Grailog: Mapping Generalized Graphs to Computational Logic


Harold Boley
NRC-ICT Fredericton
Faculty of Computer Science, University of New Brunswick Canada

Symposium on Natural/Unconventional Computing and its Philosophical Significance, AISB/IACAP World Congress - Alan Turing 2012, 2-6 July 2012, Birmingham, UK

Thanks for feedback on various versions and parts of this presentation:
- The Grailog User Interface for Knowledge Bases of Ontologies & Rules
  OMG Technical Meeting, Ontology PSIG, Cambridge, MA, 21 June 2012
- Grailog: Knowledge Representation with Extended Graphs for Extended Logics
  SAP Enterprise Semantics Forum, 24 April 2012
- Grailog: Towards a Knowledge Visualization Standard
  BMIR Research Colloquium, Stanford, CA, 4 April 2012
- PARC Research Talk, Palo Alto, CA, 29 March 2012

RuleML/Grailog: The Rule Metalogic Visualized with Generalized Graphs
PhiloWeb 2011, Thessaloniki, Greece, 5 October 2011
Abstract

Human intuition is often supported by graph-like knowledge constructs depicting objects as (atomic) nodes and (binary) relationships as directed labeled arcs. Following the AI tradition of simple semantic networks and the Semantic Web use of RDF triple stores, philosophical and domain knowledge could in principle be specified as a single directed labeled graph. However, such graphs cannot directly represent nested structures, non-binary relationships, and relation descriptions; these advanced features require encoded (‘contrived’) constructs with auxiliary nodes and relationships, which also need to be kept separate from direct (‘natural’) constructs. Therefore, various extensions of directed labeled graphs have been proposed for knowledge representation, including graph partitionings (possibly interfaced as complex nodes), n-ary relationships as directed labeled hyperarcs, and (hyper)arc labels used as nodes of other (hyper)arcs. Meanwhile, a lot of AI and Semantic Web research and development has gone into extended logics for knowledge representation such as description logics, general modal logics, and higher-order logics. The talk demonstrates how knowledge representation with graphs and logics can be reconciled. It proceeds from simple to extended graphs for logics needed in Philosophy, Cognitive Science, AI, and the Semantic Web. Along with its visual introduction, each graph construct is mapped to its corresponding symbolic logic construct. This has led to the development of the knowledge representation language Grailog as part of the Web-rule industry standard RuleML. By serializing Grailog knowledge in RuleML/XML (http://ruleml.org/#Grailog), it will become interchangeable between Web-based engines for Computational Logic.
Remove Barrier to Entry for Logic: Graph Visualization of Knowledge

- From 1-dimensional *symbol-logic* knowledge specification to 2-dimensional *graph-logic visualization* in a convenient 2D syntax
  - Supports *human in the loop* in knowledge elicitation, validation, and processing

- Combinable with graph transformations for efficient *implementation* of specifications & for visualizing model-theoretic *semantics*
  - Deep names, as graph nodes, mapped directly/injectively to elements of semantic interpretation
Grailog

**Graph inscribed logic** invokes imagery for logic.

Proposed cognitively motivated graph standard for visual-logic knowledge:

Easy to learn and draw, read and remember, e.g. for eScience, eLearning, and eBusiness.

Generalized-graph framework as one uniform user interface to major (Semantic Web) logics:

Pick & choose subset for each knowledge base, map to/fro RuleML sublanguage and UML+OCL, and access via API4KB protocol.
Generalized Graphs for the Representation and Mapping of Logic Languages

- We have used generalized graphs for representing various logic languages, where basically:
  - Graph nodes (vertices) represent individuals, classes, etc.
  - Graph arcs (edges) represent relations

- Next slides: What are the principles of this representation and what graph generalizations are required?

