Lifting Imprecise Values

Gerhard Navratil, Farid Karimipour, Andrew U. Frank

Institute for Geoinformation and Cartography, Vienna University of Technology, Gusshausstr. 27-29, A-1040 Vienna, Austria,
[navratil,karimipour,frank]@geoinfo.tuwien.ac.at

Abstract

The article presents a conceptual framework for computations with imprecise values. Typically, the treatment of imprecise values differs from the treatment of precise values. While precise computations use a single number to characterize a value, computations with imprecise values must deal with several numbers for each value. This results in significant changes in the program code because values are represented, e.g., by expectation and standard deviation and both values must be considered within the computations. It would be desirable to have a solution where only limited changes in very specific places of the code are necessary. The mathematical concept of lifting may lead to such a solution.

1 Introduction

The integration of quality descriptions is one of the most important practical problems for the GIS research and development community. All data in a GIS have limited precision leading to a corresponding level of uncertainty about the true values. These uncertainties spread if the data are used for other computations. The determination of the result's uncertainty is crucial for the user. Results are only meaningful if the possible deviations of the result do not change the result significantly. Assessment of the un-

certainty of the result requires knowledge on the uncertainty of the original data. This knowledge must then be carried along with each processing step.

We propose a conceptual framework that deals with the problem of error propagation in a mathematically clean and simple way. Frank demonstrated the use of functors to lift map algebra from a pure spatial context to a spatio-temporal context (Erwig and Schneider, 1999; Frank, 2005). The same concept has been used to lift the basic algebraic operations for numbers to normally distributed values (Navratil, 2006). Since normally distributed values are only one kind of description for imprecision we extend the concept to other kinds of descriptions.

The article is structured as follows: In section we discuss different approaches to describe imprecise values. Section 3 shows how errors propagate and how the result can be computed. Sections 4 and 5 introduce the concept of lifting and show an implementation of lifting for error propagation. The paper concludes with a discussion of aspects that need to be addressed in the future.

2 Imprecise Values

The results of measurements are not precise numbers (Viertl, 2002). The limitation of precision propagates when using these values in mathematical models. Several methods have been developed to cope with that problem. We arbitrarily selected the three different models normally distributed values, intervals, and fuzzy values for the discussion in the remainder of the paper. Other models, e.g., for skewed models were left for future research.

2.1 Normally Distributed Values

Observations of geometric qualities like distances or angles are usually assumed to be normally distributed. Maybe even other physical qualities like density can be assumed to be normally distributed. The reason for this assumption is the central limit theorem. It states that, if the sum of many independent and identically distributed variables has a finite variance, then the sum will be approximately normally distributed. Two variables are independent if the probability for the occurrence of one event is independent from the result of the other event. An example for independent variables is the numbers resulting from rolling dices. Weather conditions in a specified location on successive days are dependent events.

Normal distribution is defined by two parameters, the expected value μ and the statistical dispersion σ. Measures for the statistical dispersion are variance and standard distribution as the positive square root of the variance. In the following the variance will be used.

Normally distributed values follow the density function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

The area below the density function is a probability measure. The area for an arbitrary interval is the probability that a randomly picked value belonging to this distribution lies within the interval. The density function is defined for the interval]-∞,+∞[and is symmetric (compare Fig. 1).

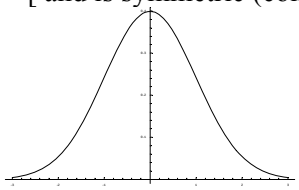

Fig. 1. Density function for a normalized ($\mu = 0$, $\sigma = 1$) normal distribution. The horizontal axis is the outcome *x* and the vertical axis specifies the density *f(x)*.

Measures for dependency are covariance and correlation. Covariance describes the common variation of two observations. The units of measurement depend on the units of measurement of the observations. The correlation on the other hand is dimensionless and describes the linear dependence between the two observations. The Pearson product-moment correlation coefficient r_{xy} is defined as:

$$r_{xy} = \frac{\sigma_{xy}}{\sqrt{\sigma_x^2}\sqrt{\sigma_y^2}} \tag{2}$$

Independent parameters have a correlation $r_{xy}=0$ and a covariance $\sigma_{xy}=0$.

