Working Paper Series
ISSN 1170-487X

Proving the existence of solutions in logical arithmetic

by John G. Cleary

Working Paper 93/5

October, 1993

Proving the Existence of Solutions in Logical Arithmetic

John G. Cleary,
University of Waikato,
Hamilton, New Zealand.
email: jcleary@waikato.ac.nz

Abstract

Logical arithmetic is a logically correct technique for real arithmetic in Prolog which uses constraints over interval representations for its implementation. Four problems with the technique are considered: answers are conditional and uninformative; iterative computations may lead to unboundedly large constraint networks; it is difficult and ineffective to deal with negation; and computing extrema is often not effective. A solution to these problems is proposed in the form of "existential intervals" which record the existence of a solution to a set of constraints within an interval. It is shown how to operate on existential intervals and how they solve the four problems.

Introduction

Logical arithmetic is a technique for embedding arithmetic within logic programming languages such as Prolog. It was first described in [Cleary, 1987]. Implementations have been completed for a number of Prolog systems [Sidebottom and Havens,1992], [Older and Vellino, 1990], [Older and Vellino, 1993], [Hyvönen, 1992] gives a good survey of constraint systems over intervals. The other major proposal for arithmetic in Prolog is CLP(R) [Lee and van Emden,1992], [Jaffar and Lassez, 1987a], [Jaffar and Lassez, 1987b]. This system uses symbolic manipulation of equations as well as a built-in linear equation solver. When sets of equations are in linear form (or if they become linear during computation) then the equations are automatically solved. The biggest difference between logical arithmetic and CLP(R) is that it cannot deal with non-linear sets of equations. It also has a serious flaw, in that it may report there are no solutions, when there are, as a result of rounding errors in the underlying floating point arithmetic.

Interval arithmetic, as presented in [Aberth, 1988], [Alefeld and Herzberger, 1983], [Moore, 1966], [Ratscheck and Rokne, 1984], [Ratscheck and Rokne, 1988] performs operations on intervals of real numbers. Thus the +, -, *, and (more problematically) / all have their corresponding interval counterparts. Such arithmetic has traditionally been dealt with functionally. That is, an operation such as + is applied to two intervals and the result is an interval.

Logical Arithmetic

In logical arithmetic statements are represented as constraints on logical variables. For example, the statement:

```
    X is 2*Y + Z
```

is represented by the set of relational constraints:

```
    multiply(2,Y,T1), add(T1,Z,X).
```

(where `T1` is a new temporary variable). The intended meaning is that `T1`, `X`, `Y` and `Z` are real values which satisfy the constraints.

In order to be able to effectively compute with such multi-variable constraints interval constraints are placed on single variables. For example, `Y` and `Z` might be constrained as:

```
    Y≥1.0, Z≥3.5
```

Using these and the `multiply` and `add` constraints it is possible to deduce that `X≥5.5`. At any one point in the calculation there will be at most two such inequalities which bound a variable above or below by some floating point constant. For example, if `X≤5.5` and `X≤6.0` this can be simplified to just `X≤5.5`. Individual constraints such as `add` are able to locally relax the intervals associated with its parameters. Fig. 1 shows an example of relaxing `add` with an initial set of interval constraints and the resulting constraints after relaxation. References such as [Cleary,1987] give details of how to compute such local relaxations. By ensuring that the resulting intervals are rounded out it is possible to avoid losing any potential solutions and never to make incorrect deductions.

Initial Constraints:			
add(X,Y,Z),	0<X, X<4,	1<Y, Y<4,	7<Z, Z<9
Relaxation:			
	(7-4)=3<X X<(9-1)=8		
		(7-4)=3<Y Y<(9-0)=9	
			(0+1)=1<X X<(4+4)=8
Derived Constraints:			
add(X,Y,Z),	3<X, X<4,	3<Y, Y<4,	7<Z, Z<8

Figure 1. Local Relaxation of add Constraint.

