

# **MODELING AND QUERYING HISTORY OF MOVEMENT: ABSTRACT DATA MODEL**

## **Part 1**

# An Abstract Model

**Design Goals:** Design a system of data types & operations which is

- closed
- simple
- powerful

**Closed:** under application of type constructors, in particular:

- For all base types of interest, we have corresponding time-dependent (temporal, “moving”) types.
- For all temporal types, we have types to represent their *domain* and *range* projections.

**Simple:**

- There are lots of types around, avoid proliferation of operations → use generic operations as much as possible.
- Explore the space of possible operations systematically.
- Achieve consistency of non-temporal and temporal types.

**Powerful:** more or less a result of the previous two.

# Data Types

## Standard Types

int, real, bool, string

## Spatial Types

point



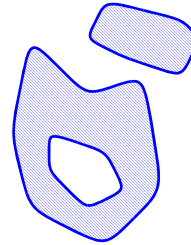
points



line



region



point A point in the Euclidean plane, or undefined.

points A finite set of points from the plane.

line A finite set of continuous curves in the plane. The “interiors” of any two curves are disjoint.

region A finite set of disjoint faces; each face  $F$  a regular closed ( $F = \overline{F^\circ}$ ) set of points from the plane with non-empty, connected interior (may have holes).

## Data Types (cont.)

### Temporal Types

instant isomorphic to real

### Type Constructors

$\alpha$  a standard type or a spatial type. (int, real, bool, string, point, points, line, region)

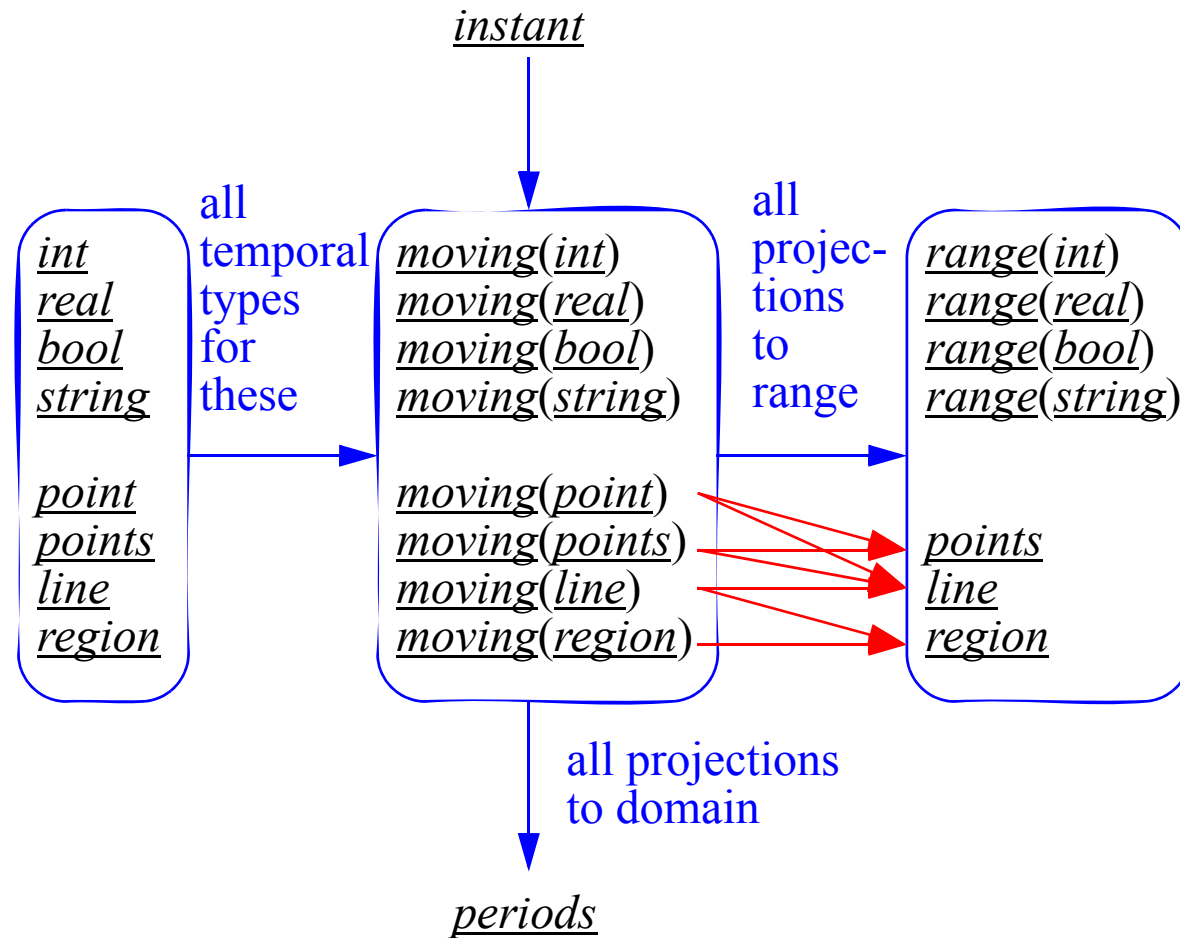
moving( $\alpha$ ) A value is a partial function  $f: A_{\text{instant}} \rightarrow A_{\alpha}$ .

