Data Types and Variations

Elisabeth Lempa
May 24, 2018
One important question of Computer Science can be posed as follows:

*How can we organize data so that it models the real world in a consistent and meaningful way?*
Idea: First Order Predicate Logic comes with a powerful semantic system.
Idea: First Order Predicate Logic comes with a powerful semantic system.

If we can find a way to reduce our constructs for data-organization to formulas of predicate logic, we can use this semantic system!
Consider this simple representation of a certain Dinosaur data object.

Dinosaur

- name : Steggy
- genus : Stegosaurus
- size_m : 6.13
Consider this simple representation of a certain *Dinosaur* data object.

Dinosaur

- name : Steggy
- genus : Stegosaurus
- size_m : 6.13

This could be mapped to the atomic formula

\[
\text{dinosaur}(\text{Steggy}, \text{Stegosaurus}, 6.13)
\]
Data Objects as Formulas

Consider this simple representation of a certain **Dinosaur** data object.

Dinosaur

- name: Steggy
- genus: Stegosaurus
- size_m: 6.13

This could be mapped to the atomic formula

\[ \text{dinosaur}(\text{Steggy}, \text{Stegosaurus}, 6.13) \]

Or the compound formula

\[ \text{dinosaur}(c) \land (\text{name}(c) = \text{Steggy}) \land (\text{genus}(c) = \text{Stegosaurus}) \land (\text{size}_m(c) = 6.13) \]
Consider this example of unrequited dinosaur love:

Dinosaur
  name    : Steggy
  genus   : Stegosaurus
  size_m  : 6.13
  likes   : Abby
Consider this example of unrequited dinosaur love:

<table>
<thead>
<tr>
<th>Dinosaur</th>
<th></th>
<th>Dinosaur</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Steggy</td>
<td>name</td>
<td>Abby</td>
</tr>
<tr>
<td>genus</td>
<td>Stegosaurus</td>
<td>genus</td>
<td>Apatosaurus</td>
</tr>
<tr>
<td>size_m</td>
<td>6.13</td>
<td>size_m</td>
<td>22.4</td>
</tr>
<tr>
<td>likes</td>
<td>Abby</td>
<td>likes</td>
<td>SELF</td>
</tr>
</tbody>
</table>

The word SELF refers to the very syntactic unit in which it occurs. As first-order predicate logic does not support cyclic terms, this cannot be translated directly into logic!
Consider this example of unrequited dinosaur love:

<table>
<thead>
<tr>
<th>Dinosaur</th>
<th>Dinosaur</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>: Abby</td>
</tr>
<tr>
<td>genus</td>
<td>genus</td>
</tr>
<tr>
<td></td>
<td>: Apatosaurus</td>
</tr>
<tr>
<td>size_m</td>
<td>size_m</td>
</tr>
<tr>
<td></td>
<td>: 22.4</td>
</tr>
<tr>
<td>likes</td>
<td>likes</td>
</tr>
<tr>
<td></td>
<td>: SELF</td>
</tr>
</tbody>
</table>

The word SELF refers to the very syntactic unit in which it occurs. As first-order predicate logic does not support cyclic terms, this cannot be translated directly into logic!
Relational Databases

While logic is concerned with **formalizing propositions** (i.e. statements that are either true or false):

\[ \text{dinosaur} (\text{Steggy}, \text{Stegosaurus}, 6.13) \]

"Steggy the Stegosaurus is a 6.13m long dinosaur."
Relational Databases

While logic is concerned with **formalizing propositions** (i.e. statements that are either true or false):

\[ \text{dinosaur}(\text{Steggy}, \text{Stegosaurus}, 6.13) \]

\[ \text{can be interpreted to mean:} \]

\[ \text{“Steggy the Stegosaurus is a 6.13m long dinosaur.”} \]

Relational databases handle **tables filled with data**, that is typically organized in tuples:

<table>
<thead>
<tr>
<th>Dinosaurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>Steggy</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Relational Databases

While logic is concerned with formalizing propositions (i.e. statements that are either true or false):

\[ \text{dinosaur}(\text{Steggy, Stegosaurus, 6.13}) \] can be interpreted to mean:

“Steggy the Stegosaurus is a 6.13m long dinosaur.”