As can be seen from the examples above and from Fig. 1 this system of arithmetic follows the tradition of Prolog in allowing any parameter to function as an input or ouput. Indeed in Fig 1. all three intervals end up smaller after the relaxation. However, some systems of constraints might seem to be problematic. For example, the constraint `multiply(X,X,2)` has as its only solutions the two square roots of 2. There is a general search technique called *splitting* for solving such non-linear equations. The interval for a variable such as `X` is split into two parts, and the parts visited by backtracking. Fig. 2 shows the splitting process for this example. It starts with the only constraint on `X` being `multiply(X,X,2)` which is unable to introduce any new constraints by local relaxation. At the first split the two constraints `X≥0` or `X<0` are introduced by backtracking. On the `X≥0` branch local relaxation is still unable to progress and a further split to `1>X,X≥0` and `X≥1` is introduced. The `1>X,X≥0` branch immediately fails as no solutions are possible. The `X≥1` branch is able to relax the multiply constraint to give `2≥X, X≥1`. This is then further split into the two intervals `2≥X, X≥1.5`, and `1.5>X, X≥1.0` and so on. The splitting continues until the limit of machine precision is reached (in this case fixed point arithmetic with two decimal digits of precision is assumed for the sake of intelligible examples).

With a fixed word size it is trivial that this is a complete procedure for solving equations to within the limits of the available machine precision. In the case of polynomials (such as the square root example above) where there are no multiple roots the above procedure converges quadratically (which is remarkable given that it does not explicitly compute the derivative). However, as we will see there are still difficulties with this procedure.