intime( $\alpha$ ) A value is a pair from  $A_{\text{instant}} \times A_{\alpha}$ .

$\alpha$  an ordered domain. (int, real, bool, string, instant)

range( $\alpha$ ) A value is a finite set of disjoint, non-adjacent intervals over  $\alpha$ . Each interval can be closed, open, or half-open.

## Data Types (cont.)



## Data Types (cont.)

### Type System as a Signature

<i>Type constructor</i>	<i>Signature</i>
<u>int</u> , <u>real</u> , <u>string</u> , <u>bool</u>	→ BASE
<u>point</u> , <u>points</u> , <u>line</u> , <u>region</u>	→ SPATIAL
<u>instant</u>	→ TIME
<u>range</u>	BASE $\cup$ TIME → RANGE
<u>moving</u> , <u>intime</u>	BASE $\cup$ SPATIAL → TEMPORAL

## Formal Definition of Data Types

Semantics of types: define **carrier sets**.

**Definition:** The carrier sets for the base types are:

$$A_{\underline{int}} := Z \cup \{\perp\},$$

$$A_{\underline{real}} := R \cup \{\perp\},$$

$$A_{\underline{string}} := V^* \cup \{\perp\}, \text{ where } V \text{ is a finite alphabet,}$$

$$A_{\underline{bool}} := \{FALSE, TRUE\} \cup \{\perp\}.$$

**Definition:** The carrier sets for the types point and points are:

$$A_{\underline{point}} := R^2 \cup \{\perp\},$$

$$A_{\underline{points}} := \{P \subset R^2 \mid P \text{ is finite}\}.$$

## Formal Definition of Data Types (cont.)

Definitions for *line* and *region* already fairly complex.

For *region*:

**Definition:** Let  $S$  be a class of simple curves. The carrier set of *region* is

$$A_{\text{region}} := \{Q \subset \mathbf{R}^2 \mid \exists R \in RC(S) : Q = \text{points}(R)\}$$

$RC(S)$ : The set of *R-complexes* over  $S$ .

An *R-complex* is a finite set of non-empty, regular closed sets that are pairwise *quasi-disjoint*.

*Quasi-disjoint*: Two regular closed sets  $Q, R$  are called *quasi-disjoint* iff  $Q \cap R$  is finite.

## Formal Definition of Data Types (cont.)

Type constructor *moving*:

**Definition:** Let  $\alpha$ , with carrier set  $A_\alpha$ , be a type to which type constructor *moving* is applicable. Then the carrier set for *moving*( $\alpha$ ) is

$$A_{\underline{\text{moving}}(\alpha)} := \{f \mid f: \bar{A}_{\text{instant}} \rightarrow \bar{A}_\alpha \text{ is a partial function} \wedge \Gamma(f) \text{ is finite}\}$$

$\Gamma(f)$  is finite:  $f$  has only a finite number of “continuous components”.

