From Data to Knowledge through Grailog Visualization


Harold Boley
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada

Decision CAMP, 4-6 November 2013
eBay Town Hall, San Jose, CA

Editorial Update: 11 November 2013
Acknowledgements

Thanks for feedback on various versions and parts of this presentation (the long version has all parts, hence gapless slide numbers):

From Data to Knowledge through Grailog Visualization
ISO 15926 and Semantic Technologies 2013 Conference, Sogndal, Norway, 5-6 September 2013

Grailog 1.0: Graph-Logic Visualization of Ontologies and Rules
The 7th International Web Rule Symposium (RuleML 2013), University of Washington, Seattle, WA, 11-13 July 2013

The Grailog Systematics for Visual-Logic Knowledge Representation with Generalized Graphs
Faculty of Computer Science Seminar Series, University of New Brunswick, Fredericton, Canada, 26 September 2012
High Performance Computing Center Stuttgart (HLRS), Stuttgart, Germany, 14 August 2012

Grailog: Mapping Generalized Graphs to Computational Logic
Symposium on Natural/Unconventional Computing and its Philosophical Significance, AISB/IACAP World Congress - Alan Turing 2012, 2-6 July 2012, Birmingham, UK

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

Grailog: Graph inscribed logic
Course about Logical Foundations of Cognitive Science, TU Vienna, Austria, 20 October -10 December 2008
Abstract

Directed labeled graphs (DLGs) provide a good starting point for visual data & knowledge representation but cannot straightforwardly represent nested structures, non-binary relationships, and relation descriptions. These advanced features require encoded constructs with auxiliary nodes and relationships, which also need to be kept separate from straightforward constructs. Therefore, various extensions of DLGs have been proposed for data & 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 / Semantic Web research and development on ontologies & rules has gone into extended logics for knowledge representation such as object (frame) logics, description logics, general modal logics, and higher-order logics. The slides demonstrate how data & knowledge representation with graphs and logics can be reconciled. They proceed from simple to extended graphs for logics needed in AI and the Semantic Web. Along with its visual introduction, each graph construct is mapped to its corresponding symbolic logic construct. These graph-logic extensions constitute a systematics defined by orthogonal axes, which has led to the Grailog 1.0 language as part of the Web-rule industry standard RuleML 1.0 (http://wiki.ruleml.org/index.php/Grailog). While Grailog’s DLG sublanguage corresponds to binary-associative memories, its hypergraph sublanguage corresponds to n-ary content-addressable memories, and its complex-node modules offer various further opportunities for parallel processing.
Visualization of Data

- *Useful* in many areas, *needed* for big data
- Gain **knowledge insights** from **data analytics**, ideally with the entire pipeline visualized
- Statistical visualization → Logical visualization

Sample data visualization ([http://wordle.net](http://wordle.net)): Word cloud for frequency of words from BMIR abstract of this talk
Visualization of Data & Knowledge: Graphs Make Logic Low-Entry-Cost

• From 1-dimensional *symbol-logic* knowledge specification to 2-dimensional *graph-logic visualization* in a systematic 2D syntax
  – Supports human in the loop across knowledge elicitation, specification, validation, and reasoning

• Combinable with graph transformation, (‘associative’) indexing & parallel processing for efficient *implementation* of specifications

• Move towards model-theoretic *semantics*
  – Unique names, as graph nodes, mapped directly/injectively to elements of semantic interpretation
Rule MetaLogic Provides Family of Language Standards for Web Data & Knowledge Interchange

- Developed on the Web: http://ruleml.org/metalogic
- Principal (family-uniform) and variant semantics
- Family-uniform syntaxes for humans and machines
Three RuleML Syntaxes (1)

Syntax

Visualization

RuleML/Grailog

Symbolic

Presentation

RuleML/POSL

Serialization

RuleML/XML
Three RuleML Syntaxes (2)

**Serialization** RuleML/XML:
Specified in XML Schema and recently in Relax NG:
http://ruleml.org

**Presentation** RuleML/POSL:
Integrates Prolog and F-logic, and translates to RuleML/XML:

**Visualization** RuleML/Grailog:
Based on Directed Recursive Labelnode Hypergraphs (DRLHs):
http://www.dfki.uni-kl.de/~boley/drlhops.abs.html
Grailog

Graph inscribed logic provides intuition for logic

Advanced cognitively motivated systematic

graph standard for visual-logic data & knowledge:

