Specification and Verification of Spatial Data Types with B-Toolkit

Kim Yong Chun and Dang Van Hung
The United Nations University
International Institute for Software Technology
P. O. Box 3058, Macau
dvh@iist.unu.edu

Abstract

Spatial data types provide a fundamental abstraction for modelling the geometric structures of objects in space, their relationships, properties and operations. In this work, we present a formal specification and verification of spatial data types with the **B-Toolkit**. *We give a formal specification of a* realm *and operations over it using Abstract Machine Notations (***AMN***) of* **B**. *We then refine and implement a realm update operator in* **B**, *and verify formally an implementation of a realm update operator with* **B-Toolkit**.

1 Introduction

The **B-Toolkit** is an integrated tool which is designed to support the rigorous or formal development of software systems using **B**-method [1]. **B**-method, like many other formal methods, provides a formal specification language and powerful tool support for verifying a system. The verification of a system with the **B-Toolkit** is carried out by discharging the proof obligations with both the *auto-prover* and *inter-prover*. The use of auto-prover results in the proof obligations many of which are discharged automatically. Undischarged proof obligations within the auto-prover can be proved with the inter-prover by adding some user theories.

A number of examples of the interactive proof with **B-Toolkit** has been presented so far, most of them are simple. It seems that no attempt has been made with using B-toolkit to perform a complicated proof which uses many user theories. In practice, one may often encounter some obligations of which proofs are not so easy. The purpose of this work is twofold: to verify the basic algorithms for maintaining a realm, the underlying structure for spatial data types, and to present a non-trivial example of interactive proof through a realm case study.

A *realm* [4] provides a fundamental abstraction for modelling geometric structure of objects in space, their relationships, properties and operations. Spatial data in our real world may often be continuous but in computer graphics applications, the internal calculations are usually done in a finite domain. The concept of a realm was introduced for transforming geometric concepts and algorithms from the continuous domain to the discrete one. In [4], the authors proposed a definition of a realm and algorithms for fundamental operations over realms without giving a formal specification. As a result, there is no way for them to prove the correctness of those algorithms in their framework. In [5], the authors proposed a formal specification of a realm and fundamental operations over realms using the Raise Specificvation Language **RSL**, but they have not carried out a formal verification because **RSL** did not have a good tool support for doing proof at that time.

In this work, we focus on a formal verification of the fundamental operations over realm using **B-Toolkit**. To do this, we must give abstract specifications, refinements and implementations of a realm and the operations on it first.

Our paper is organised as follows. Section 2 presents a formal specification of a realm and an extended realm in **B**. In section 3, we present a specification, refinements and an implementation of the insertion operator for inserting a grid segments into a realm. Section 4 describes our interactive proof of obligations arising from the refinements. The last section is the conclusion of our paper.

2 A Specification of Realms in B

A realm is a planar graph over a finite resolution grid consisting of a finite set of grid points and non-intersecting line segments. This structure is defined by users dynamically for the underlying spatial data types. In this section we present a formal specification for a realm in **B**.

2.1 Grid, Grid Points and Grid Segments

Given a non-zero natural number *n*, a *grid* is the set of points in a two dimensional space with integer coordinates ranging from 0 to $n-1$. Each element of a grid is called a grid point or N-point.

Machine *GridPoint*, parameterised by parameter *maxint* encapsulates the *grid points*.

MACHINE *GridPoint* (*maxint*)
CONSTRAINTS $maxint \in \mathbb{N}_1$
SETS *G_POINT*
CONSTANTS *xcd* , *ycd* , *mk_gpoint*
PROPERTIES
$xcd \in G\_POINT \rightarrow NN \wedge$
$ycd \in G\_POINT \rightarrow NN \wedge$
$mk\_gpoint \in NN \times NN \rightarrow G\_POINT$
DEFINITIONS $NN \;\widehat{=}\; 0 \,..\, maxint$
END

A segment in two-dimensional space with (different) *N*-points as its end points is called an *N-segment* or a *grid segment* and is represented as a set of two *N*-points. Given two different *N*-points p and q, we use the function *mk_segment* to define the segment of which end points are p and q. Let machine *GridSegment* encapsulate the set of *grid segments*. This machine is written similarly to the previous one. We refer the readers to our report [2] for its specification. In this machine, the constants *left* and *right* represent the left and right end point of a grid segment respectively. We give the left-right order between end points of a segment as follows. If there is a vertical segment separating them, then the *left end point* is the one lying on the left of the segment, and the *right end point* the other. Otherwise, i.e. they lie on a vertical segment, the lower point is called *left* and the other is called *right*.