- Later slides: How are these graphs mapped (invertibly) to logic, thus specifying Grailog as a ‘GUI’ for RuleML?
Grailog Principles

• Graphs should make it easier for humans to read and write logic constructs by exploiting a 2-dimensional representation with shorthand & normal forms, from Controlled English to logic

• Graphs should be *natural extensions* (e.g. n-ary) of Directed Labeled Graphs (DLGs), often used to represent simple semantic nets, i.e. of atomic ground formulas in function-free dyadic predicate logic (cf. binary Datalog ground facts, RDF triples, the Open Graph, and the Knowledge Graph)

• Graphs should allow *stepwise refinements* for all logic constructs: Description logic constructors, F-logic frames, general PSOA RuleML terms, etc.

• Extensions to boxes & links should be *orthogonal*
Informal Grailog Preview: Searle’s Chinese Room Argument

John Searle (emphasis added):

• “... whatever purely formal principles you put into the computer, they will not be sufficient for understanding, since a human will be able to follow the formal principles without understanding anything.”

(Minds, Brains and Programs, 1980)
Searle’s Chinese Room Scenario: Grailog for Visual Controlled English

Classes with relations

- **Language**
  - **English**
  - **Chinese**
  - **ruleset**
  - **text**
  - **question**
  - **reply**

Instances with relations

- **Searle**
- **Wang**
- **Searle-reply\_i**
- **Wang-reply\_i**

Relations:
- **subClassOf**
- **hasInstance**
- **negation**
- **lang hasLanguage**
- **understand**
- **apply**
- **to**
- **use**
- **with**
- **for**

Visually, the diagram depicts the structure and relationships between different classes and instances, with arrows indicating the flow of concepts and relations.
Grailog Generalizations

- **Directed hypergraphs:** For n-ary relations, directed (binary) arcs should be generalized to directed (n-ary) *hyperarcs*, e.g. representing relational tuples.

- **Recursive (hierarchical) graphs:** For nested terms and formulas, modal logics, and modularization, ‘flat’ graphs should be generalized to allow other graphs as *complex nodes* to any level of ‘depth’.

- **Labelnode graphs:** For allowing hybrid logics describing both instances and predicates, arc *labels* should also become usable as *nodes*.
Graphical Elements: Names

• Written into boxes (nodes):
  **Deep** (canonical, distinct) names
  – (Occurrence-)restricted
    Unique Name Assumption (rUNA)
    via Deep Name Occurrence (DNO)

• Written onto boxes (node labels):
  **Shallow** (alternate, ‘aka’) names
  – (Occurrence-)restricted
    Non-unique Name Assumption (rNNA)
    via Shallow Name Occurrence (SNO)
Instances: Individual Constants with Deep Name Specification

General: Graph (node) Logic

Examples: Graph Logic

Warren Buffett Warren Buffett
General Electric General Electric

US$ 3 000 000 000 US$ 3 000 000 000 000
Instances: Individual Constants with Shallow Name Specification

General: Graph (node)  Logic (vertical bar marks shallowness)

Examples: Graph  Logic

WB  /WB
GE  /GE
US$ 3B  /US$ 3B
Parameters: Individual Variables

General: Graph (\textit{hatched} node) Logic (\textit{italics} font, \texttt{POSL} uses "?" prefix)

Examples: Graph

\begin{tabular}{ccc}
\text{variable} & \text{variable} \\
$X$ & $X$ \\
$Y$ & $Y$ \\
$A$ & $A$
\end{tabular}
Predicates: Binary Relations (1)

General: Graph \((labeled\ arc)\)

Example: Graph

Warren Buffett \(\xRightarrow{\text{Trust}}\) General Electric

Logic

\(\text{binrel}(\text{inst}_1, \text{inst}_2)\)
Predicates: Binary Relations (2)

General: Graph (labeled arc) Logic

Example: Graph Logic

\[ \text{binrel}(\text{var}_1, \text{var}_2) \]

\[ \text{Trust}(X, Y) \]
Predicates: n-ary Relations (n>1)

General: Graph (hyperarc)

Example: Graph (n=3)

Logic

rel(inst₁, inst₂, ..., instₙ⁻¹, instₙ)

Invest(WB, GE, US$ 3·10⁹)
Implicit Conjunction of Formula Graphs: Co-Occurrence on Graph Top-Level

