

MODELING AND QUERYING HISTORY OF MOVEMENT:

ABSTRACT DATA MODEL

Part 3

The Elusive *when* Operation

Imagine there would be an operation:

$\underline{moving}(\alpha) \times (\alpha \rightarrow \underline{bool}) \rightarrow \underline{moving}(\alpha) \quad \mathbf{when}$

“Restrict a moving region to the times when its area was greater than 1000!”

```
mr when[fun (r: region) area(r) > 1000]
```

The Elusive *when* Operation

Imagine there would be an operation:

$\underline{moving}(\alpha) \times (\alpha \rightarrow \underline{bool}) \rightarrow \underline{moving}(\alpha) \quad \mathbf{when}$

“Restrict a moving region to the times when its area was greater than 1000!”

```
mr when[fun (r: region) area(r) > 1000]
```

Does this make any sense?

Continuous domain. Looping over infinitely many instances for checking the predicate is inherently impossible.

Impossible to implement?

The Elusive *when* Implementation

$\underline{moving}(\alpha) \times (\alpha \rightarrow \underline{bool}) \rightarrow \underline{moving}(\alpha) \quad \mathbf{when}$

“Restrict a moving region to the times when its area was greater than 1000!”

```
mr when[fun (r: region) area(r) > 1000]
```

Can be expressed as

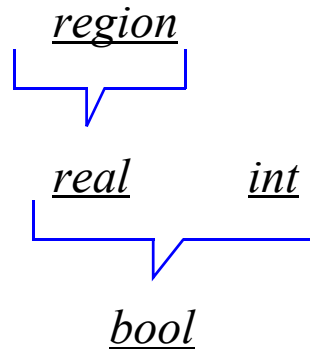
```
mr atperiods(deftime(at( area(mr) > 1000, true)
```

The Elusive *when* Implementation

$\underline{moving}(\alpha) \times (\alpha \rightarrow \underline{bool}) \rightarrow \underline{moving}(\alpha) \quad \mathbf{when}$

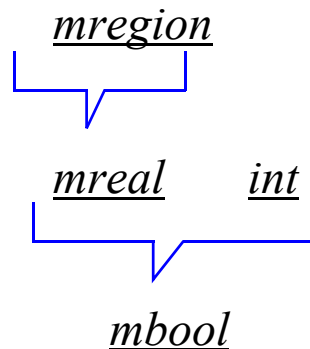
“Restrict a moving region to the times when its area was greater than 1000!”

```
mr when[fun (r: region) area(r) > 1000]
```



Can be expressed as

```
mr atperiods(deftime(at( area(mr) > 1000, true)
```



Lifting Operations to Time-Dependent Operations

$$\alpha_1 \times \alpha_2 \times \dots \times \alpha_n \rightarrow \beta \quad \mathbf{op}$$

Lifting yields operations

$$\underline{moving}(\alpha_1) \times \alpha_2 \times \dots \times \alpha_n \rightarrow \underline{moving}(\beta) \quad \mathbf{op}$$

$$\alpha_1 \times \underline{moving}(\alpha_2) \times \dots \times \alpha_n \rightarrow \underline{moving}(\beta) \quad \mathbf{op}$$

$$\underline{moving}(\alpha_1) \times \underline{moving}(\alpha_2) \times \dots \times \alpha_n \rightarrow \underline{moving}(\beta) \quad \mathbf{op}$$

...

$$\underline{moving}(\alpha_1) \times \dots \times \underline{moving}(\alpha_n) \rightarrow \underline{moving}(\beta) \quad \mathbf{op}$$

For example:

$\underline{real} \times \underline{real}$	$\rightarrow \underline{bool}$	=	$\underline{region} \times \underline{point}$	$\rightarrow \underline{point}$	intersection
$\underline{mreal} \times \underline{real}$	$\rightarrow \underline{mbool}$	=	$\underline{mregion} \times \underline{point}$	$\rightarrow \underline{mpoint}$	intersection
$\underline{real} \times \underline{mreal}$	$\rightarrow \underline{mbool}$	=	$\underline{region} \times \underline{mpoint}$	$\rightarrow \underline{mpoint}$	intersection
$\underline{mreal} \times \underline{mreal}$	$\rightarrow \underline{mbool}$	=	$\underline{mregion} \times \underline{mpoint}$	$\rightarrow \underline{mpoint}$	intersection

Lifting Operations to Time-Dependent Operations (cont.)

We defined on non-temporal types

- predicates
- set operations
- point set to point operations (aggregates)
- numeric properties of sets
- direction and distance relationships

These are all available now on temporal types!

[Very powerful language]

Lifting Operations to Time-Dependent Operations: Examples

Example: “What is the time-dependent size of hurricane Lizzy?”

```
LET LizzyArea = area(Lizzy);
```

Example: “At what time did hurricane Lizzy split into two separate parts?”

```
inst(initial(at(no_components(Lizzy), range(2,2))))
```

Lifting Operations to Time-Dependent Operations: Examples (cont.)

Example: Define derived operations **always**, **sometimes**, and **never** (using the `LET ... = FUN ...` mechanism), all with signature $\text{moving}(\text{bool}) \rightarrow \text{bool}$. Here **always** means that the $\text{moving}(\text{bool})$ has value *true* whenever it is defined (similar for the other operations). Note that $\text{false} < \text{true}$.

```
LET always = FUN (mb: mbool) not(passes(mb, false));  
LET sometimes = FUN (mb: mbool) passes(mb, true);  
LET never = FUN (mb: mbool) not(passes(mb, true));
```

“Was speed of flight KLM066 always less than 1000 km/h?”

```
LET KLM066 = .....;  
always(speed(KLM066) < 1000);
```

Lifting Operations to Time-Dependent Operations: Examples (cont.)