2.2 Fuzzy Values

Fuzzy values or fuzzy numbers are fuzzy sets whose members are real numbers in which uncertainty is represented through a non-probabilistic way (Siler and Buckley, 2005). For example, suppose you are driving and trying to keep the speed at exactly 90 km/h. In practice the speed will vary and it will be a fuzzy value around 90.

Mathematically, a fuzzy value is a function (called membership function) $\mu(x): R \to [0,1]$, which relates each number to its grade of membership. Generally, this function may have any shape. The complexity of the operations on fuzzy values depend on the shape: The more irregular the membership function the more complicated the calculations (Fodor and Bede 2006).

The so-called *L-R fuzzy values* are one of the most important and practical types of fuzzy numbers. For an L-R fuzzy value a, membership function is defined as follows (Fodor and Bede, 2006):

$$\mu(x) = \begin{cases} L(\frac{\hat{a} - x}{\underline{a}}) & x \le \hat{a} \\ R(\frac{x - \hat{a}}{\overline{a}}) & x > \hat{a} \end{cases} \tag{3}$$

where $L, R : [0,+\infty[\to [0,1]$ are two continuous, decreasing functions fulfilling $L(0) = R(0) = 1$ and $L(1) = R(1) = 0$, $\hat{a}$ is a real number with $\mu(\hat{a}) = 1$ and $\overline{a}, \underline{a}$ are two positive real numbers for which $\mu(\overline{a}) = \mu(\underline{a}) = 0$.

If functions L and R are linear, a *trapezoidal fuzzy value* will be obtained (Figure 2) represented by a quadruple $(a,b,c,d), a \le b \le c \le d$.

$$\mu(x) = \begin{cases} 0 & x < a \\ \frac{x-a}{b-a} & a \le x < b \\ 1 & b \le x \le c \\ \frac{d-x}{d-c} & c < x \le d \\ 0 & x > d \end{cases} \tag{4}$$

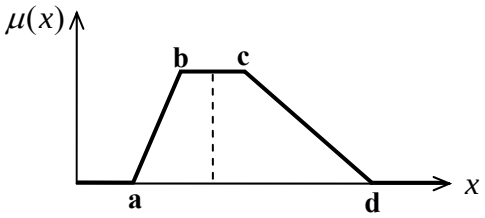

Fig. 2. A trapezoidal fuzzy number represented by (a, b, c, d)

As a particular case, a triangular fuzzy value (Figure 3) is a trapezoidal fuzzy value in which the values b and c are equal. A triangular fuzzy value can be represented either by a triple (a,b,c) or a quadruple

$(a,b,b,c), a \le b \le c$. However, quadruple representation allows us to have a unified algebra.

$$\mu(x) = \begin{cases} 0 & x < a \\ \dfrac{x-a}{b-a} & a \le x < b \\ \dfrac{c-x}{c-b} & b \le x < c \\ 0 & x > c \end{cases} \quad (5)$$

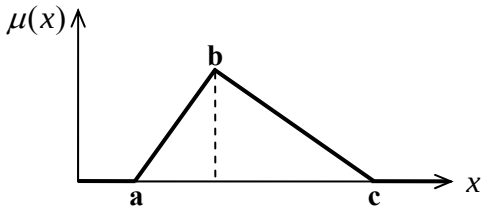

Fig. 3. A triangular fuzzy number represented by (a, b, c) or (a, b, b, c)

2.3 Interval Values

If we can state with equal confidence that a value lies somewhere between 'a' and 'b', it can be referred to as an interval [a, b] in which the occurrence probability for all elements are the same (Hayes, 2003). For example, if you are measuring a distance, you can never be certain about the result, but you may be able to say that it is certainly between 76 and 78 meters.

As Figure 4 shows, an interval value can be considered as a very simple fuzzy value with a binary membership function $\mu(x): R \to \{0,1\}$:

$$\mu(x) = \begin{cases} 1 & a \le x \le b \\ 0 & elsewhere \end{cases} \quad (6)$$

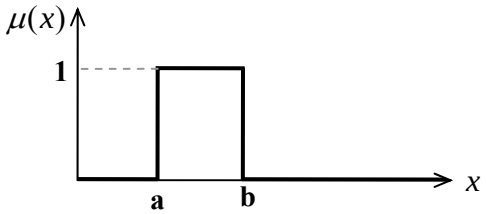

Fig. 4. An Interval value represented by [a, b]

Sometimes an interval is shown by the notation $[\underline{x}, \bar{x}]$ to emphasize the lower and upper limitations (Hayes, 2003).