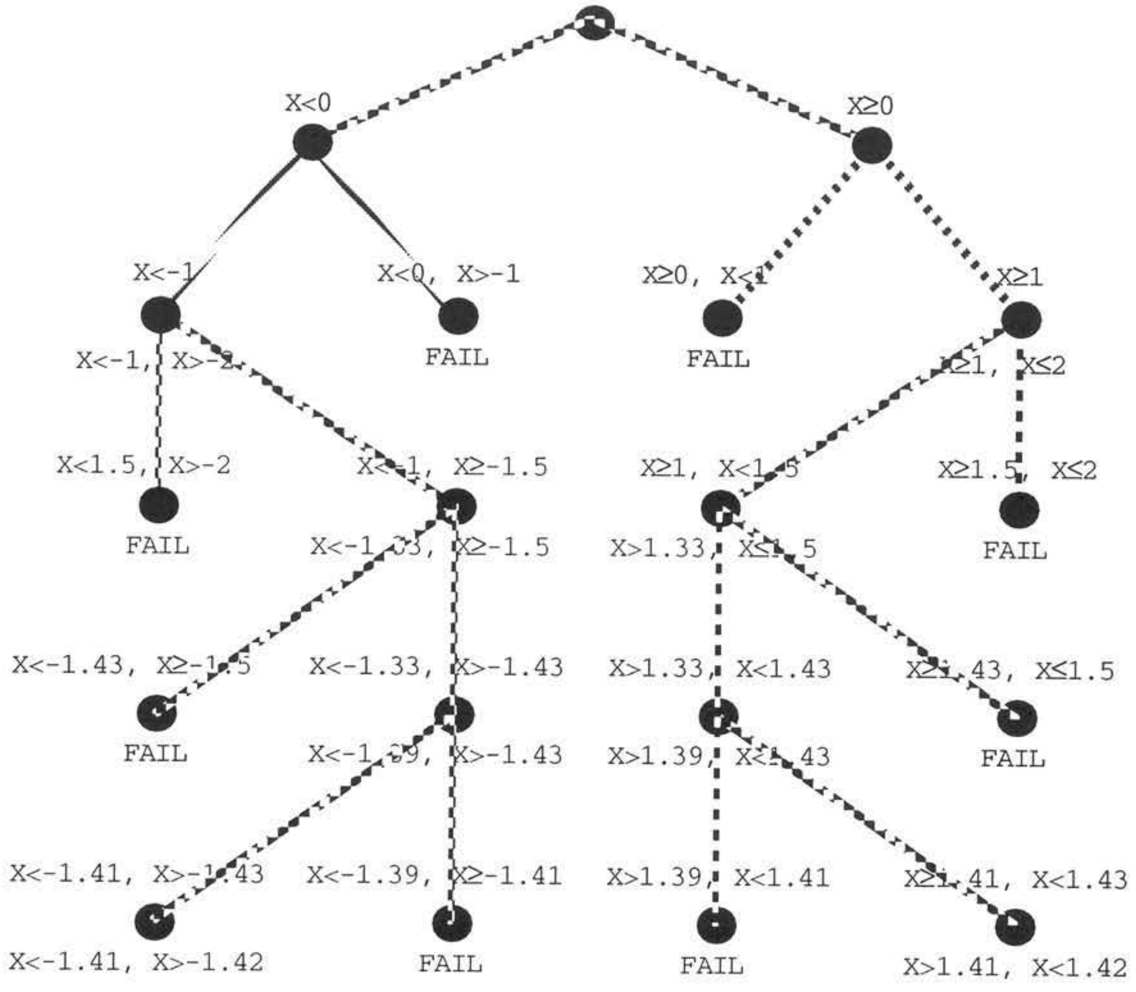

Figure 2. Splitting to Solve `multiply(X,X,2)`

P

roblems

Problem 1 - No Actual Solutions

Unlike the usual case in logic programming results in logical arithmetic are often not discrete values. As a result, answers to queries are often conditional. Consider the case when it is known that `1<X<2` and `3<Y<4`. Let `Z` be the sum of these two numbers then it is possible to deduce that `4<Z<6`. Expressed in logical arithmetic and written as a set of constraints this gives the following query:

```
        :- 1<X, X<2, 3<Y, Y<4, add(X,Y,Z).
```

The result of this query is the following conditional answer:

```
        :- 1<X, X<2, 3<Y, Y<4, add(X,Y,Z), 4<Z, Z<6..
```

This does not claim that there is a solution, it just deduces some facts that might be true if there is an overall solution. This is a relatively simple example of this problem. Some non-trivial examples are given in the rest of this subsection and in Problem 4.

To see the next problem it is necessary to examine in more detail the answer returned by a query such as the one in Fig. 2. The result is two sets of constraints:

```
:-multiply(X,X,2), 1.41<X, X< 1.42
:-multiply(X,X,2),-1.42<X, X<-1.41
```

Note that neither of these says what is a solution merely that if `X` lies between 1.41 and 1.42 (or -1.42 and -1.41) and `multiply(X,X,2)` is a solution then `X` is a solution. This is not very informative. Indeed more is known about what is not a solution than what is. Thus it is known that there is no possible solution above 1.42. It happens that there is a unique square root of 2 lying between 1.41 and 1.42 but this is not readily apparent from this answer. It would be much more satisfactory if the answer returned were something of the form:

there exists a real `c, 1.41<c, c<1.42` which is a solution to `multiply(c,c,2)`.

This problem is not trivial. Consider for example:

:- 0 is $9*X^2$ - 12*X + 4.01.[1]

This has no solution although this is not readily apparent. Relaxation and splitting returns the answer:

0.66<X, X<0.67, 0 is $9*X^2$ - 12*X + 4.01.

Within the limits of machine arithmetic assumed (two decimal places) the expression $9*X^2$ - 12*X + 4.01 evaluates to the interval `-0.16` to `0.14`. That is, the system is unable to exclude 0 as a possible solution when `X` lies in the interval `0.66` to `0.67` (In fact, symbolic reasoning shows that there are no real solutions to this equation).

Problem 2 - Negation

Consider the problem of deciding the goal `not(sqrt(-2))` using negation by failure. `sqrt` is defined by the clause:

```
sqrt(Y) :- Y is X*X.
```

That is `sqrt(Y)` succeeds iff `Y` has a square root. The call `sqrt(-2)` quickly fails after the first level of splitting and hence `not(sqrt(-2))` `succeeds`. However, the goal `not(sqrt(2))` is not so easily dealt with. As above two possible solutions are yielded but there is no guarantee that they are in fact solutions. As a result the goal `not(sqrt(2))` cannot be failed.

