Analytic Constraint Solving and Interval Arithmetic

Timothy J. Hickey

Michtom School of Computer Science, Brandeis University, USA

Abstract

In this paper we describe the syntax, semantics, and implementation of the constraint logic programming language CLP(F) and we prove that the implementation is sound. A CLP(F) constraint is a conjunction of equations and inequations in a first order theory of analytic univariate functions over the reals. The theory allows vector-valued functions over closed intervals to be constrained in several ways, including specifying functional equations (possibly involving the differentiation operator) that must hold at each point in the domain, arithmetic constraints on the value of a function at a particular point in its domain, and bounds on the range of a function over its domain. After describing the syntax and semantics of the constraint language for CLP(F) and giving several examples, we show how to convert these analytic constraints into a subclass of simpler functional constraints which involve neither differentiation nor evaluation of functions. We then present an algorithm for solving these latter constraints and prove that it is sound. This implies the soundness of the CLP(F) interpreter. We also provide some timing results from an implementation of CLP(F) based on GNU Prolog. The current implementation is able to solve a wide variety of analytic constraints, but on particular classes of constraints (such as initial value problems for autonomous ODEs), it is not competitive with other non-constraint based, interval solvers such as Lohner's AWA system. CLP(F) should be viewed as a first step toward the long term goal of developing a practical, declarative, logic-based approach to numerical analysis.

1 Introduction

Numerical computation involves computing floating point approximations to a finite set of real values which are defined in terms of a mathematical model $\mathcal{M}$. This model typically involves fairly complex mathematical objects such as multidimensional real or complex functions, partial differential equations, integral equations, Hilbert space operators, etc., but these higher level objects are used simply to place constraints on the values of the computed floating point variables.

We take the view that the goal of most numerical computation is to solve a system of constraints of the form $\exists f.\mathcal{C}(x,f)$ where x is a vector of real variables, f is a vector of variables ranging over other mathematical objects (functions, sets, operators, functionals, etc.) and $\mathcal{C}$ is a conjunction of constraints on x and f. The particular constraint language depends on the problem but usually includes differential equations (ordinary or partial), integration, vectors, and functional evaluation.

Conventional numerical analysis typically attempts to find approximate solutions to such constraints, but determining a posteriori error bounds is often exceedingly difficult to do rigorously. We are more interested in methods which generate answers with guaranteed error bounds, in the sense that the algorithm computes the solution as a union of intervals such that each mathematical solution of the constraint lies within one of the intervals.

In this paper we describe the language CLP(F), which represents a first step toward the long term goal of developing a suite of practical tools for this type of constraint-based numerical computation. An early version of the CLP(F) language was informally described (without proofs) in a workshop [9]. Since that time, the CLP(F) constraint language has been expanded to include general evaluation and range bound constraints, and it has been ported to GNU prolog [8] for the Linux x86 platform. In this paper we describe the current CLP(F) language and its implementation and we provide a proof of the soundness of the CLP(F) constraint solver.

The CLP(F) language is an example of a new approach to scientific programming which we call analytic constraint logic programming (ACLP). The idea behind ACLP is that it provides an interval-based constraint language in which higher order mathematical objects (e.g. ODEs, PDEs, function transforms, etc.) can be used to define scientifically interesting constraints on real numbers.

2 Validated Numerical Computing

To better highlight the importance and difficulty of solving analytic constraints we first present a short overview of the current approaches to validated numeric computing.

Condition numbers. The traditional approach to proving the correctness of numerical algorithms is best exemplified in the recent paper by Cucker and Smale on condition numbers for semi-algebraic systems [6]. They consider the problem of solving systems of polynomial equations and inequations and they propose an algorithm which is correct with respect to an ideal machine which can perform real computation without roundoff error. They then compute a condition number for the problem which determines a minimum level of precision in a machine computation needed to obtain a given level of precision in the result, the condition number (if finite) also can be used to bound the computation time. The irony of this approach (as Cucker and Smale acknowledge) is that it seems to be at least as difficult, in general, to compute the condition number as to solve the original problem.

From a practical point of view, Acton's monograph [1] is more reflective of the general practice of numerical analysis among scientists. The text is filled with sage advice about avoiding situations that lead to catastrophic roundoff error, but the idea of proving the (mathematical) correctness of a computed result is not even mentioned.

Interval arithmetic. An alternate approach to numerical computation pioneered by Moore in the 60's [19, 20] is to use interval arithmetic to bound the roundoff error in each step. This requires making use of the underlying directed rounding operations of the CPU so that the set resulting from using interval arithmetic operations to evaluate an arithmetic expression on floating point numbers is guaranteed to contain the mathematical result of that evaluation [13]. Multiple-precision arithmetic is also sometimes used in interval arithmetic systems. For example, in [14] a C toolbox is described which provides access to interval variables and a rich set of interval operators. This approach has been used to implement algorithms for solving a wide range of mathematical problems, including ODEs, PDEs, integral equations, etc. [5]. One disadvantage of the interval arithmetic approach is that the programming language becomes more complex. Not only must one use the various floating point and integer variables, but one must now also employ interval variables.

One of the most promising applications of interval arithmetic is to the problem of computing validated solutions to the initial value problem for ordinary differential equations. Considerable success has been achieved in developing relatively efficient techniques that produce tight (and validated) bounds on the solutions of a wide class of ODEs [19, 21]. The most effective system currently available is Lohner's AWA system [18].

Numerica. The Numerica system [25] is an excellent example of a commercially viable interval-based system. It has been specially designed for solving non-linear global optimization problems. Numerica is competitive with traditional methods in terms of performance, but it produces qualitatively superior results. For example, it can be proved that the underlying algorithms are guaranteed not to miss any of the solutions of a non-linear system. Moreover, the potential solution set is represented as a finite set of boxes. For some boxes, the system can prove that there exists a single solution in the box. This is important for global optimization problems.

The global optimization methods introduced in Numerica have also been applied to the problem of solving certain classes of ODEs with encouraging results [7, 16].

Interval Arithmetic Constraints. A more recent approach to numerical computation is to build a general interval-based constraint solver into the language itself and then to use this language to build solvers for particular classes of constraints. In these languages, the mathematical model appears exactly as a set of constraints interspersed throughout the program. Moreover, since the built-in solver will likely be used to solve many problems, considerable effort can be put into verifying its correctness and optimizing it on various platforms.

For example, the CLP(BNR) system [24, 2] is a constraint logic programming language which supports interval arithmetic constraints. It was developed in the late 80's at Bell Northern Research and has been used to build solvers for many types of constraint problems, including scheduling and numerical integration ([4], Ch.12).

Recently, a prototype of a small subset of the Numerica solver has been implemented in a few pages of code written in the CLP(intervals) language CLIP, [11]. The resulting code is easy to prove correct, but is currently not as efficient as Numerica.

Overview. In this paper we consider a natural extension of the interval arithmetic paradigm in which the constraint language is extended to include function variables which range over the space of infinitely differentiable functions on a bounded interval. The language is also extended to include various function operators (such as differentiation and evaluation at a point) and, as a notational convenience, we allow vector variables and vector operations (with the operations defined componentwise). The vector extensions can be easily translated into scalar and function operations, and they significantly improve the expressiveness of the language.

The main theoretical result of this paper is the correctness proof of a simple constraint solving algorithm which reduces analytic constraint solving to a variant of interval arithmetic constraint solving. We also provide several examples which demonstrate the viability of this approach.

3 The CLP(F) Constraint Language

Let $C_\infty([a,b])$ denote the set of all real valued functions on the closed and bounded interval $[a,b]$ which are infinitely differentiable. In this section we present a constraint language which can be used to constrain functions $f \in C_\infty([a,b])$ and real variables $x \in \mathcal{R}$. We will actually present three nested languages. At the lowest level is the language of arithmetic constraints on reals. We expand this language to form the language of functional constraints by adding function variables, arithmetic functional equations, and range constraints. Finally, we define the language of analytic constraints by adding a differentiation operator, an operator to evaluate a function at a point, and vector variables and operators to the functional constraint language.

3.1 Arithmetic Constraints

Arithmetic constraints are simply atoms in a particular first order theory of the reals. This theory contains some specified set of relation, function, constant, and variable symbols. In the current version of CLP(F) the allowable symbols include:

```
Relations:  X=Y, X<Y, X=<Y, X>Y, X>=Y, X\==Y
Constants:  pi, 0, -1, 1.23, 1.0E100, ...
Functions: X+Y, X-Y, X*Y, X/Y, -X, abs(X),
```

```
          exp(X), log(X), X^n, X**Y,
          sin(X), cos(X), tan(X),
          asin(X), acos(X), atan(X),

Variables: X,Y,... (any capitalized identifiers)
```

3.2 Functional Constraints

Functional constraints are an extension of arithmetic constraints in which we allow both real and function variables. These variables are declared using type constraints. By default, all untyped variables are considered to be reals. We also allow any arithmetic constraint as in the previous section, but where the variables can be of either real or function type. Finally, we allow range constraints which specify that a variable x or the range of a function f is contained in the range of g. The syntax of these constraints is described below.

Type Constraints. The type constraint has the form:

```
type([F1,...,Fn],TYPE)
```

where the `Fi` are variables and `TYPE` is a type specification which is either `real` for real variables or `function(A,B)` for infinitely differentiable functions on the interval $[A, B]$.

Functional Equations. In CLP(F), a relation between function variables is interpreted to hold pointwise. Thus, the constraint

```
F = G
```

where `F,G` are expressions in the typed language of analytic constraints, holds if F and G are explicitly declared to be defined on the same domain D, and if $F(t) = G(t)$ for all $t \in D$. An equation $F = X$ where X is real variable is transformed into a functional equation by viewing the real X as a constant function, i.e., it asserts that $F(t) = X$ for all t in the domain of F. Similarly, the relation `F < X` is interpreted to mean that $F(t) < X$ for all t in the domain of F.