General: Graph (\(m\) hyperarcs)

\[
\begin{align*}
\text{inst}_{1,1} & \xrightarrow{\text{rel}_1} \text{inst}_{1,2} \rightarrow \ldots \rightarrow \text{inst}_{1,n^1} \\
\text{inst}_{m,1} & \xrightarrow{\text{rel}_m} \text{inst}_{m,2} \rightarrow \ldots \rightarrow \text{inst}_{m,n^m}
\end{align*}
\]

Logic

\[
\begin{align*}
\text{rel}_1(\text{inst}_{1,1}, \text{inst}_{1,2}, \\
\ldots, \text{inst}_{1,n^1}) \wedge \\
\ldots \wedge \\
\text{rel}_m(\text{inst}_{m,1}, \text{inst}_{m,2}, \\
\ldots, \text{inst}_{m,n^m})
\end{align*}
\]

Example: Graph (2 hyperarcs)

Logic

\[
\begin{align*}
\text{Invest} & \rightarrow \text{US$3 \cdot 10^9$} \\
\text{Invest} & \rightarrow \text{US$2 \cdot 10^4$}
\end{align*}
\]

Invest(\(/\text{WB}, /\text{GE}, \text{US$3 \cdot 10^9$}) \wedge \\
Invest(\(/\text{JS}, /\text{VW}, \text{US$2 \cdot 10^4$})
Explicit Conjunction of Formula Graphs: Co-Occurrence in Complex (And) Node

General: Graph \((m\text{ hyperarcs})\)

Logic
\[(rel_1(inst_{1,1}, inst_{1,2}, \ldots, inst_{1,n^1}) \land \ldots \land rel_m(inst_{m,1}, inst_{m,2}, \ldots, inst_{m,n^m}))\]

Example: Graph \((2\text{ hyperarcs})\)

Logic
\[(\text{Invest}(/WB, /GE, \text{US}\$3 \cdot 10^9) \land \text{Invest}(/JS, /VW, \text{US}\$2 \cdot 10^4))\]
Disjunction of Formula Graphs: Co-Occurrence in Or Node

General: Graph (wavy)

\[
\begin{align*}
\text{inst}_{1,1} & \quad \text{rel}_1 \quad \text{inst}_{1,2} \quad \cdots \quad \rightarrow \quad \text{inst}_{1,n^1} \\
\text{inst}_{m,1} & \quad \text{rel}_m \quad \text{inst}_{m,2} \quad \cdots \quad \rightarrow \quad \text{inst}_{m,n^m}
\end{align*}
\]

Logic

\[
(\text{rel}_1(\text{inst}_{1,1}, \text{inst}_{1,2}, \ldots, \text{inst}_{1,n^1}) \lor \ldots \lor \\
\text{rel}_m(\text{inst}_{m,1}, \text{inst}_{m,2}, \ldots, \text{inst}_{m,n^m}) )
\]

Example: Graph

Logic

\[
(\text{Invest}(/\text{WB}, /\text{GE}, \text{US$3 \cdot 10^9$}) \lor \text{Invest}(/\text{JS}, /\text{VW}, \text{US$2 \cdot 10^4$}))
\]
From Hyperarc Crossings to Node Copies as a Normalization Sequence (1)

Hypergraph (2 hyperarcs, crossing outside nodes)

DLG (4 arcs, do not specify to whom Latin is shown or taught)

<table>
<thead>
<tr>
<th>John</th>
<th>Show</th>
<th>Latin</th>
<th>Paul</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Teach</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>John</th>
<th>Show</th>
<th>Latin</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Teach</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latin</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Paul</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
From Hyperarc Crossings to Node Copies as a Normalization Sequence (1*)

Hypergraph (2 hyperarcs, crossing inside a node)

DLG (4 arcs, do not specify to whom Latin is shown or taught)
From Hyperarc Crossings to Node Copies as a Normalization Sequence (1**)

Hypergraph (2 hyperarcs, parallel-cutting a node)

DLG (4 arcs, do not specify to whom Latin is shown or taught)
From Hyperarc Crossings to Node Copies as a Normalization Sequence (1***)

Hypergraph (2 hyperarcs, employing a node copy)