Problem 3 - Computation of Extreme Points

A common problem in numerical computation is to find the extreme points of a function [Ratscheck and Rokne, 1988]. Consider for example the problem of finding the maximum of X^2 over the range `0<X<1.0`. Reformulated this is a search for the maximum value of `Y` given the constraints `multiply(X,X,Y), 0<X, X<1`. A search for the maximum can be undertaken using splitting. Such a search yields the following results:

```
multiply(X,X,Y), 0.99≤X, X<1.00, 0.98<Y, Y<1.00
multiply(X,X,Y), 0.98≤X, X<0.99, 0.96<Y, Y<0.99
multiply(X,X,Y), 0.97≤X, X<0.98, 0.94<Y, Y<0.97
multiply(X,X,Y), 0.96≤X, X<0.97, 0.92<Y, Y<0.95
...
multiply(X,X,Y), 0.00<X, X<0.01, 0.00<Y, Y<0.01
```

Unfortunately, nothing about the maximum can be deduced from such a set of results. None of the individual results actually guarantees that there are <u>any</u> solutions to the constraints.

[1]In its expanded relational form this example gives:

```
:- add(4.01,T1,0), add(T3,T2,T1), multiply(12,X,T2),
        multiply(9,T4,T3), multiply(X,X,T4).
```

In complex examples such as this I will take the short form to be an abbreviation for this longer relational form.

What is desirable from such a set of results would be a guarantee that there exists a solution for X and Y in the first result:

```
    multiply(X,X,Y), 0.99≤X, X<1.00, 0.98<Y, Y<1.00
```

Given the existence of this solution it is possible to eliminate all the other solutions except the second which overlaps with the first:

```
    multiply(X,X,Y), 0.98≤X, X<0.99, 0.96<Y, Y<0.99
```

Given just these two solutions and an existence proof it would be possible to deduce that the maximum for Y lies between `0.98` and `1.00` and that the maximum occurs somewhere in the interval `0.98≤X<1.00`.

Problem 4 - Execution Efficiency

Given a long iterative calculation, for example, the following Prolog predicate and query:

```
    :- p(0.42).
    p(X):- 0≤X, X≤1, print(X), Y is 4*X*(1-X), p(Y).
```

Given tail-recursive optimization it might be expected that this program would execute in a fixed amount of space and print out an infinite number of consecutive values for X. However, the successive values of X are linked into a single constraint network which grows linearly with the number of iterations. Not only does the network grow so does the amount of computation as the range constraint on X (0≤X, X≤1) will cause relaxation operations to propagate backwards down the chain of constraints making each computation step proportional to the number of iterations so far.

Existence of Solutions

The first step in the resolution of these problems stems from results available about the primitive constraints such as `multiply` and `add`. Consider for example the constraint `multiply(X,X,Y)`. There are two existence results known about such a constraint. First given any value of X there is a value of Y satisfying the constraint. This follows simply from the fact that multiplication is defined for all values of its input. It is also true that for any given positive value of Y there is a value of X which satisfies the constraint. This is not as trivial and follows from the fact that X^2 is a continuous function with a minimum of 0 and a maximum of ∞.

A stronger statement about X^2 is possible. Because the function is monotone for X positive or negative then it is possible to bracket the square root of Y. Given a value for Y say y, if there is a pair x_l and x_h such that $x_l^2 \le y \le x_h^2$ and $0 \le x_l^2$ then there exists an x such that $y=x^2$ and $x_l \le x \le x_h$. Stated formally the result for the positive branch is:

$$\forall\ 0 \le x_l \le x_h,\ x_l^2 \le Y \le x_h^2 \Rightarrow \exists\ X\ \texttt{multiply(X,X,Y)},\ x_l \le X \le x_l \qquad (1)$$

Substituting 2 for Y gives:

$$\forall\ 0 \le x_l \le x_h,\ x_l^2 \le 2 \le x_h^2 \Rightarrow \exists\ X\ \texttt{multiply(X,X,2)},\ x_l \le X \le x_l$$

This says that if a bracketing pair x_l , x_h can be found then a region can be identified in which a square root for 2 is guaranteed to exist . Substitute 1.41 for x_l and 1.42 for x_h. Computing with finite precision interval arithmetic $1.98 \le 1.41^2 \le 1.99$, that is it is guaranteed that $1.41^2 \le 2$ and similarly it is guaranteed that $2 \le 1.42^2$. Thus the left hand side of the implication above is satisfied and the following can be deduced:

$$\exists\ X\ \texttt{multiply(X,X,2)},\ 1.41 \le X \le 1.42$$

Recasting this using a Skolem constant c gives three facts:

```
multiply(c,c,2).
1.41≤c.
c≤1.42.
```