Relational databases handle tables filled with data, that is typically organized in tuples:

<table>
<thead>
<tr>
<th>Dinosaurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>Steggy</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

It is easy to see that these two formalisms are transferable into each other! (Predicates are even sometimes called relational symbols for this precise reason.)
A relational calculus query has the form:

\[ \{ u \mid \varphi \} \]

Where \( u \) is a list of terms and \( \varphi \) is a logical formula.
A **relational calculus query** has the form:

\[ \{ u \mid \varphi \} \]

Where \( u \) is a list of terms and \( \varphi \) is a logical formula.

It looks (and works) like set-comprehension: The query \( \{ u \mid \varphi \} \) yields all such lists of terms \( u \), so that its variables satisfy the formula \( \varphi \).
A **relational calculus query** has the form:

\[ \{ u \mid \varphi \} \]

Where \( u \) is a list of terms and \( \varphi \) is a logical formula.

It looks (and works) like set-comprehension: The query \( \{ u \mid \varphi \} \) yields all such lists of terms \( u \), so that its variables satisfy the formula \( \varphi \). Example:

**Find all Dinosaurs which are Stegosauri and larger than 6 meters.**

\[ \{ (a, b, c) \mid (a, b, c) \in Dinosaurs, b = 'Stegosaurus', c > 6.0 \} \]
Consider these two terms:

1. \( \text{dinosaur}(\text{Steggy}, \text{Stegosaurus}, 6.13) \)
2. \( \text{dinosaur}(\text{Stegosaurus}, \text{Steggy}, 6.13) \)

Changing the order of the predicate's arguments obviously results in a representation of different dinosaurs:

1. is a Stegosaurus named Steggy
2. is a Steggy-Dinosaur (whatever that is) named Stegosaurus (which is not a very good name.)

For large numbers of arguments, especially if they aren't as easily distinguishable, this can become very inconvenient to use, very fast.
Positioning one’s arguments correctly

Consider these two terms:

1. $dinosaur(Steggy, Stegosaur, 6.13)$
2. $dinosaur(Stegosaur, Steggy, 6.13)$

Changing the order of the predicate’s arguments obviously results in representation of different dinosaurs:

1. is a Stegosaurus named Steggy
2. is a Steggy-Dinosaur (whatever that is) named Stegosaurus (which is not a very good name.)
Positioning one’s arguments correctly

Consider these two terms:

1. \(\text{dinosaur}(\text{Steggy, Stegosaur}, 6.13)\)
2. \(\text{dinosaur}(\text{Stegosaur, Steggy}, 6.13)\)

Changing the order of the predicate’s arguments obviously results in representation of different dinosaurs:

1. is a Stegosaurus named Steggy
2. is a Steggy-Dinosaur (whatever that is) named Stegosaurus (which is not a very good name.)

For large number of arguments, especially if they aren’t as easily distinguishable, this can become very inconvenient to use, very fast.
This can be mitigated by allowing to associate an identifier (a role) with each argument position.

\[ \text{dinosaur}(\text{name } \rightarrow \text{Steggy}, \text{genus } \rightarrow \text{Stegosaur}, \text{size } \rightarrow \text{6.13}) \]
This can be mitigated by allowing to associate an identifier (a role) with each argument position.

\[
dinosaur(name \rightarrow \text{Steggy}, \text{genus} \rightarrow \text{Stegosaur}, \text{size} \rightarrow 6.13)
\]

- This allows for the position of the arguments to be neglected.
- Improves readability without changing the expressive power, as this notation can be easily transcribed back into position-based notation by simply mapping each role name to a position number.
- This role-based notation can also easily be applied to relational databases.
What does the following formula mean?

\[ \forall x \exists y \text{parent}(x, y) \]
If we want to express the range of our variables, we can formalize this as:

$$\forall x \ (\text{person}(x) \Rightarrow \exists y \ (\text{person}(y) \land \text{parent}(x, y)))$$

This gets tedious very fast. We can formalize this notion of adding an atom that restricts each quantified variable.
Definition (Range Restricted Formulas or RR-Formulas)

1. Each quantifier-free $L$-Formula is an RR-Formula.
2. Each $L$-Formula constructed from a connective and an appropriate number of RR-Formulas is an RR-Formula.
3. If $\phi$ is an RR-Formula and $A$ is an atom and $x$ is a subset of the free variables in $A$, then $\forall x(A \Rightarrow \phi)$ and $\exists x(A \land \phi)$ are RR-Formulas. We will then call $A$ the range for the variables in $x$. 
RR-Formula Examples

\[ p(x) \]

\[(p(x) \Rightarrow s(x)) \land q(x, y) \Rightarrow (r(x) \lor r(y)) \]

\[ \forall x \ (p(x) \lor q(x)) \]

\[ \forall x \ (p(x) \Rightarrow q(x, y)) \]
Following this notion, we can associate each term of our logical system with a sort.