In the rest of this paper, we will frequently use the above basic type machine *GridPoint* together with the machine *GirdSegment*. For convenience, we introduce the machine *Grid* which is formed from the sum of specifications of theses two machines (here we assume that $maxint$ is 5000).

MACHINE *Grid*
INCLUDES *GridPoint* (*5000*) , *GridSegment*
END

We express the geometric relations between a grid point and a grid segment, and between two grid points, and also between two grid segments by the constants *rel_ps rel_pp* and *rel_ss*. For example, two segments s and t intersect iff $rel\_ss(s,t) = ss\_intersect$. This means that s and t have exactly one common point which is not a common end point. A grid point p is inside a grid segment s iff $rel\_ps(p,s) = ps\_in$. This means that p lies on the line segment connecting two end points of s, and p is not an end point of s. The value of these constants has the intuitive meaning. Because of the space limit, their specification is omitted here.

2.2 Realms

Now, we are ready to give the definition of a realm.

Definition 1 *A realm R is a set of grid points and grid segments defined over a finite grid such that:*

- *all (grid) end points of the segments of R are also in R,*
- *there is no point of R which is inside a segment of R*
- *two different segments of R do not intersect.*

A **B**-specification of a realm is given by the machine *Realm*.

MACHINE *Realm*
SEES *Grid* , *Relations1* , *Bool_TYPE*
SETS *SP* **;** *REALM*
CONSTANTS *pot* , *seg* , *is_realm*
PROPERTIES
$REALM = \{\, rm \mid rm \in SP \wedge$
$\quad is\_realm (rm) = TRUE \,\} \wedge$
$pot \in SP \rightarrow \mathbb{F} (G\_POINT) \wedge$
$seg \in SP \rightarrow \mathbb{F} (G\_SEGMENT) \wedge$
$is\_realm \in SP \rightarrow BOOL \wedge$
$\forall\, sp \,.\, (sp \in SP \Rightarrow$
$\quad is\_realm (sp) = TRUE \Leftrightarrow$
$\quad\quad (\forall\, ss \,.\, (ss \in seg (sp) \Rightarrow$
$\quad\quad \{\, left (ss) , right (ss) \,\} \subseteq pot (sp)) \wedge$
$\quad \forall\, (pp , ss) \,.\, (pp \in pot (sp) \wedge ss \in seg (sp) \Rightarrow$
$\quad \neg (rel\_ps (pp , ss) = ps\_in)) \wedge$
$\quad \forall\, (ss , tt) \,.\, (ss \in seg (sp) \wedge tt \in seg (sp) \Rightarrow$
$\quad \neg (rel\_ss (ss , tt) = ss\_intersect))))$
END

In the machine *Realm*, the set *SP* will be instantiated as the union type of two types *G_POINT* and *G_SEGMENT*. The set *REALM* encapsulates a realm which consists of grid points and grid segments. The constants *pot* and *seg* represent the realm grid points and the realm grid segments respectively. The basic properties of a realm are represented by the constant *is_realm*. Although the definition of realms is simple, it is not trivial to maintain the basic properties of a realm during modification. Fundamental operations for modification of realms are the insertion and deletion of points and segments. These operations must preserve the the topological structure presented over the grid as well as be close to their intuitive meaning. But this is almost impossible for a realm structure just defined above. For making this possible, some additional constraints are needed for a realm.

2.3 Extended Realms

The application data at the lowest level of abstraction can be viewed as a set of points and intersecting line segments. When these data are transformed into a realm, the intersecting line segments will not be allowed. Therefore, these intersecting segments must be transformed into non-intersecting segments. In general, the intersection points of the segments that intersect may not lie on the grid. If the non-grid intersection point is reported, we move it to a nearest grid point. Then a segment may be transformed into a chain of non-intersecting segments. However, we will face some problems with this approach. In [3, 4], the fundamental problems were reported, we brief them below without detailed explanation.