Example: Define a **closest** operator with signature $\text{moving}(\text{point}) \times \text{point} \rightarrow \text{intime}(\text{point})$ returning the time and position when a moving point is (for the first time) closest to a given static point.

```
LET closest =
```

```
FUN (mp:mpoint, p:point) atinstant(mp, inst(initial(atmin(distance(mp, p)))));
```

Table that holds positions of sites:

```
site (name: string, pos: point)
```

“At what time was flight LH078 closest to the Eiffel tower?”

```
LET LH078 = ...;
```

```
LET TheEiffelTower = ELEMENT (SELECT pos FROM site WHERE name='TheEiffelTower');  
inst(closest(LH078, TheEiffelTower));
```

Lifting Operations to Time-Dependent Operations: Examples (cont.)

Example: Table listing the locations of airports:

```
airport(id: string, pos: point)
```

“For each airport, what the minimal distance it had from the center of hurricane Lizzy?”

```
SELECT id, val(initial(atmin(distance(center(Lizzy), pos)))) FROM airport;
```

Example: “At what times was hurricane Lizzy heading north (let’s say, within 10 degrees of the exact north direction)?”

```
LET degreeRange = range(80, 100);  
LET dirLizzy = mdirection(center(Lizzy));  
deftime(at(dirLizzy, degreeRange));
```

Operations on Sets of Objects

Sometimes necessary to access *components*.

Components are (as in **no_components**):

- single interval of range(α)
- single point of points
- connected component of line graph
- face of a region
- maximal continuous part of a moving(α) function value.

Operations on Sets of Objects (cont.)

Example:

Table that contains histories of volcano eruptions:

```
volcano (name:string, eruptions:mpoint)
```

“Find the volcano eruptions that lasted for more than two days!”

Here we need to “extract” single volcano eruptions from each history of eruptions.

Operations on Sets of Objects (cont.)

Here a **decompose** operation is needed to access *components*.

In principle: $\alpha \rightarrow \underline{set}(\alpha)$

e.g. $\underline{region} \rightarrow \underline{set}(\underline{region})$

But: In the relational design, 1st normal form prohibits set attribute types. However, **decompose** operation should be widely usable, also in the relational model, hence formulated as a relation operation.

<i>Operation</i>	<i>Signature</i>	<i>Syntax</i>
decompose	$\underline{set}(\omega_1) \times (\omega_1 \rightarrow \sigma) \times \underline{ident} \rightarrow \underline{set}(\omega_2)$ $\underline{set}(\omega_1) \times (\omega_1 \rightarrow \underline{moving}(\alpha)) \times \underline{ident} \rightarrow \underline{set}(\omega_2)$	$arg_1 \ op[arg_2, arg_3]$

Operations on Sets of Objects: Examples

Example:

Consider the *country* relation :

```
country (name: string, area: region)
```

The expression

```
country decompose[area, part]
```

returns a relation with schema

```
(name: string, area: region, part: region)
```

Each tuple in the result will contain in its *part* attribute one component of the *area* attribute.

Operations on Sets of Objects: Examples (cont.)

Example:

Table that contains histories of volcano eruptions:

```
volcano (name:string, eruptions:mpoint)
```

“Find the volcano eruptions that lasted for more than two days”

```
LET oneday = duration(day(2000, 1, 1));  
LET volcano2 = volcano decompose[eruptions, one_eruption];
```

The table has schema:

```
(name: string, eruptions: mpoint, one_eruption: mpoint)
```

```
SELECT eruption FROM volcano2 WHERE duration(deftime(one_eruption)) > 2 * oneday
```

Continuity of Moving Type Values: Idea

Reasons to consider continuity:

- We want to implement our temporal types by “simple” functions: each *continuous* component should be implemented independently.
- In analyzing a value of a temporal type, it may be interesting when “discrete” changes occur (e.g., when size of a region suddenly increased)

What does continuity mean? Usually defined on real functions, hence clear for *moving(real)*. What about the other types?

Capture “discrete” changes, e.g.:

- a new point appears in a *points* value
- *line* “suddenly turns” without sweeping the intermediate positions of the plane
- a *region* suddenly has a new position or shape

Idea: introduce a measure of dissimilarity that is 0 when values are equal, and approaches 0 when values get more and more similar.

Continuity of Moving Type Values: Formal Definition

Definition: Let α be a type and $\psi : \bar{A}_\alpha \times \bar{A}_\alpha \rightarrow \mathbb{R}$. Let $f: \bar{A}_{\text{instant}} \rightarrow \bar{A}_\alpha$, and $t \in \bar{A}_{\text{instant}}$. f is ψ -continuous in t

$$:\Leftrightarrow \forall \gamma > 0 \exists \varepsilon > 0 \text{ such that } \forall \delta < \varepsilon : \psi(f(t \pm \delta), f(t)) < \gamma$$

where $\gamma, \delta, \varepsilon \in \mathbb{R}$.

Definition: Here are definitions of ψ for some of the data types α to which the *moving* constructor is applicable:

$$\alpha \in \{\underline{int}, \underline{string}, \underline{bool}\}$$

$$\psi(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases}$$

$$\alpha = \underline{real} :$$

$$\psi(x, y) = |x - y|$$

$$\alpha = \underline{point} :$$

$$\psi(p_1, p_2) = d(p_1, p_2)$$

$$\alpha = \underline{region} :$$

$$\psi(r_1, r_2) = \text{area}(r1 \setminus r2) + \text{area}(r2 \setminus r1)$$

Here $d(p_1, p_2)$ denotes the Euclidean distance between points p_1 and p_2 .