The Architecture of the GALILEO Project

Michael Chan
(Joint work with Alan Bundy and Jos Lehmann)

School of Informatics
University of Edinburgh

ARCOE 2009
Overview

- Current implementation:
  - Implemented in $\lambda$Prolog (using Teyjus).
  - Uses deep embedding.
  - Ad hoc and inference is not enabled.

- Design goals for the new architecture:
  - Need access to higher-order inference engines to prove trigger formulae
  - Modularise the ontologies and use development graphs for maintaining the ontologies and managing change.

- Development graph is based on institution theory, where nodes are ontologies and arrows are morphisms.

- Hets:
  - An analysis tool for Casl and enables theorems to be proved using ATPs.
  - Contains an implementation of development graphs.
Before Joseph Black's investigations, heat and temperature conflated.

Leads to paradox when heat is reduced but temperature is constant!
The \( \lambda \)Prolog Implementation

<table>
<thead>
<tr>
<th>Logic</th>
<th>( \lambda )Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O_t \vdash \text{Heat}(H_2O, \text{Start}(\text{Freeze})) = \text{Heat}(H_2O, \text{Start}(\text{Freeze})) )</td>
<td>axioms ont_t (applic equal (applic heat (h2o :: (applic start freeze) :: nil) :: :: nil)) :: nil) ::</td>
</tr>
<tr>
<td>( O_t \vdash \text{Heat}(H_2O, \text{Start}(\text{Freeze})) &gt; \text{Heat}(H_2O, \text{End}(\text{Freeze})) )</td>
<td>applic greater (applic heat (h2o :: (applic start freeze) :: nil) :: (applic heat (h2o :: applic endd freeze :: nil)) :: nil) :: nil) ::</td>
</tr>
</tbody>
</table>

- The ontologies are deeply-embedded.
- Cannot do inference.
  - All theorems are coded as elements of a list of axioms.
  - Cannot derive the predictive theory for freezing.
- Difficult to extend an ontology from another.
  - The lack of subtyping forces the types for heat and energy to be distinct.
- Difficult to read.
The **CASL** Specification

<table>
<thead>
<tr>
<th>Logic</th>
<th>CASL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_t \vdash \text{Heat}(H_2O, \text{Start}(\text{Freeze})) = \text{Heat}(H_2O, \text{Start}(\text{Freeze}))$</td>
<td>spec ClassicalEnergyFlow = BasicPhys then sorts Energy &lt; Int; Heat &lt; Energy; Energy_Release &lt; Process; ... ops HeatAmt : Obj * Time -&gt; Heat end</td>
</tr>
<tr>
<td>$O_t \vdash \text{Heat}(H_2O, \text{Start}(\text{Freeze})) \neq \text{Heat}(H_2O, \text{End}(\text{Freeze}))$</td>
<td>spec Energy_Release = ClassicalEnergyFlow then forall o:Obj; t1,t2:Time . t1 B t2 &lt;=&gt; HeatAmt(o,t1) &gt; HeatAmt(o,t2) . t1 = t2 &lt;=&gt; HeatAmt(o,t1) = HeatAmt(o,t2) . t1 Bi t2 &lt;=&gt; HeatAmt(o,t1) &lt; HeatAmt(o,t2) end</td>
</tr>
<tr>
<td>$O_s \vdash \text{Heat}(H_2O, \text{Start}(\text{Freeze})) = \text{Heat}(H_2O, \text{End}(\text{Freeze}))$</td>
<td>spec Os = Ot then %implies %implies %implies forall h:H2O; f:Freeze . HeatAmt(h, Start(f)) &gt; HeatAmt(h, End(f)) %%(prediction)% end</td>
</tr>
</tbody>
</table>

- Nodes can import the axiomatisation of other nodes.
- Subtypes can be defined as subsorts.
  - Heat can now be related to energy.
  - Permits a richer description.
The specification is to be analysed using Hets

- We will still need to define axioms, unless we mechanise the workings of Newtonian theory, etc.
- With support for reasoning, the axioms defined in CASL can be more general, e.g.,
  \[
  \forall o:Obj; t1,t2:Time.
  \text{t1} \leq \text{t2} \iff \text{Heat}(o,\text{t1}) > \text{Heat}(o,\text{t2})
  \]
  vs.
  \[
  \text{applic greater (applic heat (h2o :: (applic start freeze) :: nil) :: (applic heat (h2o :: applic endd freeze :: nil)) :: nil)}
  \]

- For higher-order logic, we will use HasCASL.
The ontologies are modularised.

- 0t and 0s extend from other nodes and are implicitly defined.
- The prediction and observation become proof obligations in Energy_Release and 0t, respectively.