- Drifting Lines: A segment may drift through a series of intersections by an arbitrary distance from its original position,
- Topological Inversions: The topological relation between a grid point and a grid segment, e.g. above-below relation may be inverted by transforming a segment into a chain of segments.

In their paper [4], Ralf Hartmut Güting and Markus Schneider extended the concept of a realm by using the concept *envelope* of a segment, which was first introduced by Green and Yao [3], so that the above problems could be avoided within it. They defined an envelope roughly as the collection of grid points that are immediately above, below, or on a grid segment. When the envelopes are introduced into a realm, the movement of the intersection of a given segment *s* with some other segment to a nearest grid point of its will make *s* to pass through some point *P* in its envelope. This kind of movement is fulfilled by *redrawing s* with some *polygonal line* within the envelope rather than by simply connecting *P* with the start and end points of *s*. If we adopt this technique for realms, then redrawing a segment can never drift to the other side of a realm point. A precise definition and formal specification of an envelope has been given by Dang and Ngo in [5]. In this work, we employ their formal definition for envelopes. Before giving the complete definition for envelopes, we need to first define a concept, *polygonal lines*, which is already mentioned above. A *simple polygonal line* $pl = [p_1, p_2, \ldots, p_m]$, $m \geq 2$, is a sequence of different *N*-points such that the sequences of their *x-coordinates* and *y-coordinates* are monotonic, respectively. Each *N*-point p_i, $i \in \{1, \ldots, m-1\}$ is called a *vertex* of *pl* and each *N*-segment formed from two vertexes p_i and p_{i+1} ($1 \leq i \leq m$), is called an *edge* of *pl*. A machine *SimplePolygonalLine* is introduced for this concept and is omitted here. We refer the readers to [2] for its complete specification in **AMN**.

In that machine, the set *SIMPLE_PL* is a type which consists of the simple polygonal lines. The monotonic property is represented by the constant *is_simple*. The constants *elems_pl* represents the grid points which are vertexes of a simple polygonal line and *segs_pl* represents the grid segments which are edges of a simple polygonal line.

An envelope of a segment is defined by using of a concept *thick cover* for a segment. A thick cover of *N*-segment *s* can be considered as a set of polygonal line connecting the end points of *s* which topologically preserve possible *redrawings* of *s* caused by rounding the intersection with other segments to an *N*-point. The definition of redrawing of a grid segment is given as:

Definition 2 *A simple polygonal line pl is a redrawing of a grid segment s iff it satisfies:*

- *pl joins two end points of s and passes all grid points which lie on s,*
- *pl preserves the topological structure, i.e. for any grid point g, if g is above s then g is not below pl, and if g is below s then g is not above pl.*

We introduce the Boolean function *is_redraw* in **AMN** to check whether a simple polygonal line is a redrawing of a segment or not (not described here).

Definition 3 *A thick cover of a segment s is a set B of grid points such that the following conditions are satisfied:*

1. *(Covering s) The end points of s are in B,*
2. *(Including all possible intersections with other segments) All the nearest grid points of true intersection of s with other segment t are also in B,*
3. *(Including all possible redrawing of segments when intersected with other segments) For any grid point p in B, there is a simple polygonal line pl which is a redrawing of s via p, and all the nearest points of pl are in B.*

As we have mentioned above, the true intersection of a grid segment *s* with other segment *t* may not be a grid point. We can easily show that the coordinates of the true intersection of two grid segments can be always represented by rational numbers. In order to denote the true intersection of two grid segments, we have to introduce the type machine *RationalPoint* in **AMN** to define a set of rational numbers and the operations on them. This is because that **B** does not allow the use of reals. Fortunately, the rational numbers can be defined based on integers which are allowed in **B**.

This machine includes the constant *round* which is the function returning the nearest grid point of a rational point. This machine has been refined to be more detailed in [2].

We then gives a specification of a thick cover in which we introduced Boolean function *is_thick_cover* to check whether a set of grid points is a thick cover of a segment.