Range Constraints. Once a function has been declared, CLP(F) allows a limited set of operations which can be used to constrain the range of that function. For example, the constraint

```
F in G,
```

constrains F, G to be functions on the same domain D such that $F(D) \subset G(D)$. We call $F(D)$ the range of F and denote it $range(F)$. Similarly, we can constrain a real variable X to lie in the range of a function F using the constraint:

```
X in F
```

Finally, real numbers and real functions can be constrained to take values in the interval $[A, B]$ using the following constraints:

```
X in [A,B],    F in [A,B]
```

where `A,B` are real variables or constants. The `[A,B]` notation is a notational convenience. We could express these two constraints without the `[A,B]` notation as follows:

```
X in G, F in G,  A=<G,  G=<B
```

where `G` is an auxiliary function variable with the same domain as `F`.

3.3 Analytic Constraints

We define analytic constraints by adding three features to functional constraints: a differentiation operator, a point evaluation operator on functions, and vector variables. The precise syntax is described below.

The Differentiation Operator. Since we are assuming that all functions are infinitely differentiable, if $f \in C_\infty([a, b])$, then its nth derivative $f^{(n)}$ is also. In CLP(F), we use the expression

```
ddt(F,n)
```

to represent the nth derivative of F, where n must be an integer constant.

Evaluation Constraints. Functions can also be constrained by constraining their values at given points. This is done using the evaluation constraint:

```
eval(F,X) = Y
```

where `X` and `Y` are real variables.

Arithmetic Vector Constraints. The syntax for type declaration is extended to have the form:

```
type([F1,...,Fn],TYPE)
```

where `TYPE` is a type specification which is either `real` for real variables, `function(A,B)` for infinitely differentiable functions on the interval $[A, B]$, or `vector(N,TYPE)`, where `N` is an integer constant, and `TYPE` is a type. Thus, the type language for analytic constraints is given by

$$T \;\rightarrow\; \texttt{real} \mid \texttt{function}(a, b) \mid \texttt{vector}(n, T)$$

where a, b are real constants or variables, and n is an integer constant.

We have also included three vector functions: the norm, dot product, and vector constructor.

```
norm(V), dot(V,W), v([A1,....,An])
```

In the last, the expressions `Ai` must all have the same type.

Operator overloading. To increase the expressiveness of the language we have allowed the arithmetic operators to be overloaded so that, for example, one can use the same operator "*" to express multiplication of two reals, two functions, a real and a function, a real and a vector, etc. This overloading is defined by the coercion rules shown in Fig. 1.

The Identity Function. Finally, as a notational convenience we introduce a predicate `identity(T)`, for a function variable T, which is true if `T` is the identity function. This relation is equivalent to

```
ddt(T,1)=1, ddt(T,2)=0,
eval(T,a)=a, eval(T,b)=b,
T in [a,b]
```

assuming `T` is of type `function(a,b)`.

- $g(f)$ denotes the function $t \mapsto g(f(t))$
- $g(v)$ denotes the vector $(g(e_1), \ldots, g(e_n))$
- $x \diamond f$ denotes the function $t \mapsto x \diamond f(t)$
- $f \diamond x$ denotes the function $t \mapsto f(t) \diamond x$
- $f_1 \diamond f_2$ denotes the function $t \mapsto f_1(t) \diamond f_2(t)$
- $x \diamond v$ denotes the vector $(x \diamond e_1, \ldots, x \diamond e_n)$
- $f \diamond v$ denotes the vector $(f \diamond e_1, \ldots, f \diamond e_n)$
- $v \diamond x$ denotes the vector $(e_1 \diamond x, \ldots, e_n \diamond x)$
- $v \diamond f$ denotes the vector $(e_1 \diamond f, \ldots, e_n \diamond f)$
- $v \diamond v'$ denotes the vector $(e_1 \diamond e'_1, \ldots, e_n \diamond e'_n)$

Figure 1: Coercion rules for arguments of non-real type where $x, f, v = (e_1, \ldots, e_n)$ are real, function, and vector expressions, respectively and $g(x)$ (resp. $x \diamond x'$) denotes a real function of one (resp. two) arguments.

3.4 The Semantics of Analytic Constraints

Let $\mathcal{C}(X, F)$ be an analytic constraint where $X = (X_1, \ldots, X_p)$, and $F = (F_1, \ldots, F_r)$ are tuples of real and function variables respectively. We let $\mathcal{S}_{an}(\mathcal{C}(X, F))$ denote the set of all solutions to this constraint:

$$\mathcal{S}_{an}(\mathcal{C}(X, F)) = \{(x, f) \in \mathcal{R}^p \times C^r_\infty : \mathcal{C}(x, f)\}$$

An interval-based analytic constraint solver is a function $\mathcal{A}$ which maps such an analytic constraint $\mathcal{C}(X, F)$ to a p-tuple I of intervals. We say that such a solver is **sound** if for every solution (x, f) of $\mathcal{C}(X, F)$ we have $x \in \mathcal{A}(\mathcal{C}(X, F))$. Formally, $\mathcal{A}$ is a **sound** interval-based analytic constraint solver if

$$\pi_X(\mathcal{S}_{an}(\mathcal{C}(X, F))) \subset \mathcal{A}(\mathcal{C}(X, F))$$

holds for every analytic constraint $\mathcal{C}(X, F)$.

4 Examples of Analytic Constraints

In this section we present several examples of analytic constraints and we show the answer constraints (and elapsed time) generated by the clip-1.0.2 implementation of CLP(F) running on a 500MHz Gateway 200 Pentium 3 PC under Linux RedHat 6.0. This interpreter is available [10] under a GPL copyleft license. It is written in the CLP(intervals) language CLIP, [11], which is itself a foreign file extension of GNU Prolog [8].

Taylor Constraints. The simplest analytic constraints involve inferring properties of the values of a function at specified points, based on knowledge about its values at other points and about the ranges of some of its derivatives. The fundamental result in this context is Taylor's Formula with Remainder (stated below) and so we call these Taylor constraints.

Lemma 1 (The Taylor Formula with Remainder) *Let $f \in C_\infty([a, b])$ where $I = [a, b]$ is a closed bounded interval. For any $t \in I$ and any $n \geq 0$, there is a function $\xi_{f,n,t}$, called Taylor's error function, which satisfies the following properties for any $x \in I$,*

$$\xi_{f,n,t}(x) \in I$$

$$f(x) = \left(\sum_{i=0}^{n-1} \frac{f^{(i)}(t)}{i!}(x - t)^i \right) + \frac{f^{(n)}(\xi_{f,n,t}(x))}{n!}(x - t)^n$$

This last equation is called the Taylor formula for f of degree n centered at t.

Consider, for example, the problem of making inferences about the possible value of $f(0.5) = x$, where $f \in C_\infty([0, 1])$ is a function which is zero at 0 and 1, and whose second derivative lies in a specified interval S, i.e., $f''([0, 1]) \in S$. Mathematically, this problem is captured by the predicate C defined as follows:

$$\begin{aligned} C(x, a, b) \quad \equiv \quad & \exists f \in C_\infty([0, 1]) \\ & f(0) = f(1) = 0, \quad f(0.5) = x \\ & \forall t \in [0, 1], a \leq f''(t) \leq b \end{aligned}$$

We can solve this problem in CLP(F) for $S = [16, 20]$ by posing the following query:

```
?-  type([F],function(0,1)),
  {[ddt(F,2) in [16,20],
    eval(F,0)=0, eval(F,1)=0,
    eval(F,0.5)=X]}.
>> X = [-3,-1.5]    (0.06 sec)
```

Note that in CLP(F), the type constraints appear before the functional equations, which must be enclosed in `{[...]}` curly-square brackets.

We could also express the predicate $C(x, a, b)$ as a CLP(F) procedure as follows:

```
c(X,A,B) :-
    type([F],function(0,1)),
  {[ddt(F,2) in [A,B],
    eval(F,0)=0, eval(F,1)=0,
    eval(F,0.5)=X]}.
```

Once this procedure is consulted, it can used to solve the specified problem as follows:

```
?- c(X,16,20).
>> X = [-3,-1.5]    (0.05 sec)
```

The CLP(F) interpreter sets up these constraints and infers that X is in the range $[-3, -1.5]$. The interpretation of this answer is

$$\forall x \in R.\ C(x, 16, 20) \Rightarrow (-3 \leq x \leq -1.5)$$

This follows easily from the Taylor formula with remainder. Indeed, using the Taylor formula of degree two, we see that if f is any solution to the constraint then we must have

$$f(1) = f(0) + 1 * (f'(0) + 1/2 * f''(\xi_{f,2,0}(1)))$$

Since $\xi_{f,2,0}(1) \in [0, 1]$, and $range(f'') \subset [16, 20]$, and $f(1) = f(0) = 0$, we infer that

$$0 = f'(0) + 1/2 * z, \quad z \in [16, 20]$$

So $f'(0) \in [-10, -8]$. We can now apply the Taylor formula again to compute $f(0.5)$:

$$\begin{aligned} f(0.5) &= f(0) + 0.5 * (f'(0) + 0.5/2 * (f''(\xi_{f,2,0}(0.5)))) \\ &\in 0.5 * ([-10, -8] + 0.25 * [16, 20]) \\ &= 0.5 * (-10, -8] + [4, 5]) \ = \ [-3, -1.5] \end{aligned}$$

In this case, two instances of the Taylor formula were applied to infer the constraint on X. This is typical of the current CLP(F) implementation in which we simultaneously assert several instances of the Taylor formula whenever a function is declared or a function is evaluated at a point.

Automatic Differentiation. Perhaps the simplest example of solving constraints using functional equations is automatic differentiation, as illustrated by the following example which computes the second derivative of $\exp(\sin(x))$ at 0.5:

```
?-  type([F,T],function(0,1)), identity(T),
  {[F = exp(sin(T)), eval(ddt(F,2),0.5)=X]}.
>> X =  0.469564...    (0.4 sec)
```

Note that the final ellipsis (...) indicates that all decimal digits to that point are correct, i.e.,

```
X =  0.469564...    iff  X in [0.469564,0.469565)
```

The user can set various tuning parameters that will affect the number of digits of precision produced by the solver. For example, if the directive `set_degree(25)` had been issued before the previous example, the system would have used a degree 25 Taylor series to approximate the functions `F,T`, rather than the default degree 20 approximation and hence would have returned the more precise result `0.469564399265734...` in 1.52 seconds, which is an interval whose width is two units in the last place (2 ULP), i.e., it contains just three double precision numbers.

