Reliable Computing 2 (2) (1996), pp. 97–102

Mechanising the theory of intervals using OBJ3

Marcilia A. Campos, Augusto C. A. Sampaio, and Alexandre H. F. Brainer

This work uses the OBJ3 term rewriting system to formalize the type interval through some of its algebraic and topological properties. The intended contribution is not the discovery of new theorems related to numerical or interval arithmetic. Rather, our aim is to formally specify the type interval with the view of performing mechanical (automated) reasoning about its properties.

Автоматизация выводов в теории интервалов с использованием OBJ3

М. Кампос, А. Сампайо, А. Брайнер

С помощью системы переписывания термов OBJ3 производится формализация интервального типа через некоторые из его алгебраических и топологических свойств. Авторы не ставят себе целью установление новых теорем, относящихся к области численной или интервальной арифметики. Целью работы является формальное определение интервального типа и обеспечение возможности автоматизированного получения сведений о его свойствах.

1. Introduction

Application programs that have been formally proved correct can only be reliable if they run on hardware that has been formally proved correct, under correct basic software, such as operating systems, environments and tools.

If the real numbers are represented with fixed precision (single or double), the most common form of representing them is using floating-point numbers. The system of floating-point numbers neither enables to preserve algebraic properties nor permits controlling and evaluating numerical errors that may arise in a sequence of arithmetic operations.

Moore [7] has defined the set of real intervals. Each element of this set is a compact subset of the real line. The interval arithmetic operations with high accuracy [5, 7] use the optimal dot product and interval arithmetic as essential tools in addition to floating-point arithmetic. Interval arithmetic allows the computation of guaranteed bounds for the solutions of a problem. Validation by interval arithmetic is necessary to get results of high and guaranteed accuracy.

The programming languages Pascal–XSC [4], ACRITH–XSC [9], and C–XSC [6] are extensions of the programming languages Pascal, FORTRAN, and C++, respectively. These extensions, beyond the usual primitive data types, have the type `interval` and the arithmetic operations are defined by the horizontal method [5]. The availability of the type `interval` raises the computational cost, but also increases the reliability of numerical results.

This paper introduces the type interval into a programming language, but with a different objective. The main goal here is proving theoretical properties about intervals. This work uses OBJ3 [3] to formalize the type interval through some of its algebraic and topological properties.

OBJ3 is a functional language which includes mechanisms for theorem proving. From the point of view of the interval mathematics the proofs here can be found in the references about the subject [1, 7]. Therefore, the intended contribution is not the discovery of new theorems related to numerical or interval arithmetic. Rather, our aim is to formally specify the type interval with the view of performing mechanical (automated) reasoning about its properties. It is also worth emphasizing that the *directed roundings* (∇, Δ) [5] are not considered, because we are concerned with the properties of the interval arithmetic, and not with numerical algorithms.

This paper is organized in the following way. Section 2 shows the properties about intervals that have been proved. Section 3 gives an overview of the OBJ3 language. Section 4 describes the formalization of the type interval using OBJ3. Section 5 summarizes our achievements and suggests topics for further research.

2. The properties

Let X, Y and Z be intervals. The following properties [1, 7] have been proved.

- $X + Y = Y + X$.
- $X + (Y + Z) = (X + Y) + Z$.
- $X + 0 = X$.
- $0 \in X - X$.
- $X * Y = Y * X$.
- $X * (Y * Z) = (X * Y) * Z$.
- $X * 1 = X$.
- $1 \in X/X$, $0 \notin X$.
- $X * (Y + Z) \subseteq X * Y + X * Z$.
- $X \cap Y = Y \cap X$.
- $X \cap (Y \cap Z) = (X \cap Y) \cap Z$.
- $X \cup Y = Y \cup X$.
- $X \cup (Y \cup Z) = (X \cup Y) \cup Z$.
- $X \subseteq Y \Rightarrow w(X) \leq w(Y)$.
- $w(X + Y) = w(X) + w(Y)$.
- $w(X - Y) = w(X) + w(Y)$.
- $d(X, Y) = 0 \Leftrightarrow X = Y$.
- $d(X, Y) = d(Y, X)$.
- $d(X, Z) \leq d(X, Y) + d(Y, Z)$.
- $|X| \geq 0$.
- $|X| = 0 \Leftrightarrow X = [0, 0]$.
- $|X + Y| \leq |X| + |Y|$.
- $|X * Y| = |X| * |Y|$.
- $X \subseteq Y \Rightarrow |X| \leq |Y|$.

3. OBJ3

OBJ3 [3] is a general-purpose declarative language, especially useful for specification and prototyping. A specification in OBJ3 is a collection of modules of two kinds: theories and objects. A theory has loose semantics, in the sense that it defines a variety of models. An object has tight or standard semantics; it defines, up to isomorphism, a specific model—its initial algebra. For example, the usual specification of the natural numbers as an object would describe precisely what is understood by the natural numbers. As a theory, it would describe any model that satisfies the required properties; for instance, integers would be a model. One of the consequences of the distinction between a theory and an object is that it is valid to use induction for an object, but not for a theory.