An abstract definition for an envelope and a formal specification of the function *env* to encapsulate an envelope of a segment are given as

Definition 4 *An envelope of a segment s is a thick cover of s that does not include any other thick cover of s.*

```
$env \in G\_SEGMENT \rightarrow \mathbb{F}\ (\ G\_POINT\ ) \wedge$
$\forall\ ss\ .\ (\ ss \in G\_SEGMENT \Rightarrow$
    $env\ (\ ss\ ) = \{\ pp\ \ |\ \ pp \in G\_POINT \wedge$
        $\forall\ BB\ .\ (\ BB \in \mathbb{F}\ (\ G\_POINT\ ) \wedge$
        $is\_thick\_cover\ (\ BB\ ,\ ss\ ) = TRUE \Rightarrow$
            $pp \in BB\ )\ \}\ )$
```

So, an envelope of a segment is unique. The *proper envelope* of a segment is defined as the set of points in its envelope which are not its end points. This is represented by function *proper_env*. The functions mentioned above are included in a single machine *Relations2* in [2].

As we have mentioned earlier, when we insert a new segment into a realm, this segment may intersect with another segment of the realm. If the intersection point is a grid point, this point is added to the realm, and segments are broken into smaller segments to maintain the definition of realms. If the intersection point is not a grid point, its nearest grid point must be included in the common part of their envelopes. Since we identify a segment with its envelope, we can take this point as the grid intersection point, and then break "thick" segments into smaller segments. When we identify a segment with its envelope, the constraint "*there is no point of R which is inside a segment of R*" becomes "*there is no point in the realm included in the proper envelope of any of its segments*". This leads to the concept of *extended realm*.

Definition 5 *An extended realm E is a realm in which there is no point in E included in the proper envelope of a segment in E.*

A formal specification of extended realms is given by the machine *ExtendedRealms*.

```
**MACHINE**   ExtendedRealms
**SEES**   Realm , Grid , Relations1 , Relations2 ,
        SimplePolygonalLine , Bool_TYPE
**SETS**   EXREALM
**CONSTANTS**   is_exrealm , is_equal_rm
**PROPERTIES**
    $EXREALM = \{\ ex\ \ |\ \ ex \in SP \wedge$
        $is\_exrealm\ (\ ex\ ) = TRUE\ \} \wedge$
    $is\_exrealm \in SP \rightarrow BOOL \wedge$
    $is\_equal\_rm \in EXREALM \times EXREALM \rightarrow BOOL \wedge$
    $\forall\ sp\ .\ (\ sp \in SP \Rightarrow$
        $is\_exrealm\ (\ sp\ ) = TRUE \Leftrightarrow$
          $(\ is\_realm\ (\ sp\ ) = TRUE \wedge$
        $\forall\ (\ pp\ ,\ ss\ )\ .\ (\ pp \in pot\ (\ sp\ ) \wedge ss \in seg\ (\ sp\ ) \Rightarrow$
        $pp \notin proper\_env\ (\ ss\ )\ )\ )\ )\ ) \wedge$
    $\forall\ (\ ex_1\ ,\ ex_2\ )\ .$
    $(\ ex_1 \in EXREALM \wedge ex_2 \in EXREALM \Rightarrow$
        $is\_equal\_rm\ (\ ex_1\ ,\ ex_2\ ) = TRUE \Leftrightarrow$
    $(\ pot\ (ex_1) = pot\ (ex_2) \wedge seg\ (\ ex_1\ ) = seg\ (\ ex_2\ )\ )\ )$
**END**
```

3 Grid Segment Insertion Operator

When inserting an element *sp* (a grid point or grid segment) into an extended realm *R*, we expect to generate a new extended realm *R'* to include the inserted element and the elements of the old one. However, taking the set union of *R* and *sp* may not result in an extended realm. As we have shown earlier, only redrawing segments is used to maintain the constraints for realms. Because redrawing leads to loosening of the precision, we should use it with care. So, intuitively, our requirement for the insertion of an element *sp* (a grid point or a grid segment) into an extended realm *R* is to achieve a new extended realm *R'* which is minimal (according to set inclusion) among those extended realms that include the points of *R* and *sp* such that each segment of *R*∪*sp* should have a redrawing in *R'*. In this work, we consider only the case in which *sp* is a grid segment because the process of insertion of a grid point is rather simple and is a part of the process of the insertion of a grid segment.