3 Propagation of Imprecision

Observations are used for analysis and the results are often used in decision-making processes. Imprecision in the observations will propagate through the analysis and result in imprecise results. Knowledge on the imprecision is necessary for consideration in the decision-making. Much work has been done in the field of imprecision and error propagation (Heuvelink, 1998; Bachmann and Allgöwer, 2002; Heuvelink and Burrough, 2002; Karssenberg and de Jong, 2005). In the following we show the basic operations used for the three types of imprecise values.

3.1 Operations on Normally Distributed Values

Let us assume we have normally distributed values $x_1, \ldots, x_n$ and apply a function to these values. The function result is determined by applying the function to the expected values, but what will be the variation? There are several methods to assess the result of error propagation (Heuvelink, 1998, pp. 36-42). The mathematically correct solution is the strict, algebraic computation of the distribution, which results from applying the function. However, the practicality of the solution is hampered by its complexity.

A different approach is the first order Taylor series method, which assumes that the functions can be differentiated. The first order Taylor series method replaces the function by its tangent and produces useful results if the values contain only small deviations. The computation of the first order Taylor series method for a function *f* is defined as

$$\sigma_f^2 = \mathbf{F}^T \boldsymbol{\Sigma}_{xx} \mathbf{F} \tag{7}$$

where **F** is the Jacobi matrix for the function f and $\mathbf{\Sigma_{xx}}$ is the variance-covariance matrix for the parameters.

The assumption of independent parameters simplifies the computation. The variance-covariance matrix becomes a matrix in diagonal form and (7) can be simplified to

$$\sigma_f^2 = \sum_i \sigma_i^2 \left(\frac{\partial f}{\partial x_i} \right)^2 \tag{8}$$

Monte Carlo simulation is a numerical solution and leads to the same result as the mathematically correct solution if the number of computations is large enough. However, computational costs are high since computations must be repeated frequently to assess the distribution of the result.

3.2 Operations on Fuzzy Values

To operate on fuzzy values with general membership functions, we need the concept of α-cut interval, which is defined for a fuzzy value u and $0 < \alpha \leq 1$ as shown in Figure 5. The definition is (Fodor and Bede, 2006):

$$[u]^\alpha = [\underline{u}^\alpha, \overline{u}^\alpha] = \{x \in R \mid \mu(x) \geq \alpha\} \tag{9}$$

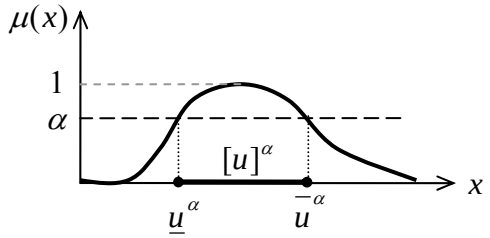

Fig. 5. The concept of an α-cut interval

Then if u and v are two fuzzy values and f and $\circ$ are two operators with one and two operands, respectively, the following rules are used to find the result for an α-cut (Fodor and Bede, 2006):

$$\begin{aligned} \underline{f(u)}^\alpha &= \min\left\{ f(\underline{u}^\alpha), f(\overline{u}^\alpha) \right\}, \\ \overline{f(u)}^\alpha &= \max\left\{ f(\underline{u}^\alpha), f(\overline{u}^\alpha) \right\}, \end{aligned} \tag{10}$$

$$\begin{aligned}\underline{(u \circ v)}^{\alpha} &= \min\left\{\underline{u}^{\alpha} \circ \underline{v}^{\alpha}, \underline{u}^{\alpha} \circ \bar{v}^{\alpha}, \bar{u}^{\alpha} \circ \underline{v}^{\alpha}, \bar{u}^{\alpha} \circ \bar{v}^{\alpha}\right\}, \\ \overline{(u \circ v)}^{\alpha} &= \max\left\{\underline{u}^{\alpha} \circ \underline{v}^{\alpha}, \underline{u}^{\alpha} \circ \bar{v}^{\alpha}, \bar{u}^{\alpha} \circ \underline{v}^{\alpha}, \bar{u}^{\alpha} \circ \bar{v}^{\alpha}\right\}.\end{aligned} \tag{11}$$