Features orthogonal → easy to learn,

 e.g. for (Business) Analytics

Generalized-graph framework as one uniform

2D syntax for major (Semantic Web) logics:

Pick subset for each targeted knowledge base,

map to/fro RuleML sublanguage, and exchange

& validate it, posing queries again in Grailog
Note on Grailog and API4KB

- Besides mapping Grailog to/fro RuleML, RDF and UML+OCL can be targeted, with uniform access to be provided by API4KB.
- Grailog and API4KB strive to cover main data & knowledge representation paradigms:
  - RDF (directed-labeled-graph) and Relational (Datalog-fact-like) data
  - Ontology (RDFS and description-logic) and Rule (Horn- and general-logic) knowledge
- An API can be (initially) designed and tested with a human in the loop much like a GUI.
Generalized Graphs to Represent and Map Logic Languages According to Grailog 1.0 Systematics

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

• 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 knowledge?

• Final slides: What is the systematics of Grailog features?
Grailog Principles

• Graphs should make it easier for humans to read and write logic constructs via 2D state-of-the-art 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 relationships

Language

SubClassOf
HasInstance

negation

lang hasLanguage

Instances with relationships

Searle
Wang
Searle-reply_i
Wang-reply_i

understand

English

apply
ruleset
to
use

Chinese

with

question

for

reply

HasInstance

distinguishable

understand

understand

understand

lang
to
use

langs

langs

langs

langs

langs
Grailog Generalizations

- **Directed hypergraphs:** For n-ary relationships, directed relation-labeled (binary) arcs will be generalized to directed relation-labeled (n-ary) hyperarcs, e.g. representing relational-database tuples.

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

- **Labelnode graphs:** For allowing higher-order logics describing both instances and relations (predicates), arc labels will also become usable as nodes.
Graphical Elements: Names

• Written into boxes (nodes): Unique (canonical, distinct) names
  – Unique Name Assumption (UNA) refined to Unique Name Specification (UNS)

• Written onto boxes (node labels): Non-unique (alternate, ‘aka’) names
  – Non-unique Name Assumption (NNA) refined to Non-unique Name Specification (NNS)

• Grailog combines UNS and NNS: $x$NS, with $x = U$ or $N$
Instances: Individual Constants with Unique Name Specifications

General: Graph (node) $\rightarrow$ Logic

$$\text{unique} \quad \text{unique}$$

Examples: Graph $\rightarrow$ Logic

$$\text{Warren Buffett} \quad \text{Warren Buffett}$$

$$\text{General Electric} \quad \text{General Electric}$$

$$\text{US$ 3 000 000 000} \quad \text{US$ 3 000 000 000}$$
Instances: Individual Constants with Non-unique Name Specifications

General: Graph (node)  Logic (vertical bar for non-uniqueness)

Examples: Graph

<table>
<thead>
<tr>
<th>WB</th>
<th>/WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE</td>
<td>/GE</td>
</tr>
<tr>
<td>US$ 3B</td>
<td>/US$ 3B</td>
</tr>
</tbody>
</table>
Graphical Elements: Hatching Patterns

- No hatching (boxes): Constant
- Hatching (elementary boxes): Variable
Parameters: Individual Variables

General: Graph (*hatched* node) Logic (*italics* font, POSL uses “?” prefix)

Examples: Graph

- $X$
- $Y$
- $A$

Logic

- $X$
- $Y$
- $A$
Predicates: Binary Relations (1)

General: Graph (labeled arc) Logic

Example: Graph Logic

Trust(Warren Buffett, General Electric)
Predicates: Binary Relations (2)

General: Graph (labeled arc) Logic

Example: Graph Logic

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

\[ \text{Trust}(X, Y) \]
Ground Equality: Identifying Pairs of Constants

General: Graph \((unlabeled\ undirected\ arc)\)

Example: Graph \(\text{inst}_1 = \text{inst}_2\)

Inspired by Charles Sanders Peirce’s line of identity, as a co-reference link
Ground Equality: Defining Constants with Constants

General: Graph (unlabeled, undirected, colon-tailed arc)

\[
\begin{array}{c}
\text{inst}_1 \\
\downarrow \\
\text{inst}_2 \\
\end{array}
\]

Logic (with oriented equality)

\[
\text{inst}_1 := \text{inst}_2
\]

Example: Graph Logic

\[
\begin{array}{c}
\text{GE} \\
\downarrow \\
\text{General Electric} \\
\end{array}
\]

\[
/\text{GE} := \text{General Electric}
\]
Ground Equality: Defining Symbolic Constants as IRIs