A module (an object or a theory) is the unit of a specification. It comprises a signature and a set of (possibly conditional) equations—the axioms. The equations are regarded as rewrite rules and computation is accomplished by term rewriting, in the usual way.

An elaborate notation for defining signatures is provided. As OBJ3 is based upon order sorted algebra, it provides a notion of subsorts which is extremely convenient in practice. For example, by declaring `Integer` as a subsort of `Float` no conversion function is needed to turn an integer into a float.

A general mixfix syntax can be used to define operators; users can define any syntax including prefix, postfix and infix. The argument and value sorts of an operator are declared when its syntactic form is declared.

Moreover, operators may have attributes describing useful properties such as associativity, commutativity and identity. This makes it possible to document the main properties of a given operator at the declaration level. As a consequence, the number of equations that need to be input by the user is reduced considerably in some cases. Most importantly, OBJ3 provides rewriting modulo these attributes.

Modules may be parametrized by theories which define the structure and properties required by an actual parameter for meaningful instantiation. The instantiation of a generic module requires a view—a mapping from the entities in the requirement theory to the entities in the actual parameter (module). As a simple example, an object to describe sets of arbitrary elements, say SET, should be parametrized by a theory which requires the argument module to have (at least) one sort. Then this module can be instantiated to obtain sets of elements of the desired sort.

Apart from the mechanisms for defining generic modules and instantiating them, OBJ3 provides means for modules to import other modules and for combining modules. For example, `A + B` creates a new module which combines the signature and the equations of `A` and `B`.

OBJ3 can also be used as a theorem prover. In particular, computation is accomplished by term rewriting which is a widely accepted method of deduction. If the rewrite rules of the application theory are confluent (or Church-Russer) and terminating, the exhaustive application of the rules works as a decision procedure for the theory: the equivalence of two expressions can always be determined by reducing both to normal form (an expression that can be reduced no longer); the equivalence holds if the two normal forms are (syntactically) the same.

4. Formalizing and implementing intervals

The theory INTERVAL must include the boolean (BOOL) and real (REAL) values with their respective operators and usual properties. OBJ3 has a built-in module which implements the boolean algebra (BOOL). The module REAL as used in this work is an incomplete specification of the real numbers (but which is enough for our purposes), where we introduced a sort Real and some operations such as addition, multiplication and subtraction with the relevant properties. For conciseness, we will not present BOOL nor REAL here; only the module which describes intervals is presented for the purpose of illustration.

Furthermore, given the features of the interval operations, it is necessary to use real sets. This may be done by specifying the theory of sets as a generic module (parametrized by the type of the elements). Then this generic module may be instantiated with the module REAL. Apart from the declaration of the type of elements, it is necessary to define a total order between elements, because the interval theory uses properties of sets, such as the maximum (max) and the minimum (min) of a set.

The theory INTERVAL contains a description of the type (sort) INTERVAL as well as its unary and binary operators. The module BOOL is imported using the clause protecting, which means that INTERVAL adds no new elements to this algebra and that no distinct elements in BOOL (such as true and false) are identified in INTERVAL. As the module SET is generic, it can only be imported into INTERVAL after being instantiated; in our particular case, this module is instantiated with the module REAL, as we are interested in sets of real numbers.