A formal specification of the insertion operation of a grid segment into an extended realm validated to the above intuition is given by the machine *InsertSegment_Ops*.

```
**MACHINE**   InsertSegment_Ops
**SEES**   Realm , ExtendedRealms , GridSegment ,
        SimplePolygonalLine , Relations2 ,
        Bool_TYPE , Bool_TYPE_Ops
**CONSTANTS**   rel_ee
**PROPERTIES**
  $rel\_ee \in EXREALM \times EXREALM \times$
            $G\_SEGMENT \rightarrow BOOL \wedge$
  $\forall\ (\ ex_1\ ,\ ex_2\ ,\ gs\ )\ .$
      $(\ ex_1 \in EXREALM \wedge ex_2 \in EXREALM \wedge$
            $gs \in G\_SEGMENT \Rightarrow$
      $rel\_ee\ (\ ex_1\ ,\ ex_2\ ,\ gs\ ) = TRUE \Leftrightarrow$
        $(\ pot\ (\ ex_1\ ) \subseteq pot\ (\ ex_2\ ) \wedge$
        $\forall$ *vv* . ( *vv* $\in$ *G_SEGMENT* $\wedge$
            *vv* $\in$ *seg* ( $ex_1$ ) $\cup$ { *ss* } $\Rightarrow$
                $\exists$ *pl* . ( *pl* $\in$ *SIMPLE_PL* $\wedge$
                *is_redraw* ( *pl* , *vv* ) $=$ *TRUE* $\wedge$
                *elems_pl* ( *pl* ) $\subseteq$ *pot* ( $ex_2$ ) $\wedge$
                *segs_pl* ( *pl* ) $\subseteq$ *seg* ( $ex_2$ ) ) ) ) )

OPERATIONS
  um $\longleftarrow$ InsertSegment ( ss , rm )  $\widehat{=}$
    PRE   ss $\in$ G_SEGMENT $\wedge$ rm $\in$ EXREALM
    THEN
      ANY   vm   WHERE
        vm $\in$ EXREALM $\wedge$
        rel_ee ( rm , vm , ss ) $=$ TRUE $\wedge$
        $\forall$ xm . ( xm $\in$ EXREALM $\wedge$
            rel_ee ( rm , xm , ss ) $=$ TRUE $\wedge$
              pot ( xm ) $\subseteq$ pot ( vm ) $\vee$
                  seg ( xm ) $\subseteq$ seg ( vm ) $\Rightarrow$
                is_equal_rm ( xm , vm ) $=$ TRUE )
      THEN
        um := vm
      END
    END
END
```

The constant *rel_ee* in the machine *InsertSegment_Ops* is a Boolean function saying that given an extended realm ex_1 and a grid segment *gs*, the extended realm ex_2 is a modification of ex_1 to include points and redrawing segments of segments of ex_1 and *gs*. An algorithm for the insertion of a segment into an extended realm was presented by Ralf Hartmut Güting and Markus Schneider [1]. We have written a refinement of the machine *InsertSegment_Ops* as a machine *InsertSegment_Ops_R* (see [2]). Now, we give an implementation of the machine *InsertSegment_Ops_R*, adopting the algorithm proposed in [1].

By this algorithm, when inserting a segment *s* into a realm, all realm points which lie on the proper envelope of *s* are marked and also all segments which intersect with *s* together with the nearest grid point of the intersection. And then, all these marked segments and *s* are redrawn to pass through their grid intersection point and all marked points. In refining the machine *InsertSegment_Ops* towards the implementation, we import two machines *RedrawingSegments* and *Redraw_Ops*. We also import three library machines: the *renamed set object library machine*, the two *renamed variable library machines*. These machines are used in the operation process as "buffer machines". Because of the space limit, we omit these machines and their implementations here and refer readers to our report [2]. The implementation of InsertSegment Operator is given in Fig 1.

4 Verification of *InsertSegment_Ops*

We have given a specification of *InsertSegment Operator*, validated it against the informal requirement, refined this machine, and also given an implementation for it in **AMN**. Now if we can prove the refinement relation from the specification toward the implementation, then our verification is done. We have done this with **B**-toolkit in a UNU/IIST project and have described our proof in detail in [2]. Because of space limit, we do not present the proof here.

5 Conclusions

We have presented a formal specification of a realm, operations on it, and a verification of an implementations using **B**-Method. This work has shown that the rigorous approach to a geometric problem could be well applied using **B**. We study some techniques for refinements and implementations via some operations specified with **B** on realms. Our refinement work has shown that performing operational refinement before data refinement might be convenient in the development of complicated systems. We have presented a non-trivial example of interactive proofs (based on the specification and refinement of a realm and operations over it) with **B-toolkit**. Our work has shown that quite complicated proofs could be done with the inter-prover using a number of suitable user-defined proof rules.