General: Graph (unlabeled, undirected, colon-tailed arc) Logic (with oriented equality, webized)

\[
\text{inst} : \text{IRI}
\]

Example: Graph Logic

\[
\text{GenElec} : \text{http://www.ge.com/}
\]

\[
/\text{GenElec} := \text{http://www.ge.com/}
\]

Definitional equality can also be used for the prefix part of the CURIE notation.
Negated Predicates: Binary Relations (Shorthand)

General: Graph *(dashed arc)* Logic

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

Example: Graph Logic

\[ \text{Trust}(\text{Joe Smallstock, General Electric}) \]
Negated Predicates: Binary Relations (Long Form)

General: Graph *(dashed box)* Logic

Example: Graph Logic

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

\[ \neg (\text{Trust}(\text{Joe Smallstock}, \text{General Electric})) \]
Ground Inequality: Pairwise Difference (Shorthand)

General: Graph (dashed unlabeled undirected arc)

Example: Graph

Joe Smallstock \quad \text{---} \quad Warren Buffett

Logic (with equality)

inst_1 \neq inst_2

Joe Smallstock \neq Warren Buffett
Ground Inequality: Pairwise Difference (Long Form)

General: Graph (dashed box, unlabeled undirected arc)

Example: Graph

Logic (with equality)

\( \neg (\text{inst}_1 = \text{inst}_2) \)

Logic (with equality)

\( \neg (\text{Joe Smallstock} = \text{Warren Buffett}) \)
Graphical Elements: Arrows (1)

- Labeled arrows (directed links) for arcs and hyperarcs (where hyperarcs ‘cut through’ nodes intermediate between first and last)
Predicates: n-ary Relations (n>1)

General: Graph (hyperarc)

Example: Graph (n=3)

Logic

Invest(/WB, /GE, US$ 3·10^9)
Negated Predicates: n-ary Relations (Shorthand)

General: Graph \((dashed \text{ hyperarc})\) Logic

Example: Graph \((n=3)\)

\[ \text{Logic} \]
\[ \neg \text{Invest(} \text{WB, GE, US$} \ 4 \cdot 10^9 \text{)} \]
Negated Predicates: n-ary Relations (Long Form)

General: Graph \((dashed\ box)\)

Example: Graph \((n=3)\)

Logic

\[ \neg (\text{rel}(\text{inst}_1, \text{inst}_2, \ldots, \text{inst}_{n-1}, \text{inst}_n)) \]

\[ \neg (\text{Invest}(/\text{WB}, /\text{GE}, \text{US}\$, 4 \cdot 10^9)) \]
 Implicit Conjunction of Formula Graphs: Co-Occurrence on Graph Top-Level

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

\[
\begin{align*}
\text{inst}_{1,1} & \quad \text{rel}_1 & \quad \text{inst}_{1,2} & \quad \ldots & \quad \text{inst}_{1,n_1} \\
\text{inst}_{m,1} & \quad \text{rel}_m & \quad \text{inst}_{m,2} & \quad \ldots & \quad \text{inst}_{m,n_m}
\end{align*}
\]

Logic

\[
rel_1(\text{inst}_{1,1}, \text{inst}_{1,2}, \ldots, \text{inst}_{1,n_1}) \land \ldots \land rel_m(\text{inst}_{m,1}, \text{inst}_{m,2}, \ldots, \text{inst}_{m,n_m})
\]

Example: Graph (2 hyperarcs)

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

Logic

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

General: Graph (solid+linear)

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

Logic

\[(Invest(/WB, /GE, US\$ 3 \cdot 10^9) \land Invest(/JS, /VW, US\$ 2 \cdot 10^4))\]
Not of And of Formula Graphs: Co-Ocurrence in a Not’s And Node

General: Graph \((\text{dashed/solid+linear})\) Logic

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

Example: Graph

\[
\neg (\text{Invest}(/\text{WB, GE, US$ 3 \cdot 10^9}) \land \text{Invest}(/\text{JS, VW, US$ 2 \cdot 10^4}))
\]
Not of And (Nand) of Formula Graphs: Co-Occurrence in Nand Node (Shorthand)