Logic (2 relations, employing a symbol copy)

\[
\begin{align*}
\text{John} & \rightarrow \text{Latin} & \rightarrow \text{Kate} \\
\text{Mary} & \rightarrow \text{Latin} & \rightarrow \text{Paul}
\end{align*}
\]

\[
\begin{align*}
\text{Show(John, Latin, Kate)} & \land \\
\text{Teach(Mary, Latin, Paul)}
\end{align*}
\]

Both ‘Latin’ occurrences remain one node even when copied for easier layout:
As a deep name, ‘Latin’ will remain unique
Predicates: Unary Relations (Classes, Concepts, Types)

General: Graph (class applied to instance node)

Example: Graph

Logic

Billionaire(Billionaire(Warren Buffett))
Class Hierarchies (Taxonomies): Subclass Relation

General: Graph (two nodes) (Description)

Logic

Example: Graph (Description)

Rich

Billionaire

class₁ \sqsubseteq class₂

subClassOf

class₁ \sqsubseteq Billionaire

class₁ \sqsubseteq Rich
Intensional Class Constructions (Ontologies): Class Intersection

General: Graph (solid node, as for conjunction)

Example: Graph

$\text{class}_1 \land \text{class}_2 \land \ldots \land \text{class}_n$

(Description)
Logic

Billionaire $\land$ Benefactor $\land$ Environmentalist

(Description)
Logic
Intensional Class Constructions (Ontologies): Class Union

General: Graph \((\text{wavy node, as for disjunction})\)

Example: Graph

\[
\text{class}_1 \cup \text{class}_2 \cup \ldots \cup \text{class}_n
\]

\[
\text{Billionaire} \cup \text{Benefactor} \cup \text{Environmentalist}
\]
Class Hierarchies (Taxonomy DAGs): Top and Bottom

General: Top (special node)

General: Bottom (special node)
Intensional Class Constructions (Ontologies): Class-Property-Restricting TBox—Existential (1*)

General: Graph (normal) (Description) Logic
\[ \exists \text{binrel}. \text{class} \]

Example: Graph (Description) Logic
\[ \exists \text{Substance}. \text{Physical} \]

A kind of schema, where Top class is specialized to have (multi-valued) attribute/property, Substance, with at least one value typed by class Physical.
Instance Assertions (Populated Ontologies): Adding ABox to Restriction TBox—Existential (1*)

General: Graph (normal) (rUNA-Description)
Logic
\[ \exists \text{binrel.class} (\text{inst}_0) \land \text{class}(\text{inst}_1) \land \text{binrel}(\text{inst}_0, \text{inst}_1) \]

Example: Graph (rUNA-Description)
Logic
\[ \exists \text{Substance}.\text{Physical} (\text{Socrates}) \land \text{Physical}(\text{P1}) \land \text{Substance}(\text{Socrates}, \text{P1}) \]
Intensional Class Constructions (Ontologies): Class-Property-Restricting TBox—Universal (1*)

General: Graph (normal) (Description)
Logic
∀binrel . class

Example: Graph (Description)
Logic
∀Substance . Physical

A kind of schema, where Top class is specialized to have (multi-valued) attribute/property, Substance, with each value typed by class Physical.
Instance Assertions (Populated Ontologies): Adding ABox to Restriction TBox—Universal (1*)

**General:** Graph (normal)

(rUNA-Description) Logic
\[
\forall \text{binrel.class(inst}_0) \land \\
\text{class(inst}_1) \land \\
\ldots \\
\text{class(inst}_n) \land \\
\text{binrel(inst}_0, \text{inst}_1) \land \\
\ldots \\
\text{binrel(inst}_0, \text{inst}_n)
\]

**Example:** Graph

(rUNA-Description) Logic
\[
\forall \text{Substance. Physical (Socrates)} \land \\
\text{Physical(P1)} \land \\
\text{Physical(P2)} \land \\
\text{Substance(Socrates, P1)} \land \\
\text{Substance(Socrates, P2)}
\]
Object-Centered Logic: Grouping Binary Relations Around Instance