The lists in (11) result from a Cartesian product. The result of a Cartesian product $u \times v$ using an operation $\circ$ is a list of all possible combinations $(u_i \circ v_i)$ of elements u_i from u with elements v_i from v. The elements of u in this case are

$$u = \left\{\underline{u}^{\alpha}, \bar{u}^{\alpha}\right\} \tag{12}$$

and the elements of v are

$$v = \min\left\{\underline{v}^{\alpha}, \bar{v}^{\alpha}\right\}. \tag{13}$$

The results of applying an operation on fuzzy values are as follows (Fodor and Bede, 2006):

$$f(u) = sort\left\{\underline{f(u)}^{0}, \overline{f(u)}^{0}, \underline{f(u)}^{1}, \overline{f(u)}^{1}\right\}, \tag{14}$$

$$(u \circ v) = sort\left\{\underline{(u \circ v)}^{0}, \overline{(u \circ v)}^{0}, \underline{(u \circ v)}^{1}, \overline{(u \circ v)}^{1}\right\}. \tag{15}$$

In the case of trapezoidal fuzzy values, $[u]^0 = [a,d]$ and $[u]^1 = [b,c]$. So the above formulas can be simplified (Fodor and Bede, 2006):

$$f(u) = sort\{f(a), f(b), f(c), f(d)\}, \tag{16}$$

$$\begin{aligned}\underline{(u \circ v)}^{0} &= \min\{a_u \circ a_v, a_u \circ d_v, d_u \circ a_v, d_u \circ d_v\}, \\ \overline{(u \circ v)}^{0} &= \max\{a_u \circ a_v, a_u \circ d_v, d_u \circ a_v, d_u \circ d_v\}, \\ \underline{(u \circ v)}^{1} &= \min\{b_u \circ b_v, b_u \circ c_v, c_u \circ b_v, c_u \circ c_v\}, \\ \overline{(u \circ v)}^{1} &= \max\{b_u \circ b_v, b_u \circ c_v, c_u \circ b_v, c_u \circ c_v\},\end{aligned} \tag{17}$$

$$(u \circ v) = sort\left\{\underline{(u \circ v)}^{0}, \overline{(u \circ v)}^{0}, \underline{(u \circ v)}^{1}, \overline{(u \circ v)}^{1}\right\}. \tag{18}$$

The result of some operations (e.g., + and -) on trapezoidal fuzzy values are also trapezoidal. However, for other operations (e.g., * and ÷) it is non-trapezoidal. In such cases, the non-trapezoidal shape of the result can be usually opted with a trapezoid although repeated operations may increase the uncertainty (Fodor and Bede, 2006).

3.3 Operations on Interval Values

Operations on interval values are defined as follows (Hayes, 2003):

$$f(u) = f[\underline{u},\overline{u}] = [\min\{f(\underline{u}), f(\overline{u})\}, \max\{f(\underline{u}), f(\overline{u})\}], \tag{19}$$

$$(u \circ v) = [\underline{u},\overline{u}] \circ [\underline{v},\overline{v}] = \\ [\min\{\underline{u} \circ \underline{v}, \underline{u} \circ \overline{v}, \overline{u} \circ \underline{v}, \overline{u} \circ \overline{v}\}, \max\{\underline{u} \circ \underline{v}, \underline{u} \circ \overline{v}, \overline{u} \circ \underline{v}, \overline{u} \circ \overline{v}\}] . \tag{20}$$

4 Lifting Error Propagation for Imprecise Values

4.1 Mathematical Concept Lifting

Lifting is a mathematical concept emerging from category theory (MacLane and Birkhoff, 1999). It uses functors to map objects and functions between these objects from one class to another class (Moggi, 1989).

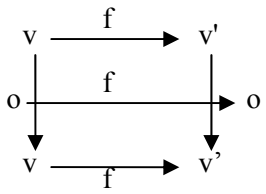

Fig. 6. Mapping of precise values to imprecise values and functions on precise values to functions on imprecise values.

Figure 6 shows the basic idea. A function *o* has an object *v* as the argument and returns another object. Assume there is another object *v'* with a mapping function *f* from *v* to *v'*. Lifting then is the concept of using the mapping function *f* not only for the objects *v* and *v'* but also for the function *o*. Thus we can write (compare Marquis, 2006)

$$f(o(v)) = o'(f(v)) = o'(v') \tag{21}$$

or in case of two objects *a* and *b*

$$f(o(a,b)) = o'(f(a), f(b)) = o'(a', b'). \tag{22}$$

The mapping function *f* is called a functor. The application of a functor on a given object or function is called lifting.