Armed with these results Problem 1 can now be reconsidered. The goal to be solved is:

```
:- multiply(X,X,2).
```

After applying splitting, two conditional results are obtained:

```
:- multiply(X,X,2),  1.41<X, X<  1.42.
:- multiply(X,X,2), -1.42<X, X< -1.41
```

These two results can be made unconditional by the answers:

```
X = c1
```

and

```
X = c2
```

together with the following auxiliary facts:

```
multiply(c1,c1,2).
1.41≤c1.
c1≤1.42.
multiply(c2,c2,2).
-1.42≤c2.
c2≤-1.41.
```

These facts show that c_1 and c_2 are both solutions to all the goals in the conditional results. That is the goals can be deleted and replaced by the bindings of `X` to c_1 and c_2. There are no residual unsatisfied goals and the answer is thus unconditional.

The same reasoning enables Problems 2 and 3 to be solved as well. In Problem 2 the goal `sqrt(2)` is solved unconditionally and so `not(sqrt(2))` can be finitely failed. In Problem 3 it can be shown that

```
multiply(X,X,Y), 0.99≤X, X<1.00, 0.98<Y, Y<1.00
```

has a solution and so the the list of interesting solutions can be pruned.

Expression (1) can also be used when `Y` is known to lie within an interval rather than having a single value. For example given the constraints:

```
:- 4≤Y, Y≤9, multiply(X,X,Y)
```

it is possible to substitute 2 for x_l and 3 for x_h. This gives the following specialization of (1):

$$\forall\ 4\leq Y\leq 9 \Rightarrow \exists\ X\ \texttt{multiply(X,X,Y)}. 2\leq X\leq \qquad (2)$$

The left hand side of the implication is satisfied and `X` can be replaced by the Skolem constant c_3`(Y)` to give the following unconditional result:

```
:- 4≤Y, Y≤9, X=c3(Y)
```

together with the following auxiliary facts:

```
2≤c3(Y).
c3(Y)≤3.
multiply(c3(Y),c3(Y),Y) :- 4≤Y, Y≤9.
```

Such generalization can be taken to its most extreme point by substituting 0 for x_l and leaving x_h unbounded. This gives the implication:

$$\forall\ 0\leq Y \Rightarrow \exists\ X\ \texttt{multiply(X,X,Y)}. 0\leq X \qquad (3)$$

That is, every `Y` greater than 0 has a positive square root. I will refer to such existence conditions which omit details of the bounds on the variables as *contracted existence conditions* and the form with the bounds as *full existence conditions*. The appendix gives a list of non-trivial existence conditions that are useful for the primitive constraints `multiply` and `add`. In the interests of space only the contracted forms are given, although the full forms are easily deduced from them.

This reasoning can be generalized to more complex equations with more than one primitive constraint. If there is a constraint of the form `Y is f(X)` (for some function `f`). Then on an interval where `f` is continuous and monotonic (has a positive or negative derivative) there is guaranteed to be a solution. The following expression casts this in the same form that was used in expression (1):

$$\forall\ \forall z\left(x_a \le z \le x_b \Rightarrow \frac{\partial f(z)}{\partial z} \ge 0\right),\ x_a \le x_l \le x_h \le x_b,\ f(x_l) \le Y \le f(x_h) \Rightarrow \exists\ X\ \ Y \text{ is } f(X),\ x_l \le X \le x_l \tag{4a}$$

Simplifying as in equation (3) yields:

$$\forall\ \forall z\left(x_a \le z \le x_b \Rightarrow \frac{\partial f(z)}{\partial z} \ge 0\right),\ f(x_a) \le Y \le f(x_b) \Rightarrow \exists\ X\ \ Y \text{ is } f(X) \tag{4b}$$

To use this general result as part of a numerical computation it is necessary to compute the derivative of `f` to ensure that it is in a monotonic region as well as computing `f` at the end-points x_l and x_h. Such computations must be done carefully using intervals and appropriate rounding to ensure that the conditions are provably true.