We can just as easily use CLP(F) to compute the points at which one of the derivatives of f has a particular value, say $f''(x) = 0.0$:

```
?-  type([F,T],function(0.5,1.0)), identity(T),
  {[F = exp(sin(T)), eval(ddt(F,2),X)=0.0]}.
>> X = 0.66623943249251...    (0.42 sec)
```

Integration. Integration is also fairly easy to describe as an analytic constraint. For example the following predicate constrains I to be the integral of F from A to B,

```
integrate(F,T,A,B,I) :-
  type([G,T],function(A,B)),identity(T),
{[ddt(G,1)=F, eval(G,A)=0, eval(G,B)=I ]}.
```

For example, we compute $\int_{0.2}^{0.4} e^{sin(t)}dt$ by

```
?- integrate(exp(sin(T)),T,0.2,0.4,X).
>> X = 0.26904062715668... (0.41 sec)
```

Constrained ODEs. Finally, we present a few examples of ODE constraints related to the 2D problem of determining the astrodynamics of a satellite orbiting a central body. Let $p(t)$ denote the relative position of the satellite at time t and assume that none of the position and velocity components $p_x(t), p_y(t), p'_x(t), p'_y(t)$ have magnitudes exceeding N over the interval. Consider the relationship among the gravitational constant g of the system, the bound N, and the positions p_i at three times $t_1 < t_2 < t_3$. This is a relationship among 3 vector variables (p_i) and five real variables g, t_1, t_2, t_3, N, which is captured by the following analytic constraint: $\gamma_N(t_1, p_1, t_2, p_2, t_3, p_3, g, N) \equiv$

$$
\begin{aligned}
\exists f \in C_\infty([t_1,t_3])^2. \quad \text{s.t.} \quad & f'' = -g * f / \parallel f \parallel^{3/2} \\
\wedge \quad & \forall t \in [t_1,t_3]. f(t), f'(t) \in [-N, N] \\
\wedge \quad & f(t_1) = p_1 \\
\wedge \quad & f(t_2) = p_2 \\
\wedge \quad & f(t_3) = p_3
\end{aligned}
$$

This constraint can be modeled by the following CLP(F) procedure:

```
gauss(T1,P1,T2,P2,T3,P3,G,N) :-
  type([F,F1,F2],v(2,function(T1,T3))),
{[F1=ddt(F,1), F2 = ddt(F,2),
  F2 = - G* F/norm(F)**1.5,
  F1 in [-N,N],
  F  in [-N,N],
  eval(F,T1)= P1,
  eval(F,T2)= P2,
  eval(F,T3)= P3]}.
```

One interesting application of this constraint is to let $G = 1.0$ and $N = 4$ and determine the position of the body at a specified time, say $t_2 = 0.05$ from its position at two other specified times, say $p(0.0) = (0, 1)$ and $p(0.10) = (0.03, 1.04)$. This can be solved by the following CLP(F) query:

```
?-  gauss(0.00,v([0,1]),
          0.05,v([A,B]),
          0.10,v([0.03,1.04]),
          1.0,4.0).
>> A =  0.01501745951836...
   B =  1.02119889722410... (5.53 sec)
```

Perhaps more interesting is the fact that we can also give `P1,P2,P3` and `T1,T3` and solve for `T2` and `G`.

```
?-{G in [0.5,10.0]},
  gauss(0.0,v([0.00,1.00]),
          X,v([0.01501,1.02222]),
        0.1,v([0.03,1.04]),
        G,4.0).
>> X = 0.049924149117...
   G = 1.88019823...  (16.46 sec)
```

As we will see later, the CLP(F) constraint solver does not do any analysis of the problem to determine, e.g., that the first ODE example was an ODE shooting problem and the second was a parameter determination problem. The solver simply converts the analytic constraint to a set of primitive interval arithmetic constraints and then iteratively applies general contractors corresponding to these constraints to reduce the size of the intervals until a fixed point is reached.

5 Comparison of Analytic Constraints and IA methods

In this section we consider an example of an initial value problem for a simple ODE and show two approaches to solving this ODE soundly. The first uses the standard interval arithmetic approach first proposed by Moore [19] and now used in the AWA system [18], the second uses the interval constraint approach advocated in this paper.

Solvers based on the interval arithmetic approach (e.g. AWA) are far superior to CLP(F) for this class of problem. They are faster, provide greater accuracy, and can also prove that the ODE has a solution. The CLP(F) system to its credit can handle a wider variety of problems involving analytic constraints and has a simple implementation which makes it easier to check for correctness of the implementation.

5.1 An Example

Consider the following simple analytic constraint $\mathcal{C}$:

```
?- type([F],function(0,0.1)),
 {[ddt(F,1)= -F**2,  eval(F,0)=1, eval(F,0.1)= X,
   F in [0,1]]}.
```

With our semantics for analytic constraints, a solution to this query is a pair (x, f) where x is a real number, and f is an infinitely differentiable function on $[0, 1]$, which satisfies

$$f' = -f^2, \tag{1}$$
$$f(0) = 1, \tag{2}$$
$$f(0.1) = x, \tag{3}$$
$$\forall s \in [0, 0.1].\ f(s) \in [0, 1] \tag{4}$$

A sound answer to the analytic constraint $\mathcal{C}$ is an interval $I \subset \mathcal{R}$ which contains all x for which there is an infinitely differentiable function f on $[0, 0.1]$ with (x, f) satisfying Eqs. (1)-(4).

There is only one solution (x, f) to this analytic constraint, and it has $f(s) = (1+s)^{-1}$ and $x = 10/11$. The CLP(F) interpreter returns an answer

```
>> X = 0.909090909090...   (0.41 sec)
```

which is a narrow interval containing 10/11.

This type of interval-based initial value problem (IVP) has been intensively studied by several interval arithmetic researchers over the past thirty years (See, for example [21], for a recent survey of such techniques).

In the next two sections we show how this particular IVP can be solved using standard interval arithmetic methods and compare that to the method used by CLP(F). We then consider two generalizations of this problem. One where interval arithmetic methods are clearly superior, and one where CLP(F) holds the upper hand.

5.2 The classical IA approach

An interval Initial Value Problem (IVP) has the form

$$y' = D(y) \ \wedge\ y(t_0) \in y_0 \ \Rightarrow\ y(t_1) \in y_1$$

where a function D, an interval y_0 and reals t_0, t_1 are given, and the goal is to compute an interval y_1 such that if y is any solution to the IVP with $y(t_0)$ in y_0, then $y(t_1)$ must be in y_1. The standard "1-step" method of solving such a problem consists of two phases. In the first phase, one constructs an interval u_1 and proves that the ODE has a solution in $[t_0, t_1]$ and that for all t in this interval $y(t) \in u_1$. In particular, this implies that $y(t_1) \in u_1$. In the second phase, one uses u_1 to bound the remainder in the Taylor formula for y and this results in a (hopefully narrower) interval y_1 containing $y(t_1)$.

The existence phase of IA IVP solving. In the first phase, the Picard-Lindelöf operator

$$(T(y))(t) = y_0 + \int_{t_0}^{t} D(y(s))ds$$

and the Banach fixed-point theorem are used to prove that the ODE has a solution in $[t_0, t_1]$. The first step is construct an interval u_1 which one hopes contains $y(I)$ for any solution y to the ODE over the interval $I = [t_0, t_1]$, i.e.,

$$y(I) \subset u_1 \tag{5}$$

We next make an estimate of the integral and infer that for any function y on I one has

$$T(y)(I) \subset y_0 + D(u_1) * I \tag{6}$$

If the interval u_1 has the property that $y_0 + D(u_1) * I \subset u_1$, then the Picard-Lindelöf operator maps any function y satisfying (5) into a function $T(y)$ that also satisfies (5). Moreover, one can construct a norm such that this is a contracting map and so the Banach fixed-point theorem implies that T has a unique fixed point y which must satisfy

$$y(I) \subset y_0 + D(u_1) * I \tag{7}$$

(See [18] for a proof.) This process can be iterated by replacing u_1 with the (potentially) smaller interval $y_0 + D(u_1) * I$ and testing whether $y_0 + D(u_1) * I \subset u_1$ holds.

The tightening phase of IA IVP solving. In the second phase, the Taylor formula with remainder is used to compute a subinterval $y_1 \subset u_1$ which contains $y(t_1)$. First, the functional equation $y' = D(y)$ is iterated to find equations for the higher derivatives:

$$y^{(i)}(t) = D^{[i]}(y(t)) \tag{8}$$

for easily computed functions $D^{[k]}$. A straightforward way to compute y_1 is to evaluate the Taylor formula

$$y(t_1) = \left(\sum_{i=0}^{k-1} \frac{y^{(i)}(t_0)}{i!}(t_1 - t_0)^i \right) \ +\ \frac{y^{(k)}(\xi)}{k!}(t_1 - t_0)^k$$

using interval arithmetic, where the coefficients $y^{(i)}(t_0)$ are computed using Eq. (8) for $i < k$ and by using

$$y^{(k)}(\xi) \in y^{(k)}([t_0, t_1]) = D^{[k]}(y([t_0, t_1])) \subset D^{[k]}(u_1)$$

to find an interval enclosing the remainder term. Thus, the interval for y_1 can be computed from y_0 and u_1 using the formula:

$y(t_1) \in$

$$\left(\sum_{i=0}^{k-1} \frac{D^{[i]}(y_0)}{i!}(t_1 - t_0)^i \right) \ +\ \frac{D^{[k]}(u_1)}{k!}(t_1 - t_0)^k \tag{9}$$

The intervals $D^{[i]}(y_0)$ are often computed from the interval y_0 by using interval arithmetic to evaluate a Taylor expansion of $D^{[i]}(y)$ about the center of the interval y_0. This technique often results in a narrower interval than straightforward evaluation of $D^{[i]}(y_0)$ would produce.