General: Graph \((\textit{dashed+linear})\) Logic

\[\neg(\textit{rel}_1(\textit{inst}_{1,1}, \textit{inst}_{1,2}, \ldots, \textit{inst}_{1,n_1}) \land \ldots \land \textit{rel}_m(\textit{inst}_{m,1}, \textit{inst}_{m,2}, \ldots, \textit{inst}_{m,n_m}))\]

Example: Graph

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

General: Graph (solid+wavy)

Logic

$(rel_1(inst_{1,1}, inst_{1,2}, \ldots, inst_{1,n^1}) \lor \ldots \lor rel_m(inst_{m,1}, inst_{m,2}, \ldots, inst_{m,n^m}))$

Example: Graph

Logic

$(Invest(/WB, /GE, US\$ 3 \cdot 10^9) \lor Invest(/JS, /VW, US\$ 2 \cdot 10^4))$
Not of Or of Formula Graphs: Co-Ocurrence in a Not’s Or Node

General: Graph \((\text{dashed/solid+linear/wavy})\) Logic

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

Logic

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

Example: Graph Logic

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

\[
-(\text{Invest}(/\text{WB}, /\text{GE}, \text{US$ 3 \cdot 10^9$}) \lor \text{Invest}(/\text{JS}, /\text{VW}, \text{US$ 2 \cdot 10^4$}))
\]
Not of Or (Nor) of Formula Graphs: Co-Occurrence in Nor Node

General: Graph *(dashed+wavy)* Logic

$$\neg (\text{rel}_1(\text{inst}_{1,1}, \text{inst}_{1,2}, ..., \text{inst}_{1,n^1}) \lor \lor \lor \text{rel}_m(\text{inst}_{m,1}, \text{inst}_{m,2}, ..., \text{inst}_{m,n^m}))$$

Example: Graph Logic

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

Hypergraph (2 hyperarcs, crossing inside a node)

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

Symbolic Controlled English

“John shows Latin to Kate. Mary teaches Latin to Paul.”
From Hyperarc Crossings to Node Copies as a Normalization Sequence (1*)

Hypergraph (2 hyperarcs, crossing outside nodes)

- John Show Latin
- Mary Teach Latin
  - Paul
  - Kate

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

- John Show Latin to Paul
- Mary Teach Latin to Kate
From Hyperarc Crossings to Node Copies as a Normalization Sequence (1**) 

Hypergraph (2 hyperarcs, parallel-cutting a node) 

John 
Show 
Latin 
Mary 
Teach 
Kate 
Paul

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

John 
Show 
Latin 
John 
Show 
Latin 
Mary 
Teach 
Kate 
Mary 
Teach 
Paul

The hyperarc for, e.g., ternary Show(John, Latin, Kate) can be seen as the path composition of 2 arcs for binary Show(John, Latin) and binary to(Latin, Kate)
From Hyperarc Crossings to Node Copies — Insert on Correct Binary Reduction

Hypergraph (2 hyperarcs, parallel-cutting a node)

DLG (8 arcs with 4 ‘reified’ relation/ship nodes to point to arguments)
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)

John Show Latin Kate
Mary Teach Latin Paul

Show(John, Latin, Kate) ∧ Teach(Mary, Latin, Paul)

Both ‘Latin’ occurrences remain one node even when copied for easier layout: Having a unique name, ‘Latin’ copies can be merged again. This “fully node copied” normal form can help to learn the symbolic form, is implemented by Grailog KS Viz, and demoed in the Loan Processor test suite
From Predicate Labels on Hyperarcs to Labelnodes Starting Hyperarcs

General: Graph (hyperarc with rect4vex-shaped labelnode)  
Logic \( \text{rel}(\text{inst}_1, \text{inst}_2, \ldots, \text{inst}_{n-1}, \text{inst}_n) \)

Example: Graph (n=3)  
Logic Invest(/WB, /GE, US$ 3 \cdot 10^9)
From Predicate Labels on Arcs to Labelnodes Starting Binary Hyperarcs

General: Graph (arc)  Logic

Example: Graph

Logic

Trust(Warren Buffett, General Electric)
Arities Smaller than in Binary DLGs: Labelnodes Starting Unary Hyperarcs (cf. Slide on Classes, Concepts, Types)

General: Graph

Logic

Example: Graph

Logic

Frugal(Warren Buffett)
Labelnodes Starting Hyperarcs Enable Second-Order Predicates (e.g. Binary): Hyperarcs with Labelnode Arguments

General: Graph (2-adorned rect-4vex: 2nd order)

