9 Representation
Syntax is Representation

Syntax: Agreed-upon representation for semantic concepts
Semantics is Representation

Semantic Domain

Abstraction

Representations
9 Representation

Representation as a tool of thought
Languages and DSLs are domain representations
Example 1: Spatial Query Language
Example 2: 2D Animations
Example 3: Data Collection & Analysis
A Number Game

• Given: 9 number cards 1, 2, 3, ..., 9
• Two players, A and B, alternate in picking cards
• The player who first can form a sum of 15 wins

Example game:
A takes 8, B takes 2, A takes 4, B takes 3, A takes 5

Question:
Which move should B make next?
Another Game

• Tick-tack-toe: two players, A and B, alternate in placing Xs and Os in a 3-by-3 grid.

• The player who first can form a row, column, or diagonal wins

Example game:

```
X O X
X
O
```

Question:
Which move should B make next?
Both Games Are Isomorphic

• Arrange numbers in the 3-by-3 grid so that they form a magic square.

• Color selected numbers by player

Example game:

Question: Which move should B make next?

Take 6
Multiple Representations

- Mapping the arithmetic representation to a spatial representation makes the task easy for humans
- Mapping the spatial representation to an arithmetic representation makes the task easy for computers

The effectiveness of a representation depends on the cognitive abilities of the “user”
Textual versus Visual

- Humans are “spatial creatures”
  → Effective DSLs are often visual

- Computers are “non-spatial machines”
  → General-purpose languages are textual
Choice of Representation

- The right representation makes problem solving easy
- A good representation captures essential elements and leaves out the rest
- Abstract**tion** is critical: express important aspects in designing a representation

- Cognitive artifacts serve as tools of thought (most important tool: paper + reading and writing)
- “It is things that make us smart” (Don Norman)
DSLs as Tools of Thought

• A DSL is a tool of thought for a particular domain

• A good DSL design facilitates the easy representation of essential domain elements

• Abstraction is critical: A DSL should be designed around the most important domain objects

CREATIVITY CRITERION
An ideal DSL lets you explore new ideas in a domain that were not thought of before
Metalanguages

• BNF and Haskell are metalanguages for describing languages

• BNF and Haskell are metarepresentations

• Advantages of Haskell:
  Can describe semantics (BNF only does syntax)
  Allows more flexible definitions:
  - parameterized types (type constructors)
  - function types in productions
  - higher-order abstract syntax

  **CREATIVITY ASPECT**
  ... explore new ideas in the domain
Other Metalanguages

- **UML**
  
  *Advantage*: Can be understood by OO programmers
  
  *Disadvantages*: Limited like BNF, few abstractions, not scalable

- **Prolog**
  
  *Advantage*: Nice parser support, backtracking
  
  *Disadvantages*: Not higher order (but see Lambda Prolog), untyped

*Strong typing of metalanguage guarantees syntax correctness of represented language*
Other Metalanguages

• Idris
  *Advantages*: Can model sophisticated dependent type systems and typed semantics, can produce automatic correctness proofs (in some cases)
  *Disadvantage*: Niche product, limited support

• And more:
  C++ (templates)
  FreshML (support for binders)
  MetaOCaml (staged programming)
  Graph Grammars, Multiset Grammars (visual languages)
  ...
Example 1: Spatial Query Language

Example Query: “Find all cities in a certain area”

⇒ Need types for points and regions
⇒ Need operations and predicates, such as intersection and inside

```haskell
type Point = ...
type Region = ...

(/
) :: Region \rightarrow Region \rightarrow Region

inside :: Point \rightarrow Region \rightarrow Bool
```
Representation of Regions

A direct, obvious approach:

```haskell```

```haskell
type Point  = (Float,Float)
type Region = [Point]
```

Implementation of inside or intersection is hard and takes a long time!

A functional approach:

```haskell```

```haskell
type Region = Point -> Bool

(/
\) :: Region -> Region -> Region
(r /
\ s) p = r p && s p
```

Implementation is trivial and found within in a minute
More Spatial Functions

type Region = Point -> Bool

(\ \ ) :: Region -> Region -> Region
(r \ \ s) p = r p && s p

(\ /) :: Region -> Region -> Region
(r \ / s) p = r p || s p

outside :: Region -> Region
outside r = not . r

inside :: Point -> Region -> Bool
inside p r = r p

inside = flip ($)
Abstraction Opportunities

type Region = Point -> Bool

(\/
) :: Region -> Region -> Region
(r \/
 s) p = r p && s p

(\/
) :: Region -> Region -> Region
(r \
 s) p = r p || s p

Same pattern

liftB :: (Bool -> Bool -> Bool) -> Region -> Region -> Region
liftB f g h x = f (g x) (h x)

(\/
) = liftB (&&)
(\/
) = liftB (||)
Query Language

A simple database

```haskell
data City = C {name::String, pos::Point}

cs :: [City]
cs = [C "A" (1,1), C "B" (4,4)]
```

A simple query: “Find all cities in area r”

```haskell
cs `select` (pos `isInside` r)
```

```haskell
isInside :: (a -> Point) -> Region -> (a -> Bool)
isInside f r x = f x `inside` r

isInside f r x = r (f x)
isInside f r = r . f
isInside = flip (.)
```
Query Language Observations

Representation of one basic object (region) directly supports several critical operations.