At this point the first example of Problem 1 has not been dealt with. That is the query:

```
    :- 1<X, X<2, 3<Y, Y<4, add(X,Y,Z).
```

still returns the conditional answer:

```
    :- 1<X, X<2, 3<Y, Y<4, add(X,Y,Z), 4<Z, Z<6..
```

To make this answer unconditional requires that the goal `add(X,Y,Z)` somehow be satisfied. Following the same type of reasoning used earlier the following existence result is available:

$$\forall\ X\ Y\ \exists\ Z\ \ \text{add}(X,Y,Z). \tag{5}$$

That is, for every value of `X` and `Y` there is a sum which is the result of adding them. As in equation (2) `Z` can be replaced by a constant $c_4(X,Y)$ which gives the following unconditional result and auxiliary fact:

```
    :- 1<X, X<2, 3<Y, Y<4, Z=c4(X,Y).
    add(X,Y,c4(X,Y)).
```

The full version of (5) gives additional auxiliary facts:

```
    add(X,Y,c4(X,Y)):- 1<X, X<2, 3<Y, Y<4.
    4<c4(X,Y).
    c4(X,Y)<6.
```

This type of reasoning can be extended to arbitrarily complex expressions. Consider, for example, the constraints:

```
    :- 1<X, X<3, 2<Y, Y<4, Z is X*Y + X, .
```

or in relational form:

```
    :- 1<X, X<3, 2<Y, Y<4, add(X,T,Z), multiply(X,Y,T).
```

After local relaxation this gives:

```
    :- 1<X, X<3, 2<Y, Y<4, add(X,T,Z), multiply(X,Y,T),
            2<T, T<12, 3<Z, Z<15.
```

Using the full form of equation (A3) in the appendix there must be a solution to `multiply(X,Y,T)` binding `T` to a constant $c_5(X,Y)$. This allows `multiply` to be deleted from the constraints giving:

```
    :- 1<X, X<3, 2<Y, Y<4, add(X,T,Z), T=c5(X,Y), 3<Z, Z<15.
and multiply(X,Y,c5(X,Y)):- 1<X, X<3, 2<Y, Y<4.
    2<c5(X,Y).
    c5(X,Y)<12.
```

But rule (A2a) can now be applied to bind Z to a constant c_6(X, c_5(X,Y)) and delete add, finally giving the unconditional answer:

```
        :- 1<X, X<3, 2<Y, Y<4, T=c5(X,Y), Z=c6(X,c5(X,Y)).
and     add(X,c5(X,Y),c6(X,c5(X,Y))):- 1<X, X<3.
        3<c6(X,c5(X,Y)).
        c6(X,c5(X,Y))<15.
```

In this way unconditional answers can be derived for arbitrarily complex sets of constraints. Thus problem 4, where an iterative calculation builds up an unboundedly large constraint network, can be solved by repeatedly proving the existence of the solution. This keeps the size of the active constraint network constant.

This form of reasoning is effectively functional. That is, it successively eliminates the constraints derived from the leaves of an expression until a solution to the complete expression is proven. This is very close to the computations done in functional interval arithmetic. There is a small measure of added flexibility as the various rules in the appendix allow deferring the decision of which direction to propagate the existential results until after relaxation has occurred.

Future Work

To actually provide a complete system within Prolog using these results requires solving a number of control issues not explored here. For example, it is not clear when an existence result should be sought as opposed to further relaxation. It is not clear to what extent this can be automatic and to what extent it is necessary to give the programmer control of these issues. Work is currently underway to include existential intervals in a logical arithmetic system.

Acknowledgements

Part of this work was completed under a grant from the Natural Sciences and Engineering Research Council of Canada.

Appendix

This appendix gives a list of non-trivial existence conditions that are useful, including those for the primitive constraints multiply and add. In the interests of space only the contracted forms are given, although the full forms are easily deduced from them. Each rule can be used as a rewrite rule to delete a set of constraints and replace them by a binding of a variable to a real constant (together with auxiliary facts about the constant).