Logic

$$binrel2ndord(rel_1, rel_2)$$

Example: Graph

Logic

Antonym(Frugal, Prodigal)
Arities Smaller than in Binary DLGs: Labelnodes for Nullary Hyperarcs (cf. Propositional Logic)

General: Graph Logic

Example: Graph Logic

nullaryrel

nullaryrel() Sunny()
Functions ‘Actively’ Applied to Arguments (Shorthand)

General: Graph (hyperarc with rect4cave-shaped labelnode)

Logic

\[ \text{func}(\text{inst}_1, \text{inst}_2, \ldots, \text{inst}_{n-1}, \text{inst}_n) \]

Example: Graph (n=3)

Logic

Profit (\text{WB, YY, 2011})
Function Apps — Value-Returning: ‘Active’ Call (Long Form)

General: Graph (round2cave-shaped Logic enclosing box)

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

Example: Graph (n=3)

Logic
Profit(WB, YY, 2011)
Function Apps — Value-Returning: Result for Definition of Next Slide

General: Graph

Logic

\( \text{val} \)

Example: Graph

Logic

\( (n=3) \)

US$ 2 \cdot 10^6

US$ 2 \cdot 10^6
Function Apps — Value-Returning: Logic with Equality Definition (1)

General: Graph (ground)

Logic

fun(inst_1, inst_2, ..., inst_{n-1}, inst_n) := val

Example: Graph (n=3)

Logic

Profit(WB, YY, 2011) := US$ 2 \cdot 10^6
Function Apps — Value-Returning: Logic with Equality Definition (2)

General: Graph (inst/var terms) Logic

Example: Graph (n=1) Logic

Double Function Apps ▾ Value-Returning

Logic with Equality Definition (2):

\[
\text{fun} \left( \forall \var_i \right) \text{fun}(\text{term}_1, \text{term}_2, \ldots, \text{term}_n) := \text{val}
\]

Example: Graph (n=1)

\[
\begin{align*}
\text{fun} & \quad \text{term}_1 \quad \text{term}_2 \quad \ldots \quad \text{term}_n \\
\text{Double} & \quad X \quad \text{Mult} & \quad \text{term}_n \quad \text{val} \\
\end{align*}
\]

Double\((X) :=\) Mult\((X, 2)\)
Double Function Sample Call: Rewriting Trace (1)

Call/query of Double instantiates equality definition of previous slide (X=3)
Double Function Sample Call: Rewriting Trace (1’)

Graph

Logic

Mult(3, 2)

Call/query of Mult assumed to be computed by a built-in definition (3 * 2)
Double Function Sample Call: Rewriting Trace (1’’)

More in slides about *Functional-Logic Programming* with (oriented) equations
Function Apps — Value-Denoting: ‘Passive’ Data Construction

General:  Graph (\textit{rect2cave}-shaped enclosing box)

Logic (POSL)
\[
\text{fun}[\text{inst}_1, \text{inst}_2, ..., \text{inst}_{n-1}, \text{inst}_n]
\]

Example:  Graph (n=3)

Logic
Profit[/WB, /YY, 2011]
Predicates: Unary Relations (Classes, Concepts, Types)

General: Graph (class applied to instance node)

Example: Graph

Logic

class

HasInstance

inst₁

Logic

Billionaire

class(inst₁)

Billionaire(Warren Buffett)

Warren Buffett
Negated Predicates: Unary Relations

General: Graph (class *dash* - applied to instance node)

Example: Graph

Logic

\[ \neg \text{Billionaire(} \text{Joe Smallstock)} \]
Graphical Elements: Arrows (2)

- Arrows for special arcs and hyperarcs
  - HasInstance: Connects class, as labelnode, with instance (hyperarc of length 1)
    - As in DRLHs and shown earlier, labelnodes can also be used (instead of labels) for hyperarcs of length > 1
  - SubClassOf: Connects subclass, unlabeled, with superclass (arc, i.e. of length 2)
    - Implies: Hyperarc from premise(s) to conclusion
  - Object-IDentified slots and shelves: Bulleted arcs and hyperarcs
Class Hierarchies (Taxonomies): Subclass Relation

General: Graph (two nodes)

Example: Graph

- SubClassOf
- \( \text{class}_1 \) \( \text{class}_2 \)
- Billionaire \( \equiv \) Rich
Class Hierarchies (Taxonomies): Negated Subclass Relation

General: Graph (two nodes)