General: Graph (\textit{inst}_0\text{-centered})

\begin{itemize}
  \item class
  \item binrel_1 \quad \text{inst}_1
  \item \ldots
  \item \text{inst}_0 \to \text{inst}_1
  \item \text{binrel}_n \quad \text{inst}_n
\end{itemize}

Example: Graph (Socrates-centered)

\begin{itemize}
  \item Philosopher
  \item Socrates
  \item Substance
  \item Teaching
  \item P1
  \item T1
\end{itemize}

(\text{Object-Centered})

Logic

\begin{itemize}
  \item class(\text{inst}_0) \land
  \item binrel_1(\text{inst}_0, \text{inst}_1) \land
  \item \ldots
  \item binrel_n(\text{inst}_0, \text{inst}_n)
\end{itemize}

(\text{Object-Centered})

Logic

\begin{itemize}
  \item Philosopher(Socrates) \land
  \item Substance(Socrates, P1) \land
  \item Teaching(Socrates, T1)
\end{itemize}
RDF-Triple (‘Subject’-Centered) Logic: Grouping Properties Around Instance

General: Graph (\textit{inst}_0\text{-centered})

\begin{itemize}
  \item \textit{inst}_0
  \item \text{property}_1
  \item \ldots
  \item \text{property}_n
  \item \textit{inst}_1
  \item \text{property}_1
  \item \ldots
  \item \text{property}_n
  \item \textit{inst}_n
\end{itemize}

Example: Graph (Socrates-centered)

\begin{itemize}
  \item \text{Philosopher}
  \item Socrates
  \item \text{Substance}
  \item \text{Teaching}
  \item \text{P1}
  \item \text{T1}
\end{itemize}

(Subject-Centered) Logic

\begin{itemize}
  \item \{(\textit{inst}_0, \text{rdf:type}, \textit{class})\}
  \item \{(\textit{inst}_0, \text{property}_1, \textit{inst}_1)\}
  \item \ldots
  \item \{(\textit{inst}_0, \text{property}_n, \textit{inst}_n)\}
\end{itemize}

(Subject-Centered) Logic

\begin{itemize}
  \item \{(Socrates, \text{rdf:type}, \text{Philosopher})\}
  \item \{(Socrates, \text{Substance}, \text{P1})\}
  \item \{(Socrates, \text{Teaching}, \text{T1})\}
\end{itemize}
Logic of Frames (‘Records’): Associating Slots with OID-Distinguished Instance

General: Graph (bulleted arcs) (PSOA-like Frame) Logic

```
class

inst_0
  ... slot_1
      inst_1
  ... slot_n
      inst_n

inst_0 \#class(
  slot_1 -> inst_1;
  ... slot_n -> inst_n)
```

Example: Graph (PSOA-like Frame) Logic

```
Socrates
  Substance -> P1
  Teaching -> T1

Socrates\#Philosopher(
  Substance -> P1;
  Teaching -> T1)
```
Logic of Shelves (‘Arrays’): Associating Tuple(s) with OID-Distinguished Instance

General: Graph (bulleted hyperarc) (PSOA-like Shelf) Logic

Example: Graph (PSOA-like Shelf) Logic

Socrates#Philosopher(c. 469 BC, 399 BC)
Positional-Slotted-Term Logic: Associating Tuple(s)+Slots with OID-Disting’ed Instance

General: Graph (PSOA-like Positional-Slotted-Term) Logic

Example: Graph (PSOA-like Positional-Slotted-Term) Logic

Socrates#Philosopher(c. 469 BC, 399 BC; Substance->P1; Teaching->T1)
Rules: Relations Imply Relations (1)

General: Graph (ground, shorthand)

Logic

\[ rel_1(\text{inst}_{1,1}, \text{inst}_{1,2}, ..., \text{inst}_{1,n^1}) \Rightarrow \]
\[ rel_2(\text{inst}_{2,1}, \text{inst}_{2,2}, ..., \text{inst}_{2,n^2}) \]

Example: Graph

Logic

Invest(\text{/WB,} \text{/GE,} \text{US$ 3 \cdot 10^9}) \Rightarrow \]
Invest(\text{/JS,} \text{/VW,} \text{US$ 5 \cdot 10^3})
Rules: Relations Imply Relations (3)