Applying the standard IA approach to the example. In the existence phase we need to find a u_1 such that

$$y_0 + (-u_1^2) * I \subset u_1 \tag{10}$$

Observe that if $u_1 = [0, 1]$, $y_0 = 1$, and $I = [0, 0.1]$, then the inclusion (10) holds. Indeed, we have

$$1 + (-[0, 1]^2 * [0, 0.1]) = [0.9, 1.0] \subset [0, 1]$$

This implies that $y' = -y^2, y(0) = 1$ has a unique solution in $I = [0, 0.1]$ and that $y(I) \subset [0.9, 1]$. For this example, $[0.9, 1.0]$ is already a fixed point of $y(I) = 1 - y(I)^2 * I$ and so no further iteration of the Picard-Lindelöf operator is needed

Let us now consider the tightening phase using a degree two Taylor series. In practice, one would use a higher degree

Taylor series to attain a high precision answer, but the ideas are the same. In this case, Eqs. (8) and (9) specialize to yield:

$$\begin{aligned} y'(t) &= -y(t)^2 \\ y''(t) &= -2y(t)y'(t) = 2y(t)^3 \\ y(t) &\in y(0) - y(0)^2 t + 2y(I)^3 t^2/2 \end{aligned}$$

Using the initial conditions $y(0) = 1$, $y(I) \subset u_1$ we obtain

$$\begin{aligned} y(0.1) &\in 1 - 1^2 * 0.1 + 2 * [0.9, 1]^3 * 0.1^2/2 \\ &\subset [0.90729, 0.91] \end{aligned}$$

Thus, this simplest interval arithmetic approach uses the Picard-Lindelöf operator to compute a reasonable estimate u_1 for $y([t_0, t_1])$ and then uses this estimate as a bound on the Taylor remainder to compute an interval containing $y(t_1)$ from an interval containing $y(t_0)$.

5.3 The CLP(F) approach

The CLP(F) approach consists of two phases. In the first phase, we reduce the general analytic constraint $\mathcal{C}$ to a functional constraint $\phi(\mathcal{C}) = \tilde{\mathcal{C}}$ which, by construction, contains no evaluation constraints, no differentiation constraints, and no vector variables. In the second phase, we show how to solve such a system by repeatedly applying certain interval arithmetic contraction operators.

The reduction strategy. The strategy is to show that any solution (x, f) of the analytic constraint being solved, gives rise to a solution of a simpler set of functional constraints. More precisely, we will show how to use (x, f) to construct constraints on the following objects:

$$\begin{aligned} &a_0 = f(0),\ a_1 = f'(0), \\ &b_0 = f(0.1),\ v = f''(\xi(0.1)) \\ &y_0(x) = f(x),\ y_1(x) = f'(x),\ y_2(x) = f''(x), \\ &s(x) = x \in I,\ z(x) = f''(\xi(x)) \end{aligned}$$

where y_0, y_1, y_2, s, z are functions defined on $I = [0, 0.1]$ and $\xi = \xi_{f,2,0}$ is the Taylor remainder function.

The key idea is to use the analytic constraints to generate arithmetic constraints on the values of the four real variables a_0, a_1, b_0, v and functional equation and range constraints on the five functions $y_0(x), y_1(x), y_2(x), z(x), s(x)$. The two evaluation equations and one range constraint

```
eval(F,0)=1, eval(F,0.1)=X, F in [0,1]
```

generate three simple arithmetic and range constraints on $a_0 = f(0)$, $b_0 = f(0.1)$, and $y_0 = f$:

$$a_0 = 1, \quad b_0 = x, \quad y_0(I) \in [0, 1] \tag{11}$$

From the functional equation constraint

```
ddt(F,1)= -F^2
```

we generate constraints on the derivatives of f over I and at 0.

$$a_1 = -a_0^2 \tag{12}$$
$$y_1(x) = -y_0(x)^2 \tag{13}$$
$$y_2(x) = -2 * y_0(x) * y_1(x) \tag{14}$$

These constraints arise from using the Taylor series of f centered at some point x in $[0, 0.1]$ to compute the Taylor series of f' and $-f^2$, which must be equal. The Taylor series for f has the form:

$$f(s) = y_0(x) + y_1(x)(s - x) + y_2(x)/2 * (s - x)^2 + \ldots$$

Using this expression we can easily compute the Taylor series for f' and $-f^2$:

$$\begin{aligned} f'(s) &= y_1(x) + y_2(x)(s - x) + y_3(x)/2 * (s - x)^2 + \ldots \\ -f(s)^2 &= -y_0^2(x) + 2 * y_0(x) * y_1(x)(s - x) + \ldots \end{aligned}$$

Since these two series are equal, their Taylor coefficients must be equal and so we can infer Eqs. (13), (14). Similarly, Eq. (12) is generated by equating the Taylor coefficients expanded about $x = 0$. Next, the type constraint

```
type([F],function(0,0.1))
```

states that F is infinitely differentiable on $I = [0, 0.1]$ and so must satisfy the Taylor formula of degree 2.

$$f(x) = f(0) + f'(0)x + f''(\xi(x))x^2/2, \quad \forall x \in I$$

This translates immediately into a constraint on y_0, a_0, a_1, s, z for points in $I = [0, 0.1]$

$$y_0(x) = a_0 + s(x) * (a_1 + s(x) * z(x)/2) \tag{15}$$
$$z(I) \subset y_2(I) \tag{16}$$
$$s(I) \subset [0, 0.1] \tag{17}$$

Instantiating the Taylor formula at $x = 0.1$, we get a constraint on $b_0 = f(0.1)$:

$$b_0 = a_0 + 0.1 * (a_1 + 0.1 * v/2) \tag{18}$$
$$v \in z(I) \tag{19}$$

Note that we use the Horner form of these polynomials because it often results in better constraint narrowing.

Construction of the functional constraints. We can now combine Eqs. (11)-(19) into a reduced analytic constraint with two types of variables:

$$type([Y_0, Y_1, Y_2, S, Z], function(0, 0.1)) \tag{20}$$
$$type([A_0, A_1, V, X, B_0], real) \tag{21}$$

and several functional and range constraints (which we have reordered to simplify the ensuing discussion)

$$Y_0 \subset [0, 1] \tag{22}$$
$$S \subset [0, 0.1] \tag{23}$$
$$A_0 = 1 \tag{24}$$
$$A_1 = -A_0^2 \tag{25}$$
$$Y_1 = -Y_0^2 \tag{26}$$
$$Y_2 = -2 * Y_0 * Y_1 \tag{27}$$
$$Z \subset Y_2 \tag{28}$$
$$V \in Z \tag{29}$$
$$Y_0 = A_0 + S * (A_1 + S * Z/2), \tag{30}$$
$$B_0 = A_0 + 0.1 * (A_1 + 0.1 * V/2), \tag{31}$$
$$X = B_0 \tag{32}$$

Solving the functional constraints. For these constraints, we use a particularly simple constraint solving strategy whose correctness will be proved in Section 6.3. We initially assign the interval $[-\infty, \infty]$ to each real and function variable. The interval for a function variable corresponds to its range. We narrow these intervals by using the contractor

$$V \leftarrow V \cap E(U, V)$$

for each equation $V = E(U, V)$ in the constraint set, and using the contractor

$$V \leftarrow V \cap U$$

for each inclusion $V \in U$ or $V \subset U$ in the constraint set. In the general case a more sophisticated method for solving arithmetic constraints is required [12, 13].

Applying each of the constraint contractors once to this initial state yields:

$$\begin{array}{rcl} A_0 & = & 1 \\ A_1 & = & -1 \\ Y_0 & \subset & [0, 1] \\ Y_1 & \subset & [-1, 0] \\ V \in Z \subset Y_2 & \subset & [0, 2] \\ X & \in & [0.9, 0.91] \end{array}$$

Repeating this process again, a fixed point is reached:

$$\begin{array}{rcl} A_0 & = & 1 \\ A_1 & = & -1 \\ Y_0 & \subset & [0.9, 1.0] \\ Y_1 & \subset & [-1.0, -0.81] \\ V \in Z \subset Y_2 & \subset & [1.458, 2.0] \\ X & \in & [0.90729, 0.91] \end{array}$$

Tighter bounds on X can be obtained by using a higher degree. For one-step IVP problems, the simple IA method discussed above and the CLP(F) method will return the same answers, but the IA method requires less memory and time since it is specialized to solve only these problems.

5.4 An Iterated IVP problem

We now consider the more difficult problem of computing the value of $f(12.0)$, given $f' = -f^2, f(0) = 1, f(I) \in [0, 1]$ as before.

The standard interval arithmetic approach to solving IVP problems of this form is to subdivide the problem into a sequence of problems with smaller intervals:

$$y' = D(y) \ \wedge \ y(t_j) \in y_j \ \Rightarrow \ y(t_{j+1}) = y_{j+1}$$

where $t_0 = 0.0 < t_1 < \ldots < t_n = 12.0$ is a sequence of intermediate points which are generally computed dynamically and are not necessarily equally spaced. These are solved sequentially, first y_0 is used to compute an interval y_1 containing all solutions to the IVP. This is then used to computed y_2 and so on until an interval y_n is computed which contains all solutions $y(t_n)$ to the original IVP.

In the simplest approach each of these subproblems (computing t_{j+1} and y_{j+1} from t_j and y_j) is solved using a two phase process. The first phase consists of proving that the IVP has a unique solution on the interval $[t_j, t_{j+1}]$ and finding an interval u_{j+1} which contains the range of $y([t_j, t_{j+1}])$. The second phase consists of finding a subinterval y_{j+1} of u_{j+1} which contains $y(t_{j+1})$.

This simple approach is subject to the "wrapping effect" when y is a vector of functions. This effect arises from the implicit requirement that the intermediate solution sets y_j are approximated by rectilinear intervals. This results in an overestimate which by itself can result in exponential growth in the widths of the intervals for small step size. There are several techniques that can be applied to lessen this effect. For example, see Nedialkov, et.al., [21] for a survey of recent approaches, and Deville,Janssen, van Hentenryck [7, 16] for a novel approach using unconstrained optimization methods.