The relation subsort declares that intervals may be represented as sets. This facility of OBJ3 is particulary useful in the definition of intersection and union of intervals when the result of some operation may be the empty set. The constructor of the type interval ([_,_]) is a partial operator: the lower bound of the interval must be less than or equal to the upper bound. The symbols [prec 2], [prec 4], [prec 5], and [prec 6] are the precedences of the operators reciprocal, negation, multiplication and subtraction. This means, for example, that the precedence of the negation is lower than the precedence of the multiplication in terms that involve both. The additional properties of the operators are described by equations (eq) and conditional equations (cq). It is possible to associate a name (label) with the equations (see [2] for more details).

```
th INTERVAL is

 protecting BOOL .
 protecting SET[REAL] .

 sort Interval .
  subsort Interval < Set .

 op-as [_,_]    : Real Real -> Interval for [x1,x2] if x1 <= x2 .
 op _+_         : Interval Interval -> Interval [prec 6] [comm assoc id: [0,0]] .
 op -_          : Interval -> Interval [prec 4] .
 op _-_         : Interval Interval -> Interval [prec 6] .
 op _*_         : Interval Interval -> Interval [prec 5] [comm assoc id: [1,1]] .
 op _ -1        : Interval -> Interval [prec 2] .
 op _/_         : Interval Interval -> Interval [prec 5] .
 op w_          : Interval -> Real .
```

```
op absi_      : Interval -> Real .
op pm_        : Interval -> Real .
op d          : Interval Interval -> Real .
op _om_       : Interval Interval -> Bool .
op _ok_       : Interval Interval -> Bool .
op _inc_      : Interval Interval -> Bool .
op _/\_       : Interval Interval -> Interval [comm assoc] .
op _\/_       : Interval Interval -> Interval [comm assoc] .
op _pertains_ : Real Interval -> Bool .

var A B         : Interval .
var x1 x2 x3 x4 : Real .

[add]  eq [x1,x2] + [x3,x4] = [x1 + x3, x2 + x4] .
[sym]  eq - [x1,x2] = [- x2, - x1 ] .
[dif]  eq A - B = A + - B .
[eq]   eq [x1,x2] == [x3,x4] = (x1 == x3) and (x2 == x4) .
[mult] eq [x1,x2]*[x3,x4] = [min({x1 * x3} U {x1 * x4} U {x2 * x3} U {x2 * x4}) ,
                             max({x1 * x3} U {x1 * x4} U {x2 * x3} U {x2 * x4})] .
[rec]  eq [x1,x2] -1 = [1 / x2, 1 / x1] .
[div]  eq A / B = A  * B -1 .
[wid]  eq w [x1,x2] = x2 - x1 .
[abs]  eq absi [x1,x2] = max({abs(x1)} U {abs(x2)}) .
[dist] eq d([x1,x2] , [x3,x4]) = max({abs(x1 - x3)} U {abs(x2 - x4)}) .
[pm]   eq pm [x1,x2] = (x1 + x2) / 2 .
[om]   cq [x1,x2] om [x3,x4] = true if x2 < x3 .
[ok]   cq [x1,x2] ok [x3,x4] = true if x1 <= x3 and  x2 <= x4 .
[inc]  eq [x1,x2] inc [x3,x4] = if x3 <= x1 and  x2 <= x4
                                then true
                                else false
                              fi .
[int]  eq [x1,x2] /\ [x3,x4] = if x1 > x4 or x2 < x3
                                then {}
                                else [max ({x1} U {x3}), min({x2} U{x4})]
                              fi .
[uni]  eq [x1,x2] \/ [x3,x4] = if ([x1,x2] /\ [x3,x4]) == {}
                                then {}
                                else [min ({x1} U {x3}), max({x2} U{x4})]
                              fi .
[pert] cq x1 pertains [x2,x3] = true if x2 <= x1 and x1 <= x3 .

endth
```

5. Conclusions

The main purpose of this paper was to show how a term rewrite system such as OBJ3 can be used both to formalize and to prove properties of interval mathematics.

The dual nature of the type interval (number or set) was easily solved with the subsort declaration, i.e., `subsort Interval < Set` means that the set of elements having the first sort

(`Interval`) is a subset, not necessarily proper, of the set of elements having the second sort (`Set`).

The formalization of the interval theory also illustrates the facilities related to the operator declarations in OBJ3. When, for example, commutativity is defined as an attribute (at the declaration level) there are implications related to termination.

The module system stimulates an incremental presentation of the theory. It was seen that modules may be parametrized by theories and instantiated to meet particular needs. Such facility is more powerful than the ones offered by most of the specification/programming languages, where only the type of arguments is considered; OBJ3 also specifies the properties that actual parameters must satisfy.

The theory presented here introduced only the type interval. A more complete theoretical work must formalize all the spaces of the numeric computation. This can be done constructively on top of the type interval by defining new type formation operators.

The functional language ML was originally designed to serve as the programming language for the LCF theorem prover [8]. An interesting piece of work would be to do the specification of the type interval using the specification language ML, and then compare the advantages and drawbacks between both.

References

[1] Alefeld, G. and Herzberger, J. *Introduction to interval computation.* Academic Press, N.Y., 1983.

[2] Campos, M. A. *Formalizing and implementing (or proving) properties of intervals using the OBJ3 system.* Technical Report, No. 002, DI–UFPE–BR.

[3] Goguen, J., Winkler, T., Meseguer, J., Futatsigi, K., and Jouannaud, J. *Introduction to OBJ.* Technical Report, SRI International, 1993, to appear.

[4] Hammer, R., Neaga, M., and Ratz, D. *Pascal–XSC, new concepts for scientific computation and numerical data processing.* In: Adams, E. and Kulisch, U. (eds) "Scientific Computing with Automatic Result Verification **189**", Academic Press, 1993, pp. 15–44.

[5] Kulisch, U. W. and Miranker, W. L. *Computing arithmetic in theory and practice.* Academic Press, 1981.

[6] Lawo, C. *C–XSC, a programming environment for verified scientific computing.* In: Adams, E. and Kulisch, U. (eds) "Scientific Computing with Automatic Result Verification **189**", Academic Press, 1993, pp. 71–86.

[7] Moore, R. E. *Methods and applications of interval analysis.* SIAM, Philadelphia, 1979.

[8] Paulson, L. C. *ML for the working programmer.* Cambridge University Press, 1991.

[9] Walter, W. V. *ACRITH–XSC, A Fortran-like language for verified computing.* In: Adams, E. and Kulisch, U. (eds) "Scientific Computing with Automatic Result Verification **189**", Academic Press, 1993, pp. 45–70.

Received: October 30, 1995
Revised version: November 29, 1995

Universidade Federal de Pernambuco
Departamento de Informática, Caixa Postal 7851
CEP 50740–540 Recife–PE
Brasil
E-mail: {acas, ahfb, mac}@di.ufpe.br