Instead of, when considering a predicate or function symbol, only asking the question:

*How many arguments should this be applied to?*

We will now also ask:

*What sort of arguments should this be applied to?*
Many-Sorted Predicate Logic

We can extend our model as follows:

When defining a **predicate symbol**, additionally to an arity $n$, we supply an $n$-tuple of sort symbols: $(s_1 \times s_2 \times \cdots \times s_n)$.

This means, that the first argument should be of sort $s_1$, the second of sort $s_2$, and so on.
We can extend our model as follows:

When defining a **predicate symbol**, additionally to an arity $n$, we supply an $n$-tuple of sort symbols: $(s_1 \times s_2 \times \cdots \times s_n)$.

This means, that the first argument should be of sort $s_1$, the second of sort $s_2$, and so on.

When defining a **function symbol**, additionally to an arity $n$, we supply an $n+1$-tuple of sort symbols, marking the last one with a leading arrow: $(s_1 \times s_2 \times \cdots \times s_n \rightarrow s_{n+1})$.

Where the arguments are of sorts $s_1$ to $s_n$, and the result is of sort $s_{n+1}$. 
Examples.

\[
\text{parent} : \text{person} \times \text{person} \\
\text{parent} : \text{dinosaur} \times \text{dinosaur} \\
Tom : \text{person}, \ Steggy : \text{dinosaur} \\
\text{bestfriend} : \text{person} \rightarrow \text{person}
\]
By restricting our syntactic definitions of term and formula to well-sorted ones, this allows for static sort-checking:

`parent(Tom, Steggy)` would, for example, not be a valid formula of many-sorted predicate logic.
Translating back

Each many-sorted predicate logic formula can be translated back into a classical predicate logic formula.
Translating back

Each many-sorted predicate logic formula can be translated back into a classical predicate logic formula.

For each sort-symbol $s$, add a unary relation symbol $\hat{s}$:

$$\text{Steggy : } dinosaur \rightsquigarrow \text{dinosaur}(\text{Steggy})$$
Translating back

Each many-sorted predicate logic formula can be translated back into a classical predicate logic formula.

For each sort-symbol $s$, add a unary relation symbol $\widehat{s}$:

$\text{Steggy} : \text{dinosaur} \leadsto \text{dinosaur}(\text{Steggy})$

$\text{bestfriend} : \text{person} \rightarrow \text{person} \leadsto \forall x (\text{person}(\text{bestfriend}(x)) \Rightarrow \text{person}(x))$
Sorting out range restrictions

Note that these back translated formulas are always RR-formulas.

This means, again, that introducing sorts does not affect the expressive power of the system!
However, it introduces a powerful method of error detection and can vastly improve readability and shorten formulas.
We could also allow the definition of more complex, constructed sorts, for example: \texttt{Either[person, dinosaur]}
We could also allow the definition of more complex, constructed sorts, for example: Either[person, dinosaur]

\[
\text{Arthur} : \text{Either}[\text{person}, \text{dinosaur}] \leadsto \overline{\text{person}}(\text{Arthur}) \lor \overline{\text{dinosaur}}(\text{Arthur})
\]
We could also allow the definition of more complex, constructed sorts, for example: Either[person, dinosaur]

Arthur : Either[person, dinosaur] \sim \text{person}(Arthur) \lor \text{dinosaur}(Arthur)

bestfriend : person \rightarrow Either[person, dinosaur] \sim \forall x(\text{person}(\text{bestfriend}(x)) \lor \text{dinosaur}(\text{bestfriend}(x)) \Rightarrow \text{person}(x))
We can see that this concept is very similar to the concept of *types* in programming languages, and it could potentially be extended in similar ways:

1. Product-Sorts
2. Recursive Sorts
3. Higher-Order Function Symbols
4. Higher-Kinded Sorts
5. ...
Conclusion

- How can we model data objects as logical formulas? What are the limits of this model?
- How can this view be applied to relational databases?
- What are range restricted formulas, and why are they useful?
- What are sorts?
Conclusion

- How can we model data objects as logical formulas? What are the limits of this model?
- How can this view be applied to relational databases?
- What are range restricted formulas, and why are they useful?
- What are sorts?

... if this talk has been successful, you should be able to answer these now 😊