Example: Graph

Diagram:

```
class_2
   ^
  /   not SubClassOf
 /    |
class_1
 |
|
Poor
 |
|
Billionaire

(Description)
Logic

class_1 \ notSubClassOf \ class_2

(Description)
Logic
Billionaire \ not \\ Poor
```
Intensional-Class Constructions (Ontologies): Class Intersection

General: Graph (solid+linear node, as for conjunction)

Class Intersection:

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

Example: Graph

\[
\text{Billionaire} \cap \text{Benefactor} \cap \text{Environmentalist}
\]
**Intensional-Class Applications: Class Intersection**

**General:** Graph \((complex\ class\ applied\ to\ instance\ node)\)

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

\[
\text{inst}_1
\]

**Example:** Graph

\[
\text{Billionaire} \quad \text{Benefactor} \quad \text{Environmentalist}
\]

\[
\text{Warren Buffett}
\]
Intensional-Class Constructions (Ontologies): Class Union

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

Example: Graph

- Billionaire
- Benefactor
- Environmentalist

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

\[
\begin{align*}
\text{(Description)} \\
\text{Logic} \\
\text{class}_1 \cup \\
\text{class}_2 \cup \\
\ldots \cup \\
\text{class}_n
\end{align*}
\]
Intensional-Class Applications: Class Union

General: Graph (complex class applied to instance node)

Example: Graph

(Billionaire $\sqcup$ Benefactor $\sqcup$ Environmentalist)

(Warren Buffett)
Intensional Class Constructions (Ontologies): Class Complement

General: Graph
(dashed+linear node, as for negation contains node to be complemented)

Example: Graph

(Description) Logic

Atomic class (shorthand)

Arbitrary class

class

class

¬ class

¬ Billionaire

Billionaire

Billionaire
Class Hierarchies (Taxonomy DAGs): Top and Bottom

General: Top (special node) (Description)
Logic
T
(owl:Thing)

General: Bottom (special node) (Description)
Logic
⊥
(owl:Nothing)
Intensional Class Constructions (Ontologies): Class-Property Restriction—Existential (1)

General: Graph (shorthand) (Description) Logic

Example: Graph (Description) Logic

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
Intensional Class Constructions (Ontologies): Class-Property Restriction—Existential (1*)

General: Graph (normal) (Description) Logic

Example: Graph (Description) Logic

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

\[ \exists \text{binrel} . \text{class} \]

\[ \exists \text{Substance} . \text{Physical} \]
Instance Assertions (Populated Ontologies): Using Restriction for ABox—Existential (1)

General: Graph (shorthand) (xNS-Description) Logic

\[ \exists binrel. class(inst_0) \land class(inst_1) \land binrel(inst_0, inst_1) \]

Example: Graph (xNS-Description) Logic

\[ \exists Substance. Physical(Socrates) \land Physical(P1) \land Substance(Socrates, P1) \]
Instance Assertions (Populated Ontologies): Using Restriction for ABox—Existential (1*)

General: Graph (normal)  
(xNS-Description)  
Logic  
∃binrel.class(inst₀) ∧  
class(inst₁) ∧  
binrel(inst₀, inst₁)

Example: Graph  
(xNS-Description)  
Logic  
∃Substance. Physical(Socrates) ∧  
Physical(P1) ∧  
Substance(Socrates, P1)
Intensional Class Constructions (Ontologies): Class-Property Restriction—Universal (1)

General: Graph (shorthand) (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
Intensional Class Constructions (Ontologies): Class-Property Restriction—Universal (1*)

General: Graph (normal)

Example: Graph

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): Using Restriction for ABox—Universal (1)

General: Graph (shorthand)

Example: Graph

\[ \forall \text{binrel} \]

\[ \text{inst}_0 \rightarrow \text{inst}_1 \rightarrow \ldots \rightarrow \text{inst}_n \]

\[ \text{class} \]

\[ \text{T} \]

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

\[ \forall \text{Substance}\rightarrow \text{Physical} \]

\[ \text{T} \rightarrow \text{Physical} \]

\[ \text{Socrates} \rightarrow \text{P1} \rightarrow \text{P2} \]

\[ \forall \text{Substance} \cdot \text{Physical} \cdot (\text{Socrates}) \land \text{Physical}(\text{P1}) \land \text{Physical}(\text{P2}) \land \text{Substance}(\text{Socrates, P1}) \land \text{Substance}(\text{Socrates, P2}) \]
Instance Assertions (Populated Ontologies): Using Restriction for ABox—Universal (1*)