The simple CLP(F) approach to solving this problem is to write a procedure which generates constraints for each of the subintervals $[t_j, t_{j+1}]$.

```
p(T1,T2,Y1,Y2) :-
    type([F],function(T1,T2)),
  {[ddt(F,1)= -F**2, F in [0,1],
    eval(F,T1)=Y1, eval(F,T2)=Y2 ]}.

step(T1,Tn,Y1,Yn,Dx) :-
  {[Tn-T1 =< Dx ]},
  p(T1,Tn,Y1,Yn).
step(T1,Tn,Y1,Yn,Dx) :-
  {[Tn-T1 > Dx, T2 = T1+Dx]},
  p(T1,T2,Y1,Y2),
  step(T2,Tn,Y2,Yn,Dx).

test(Y) :- p(0,12,1,Y,0.1).
```

This approach has several drawbacks. First its memory requirements increase linearly with the number of subintervals. Second, it is subject to the wrapping effect and so rapidly loses precision. Finally, it is much slower than the interval arithmetic method since it often uses interval constraint solving when interval evaluation is all that is needed.

The current implementation of CLP(F) is not competitive in efficiency or precision with AWA ([18]) and similar solvers which are specialized for finding validated solutions of these IVP problems. For example, in Nedialkov's dissertation, ([22], pp. 91-93), two state-of-the-art interval arithmetic ODE IVP solvers are compared on a variation of this problem to compute $f(12)$. These solvers are able to verify that the ODE does indeed have a solution over that entire interval and to return answers with about 10 decimal digits of precision in about 0.12 seconds using a step size of 0.1 and Taylor formulas of degree 11. CLP(F), by comparison, requires 220 seconds to attain 10 digit accuracy with degree 30 Taylor formulas.

Since CLP(F) constraint solving is formulated in quite general terms, it is a nontrivial task to prove the existence of solutions to general analytic constraints, but we view it as an interesting and important problem. Another important and interesting problem is to reframe the "wrapping effect" as a constraint problem and develop CLP(F) strategies to reduce this effect.

5.5 A Parameter Determination problem

Next we consider a problem that can easily be solved by CLP(F) but for which the interval arithmetic IVP solvers are not well suited.

The problem is to solve the following analytic constraint in which the values at the endpoint are given and the exponent of F in the differential equation is to be determined.

```
?- type([F],function(0,0.1)),
 {[ddt(F,1)= -F**R, R in [0,10],
   eval(F,0)=1, eval(F,0.1)= 0.9095,
   F in [0,1]]}.
R = 2.1028663136056...   (16.56 sec)
```

This is not an IVP problem and hence lies out of the range of systems like AWA ([18]). An interesting possible direction of research would be to attempt to extend AWA-like systems so that they could handle larger classes of analytic constraints.

The CLP(F) interpreter will solve this constraint easily by generating a slightly different arithmetic constraint set. For example, in the degree 2 example Eqs. (25)-(27) will be replaced by the following:

$$\begin{aligned} A_1 &= -A_0 **R \\ Y_1 &= -Y_0 **R \\ Y_2 * Y_0 &= R * Y_1 * Y_1 \end{aligned}$$

and two new constraints will be added:

$$\begin{aligned} B_0 &= 0.9095, \\ R &\subset [0,10] \end{aligned}$$

This degree 2 system does not result in any narrowing of R, but if the corresponding degree 30 system is used, then R can be calculated to high precision as shown above.

6 The Analytic Constraint Solving Algorithm

In this section, we show how to extend the techniques shown in the examples above to solve general analytic constraints. First we show how to solve arithmetic constraints, then functional constraints, and finally we show how to reduce analytic constraints to functional constraints.

6.1 Interval Arithmetic Constraint Solving

An interval-based arithmetic constraint solver is an algorithm which takes an arithmetic constraint $\mathcal{C}(X)$ on a tuple X of variables and returns an tuple I of intervals such that every solution X to C is contained in I, i.e.,

$$\mathcal{S}(\mathcal{C}(X)) \subset I$$

where $\mathcal{S}(\mathcal{C}(X)) = \{x \in \mathcal{R}^n : \mathcal{C}(x)\}$ denotes the set of solutions to $\mathcal{C}$.

The simplest way to implement such an arithmetic constraint solver is to first transform a general constraint C into a conjunction of primitive constraints by introducing temporary variables and then to iteratively apply primitive contraction operators K_c for each primitive arithmetic constraint c. For example, to solve

```
?- {[exp(X)^2+(exp(X^2)-1)^2=25, X>0]}.
    X = 1.23731271072460...e+00 ?  (0.03 sec)
```

we first transform it into primitive constraints involving four operators (square, subtraction, addition, exp) and several temporary variables $t_0, t_1, t_2, t_3, t_4, t_5$:

$$\begin{aligned} & (e^x)^2 + (e^{x^2} - 1)^2 = 25, \;\; x > 0 \\ \Leftarrow \quad & x > 0, \; x^2 = t_0, \; exp(x) = t_1, \; exp(t_0) = t_2, \\ & t_1^2 = t_3, \; t_2 - 1 = t_4, t_4^2 = t_5, \; t_3 + t_5 = 25 \end{aligned}$$

Then, initializing all variables to $[-\infty, \infty]$, one can repeatedly apply contraction operators for each of these constraints to contract the intervals. Since the intervals are represented as IEEE floating point numbers and there are finitely many such numbers, termination is guaranteed. In practice, one often stops after some fixed number of iterations is exceeded (e.g. 10^6).

If we let I_x denote the interval associated to x, then the contractor for the constraint $\exp(x) = t_1$ is the function K which maps the current interval variable bindings I into $J = K(I)$, where

$$\begin{aligned} J_x &= I_x \cap \log(I_{t_1}) \\ J_{t_1} &= I_{t_1} \cap \exp(I_x) \\ J_y &= I_y, \;\; \forall y \notin \{x, t_1\} \\ & \quad \text{where} \\ exp([a,b]) &= [\exp_{lo}(a), \exp_{hi}(b)] \\ log([a,b]) &= [\log_{lo}(\max(a,0)), \log_{hi}(\max(b,0))] \end{aligned}$$

Here $\exp_{lo/hi}$ and $\log_{lo/hi}$ are floating point functions which return lower/upper bounds on $\exp(x)$ and $\log(x)$. Such operators are described explicitly in [12] and [17] in terms of interval arithmetic operations which are defined and proved correct in [13].

Continuing with the example, initially $I_x = I_{t_i} = [-\infty, \infty]$, so applying the contractors for $x > 0$, $x^2 = t_0$ and $\exp(x) = t_1$ contracts I_x and I_{t_0} to $[0, \infty]$ and I_{t_1} to $[1, \infty]$. Applying the contractors for the next four constraints gives us the following box I:

$$\begin{aligned} & I_x = [-\infty, \infty], \\ & I_{t_0} = I_{t_4} = I_{t_5} = [0, \infty], \; I_{t_1} = I_{t_2} = I_{t_3} = [1, \infty], \end{aligned}$$

Since t_3, t_5 are non-negative, the contractor for $t_3 + t_5 = 25$ contracts I_{t_3} to $[1, 25]$ and I_{t_5} to $[0, 24]$. Reapplying the contractors for the constraints (in reverse order) we find that I_{t_1} is narrowed to $[1, 5]$ and I_{t_4} is narrowed to $[0, \sqrt{24}]$. Thus, I_{t_2} is narrowed to $[1, 1 + \sqrt{24}]$, and I_{t_0} then is narrowed to $[0, \log(1 + \sqrt{24})]$ and finally x is narrowed to $[0, \sqrt{\log(1+\sqrt{24})}] \cap [0, \log(5)] = [0, \sqrt{\log(1+\sqrt{24})}]$. So we get

$$\begin{aligned} & I_{t_0} = [0, \log_{hi}(\mathrm{add}_{hi}(1, \mathrm{sqrt}_{hi}(24)))], \\ & I_{t_1} = [1, 5], \\ & I_{t_2} = [1, 1 + \mathrm{sqrt}_{hi}(24)], \\ & I_{t_3} = [1, 25], \\ & I_{t_4} = [0, \mathrm{sqrt}_{hi}(24)], \\ & I_{t_5} = [0, 24], \\ & I_x = [0, \mathrm{sqrt}_{hi}(\log_{hi}(\mathrm{add}_{hi}(1, \mathrm{sqrt}_{hi}(24))))] \\ & \quad \subset [0.0, 1.332208454935944] \end{aligned}$$

After applying about 600 of these primitive contraction operations, CLP(F) is able to contract the interval for X to a narrow interval containing only 5 floating point numbers.

6.2 Contracting Range Constraints

Several CLP(intervals) implementations, starting with BNR Prolog [23], have included a domain range constraint (`X in Y`) which has been implemented operationally as a contractor for X which simply intersects X with Y. Thus, for example, the following query contracts X but not Y:

```
?- {X = [1,5], Y = [2,6], X in Y}.
   X=[2,5], Y=[2,6] (0.001 sec)
```

If $\mathcal{C}(X)$ is an arithmetic constraint on the real variables $X_1, \ldots, X_n$ which may also contain range constraints, we let $\mathcal{A}_{arith}(\mathcal{C})$ be the answer interval $I \subset \mathcal{R}^n$ obtained by iteratively applying the interval arithmetic contractors and this range contractor until a fixed point is reached. It is not apriori clear that there will be a unique fixed point, but this can easily be proved from the monotonicity of the contraction operators (i.e. $(A \subset B) \Rightarrow (K(A) \subset K(B))$ for all interval arithmetic and range contractors K).

6.3 Functional Constraint Solving

In this section we present a sound constraint solving algorithm for functional constraints. Recall that a functional constraint $\mathcal{C}(X, F)$ is a conjunction of arithmetic constraints and range constraints on the real variables X and function variables F.

Let $\mathcal{C}(X, F)$ be a functional constraint, where X is an n-tuple of real variables and F is an m-tuple of function variables. Let Y be an m-tuple of real variables disjoint from X and let I be the answer interval obtained by solving $\mathcal{C}(X, Y)$ using the arithmetic constraint solving algorithm $\mathcal{A}_{arith}$ defined in Section 6.2:

$$I = \mathcal{A}_{arith}(\mathcal{C}(X, Y))$$

We let $\mathcal{A}_{fn}(\mathcal{C}(X, F)) = \mathcal{A}_{arith}(\mathcal{C}(X, Y))$ denote this functional constraint solving algorithm.