# Operations: Overview

## Design Goals:

- as generic as possible
- achieve consistency between operations on non-temporal and temporal types

e.g.

$$\begin{array}{lll} \underline{point} \times \underline{point} & \rightarrow \underline{real} & \mathbf{distance} \\ \underline{mpoint} \times \underline{mpoint} & \rightarrow \underline{mreal} & \mathbf{mdistance} \end{array}$$

$$\begin{aligned} & \mathbf{distance(atinstant(mp_1, t), atinstant(mp_2, t))} \\ & = \mathbf{atinstant(mdistance(mp_1, mp_2), t)} \end{aligned}$$

## Three steps:

1. Define operations on non-temporal types
2. “Lift” them all to temporal types
3. Add specific operations for temporal types

# Operations on Non-Temporal Types

Generic view: *point* and *point set* in some space

	$\pi$	$\sigma$
Space	point type	point set type
Integer	<u>int</u>	<u>range(int)</u>
Real	<u>real</u>	<u>range(real)</u>
Bool	<u>bool</u>	<u>range(bool)</u>
String	<u>string</u>	<u>range(string)</u>
Time	<u>instant</u>	<u>periods</u>
2D Space	<u>point</u>	<u>points, line, region</u>

1D Spaces

$\pi \times \sigma$	$\rightarrow$ <u>bool</u>	<b>inside</b>	instantiates to
<u>int</u> $\times$ <u>range(int)</u>	$\rightarrow$ <u>bool</u>	<b>inside</b>	
<u>bool</u> $\times$ <u>range(bool)</u>	$\rightarrow$ <u>bool</u>		
<u>instant</u> $\times$ <u>periods</u>	$\rightarrow$ <u>bool</u>		
<u>point</u> $\times$ <u>line</u>	$\rightarrow$ <u>bool</u>		etc.

## Operations on Non-Temporal Types (cont.)

Class	Operations
Predicates	<b>isempty</b> <b>=, /=, intersects, inside</b> <b>&lt;, &lt;=, &gt;=, &gt;, before</b> <b>touches, attached, overlaps, on_border, in_interior</b>
Set Operations	<b>intersection, union, minus</b> <b>crossings, touch_points, common_border</b>
Aggregation	<b>min, max, avg, center, single</b>
Numeric	<b>no_components, size, perimeter, duration, length, area</b>
Distance and Direction	<b>distance, direction</b>
Boolean	<b>and, or, not</b>

## Operations on Non-Temporal Types (cont.): Predicates

### Unary

<i>Operation</i>	<i>Signature</i>	<i>Semantics</i>
<b>isempty[undefined]</b>	$\pi \rightarrow \underline{bool}$	$u = \perp$
	$\sigma \rightarrow \underline{bool}$	$U = \emptyset$

### Binary

	<i>Sets</i>	<i>Order (1D Spaces)</i>	<i>Topology</i>
<i>point vs. point</i>	$u = v, u \neq v$	$u < v, u \leq v,$ $u \geq v, u > v$	
<i>point set vs. point set</i>	$U = V, U \neq V$ $U \cap V \neq \emptyset$ ( <b>intersects</b> ) $U \subseteq V$ ( <b>inside</b> )	$\forall u \in U, \forall v \in V : u \leq v$ <b>(before)</b>	$\partial U \cap \partial V \neq \emptyset$ ( <b>touches</b> ) $\partial U \cap V^\circ \neq \emptyset$ ( <b>attached</b> ) $U^\circ \cap V^\circ \neq \emptyset$ ( <b>overlaps</b> )
<i>point vs. point set</i>	$u \in V$ ( <b>inside</b> )	$\forall u \in U : u \leq v$ ( <b>before</b> ) $\forall v \in V : u \leq v$	$u \in \partial V$ ( <b>on_border</b> ) $u \in V^\circ$ ( <b>in_interior</b> )

## Operations on Non-Temporal Types: Set Operations

<i>Operation</i>	<i>Signature</i>	
(i) <b>intersection, minus</b>	$\pi \times \pi$	$\rightarrow \pi$
(ii) <b>intersection</b>	$\pi \otimes \sigma$	$\rightarrow \pi$
<b>minus</b>	$\pi \times \sigma$	$\rightarrow \pi$
	$\sigma \times \pi$	$\rightarrow \sigma$
<b>union</b>	$\pi \otimes \sigma$	$\rightarrow \sigma$
(iii) <b>intersection, minus, union</b>	$\sigma \times \sigma$	$\rightarrow \sigma$ [1D]
(iv) <b>intersection</b>	$\sigma_1 \times \sigma_2$	$\rightarrow \min(\sigma_1, \sigma_2)$ [2D]
<b>minus</b>	$\sigma_1 \times \sigma_2$	$\rightarrow \sigma_1$ [2D]
<b>union</b>	$\sigma \times \sigma$	$\rightarrow \sigma$ [2D]
(v) <b>crossings</b>	<u>line</u> $\times$ <u>line</u>	$\rightarrow$ <u>points</u>
<b>touch_points</b>	<u>region</u> $\otimes$ <u>line</u>	$\rightarrow$ <u>points</u>
	<u>region</u> $\times$ <u>region</u>	$\rightarrow$ <u>points</u>
<b>common_border</b>	<u>region</u> $\otimes$ <u>line</u>	$\rightarrow$ <u>line</u>
	<u>region</u> $\times$ <u>region</u>	$\rightarrow$ <u>line</u>