The functors we deal with are pointed functors. A pointed functor has a specified natural equivalence. In our case the pointed functor maps from precise to imprecise values.

4.2 Programming Paradigm Lifting

Lifting is a concept that can simplify programming. Assume we specify a function, which takes two points and computes the distance between these points. The points are defined as pairs of floating point numbers. The function will work correctly if it is applied to the correct data type but how can we apply it to different data types, e.g., data types for moving points? In this case the coordinates of the points and the distance between the points depend on an additional parameter 'time'. In an imperative programming language that does not support polymorphism and generic programming (such as C) we may use one of the following solutions:

- we copy the function and edit the copy such that it works with the new data type or
- we take care of the differences whenever we call the function, by first clarifying the temporal aspect and then computing the distance.

In a programming environment the first method will inevitably result in a long list of slightly different versions of the same function. Each version is applicable to only one data type. The second method may result in strange errors if one of the programmers does not know about the limitations or makes a mistake in the calling routine.

Lifting provides an elegant solution to that problem. The language must be capable of two concepts:

- The language must be able to overload a function. Overloading a function allows using the same function name for different data types. Typical examples known from standard imperative languages are the basic mathematical operations, which can be applied to different numerical data types. A language used for lifting must provide this capability for all types of functions including user-defined functions.
- The language must be a second-order language, i.e., the function must support the use of functions as parameters for other functions.

Lifting only requires the definition of a functor. This functor can then be used to automatically adapt functions to the data type.

4.3 Application of Lifting for Imprecise Values

How can we apply the concept of lifting to precise and imprecise values? Starting point is a data type `a` for precise values. Parts of the definition for basic mathematic operations in Haskell (e.g., Bird, 1998) are shown here:

```
class Num a where
   (+)    :: a -> a -> a
   (*)    :: a -> a -> a
   abs    :: a -> a
   negate :: a -> a

class (Num a) => Fractional a where
   (/) x y :: a -> a -> a
   recip   :: a -> a

class (Fractional a) => Floating a where
   sqrt :: a -> a

class (Floating a) => Numbers a where
   sqr :: a -> a
```

Applying these functions to a new data type requires building the corresponding instances. These instances can be constructed in a traditional way by rewriting the code. The result for a new data type could look like the following:

```
data MyNumbers a = MyNum a

instance (Floating (MyNumbers Float)) =>
         Numbers (MyNumbers Float) where
   sqr (MyNum x) = MyNum (x * x)
```

The disadvantage of this method has been shown in section 4.2. A functor eliminates the necessity to rewrite the code. The definition for a class of functors mapping from a data type `a` to a data type `b a` is

```
class Lifts b a where
   lift0 :: a -> b a
   lift1 :: (a -> a) -> b a -> b a
   lift2 :: (a -> a -> a) -> b a -> b a -> b a
```

The function `lift0` maps a value from one class to another. The functions `lift1` and `lift2` provide the same for functions with one or two parameters. Functions with more than two parameters must be mapped recursively.

We can now define the instance of above class `Numbers` for the data type `MyNumber a` as follows

```
instance Lifts MyNumber a where
   lift0 x        = MyNum x
   lift1 op x     = MyNum (op x)
   lift2 op x1 x2 = MyNum (op x1 x2)

instance (Floating MyNumber) => Numbers MyNumber
where
   sqr x = lift1 sqr x
```

The lifting function maps the functionality of the existing data type to the new data type. This avoids rewriting the code. The benefit becomes evident if an error must be fixed in the implementation. Instead of fixing all instances, only the original instance must be fixed and this change automatically affects all other instances.

5 Sample Implementation

In this section we describe how to implement lifting for the imprecise concepts defined in section 2. The mapping of functions uses the error propagation concepts shown in section 3. We now want to use these definitions to define lifting functions as introduced in section 4.

The first step in each section is the definition of a data type. The operations necessary to propagate imprecision are concept dependent. The concepts also use different parameters to describe the imprecision. These parameters are collected in corresponding data types.