Theorem 1 *$\mathcal{A}_{fn}$ is a sound functional constraint solving algorithm. That is, if $\mathcal{C}(X, F)$ is a functional constraint for a tuple $X = (X_1 \ldots, X_n)$ of real variables and a tuple $F = (F_1, \ldots, F_m)$ of function variables, and if $I = \mathcal{A}_{fn}(\mathcal{C}(X, F)) \subset \mathcal{R}^n \times \mathcal{R}^m$ is the answer interval produced by $\mathcal{A}_{fn}$, then for any solution (x, f) of $\mathcal{C}(X, F)$ we must have*

$$x_i \in I_{X_i}, \;\; range(f_j) \subset I_{F_j} \tag{33}$$

for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$, where $range(f_j)$ denotes the range of the function f_j over its domain.

Proof. Without loss of generality, we may assume that $\mathcal{C}$ consists entirely of primitive constraints. To simplify the proof we also assume that all functions are defined on the same domain $[a, b]$, but the proof can easily be modified to get around this restriction.

Let $\mathcal{C}_0$ be the conjunction of all arithmetic constraints in $\mathcal{C}$ and let $\mathcal{C}_1$ be the conjunction of all range constraints in $\mathcal{C}$. To prove the theorem we need only verify that if I is any interval satisfying (33) then $K_C(I)$ also satisfies (33) for every contraction operator K_C with $C \in \mathcal{C}$,

First, observe that for every t in the common domain $[a, b]$ of the functions, if we let $y = f(t)$, then (x, y) is a solution to the arithmetic constraint $\mathcal{C}_0$, that is,

$$\forall t \in [a, b].\; \mathcal{C}_0(x, f(t))$$

This is an immediate consequence of the fact that the arithmetic constraints on functions are defined pointwise. Thus by the soundness of $\mathcal{A}_{arith}$ for interval arithmetic constraints, the assumption that $(x, f([a, b]) \subset I$ implies that it is also contained in $J = K_C(I)$, i.e., J_{Y_i} must contain $f_i(t)$ for all t, hence it must contain $range(f_i)$.

Next, let C be a range constraint. If C has the form `Xi in Fj`, then we must have $x_i = f_j(t)$ for some t and hence we must have

$$x_i \in I_{X_i} \cap I_{Y_i} \; = \; J_{Y_i}$$

Similarly, if C has the form `Fi in Fj`, then since (x, f) is a solution to $\mathcal{C}$, we must have $range(f_i) \subset range(f_j)$ and so

$$range(f_i) \subset I_{Y_i} \cap I_{Y_j} \; = \; J_{Y_i}$$

Finally, if C has the form `Fi in [Xi,Xk]`, then we must have $range(f_i) \subset [x_i, x_k]$ and so

$$range(f_i) \subset I_{Y_i} \cap [I_{X_i}, I_{X_k}] \; = \; J_{Y_i}$$

where $[A, B] = [min(A, B), max(A, B)]$ denotes the smallest interval containing both intervals A and B. Since every constraint in $\mathcal{C}$ is either an arithmetic constraint or a range constraint, this proves the theorem. **QED**

6.4 The Reduction Algorithm

We now present an algorithm, depending on an integer parameter $n \geq 2$, for reducing an analytic constraint $\mathcal{C}(X, F)$ to a functional constraint $\tilde{\mathcal{C}}_n(X, F, Y, G)$ in such a way that every solution (x, f) to $\mathcal{C}$ can be extended to a solution (x, f, y, g) of $\tilde{\mathcal{C}}_n$. The integer n represents the degree of the Taylor formulas which will be used in the reduction.

To simplify the discussion, we make some mild assumptions about the analytic constraint $\mathcal{C}(X, F)$ which we are going to solve. First we assume that X is a tuple of real numbers and F is a tuple of functions:

$$X = (X_1, \ldots, X_p)$$
$$F = (F_1, \ldots, F_r)$$

where all of the functions F_i are defined on the same domain $[a, b]$. It is not hard to handle functions defined on different domains or even functions whose domains are given by real variables, but it complicates the notation. We also assume that all constraints are primitive, i.e. they are either of the form $op(U_1, \ldots, U_n) = V$ for some operator op, or of the form $r(U_1, \ldots, U_n)$ for some relation r, where the U_i can be real and/or function variables. It is straightforward to transform any constraint into a set of primitive constraints simply by adding temporary variables for each subexpression in a complex constraint, e.g.,

```
ddt(F,1)=-F**R --> T1=ddt(F,1),T2=F**R,T1=-T2
```

This transformation is similar to the translation of arithmetic expressions to three address code. In particular, common subexpressions can be represented by the same temporary variable although our current implementation does not make this optimization.

The constraint variables. The main idea behind the reduction from analytic to functional constraints is to introduce a set of auxiliary function variables representing the first n derivatives of the analytic function variables and a set of real variables representing the values of these functions at various key points. We then introduce two types of constraints on these variables: constraints arising from the functional equations, and constraints arising from the Taylor formula with remainder.

The conjunction of these constraints will be the functional constraint $\tilde{\mathcal{C}}_n(X, F, Y, G)$ where X, Y are tuples of real variables and F, G are tuples of function variables

$$\begin{aligned} Y &= ((A_{ij}), (B_{ij}), (W_{ih}), (U_{ijht}), (V_{ij})) \\ G &= (\mathbf{S}(x), (\mathbf{Z}_{ijt}(x))) \end{aligned}$$

and $\tilde{\mathcal{C}}_n(X, F, Y, G)$ will be constructed so that, for any solution (x, f) to $\mathcal{C}(X, F)$, it evaluates to true whenever the variables are given the following bindings (which we denote by $\Gamma(x, f) = (x, f, y, g)$):

$$\begin{aligned} X_i &= x_i \\ \mathbf{F}_{ij}(s) &= f_i^{(j)}(s) \\ A_{ij} &= f_i^{(j)}(a) \\ B_{ij} &= f_i^{(j)}(b) \\ V_{ij} &= f_i^{(j)}(\xi_{f_i,j,a}(b)) \\ U_{ijht} &= \begin{cases} f_i^{(j)}(\xi_{f_i,j,t}(x_h)), & \text{if } x_h \in [a,b] \\ 0 & \text{else} \end{cases} \\ W_{ih} &= \begin{cases} f_i(x_h), & \text{if } x_h \in [a,b] \\ 0 & \text{else} \end{cases} \\ \mathbf{Z}_{ijt}(s) &= f_i^{(j)}(\xi_{f_i,j,t}(s)) \\ \mathbf{S}(s) &= s \end{aligned}$$

with the indices i, j, h, t ranging over the following sets

$$\begin{aligned} &i = 1, \ldots, r \\ &j = 0, \ldots, n \\ &h = 1, \ldots, p \\ &t \in \{a, b\} \end{aligned}$$

Recall that p is the number of real variables X_h in $\mathcal{C}$, r is the number of function variables F_j, and n is the degree of the Taylor series approximation that will be used. Thus, the first step is to introduce $O(rnp)$ new interval arithmetic variables. Note: in this section we will use bold face to distinguish the function variables from the real variables.

Functional Equation Constraints. We first treat the functional equation constraints, since they are in some ways the simplest. Let (x, f) be a solution of $\mathcal{C}(X, F)$, We will translate functional equations involving X_i and F_i into functional constraints on $X_i, A_{ij}, B_{ij}, \mathbf{F}_{ij}$.

Observe that the vectors A_{i*},B_{i*}, and $\mathbf{F}_{i*}$ represent the Taylor coefficients of f at a, b, and some point $x \in [a, b]$. Thus, if we let $T_t(f)$ denote the (infinite) sequence of coefficients for the Taylor series of f centered at t, then any arithmetic constraint among the f_j and x_i induces a corresponding set of arithmetic constraints on the Taylor coefficients $T_t(f_j)$ and the x_i. The special case of $t = a$ generates constraints on the real vectors A_{i*} and X_*. Similarly, $t = b$ generates the same constraints on B_{i*} and X_*, and $t = x$ generates the same constraints on the vectors $\mathbf{F}_{i*}$ of functions and the vector X_i of reals.

For example, the analytic constraint $f_i * f_j = f_k$, induces a constraint relating the Taylor coefficients of f_i, f_j, and f_k at any s in their domain:

$$f_k^{(u)}(s) = \sum_{v+w=u} \binom{u}{v} f_i^{(v)}(s) * f_j^{(w)}(s), \ \forall u$$

Therefore, the analytic constraint $F_i * F_j = F_k$ is then translated into the following set of arithmetic constraints on the Taylor coefficient functions $\mathbf{F}_{i*}$:

$$\mathbf{F}_{ku} = \sum_{v+w=u} \binom{u}{v} \mathbf{F}_{iv} * \mathbf{F}_{jw},$$

for each $u = 0, 1 \ldots, n-1$. Similar constraints will be generated for A_* and B_*. Since the constraints for A, B, and $\mathbf{F}$ are identical, we only show the constraints for $\mathbf{F}$. The other algebraic constraints are handled as follows, where the X_i are real variables and the F_i are function variables:

$$\begin{array}{lll} F_i = X_k & \Rightarrow & \mathbf{F}_{i0} = X_k, \mathbf{F}_{ij} = 0, \ \forall j > 0 \\ F_k = X_i * F_j & \Rightarrow & \mathbf{F}_{ku} = X_i * \mathbf{F}_{ju}, \ \forall u \\ F_k = F_i + F_j & \Rightarrow & \mathbf{F}_{ku} = \mathbf{F}_{iu} + \mathbf{F}_{ju}, \ \forall u \\ F_j = ddt(F_i, k) & \Rightarrow & \mathbf{F}_{i,u+k} = \mathbf{F}_{ju}, \ \forall u \end{array}$$

Other combinations, e.g. $F_k = X_i + F_j$ are handled similarly.