General: Graph (inst/var terms)

Logic

\[(\forall \text{var}_{i,j})\]
\[\text{rel}_1(\text{term}_{1,1}, \text{term}_{1,2}, \ldots, \text{term}_{1,n_1}) \Rightarrow \text{rel}_2(\text{term}_{2,1}, \text{term}_{2,2}, \ldots, \text{term}_{2,n_2})\]

Example: Graph

Logic

\[(\forall Y, A)\]
\[\text{Invest}(\text{/WB, Y, A}) \Rightarrow \text{Invest}(\text{/JS, Y, US$ 5 \cdot 10^3})\]
Rules: Conjuncts Imply Relations (1)

General: Graph (shorthand)

Logic

\[(\forall \text{var}_{i,j})\]
\[rel_1(\text{term}_{1,1}, \text{term}_{1,2}, \ldots, \text{term}_{1,n^1}) \land \]
\[rel_2(\text{term}_{2,1}, \text{term}_{2,2}, \ldots, \text{term}_{2,n^2}) \Rightarrow \]
\[rel_3(\text{term}_{3,1}, \text{term}_{3,2}, \ldots, \text{term}_{3,n^3})\]

Example: Graph

Logic

\[(\forall Y, A)\]
Invest(\text{/WB, Y, A}) \land
Trust(\text{/JS, Y}) \Rightarrow
Invest(\text{/JS, Y, US$ 5 \cdot 10^3})
Modally Embedded Propositions

General: Graph (Modal) Logic
(complex octagon node, used to ‘quarantine’ what another agent believes, wants, etc.)

Example: Graph (Modal) Logic

believe_{agent}(graph)

believe_{GE}(\text{Invest}(/\text{WB, /GE, US$ 4 \cdot 10^9}))
Beliefs and Desires as Propositional Attitudes (1)

Propositional attitude: a mental state relating a person to a proposition (which can involve other persons)

“If George desires action A and believes (the proposition) that originator O will cause A, then George supports O.”

Grailog:
Beliefs and Desires as Propositional Attitudes (2)

Example: “If John desires the negation of (state of affairs) X, then he does not desire X.”

Grailog:

While variables A and O of the earlier example are bound to an action and originator individual, variable X here is bound to an entire proposition or an arbitrarily complex set of propositions.
Conclusions

- Presented new version of Grailog, including feedback
- Graphical elements for box and arrow systematics, leaving color (except for IRIs) for other purposes, e.g. highlighting subgraphs (for retrieval & inference)
- Introducing Deep vs. Shallow Name Specification
- Focus on mapping to a family of logics as in RuleML
- Use cases from philosophy to technology to business
- Processing of earlier Grailog-like DRLHs studied in Lisp, FIT, and Relfun
- Now aligned with Web-rule industry standard RuleML
Future Work (1)

• Refining/extending Grailog, along with API4KB effort
  – Comparing with other graph formalisms, e.g. Conceptual Graphs (http://conceptualstructures.org)
  – Defining mappings to/fro UML structure diagrams + OCL, adopting UML behavior diagrams (http://www.uml.org)

• Implementing tools, e.g. as use case for (Functional) RuleML (http://ruleml.org/fun) engines
  – More mappings between graphs, logic & RuleML/XML
  – Graph indexing & querying (cf. http://www.hypergraphdb.org)
  – Graph transformations (normal form, typing homomorphism, merge, ...)
  – Advanced graph-theoretical operations (e.g., path tracing)

• Benefit from, and contribute to, Protégé visualization plug-ins such as Jambalaya/OntoGraf and OWLViz for OWL ontologies and Axiomé for SWRL rules
Future Work (2)

• Proceeding from the 2-dimensional (planar) Grailog to a 3-dimensional (spatial) one
  – Exploiting advantages of crossing-free layout, spatial shortcuts, and analogical representation of 3D worlds
  – Mitigating disadvantages of occlusion and of harder spatial orientation and navigation
• Considering the 4th (temporal) dimension of animations to visualize logical inferences, graph processing, etc.
• Submitting for standardization
• See also: http://ruleml.org/#Grailog