Addition

$\forall$ X Y $\exists$ Z add(X,Y,Z) (A1)

Subtraction

$\forall$ X Y $\exists$ Z add(X,Z,Y) (A2a)

$\forall$ X Y $\exists$ Z add(Z,X,Y) (A2b)

Multiplication

$\forall$ X Y $\exists$ Z multiply(X,Y,Z) (A3)

Division

$\forall$ X Y>0 $\exists$ Z multiply(Z,Y,X) (A4a)

$$\forall\ X\ Y<0\ \exists\ Z\ \ \texttt{multiply(Z,Y,X)} \tag{A4b}$$

$$\forall\ X\ Y>0\ \exists\ Z\ \ \texttt{multiply(Y,Z,X)} \tag{A4c}$$

$$\forall\ X\ Y<0\ \exists\ Z\ \ \texttt{multiply(Y,Z,X)} \tag{A4d}$$

Square Root

$$\forall\ 0\leq Y \Rightarrow \exists\ X\ \texttt{multiply(X,X,Y)},0\leq X \tag{A5a}$$

$$\forall\ 0\leq Y \Rightarrow \exists\ X\ \texttt{multiply(X,X,Y)},0\geq X \tag{A5b}$$

General Case

$$\forall\ \forall z\left(x_a\leq z\leq x_b \Rightarrow \frac{\partial f(z)}{\partial z}\geq 0\right),\ f(x_a)\leq Y\leq f(x_b) \Rightarrow \exists\ X\ Y\ \texttt{is f(X)} \tag{A6a}$$

$$\forall\ \forall z\left(x_a\leq z\leq x_b \Rightarrow \frac{\partial f(z)}{\partial z}\leq 0\right),\ f(x_a)\leq Y\leq f(x_b) \Rightarrow \exists\ X\ Y\ \texttt{is f(X)} \tag{A6b}$$

References

Aberth, O. (1988) *Precise Numerical Analysis*, Wm. C. Brown, Dubuque.

Alefeld, G., and Herzberger, J. (1983) *Introduction to Interval Computations*, Addison-Wesley, Reading, MA.

Cleary, J.G.(1987) "*Logical Arithmetic*," Future Computing Systems, **2**(2), pp. 125-149.

Cole, A., and Morrison, R.(1982) "*Triplex: A System for Interval Arithmetic*," Software Practice and Experience, **12**, pp. 341-350.

Hyvönen, E.(1992) "*Constraint Reasoning Based on Interval Arithmetic: the Tolerance Propagation Approach*," Artificial Intelligence, **58**, pp. 71-1123.

Jaffar, J., and Lassez, J.-L.(1987a) "*Constraint Logic Programming*," Proceedings Fourteenth Conference on Principles of Programming Languages, Munich, pp. 111-119.

Jaffar, J., and Lassez, J.-L.(1987b) "*Methodology and Implementation of a CLP System*," Fourth Int. Conf. on Logic Programming, Melbourne, Australia, July.

Lee, J.M.H., and van Emden, M.H.(1992) "*Adapting CLP(R) to Floating Point Arithmetic*," Proc. Fifth Generation Comp. Systems Conf., Tokyo.

Moore, R. (1966) *Interval Arithmetic*, Prentice-Hall, Englewood Cliffs, NJ.

Older, W., and Vellino, A.(1990) "*Extending Prolog with Constraint Arithmetic*," Proc. Canadian Conf. on Electrical and Computer Eng., Banff, Alberta.

Older, W., and Vellino, A.(1993) "*Constraint Arithmetic on Real Intervals*," in Constraint Logic Programming, Collected Research, (Benhamou, F., and Colmerauer, A. eds.), MIT Press Cambridge, MA.

Ratscheck, H., and Rokne, J. (1984) *New Computer Methods for Global Optimization*, Ellis Horwood, Chichester, England.

Ratscheck, H., and Rokne, J. (1988) *Computer Methods for the Range of Functions*, Ellis Horwood, Chichester, England.

Sidebottom G., and Havens W.(1992) "*Hierarchical Arc Consistency Applied to Numeric Processing in Constraint Logic Programming*," Computational Intelligence, **8**(4), pp. 601-623.