Functional representation is powerful, since functions can be applied.

Higher-order functions can leverage functional representations of objects.
Example 2: Visual Languages

Domain: 2D animations and 2D games on a grid-like background

⇒ Need description for movement of objects

Example Animation: “Define a train that follows train tracks”

\[
\begin{align*}
\text{move } d & \quad t & \\
| d == \text{East} & \quad & t.\text{right} == \text{HorTrack} = & \text{Right} \\
| d == \text{East} & \quad & t.\text{right} == \text{RgtTurn} = & \text{TurnRight} \\
| & \quad & & \ldots
\end{align*}
\]
Visual Rewrite Rules

AgentSheets by Alexander Repenning
Creating Rules by Analogy

“Cars moves on streets like trains move on tracks”

Output: a visual rewrite rule

A form of visual metaprogramming
Movement Built into Background

Automatic bending of movement tiles
Example 3: Data Collection & Analysis

Domain: Science & financial data collections

Example: “Collect and aggregate precipitation data”

\[
\begin{align*}
corv &= [23, 29, 12, 15, 5, 0, 0, 8, 11, 17, 25, 29] \\
salem &= [21, 26, 19, 16, 5, 1, 0, 9, 5, 21, 20, 24] \\
pdx &= [27, 28, 17, 13, 9, 3, 0, 7, 16, 15, 21, 28] \\
precip &= [corv, salem, pdx] \\
\end{align*}
\]

\[
\begin{align*}
agg~x &= (\text{maximum } x, \text{sum } x) \\
placeAgg &= \text{map}~agg~\text{precip} \\
monthAgg &= \text{map}~agg~(\text{transpose}~\text{precip}) \\
\end{align*}
\]

\[
\begin{align*}
> \text{placeAgg} \\
&= [(29, 174), (26, 167), (28, 184)] \\
\end{align*}
\]

relationship to data source is lost!
## Spatially Embedded Computing

![Excel Sheet Image]

The table below represents precipitation data for different cities over various months:

<table>
<thead>
<tr>
<th>Month</th>
<th>Corv</th>
<th>Salem</th>
<th>PDX</th>
<th>MAX</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>23</td>
<td>21</td>
<td>27</td>
<td>27</td>
<td>71</td>
</tr>
<tr>
<td>Feb</td>
<td>29</td>
<td>26</td>
<td>28</td>
<td>29</td>
<td>83</td>
</tr>
<tr>
<td>Mar</td>
<td>12</td>
<td>19</td>
<td>17</td>
<td>19</td>
<td>48</td>
</tr>
<tr>
<td>Apr</td>
<td>15</td>
<td>16</td>
<td>13</td>
<td>16</td>
<td>44</td>
</tr>
<tr>
<td>May</td>
<td>5</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>Jun</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Jul</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Aug</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>24</td>
</tr>
<tr>
<td>Sep</td>
<td>11</td>
<td>5</td>
<td>16</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Oct</td>
<td>17</td>
<td>21</td>
<td>15</td>
<td>21</td>
<td>53</td>
</tr>
<tr>
<td>Nov</td>
<td>25</td>
<td>20</td>
<td>21</td>
<td>25</td>
<td>66</td>
</tr>
<tr>
<td>Dec</td>
<td>29</td>
<td>24</td>
<td>28</td>
<td>29</td>
<td>81</td>
</tr>
<tr>
<td>MAX</td>
<td>29</td>
<td>26</td>
<td>28</td>
<td>29</td>
<td>525</td>
</tr>
<tr>
<td>SUM</td>
<td>174</td>
<td>167</td>
<td>184</td>
<td>184</td>
<td>525</td>
</tr>
</tbody>
</table>

The data shows the precipitation levels for Corvallis, Salem, and Portland (PDX) for each month, with calculations for maximum (MAX) and sum (SUM) values.
Syntax of Spreadsheets

Formulas
\[ f \in F \quad ::= \quad v \mid \omega(f, \ldots, f) \mid \rho \]

Values
\[ \rho \quad ::= \quad (\rho, \rho) \]

Operations
\[ \rho \quad ::= \quad n \mid +n \mid -n \]

References
\[ \rho \quad ::= \quad (p, p) \]

Two-dimensional grid

Pointers
\[ p \quad ::= \quad n \mid +n \mid -n \]

Absolute & relative references

Natural numbers

Spreadsheets
\[ S \in \Sigma \quad = \quad A \rightarrow F \]

 Addresses
\[ a \in A \quad ::= \quad (n, n) \]

(a, f) \in S is called a cell
Spreadsheets vs. Lambda Calculus

\[ f \in F ::= v \mid \omega(f, \ldots, f) \mid \rho \]

\[ e ::= \lambda x.e \mid e\ e \mid x \]

\[ S \in \Sigma = A \rightarrow F \]

Representations
Computational Limitations of Spreadsheets

\[ f \in F ::= v \mid \omega(f, \ldots, f) \mid \rho \]

\[ e ::= \lambda x.e \mid e e \mid x \]

first-order language
no local scope
generally, no recursion

\[ S \in \Sigma = A \rightarrow F \]