General: Graph (normal)

\[ \forall \text{binrel} \in T \rightarrow \text{class} \]

\[ \text{inst}_0 \rightarrow \text{inst}_1 \rightarrow \ldots \rightarrow \text{inst}_n \]

Example: Graph

\[ \forall \text{Substance} \rightarrow \text{Physical} \]

\[ \text{Socrates} \rightarrow \text{P1} \rightarrow \text{P2} \]

(xNS-Description) Logic

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

(xNS-Description) Logic

\[ \forall \text{Substance.Physical} \land \text{(Socrates)} \land \text{Physical(P1)} \land \text{Physical(P2)} \land \text{Substance(Socrates, P1)} \land \text{Substance(Socrates, P2)} \]
Existential vs. Universal Restriction

(Physical/Mental Assumed Disjoint: Can Be Explicated via Bottom Intersection)

Example: Graph

\[ \exists \text{Substance}. \text{Physical}(\text{Socrates}) \land \text{Physical}(\text{P1}) \land \text{Mental}(\text{P3}) \land \text{Substance}(\text{Socrates, P1}) \land \text{Substance}(\text{Socrates, P3}) \]

Example: Graph

\[ \forall \text{Substance}. \text{Physical}(\text{Socrates}) \land \text{Physical}(\text{P1}) \land \text{Mental}(\text{P3}) \land \text{Substance}(\text{Socrates, P1}) \land \text{Substance}(\text{Socrates, P3}) \]
LuckyParent Example (1)

LuckyParent ≡ Person \( \exists \text{Spouse} \). Person \( \forall \text{Child}. (\neg \text{Poor} \lor \exists \text{Child}. \text{Doctor}) \)
LuckyParent Example (1*)

\[ \text{LuckyParent} \equiv \text{Person} \exists \text{Spouse}. \text{Person} \forall \text{Child}. (\neg \text{Poor} \cup \exists \text{Child}. \text{Doctor}) \]
LuckyParent Example (1**)
LuckyParent Example (1**)
Object-Centered Logic: Grouping Binary Relations Around Instance

General:  Graph  \(\text{class}(\text{inst}_0)\) \&  \(\text{binrel}_1(\text{inst}_0, \text{inst}_1)\) \&  \(\text{binrel}_n(\text{inst}_0, \text{inst}_n)\)  

Example:  Graph  \(\text{Philosopher}(\text{Socrates})\) \&  \(\text{Substance}(\text{Socrates, P1})\) \&  \(\text{Teaching}(\text{Socrates, T1})\)
RDF-Triple (‘Subject’-Centered) Logic: Grouping Properties Around Instance

**General:**
Graph
\[(\text{inst}_0\text{-centered})\]

\[
\text{class} \to \text{inst}_0 \overset{\text{property}_1}{\longrightarrow} \cdots \overset{\text{property}_n}{\longrightarrow} \text{inst}_1 \overset{\text{inst}_n}{\longrightarrow} \]

**Example:**
Graph
\[(\text{Socrates-centered})\]

\[
\text{Philosopher} \to \text{Socrates} \overset{\text{Substance}}{\longrightarrow} \text{P1} \overset{\text{Teaching}}{\longrightarrow} \text{T1} \]

\[
\{(\text{inst}_0, \text{rdf:type, class}), (\text{inst}_0, \text{property}_1, \text{inst}_1), \cdots (\text{inst}_0, \text{property}_n, \text{inst}_n)\}
\]

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

General: Graph (bulleted arcs)

Example: Graph

(PSOA Frame) Logic

(inst₀[class]  
slot₁→inst₁;  
...  
slotₙ→instₙ)

(inst₀ ∈ class,  
slot₁ = inst₁,  
...  
slotₙ = instₙ)

(PSOA Frame) Logic

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

General: Graph
(bulleted hyperarc)

Example: Graph

(PSOA Shelf) Logic

inst_0#class( inst_1, ..., inst_m)

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

General: Graph

(PSOA Positional-Slotted-Term) Logic

Example: Graph

(PSOA Positional-Slotted-Term) Logic

Socrates#Philosopher(
c. 469 BC, 399 BC;
Substance->P1;
Teaching->T1)
Term and Formula Description:
Associating Slots with Complex Node

Complex Node (e.g. Roundangle) having Outward Slots

Complex Node (e.g. Roundangle) having Inward Slots

Slots visible in outer context. Example: Like for elementary node

Slots visible in inner context. Example: Later slide with closure slot
Rules: Relations Imply Relations (1)