5.1 Lifting Normally Distributed Values

Normally distributed values are defined by the expected value and the variance, which are stored in this order in the data set. `ND` is a constructor function, which creates the data set using the provided values.

```
data NormDist v = ND v v
```

Error propagation for normally distributed values requires the computation of partial derivatives. Numerical differentiation using linearization is a simple solution that provides an approximation. For a function f with parameters x_1, x_2, … the derivative in x_1 is

$$\frac{\partial f}{\partial x_1} = \frac{f(x_1 + \varepsilon, x_2, ...) - f(x_1, x_2, ...)}{\varepsilon} \tag{23}$$

where ε is a small value specifying the length of the interval used for the linearization. More sophisticated strategies can be found in the literature (Press, Flannery et al., 1988). The local functions `diff` respectively `diff1` and `diff2` provide lifting function (23).

```
instance Lifts NormDist Double where
   lift0 v = ND v 0
   lift1 op (ND v s) = ND (op v) (diff * s) where
      diff = ((op (v+epsilon)) - (op v)) / epsilon
   lift2 op (ND v1 s1) (ND v2 s2) =
     ND (op v1 v2) (diff1^2*s1 + diff2^2*s2)) where
      diff1 = ((op (v1+epsilon) v2) - (op v1 v2)) /
                                                 epsilon
      diff2 = ((op v1 (v2+epsilon)) - (op v1 v2)) /
                                                 epsilon
```

Lifting a precise value requires an assumption for a variance. Since the value shall be precise, the variance is set to zero. Lifting a mathematical operation `op` with one or two parameters uses numerical differentiation and formula (8) to compute the variance of the function result.

This instance then provides the functionality to easily lift the basic mathematical operations. This is done as shown in section 4.3.

5.2 Lifting Fuzzy Values

The data type for fuzzy values must store the four points necessary to define the trapezoidal representation shown in Figure 2. Other realizations like triangular distribution functions may use different data types. `F` is again the constructor function.

```
data Fuzzy    v = F  v v v v
```

The implementation for fuzzy values is equal to the examples above:

```
instance Lifts Fuzzy Double where
   lift0 v = F v v v v
   lift1 op (F a b c d) =
     listToFuzzyNum (sort cartProduct) where
       cartProduct = [(op a), (op b), (op c), (op d)]
   lift2 op (F a1 b1 c1 d1) (F a2 b2 c2 d2) =
     listToFuzzyNum (sort cartProduct) where
        cartProduct = [minimum cartProduct1,
                       maximum cartProduct1,
                       minimum cartProduct2,
                       maximum cartProduct2]
```

```
cartProduct1 = [(op a1 a2), (op a1 d2),
                (op d1 a2), (op d1 d2)]
cartProduct2 = [(op b1 b2), (op b1 c2),
                (op c1 b2), (op c1 c2)]
```

The functions `cartProduct`, `cartProduct1`, and `cartProduct2` define the Cartesian product as introduced in section 3.2.

5.3 Lifting Interval Values

The data type for the interval stores begin and end of the interval. `I` is again a constructor function.

```
data Interval v = I  v v
```

As seen in section 3, interval arithmetic is rather simple. Again we use the Cartesian product to determine the boundaries of the resulting interval.

```
instance Lifts Interval Double where
   lift0 v = I v v
   lift1 op (I a b) = I (minimum cartProduct)
                        (maximum cartProduct) where
      cartProduct = [(op a), (op b)]
   lift2 op (I a1 b1) (I a2 b2) =
    I (minimum cartProduct) (maximum cartProduct)
      where cartProduct = [(op a1 a2), (op a1 b2),
                           (op b1 a2), (op b1 b2)]
```

6 Examples for Using the Lifted Operations

How can we now use the lifted function? The simplest example is using a basic mathematic operation, e.g., by adding two numbers. This requires typing `a+b` on the command line. Depending on the definition of the parameters `a` and `b`, we get different results. Table 1 shows the results.

A more complex example is the computation of the distance between two points. The points are defined by a pair of coordinates in a plane coordinate system. The distance is then defined as

$$s_{12} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \,. \tag{24}$$

Table 1. Results of addition for different types of values.

Type	Values	Result of a+b
precise values	a = 5.0 b = 2.0	7.0
normally distributed	a = ND 5.0 0.05 b = ND 2.0 0.04	ND 7.0 0.0899
interval	a = I 4.85 5.15 b = I 1.88 2.12	I 6.73 7.27
fuzzy	a = F 4.85 4.95 5.05 5.15 b = F 1.88 1.96 2.04 2.12	F 6.73 6.91 7.09 7.27

In Haskell the definition is

```
data Point f  = Pt f f deriving Show

dist :: Numbers a => (Point a) -> (Point a) -> a
dist (Pt x1 y1) (Pt x2 y2) =
                    sqrt (sqr(x1-x2) + sqr(y1-y2))
```

We can now again use different implementations of imprecise values to test the implementation. Table 2 shows the results.