**Semantics:** Standard set operations in 1D, regularized set operations in 2D.

## Operations on Non-Temporal Types: Aggregation

Reduces point sets to sets.

<i>Operation</i>	<i>Signature</i>	<i>Semantics</i>
<b>min, max</b>	$\sigma \rightarrow \pi$ [1D]	$\min(\rho(U)), \max(\rho(U))$
<b>avg</b>	$\sigma \rightarrow \pi$ [1Dnum]	
<b>avg[center]</b>	$\sigma \rightarrow \pi$ [2D]	
<b>single</b>	$\sigma \rightarrow \pi$	if $\exists u: U = \{u\}$ then $u$ else $\perp$
<b>min[start], max[end]</b>	$\underline{periods} \rightarrow \underline{instant}$	

**Example:** “Find the point where the highway A1 crosses the river Rhine.”

```
LET RhineA1 = ELEMENT(  
  SELECT single(crossings(R.route, H.route))  
  FROM river R, highway H  
  WHERE R.name = 'Rhine' AND H.name = 'A1' )
```

```
city (name:string, pop: int, center: point)  
country (name: string, area: region)  
river (name: string, route: line)  
highway (name: string, route: line)
```

## Operations on Non-Temporal Types: Numeric Properties of Sets

<i>Operation</i>	<i>Signature</i>	
<b>no_components</b>	$\sigma$	$\rightarrow \underline{int}$
<b>size</b>	$\sigma$	$\rightarrow \underline{real}$ [cont]
<b>perimeter</b>	<u>region</u>	$\rightarrow \underline{real}$
<b>size[duration]</b>	<u>periods</u>	$\rightarrow \underline{real}$
<b>size[length]</b>	<u>line</u>	$\rightarrow \underline{real}$
<b>size[area]</b>	<u>region</u>	$\rightarrow \underline{real}$

**Example:** “How long is the common border of France and Germany?”

```
LET France = ELEMENT(SELECT area FROM country WHERE name = 'France');
LET Germany = ELEMENT(SELECT area FROM country WHERE name = 'Germany');
length(common_border(France, Germany))
```

```
city (name:string, pop: int, center: point)
country (name: string, area: region)
river (name: string, route: line)
highway (name: string, route: line)
```

## Operations on Non-Temporal Types: Distance and Direction

<i>Operation</i>	<i>Signature</i>
<b>distance</b>	$\pi \times \pi \rightarrow \underline{real}$ [cont]
	$\pi \otimes \sigma \rightarrow \underline{real}$ [cont]
	$\sigma \times \sigma \rightarrow \underline{real}$ [cont]
<b>direction</b>	$\underline{point} \times \underline{point} \rightarrow \underline{real}$

**Direction:** angle of the line from the first to the second point, measured in degrees, relative to a horizontal line (counterclockwise). If  $q$  is exactly north of  $p$  then **direction**( $p, q$ ) = 90.

**Example:** “Find the cities north of and within 200 kms of Munich!”

```
LET Munich = ELEMENT(SELECT center FROM city WHERE name = 'Munich');
SELECT name FROM city
WHERE distance(center, Munich) < 200
AND direction(Munich, center) => 45 AND direction(Munich, center) <= 135
```

```
city (name:string, pop: int, center: point)
country (name: string, area: region)
river (name: string, route: line)
highway (name: string, route: line)
```

## Operations on Non-Temporal Types: Boolean

bool → bool    **and**

bool → bool    **or**

bool → bool    **not**

# Kernel Algebra

This completes the scope of the **kernel algebra** subject to **lifting**.

Kernel Algebra:

- Types in  $\text{BASE} \cup \text{SPATIAL}$
- Operations defined above, restricted to those types.

## First Step Done

Three steps:

1. Define operations on non-temporal types **(done)**
2. “Lift” them all to temporal types → **postponed**
3. Add specific operations for temporal types → **next**