References

[1] J-R Abrial. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, Cambridge, UK., first edition, 1996.

[2] Kim Yong Chun and Dang Van Hung. Specification and Verification of Spatial Data Types with B-Toolkit. Technical Report 242, UNU/IIST, P.O. Box 3058, Macau, October 2001.

[3] Greene D. and F. Yao. Finite-resolution computational geometry. Proc. 27th IEEE Symp. on Foundations of Computer Science, pages 143–152, 1986.

[4] Ralf Hartmut Guting and Markus ScneiderM. Realm: A foundation for spatial data types in database systems. Advances in Spatial Database. Springer-Verlag, 1993.

[5] Hung Dang Van, Chris George, Tomasz Janowski, and Richard Moore. *Specification Case Studies in RAISE*. Springer-Verlag, 2002.

```
IMPLEMENTATION   InsertSegment_Ops_I
REFINES   InsertSegment_Ops_R
SEES   Grid , RationalPoint , SimplePolygonalLine , MarkedSegment , Relations2 ,
       Realm , ExtendedRealms , RedrawSegment_set_ctx , Bool_TYPE
IMPORTS
    RedrawingSegments , Redraw_Ops ,
    RedrawSegment_set_obj ( MARKED_SEG , 2 , 100 ) ,
    RedrawingSegment_Vvar ( RedrawSegment_SETOBJ ) ,
    UnredrewSegment_Vvar ( RedrawSegment_SETOBJ )
INVARIANT
  RedrawingSegment_Vvar ∈ RedrawSegment_settok ∧ UnredrewSegment_Vvar ∈ RedrawSegment_settok ∧
  redrawing_segs = RedrawSegment_setstruct ( RedrawingSegment_Vvar )
INITIALISATION
  VAR
    aa , bb , app , bpp
  IN
    aa , app ⟵ RedrawSegment_CRE_SET_OBJ ; RedrawingSegment_STO_VAR ( app ) ;
    bb , bpp ⟵ RedrawSegment_CRE_SET_OBJ ; UnredrewSegment_STO_VAR ( bpp )
  END
OPERATIONS
  um ⟵ InsertSegment ( ss , rm )  ≙

    VAR   cc , pp , qq , ll , nn , ii , tt   IN
      GetAllRedrawingSegments ( ss , rm ) ; cc := rm ;
      pp ⟵ RedrawingSegment_VAL_VAR ; qq ⟵ UnredrewSegment_VAL_VAR ;
      ll ⟵ RedrawSegment_CPY_SET_OBJ ( pp , qq ) ; qq ⟵ UnredrewSegment_VAL_VAR ;
      nn ⟵ RedrawSegment_CRD_SET_OBJ ( qq ) ; ii := nn ;
      WHILE   ii ≠ 0   DO
        tt ⟵ RedrawSegment_VAL_SET_OBJ ( qq , ii ) ; cc ⟵ RedrawMarkedSegment ( tt , cc ) ; ii := ii − 1
      INVARIANT
        nn ∈ ℕ ∧ cc ∈ EXREALM ∧ pp = RedrawingSegment_Vvar ∧
        qq = UnredrewSegment_Vvar ∧ ll ∈ BOOL ∧
        ii ∈ ℕ ∧ ss ∈ G_SEGMENT ∧ rm ∈ EXREALM ∧ ( ii = 0 ⇒ rel_ee ( rm , cc , ss ) = TRUE ∧
        ∀ dd . ( dd ∈ EXREALM ∧ rel_ee ( rm , dd , ss ) = TRUE ⇒
                      pot ( cc ) ⊆ pot ( dd ) ∧ seg ( cc ) ⊆ seg ( dd ) ) ) ∧
        RedrawingSegment_Vvar ∈ RedrawSegment_settok ∧
        UnredrewSegment_Vvar ∈ RedrawSegment_settok ∧
        redrawing_segs = RedrawSegment_setstruct ( RedrawingSegment_Vvar )
      VARIANT
        nn
      END   ;
      um := cc
    END
END
```

Figure 1. Implementation of InsertSegment Operator