Table 2. Results of distance computation for different types of values

Type	Values	Result of `dist`
precise values	ptD1, ptD2 :: Point Double ptD1 = Pt 20 20 ptD2 = Pt 12 17	8.544
normally distributed	ptN1, ptN2 :: Point (NormDist Double) ptN1 = Pt (ND 20 0.02) (ND 20 0.04) ptN2 = Pt (ND 12 0.06) (ND 17 0.01)	ND 8.544 0.076
interval	ptI1, ptI2 :: Point (Interval Double) ptI1 = Pt (I 19.94 20.06) (I 19.88 20.12) ptI2 = Pt (I 11.82 12.18) (I 16.97 17.03)	I 8.267 8.822
fuzzy	ptF1, ptF2 :: Point (Fuzzy Double) ptF1 = Pt (F 19.94 19.98 20.02 20.06) (F 19.88 19.96 20.04 20.12) ptF2 = Pt (F 11.82 12.94 12.06 12.18) (F 16.97 16.99 17.01 17.03)	F 7.595 8.302 8.524 8.822

7 Conclusions and Future Work

In this paper we use the domains of precise and imprecise values as an example. We showed different models of imprecise values in section 2 and how to propagation imprecision in each model in section 3. In section 4 we

discussed the concept of mapping between different domains. We have shown that it is possible to construct functors, which map precise values and functions on these values to imprecise values. This allows the specification of functions, which work with different kinds of values. As shown in section 5.5 the defined functors allow a simple mapping of functions from a simple domain to an extended domain. It can be generalized from the results how we can use functors to map data and functions from one domain to another domain.

References

Bachmann, A. and Allgöwer, B. (2002). Uncertainty Propagation in Wildland Fire and Behaviour Modelling. International Journal of Geographic Information Science 16(2): 115-127.

Bird, R. (1998). Introduction to Functional Programming Using Haskell. Hemel Hempstead, UK, Prentice Hall Europe.

Erwig, M. and Schneider, M. (1999). Developments in Spatio-Temporal Query Languages. Proceedings of 10th International Workshop on Database and Expert Systems Applications, Florence, Italy.

Fodor, J. and Bede, B. (2006). Arithmetics with Fuzzy Numbers: A Comparative Overview. Proceeding of 4th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence, Herl'any, Slovakia.

Frank, A.U. (2005). Map Algebra Extended with Functors for Temporal Data. Proceedings of the ER Workshop 2005 (CoMoGIS'05). J. Akoka et al. Klagenfurt, Austria, Springer-Verlag Berlin Heidelberg. LNCS 3770: 193-206.

Hayes, B. (2003). A Lucid Interval. American Scientist 91(6): 484-488.

Heuvelink, G.B.M. (1998). Error Propagation in Environmental Modelling with GIS. London, Taylor & Francis.

Heuvelink, G.B.M. and Burrough, P.A. (2002). Developments in Statistical Approaches to Uncertainty and its Propagation. International Journal of Geographic Information Science 16(2): 111-113.

Karssenberg, D. and de Jong, K. (2005). Dynamic Environmental Modelling in GIS: 2. Modelling Error Propagation. International Journal of Geographic Information Science 19(6): 623-637.

MacLane, S. and Birkhoff, G. (1999). Algebra (3rd Edition), AMS Chelsea Publishing.

Marquis, J.-P. (2006). Category Theory, Stanford Encyclopedia of Philosophy.

Moggi, E. (1989). A Category-Theoretic Account of Program Modules. Category Theory and Computer Science, Springer-Verlag, Berlin.

Navratil, G. (2006). Error Propagation for Free? GIScience, Münster, Germany, Institute for Geoinfomatics, University Münster.

Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. (1988). Numerical Recipes in C - The Art of Scientific Computing. Cambrigde, Cambridge University Press.

Siler, W. and Buckley, J.J. (2005). Fuzzy Expert Systems and Fuzzy Reasoning, Wiley Press.

Viertl, R. (2002). On the Description and Analysis of Measurements of Continuous Quantities. Kybernetika 38(3): 353-362.