CASL

```plaintext
spec ClassicalEnergyFlow = BasicPhys
then
  sorts Energy < Int; Heat < Energy;...
  ops HeatAmt : Obj * Time -> Heat
end
spec Energy_Release = ClassicalEnergyFlow
then
  forall o:Obj; t1,t2:Time
  . t1 B t2 <=> HeatAmt(o,t1) > HeatAmt(o,t2)
  . t1 = t2 <=> HeatAmt(o,t1) = HeatAmt(o,t2)
  . t1 Bi t2 <=> HeatAmt(o,t1) < HeatAmt(o,t2)
end
spec Ot = Energy_Release
then %implies
  forall h:H2O; f:Freeze
  . HeatAmt(h, Start(f)) > HeatAmt(h, End(f))
end
spec Os = Ot
then %implies
  %observation%
```

The Development Graph for Latent Heat
Figure: DGs after decomposition and subsumption.
Ontology Repair Plans

- Resolve logical conflicts between modular ontologies.
- They are composed of rules for conflict diagnosis (trigger) and transformation of ontologies (repair).
- Repair operations could split/conflate concepts, add/remove arguments, etc.
- Repairs can be represented as morphisms between the old and the new ontologies.
  - The repaired ontologies typically inherit some of the signatures and theorems of the old.
  - We may need to extend DG to consider signature changes as morphisms.
The λProlog Implementation

<table>
<thead>
<tr>
<th>Logic</th>
<th>λProlog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trigger:</strong></td>
<td>wms_trigger 01 02 Stuff Arg (applic Stuff Arg) 2 :-</td>
</tr>
<tr>
<td>$O_t \vdash \text{stuff} (\vec{s}) = v_1,$</td>
<td>turnstile 01 (applic equal (applic Stuff Arg :: Val1 :: nil)),</td>
</tr>
<tr>
<td>$O_s \vdash \text{stuff} (\vec{s}) = v_2,$</td>
<td>turnstile 02 (applic equal (applic Stuff Arg :: Val2 :: nil)),</td>
</tr>
<tr>
<td>$O_t \vdash v_1 \neq v_2$</td>
<td>turnstile 02 (applic greater (Val1 :: Val2 :: nil)).</td>
</tr>
<tr>
<td><strong>Repair:</strong></td>
<td>repair 01 02 NA1 NA2 :</td>
</tr>
<tr>
<td>$\text{Ax}(\nu(O_t)) ::= {\forall \vec{s}.\vec{\tau}.$</td>
<td>wms_trigger 01 02 S L Tot P</td>
</tr>
<tr>
<td>$\text{stuff}_{\text{invis}}(\vec{s}) ::= \text{stuff}(\vec{s}) -$</td>
<td>flip P FP</td>
</tr>
<tr>
<td>$\text{stuff}_{\text{vis}}(\vec{s})} \cup \text{Ax}(O_t)$</td>
<td>choose S L T</td>
</tr>
<tr>
<td>$\text{newstuff} \text{Tot T SVis SInvis}$</td>
<td>newstuff Tot T SVis SInvis</td>
</tr>
<tr>
<td>$\text{axioms} O1 A1$</td>
<td>axioms O1 A1</td>
</tr>
<tr>
<td>$\text{axioms} O2 A2$</td>
<td>axioms O2 A2</td>
</tr>
<tr>
<td>changeWMS P O1 A1 Tot SVis</td>
<td>changeWMS P O1 A1 Tot SVis</td>
</tr>
<tr>
<td>SInvis NA1</td>
<td>SInvis NA1</td>
</tr>
<tr>
<td>changeWMS FP O2 A2 Tot SVis</td>
<td>changeWMS FP O2 A2 Tot SVis</td>
</tr>
<tr>
<td>SInvis NA2.</td>
<td>SInvis NA2.</td>
</tr>
</tbody>
</table>

- The predicate turnstile is used to represent that a formula is a theorem of an ontology.
- Mathematical relations such as $=$ and $<$ need to be defined.
Shallow and Deep embeddings

- We cannot deeply-embed or shallowly-embed the logics.
  - With a deep embedding, we’ll lose the ability to do inference on the object-logic.
  - With a shallow embedding, we’ll lose the ability to recurse over syntactical structures.

- We can use Huet’s zippers for recursion over the syntax of a theorem.

- The recursion will need to be done either in Hets/DocTip itself or in an external module interfaced with Hets/DocTip.
Effecting Repair

- The repair is guided by external programs, possibly implemented in λProlog.
- Haskell could be an alternative, but the implementation would be less natural.
- To effect the repair, the DG is manipulated by, e.g., adding/removing nodes and adding/removing links between nodes.
What has to be Done

- We will need mechanisms for effecting the repair in DGs.
- Need an *API* connecting repair programs to Hets/DocTip.
- The changes made may need to be propagated to the GUI.
- How ontology repairs can be represented as morphisms?
- How should the recursion over syntactical structures of theorems take place?
The current implementation has several major limitations, e.g., the inability to perform inference.

The new architecture focuses on modularising ontologies and reduces most of these limitations.

So far, the implementation of the latent heat ontologies looks promising.

Work with Hets/DocTip developers to:
- Implement mechanisms that effect repair
- Build an API
- Extend morphisms to include signature changes