Although we could handle the elementary functions by explicitly computing the relationship between the Taylor coefficients of the argument and result functions, we have chosen in this implementation to reduce the elementary function constraints to arithmetic constraints. For example, to handle $F_i = exp(F_j)$ we constrain the constant terms

$$\mathbf{F}_{i0} = exp(\mathbf{F}_{j0})$$

and then we add the constraints corresponding to the induced equation, $exp(F_j)' = exp(F_j) * F_j'$:

$$\mathbf{F}_{i,u+1} = \sum_{v+w=u} \binom{u}{v} \mathbf{F}_{iv} * \mathbf{F}_{j,w+1} \ \forall u$$

Similarly, for $F_i = F_j ** X_k$, we add the corresponding constraint on the constant terms

$$\mathbf{F}_{i0} = \mathbf{F}_{j0} ** X_k$$

and then we use the relation $F_i' * F_j = X_k * F_i * F_j'$ to generate higher level constraints. For $F_j = sin(F_i), F_k = cos(F_i)$, we likewise add the corresponding constant coefficients, and then we use the equations

$$ddt(F_j, 1) = F_k * ddt(F_i, 1), \ \ ddt(F_k, 1) = -F_j * ddt(F_i, 1)$$

to induce arithmetic constraints among the higher order coefficients of F_i, F_j, F_k. Note that we are implicitly assuming that sine and cosine constraints always appear together, if not, then we add a cosine constraint and introduce the arithmetic constraints as above.

Type constraints. The functional equation constraints defined above tend to propagate information from lower derivatives to higher ones. For example, knowing $f(t) = 1$ and $f' = f$ we can infer that $f^{(n)}(t) = 1, \ \forall n$. The type constraints which we introduce in this section tend to have the effect of propagating information about the values and ranges of the derivatives of the functions from higher derivatives down to lower derivatives, and for propagating information about the values of the functions at one point to information about the values at another.

These constraints are all special cases of the Taylor formula with remainder, in which the Taylor coefficients at one endpoint and the ranges of the derivatives over the interval are used to constrain the value of the function at some point in the interval. Currently we invoke five of these special cases of the Taylor formula of degree n. Two cases corresponding to computing the values of function $\mathbf{F}_i(s)$ using the Taylor coefficients at the two endpoints. Two cases for computing the result of evaluating $\mathbf{F}_i$ at a point x_h in the domain, using the Taylor coefficients at the two endpoints, and finally one case in which the Taylor coefficients at the left and right endpoints are constrained in terms of the ranges of the derivatives over the interval.

It may be possible to obtain better convergence by adding more special cases of these constraints (e.g. the corresponding equations for each of the derivatives of $\mathbf{F}_i$), but in practice the constraints we have selected seem to be optimal. The main problem with adding additional constraints is that it can increase the execution time and the memory requirements. On the other hand, additional constraints can also sometimes improve the precision obtained by the constraint solver.

Before describing the constraints in detail, we present the modified version of the Taylor formula (Lemma 1 in Section 4) which we shall use.

Corollary 1 *Let the notation be as in Lemma 1, and in addition for all $j \geq 0$, let $z_j(s) = f^{(j)}(\xi_{f,j,t}(s))$. Then for all $s \in [a,b]$, and for every every $j \geq 0$ we have:*

$$z_j(s) = f^{(j)}(t) \;+\; (s-t) * z_{j+1}(s)/(j+1)$$

Moreover, we have $f(s) = z_0(s)$ and for every $j > 0$ $z_j(s) \in f^{(j)}([a,b])$. In particular, the Taylor formulas of degrees $0,\ldots,n$ can be collectively expressed in terms of the z_i as

$$\begin{array}{rcl} f(s) & = & z_0(s), \\ z_0(s) & = & f(t) \;+\; (s-t) * z_1(s)/1 \;\subset\; f(I) \\ z_1(s) & = & f'(t) \;+\; (s-t) * z_2(s)/2 \;\subset\; f'(I) \\ \ldots & & \ldots \\ z_{n-1}(s) & = & f^{(n-1)}(t) \;+\; (s-t) * z_n(s)/n \;\subset\; f^{(n-1)}(I) \\ z_n(s) & \subset & f^{(n)}(I) \end{array}$$

where $I = [a,b]$.

Proof. By definition of ξ, we see that $f(s) = z_0(s)$. Comparing the Taylor formulas of degrees j and $j+1$ we see that they have the same first j terms, and hence we have

$$\begin{array}{l} \dfrac{f^{(j)}(\xi_{f,j,t}(s))}{j!}(s-t)^j \quad = \\ \qquad \dfrac{f^{(j)}(t)}{j!}(s-t)^j \quad + \quad \dfrac{f^{(j+1)}(\xi_{f,j+1,t}(s))}{(j+1)!}(s-t)^{j+1} \end{array}$$

Factoring out $(s-t)^j/j!$ from both sides, we obtain:

$$f^{(j)}(\xi_{f,j,t}(s)) = f^{(j)}(t) + f^{(j+1)}(\xi_{f,j+1,t}(s))\frac{(s-t)}{(j+1)}$$

QED

Since we are assuming that all function variables F_i are declared to have type $function(a,b)$, we can apply the Taylor formula and introduce the constraints corresponding to several different specializations of the corollary:

- For each function f_i, we apply the corollary with $f = F_i$, $t = a$, $s \in [a,b]$ and obtain the following constraints which relate the variables $\mathbf{F}_{ij}$, $\mathbf{S}$ and A_{ij} using intermediate variables $\mathbf{Z}_{ija}$:

$$\begin{array}{l} \mathbf{F}_{i0} = \mathbf{Z}_{i,0,a} \\ \mathbf{Z}_{i,0,a} = (A_{i0} + (\mathbf{S} - a) * \mathbf{Z}_{i,1,a}/1) \in \mathbf{F}_{i0} \\ \ldots \\ \mathbf{Z}_{i,n-1,a} = (A_{i,n-1} + (\mathbf{S} - a) * \mathbf{Z}_{i,n,a}/n) \in \mathbf{F}_{i,n-1} \\ \mathbf{Z}_{i,n,a} \in \mathbf{F}_{in} \end{array}$$

- For each function f_i, we apply the corollary with $f = F_i$, $t = b$, $s \in [a,b]$ and obtain a constraint which relates the variables $\mathbf{F}_{ij}$,$\mathbf{S}$, and B_{ij} using intermediate variables $\mathbf{Z}_{ijb}$:

$$\begin{array}{l} \mathbf{F}_{i0} = \mathbf{Z}_{i,0,b} \\ \mathbf{Z}_{i,0,b} = (B_{i0} + (\mathbf{S} - b) * \mathbf{Z}_{i,1,b}/1) \in \mathbf{F}_{i0} \\ \ldots \\ \mathbf{Z}_{i,n-1,b} = (B_{i,n-1} + (\mathbf{S} - b) * \mathbf{Z}_{i,n,b}/n) \in \mathbf{F}_{i,n-1} \\ \mathbf{Z}_{i,n,b} \in \mathbf{F}_{in} \end{array}$$

- Finally, for each function f_i, we apply the corollary with $f = F_i$, $t = a$, $s = b$ and obtain a constraint which relates the variables A_{ij} and B_{ij} using $\mathbf{F}_{ij}$ and the intermediate variables V_{ij}:

$$\begin{array}{l} B_{i0} = V_{i,0} \\ V_{i,0} = (A_{i0} + (b - a) * V_{i,1}/1) \in \mathbf{F}_{i0} \\ \ldots \\ V_{i,n-1} = (A_{i,n-1} + (b - a) * V_{i,n}/n) \in \mathbf{F}_{i,n-1} \\ V_{i,n} \in \mathbf{F}_{in} \end{array}$$

Evaluation Constraints. Evaluation constraints have the form $eval(F_i, X_h) = X_g$ and as before we use the corollary to generate arithmetic constraints relating $W_{ih} = f_i(x_h)$ to the Taylor coefficients $A_{i*}, B_{i*}, \mathbf{F}_{i*}$ of f_i.

- $W_{ih} = X_g$ and $X_h \in [a,b]$
- Applying the corollary with $f = F_i$, $t = a$, $s = x_h$ we obtain a constraint relating X_h with A_* and $\mathbf{F}_*$ using intermediate variables U_{*a}:

$$\begin{array}{l} W_{ih} = U_{i,0,h,a} \\ U_{i,0,h,a} = (A_{i0} + \frac{(X_h - a)}{1} * U_{i,1,h,a}) \in \mathbf{F}_{i0} \\ \ldots \\ U_{i,n-1,h,a} = (A_{i,n-1} + \frac{(X_h - a)}{n} * U_{i,n,h,a}) \in \mathbf{F}_{i,n-1} \\ U_{i,n,h,a} \in \mathbf{F}_{in} \end{array}$$

- Applying the corollary with $f = F_i$, $t = b$, $s = x_h$ we obtain a constraint relating X_h with B_* and $\mathbf{F}_*$ using intermediate variables U_{*b}:

$$\begin{array}{l} W_{ih} = U_{i,0,h,b} \\ U_{i,0,h,b} = (B_{i0} + \frac{(X_h - b)}{1} * U_{i,1,h,b}) \in \mathbf{F}_{i0} \\ \ldots \\ U_{i,n-1,h,b} = (B_{i,n-1} + \frac{(X_h - b)}{n} * U_{i,n,h,b}) \in \mathbf{F}_{i,n-1} \\ U_{i,n,h,b} \in \mathbf{F}_{in} \end{array}$$

Range Constraints. Range constraints have three possible forms. Constraints of the form `Fi in [Xj,Xk]` are handled by constraining $F_i(s) \in [X_j, X_k]$:

$$\mathbf{F}_{i0} \texttt{ in } [X_j, X_k]$$

where $[X_j, X_k]$ denotes the smallest interval containing both X_j, X_k. Constraints of the form `Xh in Fi` get translated to

$$X_h \texttt{ in } \mathbf{F}_{i0}$$

Finally, constraints of the form `Fi in Fj` get translated as

$$\mathbf{F}_{i0} \texttt{ in } \mathbf{F}_{j0}$$

Range constraints coupled with Taylor constraints and functional equations are used to put bounds on the Taylor coefficients within an interval.

