] F. Benhamou and W. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 1993. (Submitted).

[3] J.G. Cleary. Logical Arithmetic. *Future Generation Computing Systems*, 2(2):125–149, 1987.

[4] E. Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32:281–331, 1987.

[5] J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Non-linear Equations.* Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[6] E.R. Hansen and R.I. Greenberg. An Interval Newton Method. *Appl. Math. Comput.*, 12:89–98, 1983.

[7] E.R. Hansen and S. Sengupta. Bounding Solutions of Systems of Equations Using Interval Analysis. *BIT*, 21:203–211, 1981.

[8] H. Hong and V. Stahl. Safe Starting Regions by Fixed Points and Tightening. Submitted for Publication, November 1993.

[9] E. Hyvönen. Constraint Reasoning Based on Interval Arithmetic. . In *Proceedings of IJCAI 1989*, 193–198, 1989.

[10] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4:187–201, 1985.

[11] J.H.M. Lee and M.H. van Emden. Interval Computation as Deduction in CHIP. *Journal of Logic Programming*, 16(3-4):255–276, 1993.

[12] A.K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[13] U. Montanari. Networks of Constraints : Fundamental Properties and Applications to Picture Processing. *Information Science*, 7(2):95–132,1974.

[14] R.E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[15] J.J. More and M.Y. Cosnard. Numerical Solution of Nonlinear Equations. *ACM Transactions on Mathematical Software*, 5:64–85, 1979.

[16] W. Older. Krawczyk Derivatives. Unpublished Note, 1991.

[17] W. Older and F. Benhamou. Programming in CLP(BNR). In *PPCP'94*, Newport, RI (USA), 1993.

[18] W. Older and A. Vellino. *Constraint Arithmetic on Real Intervals.* In *Constraint Logic Programming: Selected Papers*, F. Benhamou & A. Colmerauer eds., The MIT Press, Cambridge, MA, 1993.

[19] L.B. Rall. *Automatic Differentiation: Techniques and Applications.* Springer Lectures Notes in Computer Science, Springer Verlag, New York, 1981.

[20] G. Sidebottom and W. havens. Hierarchical Arc Consistency Applied to Numeric Processing in Constraint Logic Programming. *Computational Intelligence*, 8(4), 1992.

[21] P. Van Hentenryck, V. Saraswat, and Y. Deville. The Design, Implementation, and Evaluation of the Constraint Language `cc(FD)`. Technical Report, Brown University, December 1992.