The Identity Constraint. The constraint `identity(Fi)` gets translated into

$$\begin{array}{l} A_{i0} = a,\ A_{i1} = 1,\ A_{ik} = 0, \forall k > 1 \\ B_{i0} = b,\ B_{i1} = 1,\ B_{ik} = 0, \forall k > 1 \\ \mathbf{F}_{i0} \subset [a, b],\ \mathbf{F}_{i1} = 1,\ \mathbf{F}_{ik} = 0, \forall k > 1 \end{array}$$

6.5 The Main Theorem

Given an analytic constraint $\mathcal{C}$ and an integer $n > 1$, we let $\tilde{\mathcal{C}}_n(X, F, Y, G)$ denote the functional constraint obtained by translating the constraints of $\mathcal{C}$ as described above. Let $\mathcal{A}_{an}$ denote the algorithm which generates a tuple I of intervals for an analytic constraint $\mathcal{C}(X, F)$ by first translating $\mathcal{C}$ into a functional constraint

$$\phi_n(\mathcal{C}(X, Y)) = \tilde{\mathcal{C}}_n(X, F, Y, G)$$

for some real variables Y and function variables G, and then applying the functional constraint solving algorithm. Formally, $\mathcal{A}_{an}$ is defined by:

$$\mathcal{A}_{an}(\mathcal{C}(X, F)) = \pi_X(\mathcal{A}_{fn}(\tilde{\mathcal{C}}_n(X, F, Y, G)))$$

Theorem 2 *$\mathcal{A}_{an}$ is a sound analytic constraint solver.*

Proof. The constraint $\tilde{\mathcal{C}}_n(X, F, Y, G)$ has been constructed in such a way that if (x, f) is any solution to the analytic constraint $\mathcal{C}(X, F)$, then $\Gamma(X, F) = (x, f, y, g)$ is a solution to the functional constraint $\tilde{\mathcal{C}}_n(X, F, Y, G)$, where Y and G are the tuples of real and function variables, respectively, defined by $X_i = x_i$, $\mathbf{F}_{ij}(s) = f_i^{(j)}(s)$, and

$$\begin{array}{rcl} Y & = & ((A_{ij}), (B_{ij}), (W_{ih}), (U_{ijht}), (V_{ij})) \\ G(x) & = & (\mathbf{S}(x), (\mathbf{Z}_{ijt}(x))) \end{array}$$

By the soundness of the functional constraint solver $\mathcal{A}_{fn}$ we can infer that $x \in \pi_X(\mathcal{A}_{fn}(\tilde{\mathcal{C}}_n(X, F, Y, G)))$. Thus, we have shown that

$$\pi_X(\mathcal{S}_{an}(\mathcal{C}(X, F))) \subset \pi_X(\mathcal{A}_{fn}(\tilde{\mathcal{C}}_n(X, F, Y, G)))$$

QED

7 CLP semantics

The soundness of the CLP(F) analytic constraint solver implies a soundness property of CLP(F) itself. Indeed, the result of executing a CLP(F) query Q(X) to a program P is a sequence of constraints $C_i(X, Y_i, F_i)$, where X, Y_i are real variables, and F_i are function variables. If this is a finite set, then Jaffar and Lassez [15] have shown that

$$\mathcal{T} \cup P^* \models \forall X.Q(X) \leftrightarrow \bigvee_i \exists Y_i C_i(X, Y_i, F_i)$$

where $\mathcal{T}$ is the first order theory of the domain of analytic constraints and P^* is Clark's completion of the program P.

The CLP(F) interpreter uses the constraints C_i to generate an answer interval

$$I_i = \mathcal{A}_{an}(C_i(X, Y_i, F_i))$$

By the soundness of the CLP(F) analytic constraint solver $\mathcal{A}_{an}$ we can conclude that for every solution (x, y, f) of the analytic constraint C_i, we have (x, y) is contained in I_i, and hence that

$$\mathcal{T} \cup P^* \models \left(\forall x.Q(x) \Rightarrow x \in \bigcup_i \pi_x(\mathcal{A}_{an}(C_i(x, Y_i, F_i))) \right)$$

which extends the soundness result from constraints to programs.

8 Limitations

The signal property of CLP(F) is that it has a very expressive constraint language with a simple, sound constraint solving algorithm. Moreover, the examples selected in Section 4 demonstrate that in some favorable situations the current CLP(F) interpreter is able to produce answer intervals, in a few seconds or less, which are fairly narrow.

On the other hand, CLP(F) should be viewed as a *first* step toward the long term goal of developing efficient, high-precision analytic constraint solvers as there are several aspects of the current implementation of CLP(F) which need improvement. Solving these problems will likely require new ideas and techniques.

- It is is common to pose a query and have the computed answer be an interval with very wide bounds (often $[-\infty, \infty]$). Such a result indicates that the algorithm employed by the solver was ineffective. This problem usually occurs for the same reason that numerical algorithms produce incorrect results, e.g., the "step size" is too large, or the solver is using too few terms of the Taylor series to approximate the functions closely enough. An interesting research problem is to investigate for which classes of problems the solver is able to return tight bounds.
- CLP(F) does not verify that a solution does indeed exist. If a solution does not exist then CLP(F) may or may not detect this (e.g., by responding to a query with a "no" answer). If it does not detect the insolubility, then the fact that any solution must satisfy the computed constraints is still true, but misleading, since there are no solutions in this case. It is not clear how to prove the existence of solutions of general analytic constraints.

- For the special case of analytic constraints which happen to be initial value problems for ODEs, CLP(F) can not compete either in speed or precision with the current interval based tools such as AWA [18]. It might be possible to develop an optimizing compiler which would lessen or eliminate this gap.
- One often needs to adjust various tuning parameters to attain the speed and precision desired. For example, the degree of the power series to be used is a key parameter which affects both execution time and precision. There might be other ways of dynamically adjusting these parameters in an optimal way.
- The current CLP(F) constraint solving mechanism seems only to work for ODE's which have analytic solutions. It is not clear how to extend it to handle more complex mathematical objects and relations, such as analytic functions of several variables and PDEs, or functions with singularities at the endpoints.

Acknowledgements

I would like to thank the referees for their helpful comments based on a close reading of the text. I would also like to thank Maarten van Emden and Qun Ju for many helpful conversations about interval arithmetic constraint solving.

References

[1] Forman S. Acton, Real computing made real: Preventing Errors in Scientific and Engineering calculations, Princeton University Press, Princeton, New Jersey, 1996.

[2] Frédéric Benhamou and William J. Older, Applying interval arithmetic to real, integer, and Boolean constraints, *Journal of Logic Programming*, 32:1–24, 1997.

[3] Bell Northern Research Ltd., BNR Prolog Reference Manual, 1988.

[4] Bell Northern Research Ltd., BNR Prolog User Manual, 1988.

[5] Gerd Bohlender, Bibliography on enclosure methods and related topics, In Scientific Computing with Automatic Result Verification, U. Kulisch, E. Adams, editors, Academic Press, 1993.

[6] Cucker, F. and Smale, S. Complexity Estimates Depending on Condition and Round-Off Error, Journal of the ACM, 46:1, pp. 113-184, 1999.

[7] Deville, Y. and Janssen, M., and van Hentenrych, P. Consistency Techniques in Ordinary Differential Equations in Principles and Practice of Constraint Programming – CP98, M. Maher and J-F. Puget (eds.), Springer-Verlag, LNCS v. 1520, 1998.

[8] Daniel Dìaz, GNU Prolog, `www.gnu.org/software/prolog`, 1999.

[9] T. Hickey, CLP(F) and Constrained ODEs, in the Workshop on Constraint Languages and their use in Problem Modeling, ECRC Tech. Report (defunct), 1994.

[10] T. Hickey, The Clip 1.0.2 implementation of CLP(F), `www.cs.brandeis.edu/~tim/clip`, 1999.

[11] T. Hickey, CLIP: A CLP(Intervals) dialect for Metalevel Constraint Solving, to appear in Proceedings of the conference on Practical Applications of Declarative Languages (PADL'00), Jan. 2000.

[12] T. Hickey, H. Wu, and M.H. van Emden, A Unified Framework for Interval Constraints and Interval Arithmetic, in Principles and Practice of Constraint Programming – CP98, M. Maher and J-F. Puget (eds.), Springer-Verlag, LNCS v. 1520, pp. 250-264, 1998.

[13] T. Hickey, Q. Ju, and M.H. van Emden, Interval Arithmetic: from Principles to Implementation, Brandeis University CS Tech Report, CS-99-202, April 1999.

[14] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz, *Numerical Toolbox for Verified Computing I*, Springer-Verlag, 1993.

[15] Jaffar, J. and Lassez, J. L., Constraint Logic Programming, In Proceedings of the 14th ACM Symposium on the Principles of Programming Languages, 1987.

[16] M. Janssen and Y. Deville and P. van Hentenryck, Multistep Filtering Operations for Ordinary Differential Equations Proceedings of CP'99, Springer-Verlag, LNCS, 1999

[17] Qun Ju, A Sound Interval Constraint Logic Programming System, PhD. Dissertation, Brandeis University, 1998.

[18] R. J. Lohner, Einschielßung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen, Ph.D. thesis. Universität Karslruhe, 1988.

[19] Moore, R. E., Automatic local coordinate transformations to reduce the growth of error bounds in interval computation of solutions of ordinary differential equations, In Error in Digital Computation, Vol II, L.B. Rall, ed., Wiley, New York, 1965, pp. 103-140, 1965.

[20] Moore, R. E., Interval Analysis. Prentice-Hall, 1966.

[21] Ned S. Nedialkov and Ken R. Jackson and George F. Corliss, Validated Solutions of Initial Value Problems for Ordinary Differential Equations, Appl. Math. and Comp., to appear, 1999.

[22] Ned S. Nedialkov, Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation Ph.D. Dissertation, Univ. of Toronto, Dept. of CS. 1999.

[23] William Older and Frédéric Benhamou. Programming in CLP(BNR). Proceedings of PPCP'93, 1993.

[24] Older, W. and Vellino, A., Constraint Arithmetic on Real Intervals, in Constraint Logic Programming: Selected Research, Colmerauer, A. and Benhamou, F. (eds), MIT Press 1993.

[25] Pascal Van Hentenryck, Laurent Michel, and Yves Deville, *Numerica: A Modeling Language for Global Optimization*, MIT Press, 1997.