General: Graph (ground, shorthand)

Logic

rel_1(inst_{1,1}, inst_{1,2}, ..., inst_{1,n_1}) \Rightarrow
rel_2(inst_{2,1}, inst_{2,2}, ..., inst_{2,n_2})

Example: Graph

Logic

Invest(/WB, /GE, US$ 3 \cdot 10^9) \Rightarrow
Invest(/JS, /VW, US$ 5 \cdot 10^3)
Rules: Relations Imply Relations (1*)

General: Graph (ground, normal)

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

\[ \text{Invest(}/\text{WB, }/\text{GE, US$ 3 \cdot 10^9}/) \Rightarrow \text{Invest(}/\text{JS, }/\text{VW, US$ 5 \cdot 10^3}/) \]
Rules: Relations Imply Relations (2)

General: Graph (non-ground, where ‘Implies’ arrow creates universal closure)

Logic

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

Example: Graph

Logic

$$(\forall X, Y, A, U, V, B) \quad \text{Invest}(X, Y, A) \Rightarrow \text{Invest}(U, V, B)$$
Rules: Relations Imply Relations (3)

General: Graph (inst/var terms)

Logic

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

Example: Graph

Logic

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

General: Graph (shorthand)

Logic

\((\forall \text{var}_{i,j})\)

\(\text{rel}_1(\text{term}_{1,1}, \text{term}_{1,2}, \ldots, \text{term}_{1,n_1}) \land \ldots \land \text{rel}_2(\text{term}_{2,1}, \text{term}_{2,2}, \ldots, \text{term}_{2,n_2}) \Rightarrow \text{rel}_3(\text{term}_{3,1}, \text{term}_{3,2}, \ldots, \text{term}_{3,n_3})\)

Example: Graph

Logic

\((\forall Y, A)\)

Invest(\('/\text{WB}, Y, A') \land \text{Trust}(\('/\text{JS}, Y') \Rightarrow \text{Invest}(\('/\text{JS}, Y, \text{US}\$5 \cdot 10^3')\)\)
Rules: Conjuncts Imply Relations (1*)

General: Graph (prenormal)

Logic

\[(\forall \text{var}_{i,j})\]

\[\text{rel}_1(\text{term}_{1,1}, \text{term}_{1,2}, \ldots, \text{term}_{1,n_1}) \land \]

\[\text{rel}_2(\text{term}_{2,1}, \text{term}_{2,2}, \ldots, \text{term}_{2,n_2}) \Rightarrow \]

\[\text{rel}_3(\text{term}_{3,1}, \text{term}_{3,2}, \ldots, \text{term}_{3,n_3})\]

Example: Graph

Logic

\[(\forall Y, A)\]

\[\text{Invest}(/WB, Y, A) \land \]

\[\text{Trust}(/JS, Y) \Rightarrow \]

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

General: Graph (normal)

Logic

\[ (\forall \text{var}_{i,j}) (\text{rel}_1(\text{term}_{1,1}, \text{term}_{1,2}, \ldots, \text{term}_{1,n}) \land \text{rel}_2(\text{term}_{2,1}, \text{term}_{2,2}, \ldots, \text{term}_{2,n}) \Rightarrow \text{rel}_3(\text{term}_{3,1}, \text{term}_{3,2}, \ldots, \text{term}_{3,n})) \]

Example: Graph

Logic

\[ (\forall Y, A) (\text{Invest}(/\text{WB}, Y, A) \land \text{Trust}(/\text{JS}, Y) \Rightarrow \text{Invest}(/\text{JS}, Y, \text{US}\$ 5 \cdot 10^3)) \]
Rules: Conjuncts Imply Relations (2)

Example: RuleML/XML

<Implies closure="universal">
  <And>
    <Atom>
      <Rel>Invest</Rel>
      <Ind unique="no">WB</Ind>
      <Var>Y</Var>
      <Var>A</Var>
    </Atom>
    <Atom>
      <Rel>Trust</Rel>
      <Ind unique="no">JS</Ind>
      <Var>Y</Var>
    </Atom>
  </And>
  <Atom>
    <Rel>Invest</Rel>
    <Ind unique="no">JS</Ind>
    <Var>Y</Var>
    <Data>US$ 5 \cdot 10^3</Data>
  </Atom>
</Implies>

Logic

(\forall \ Y, \ A)
(Invest(WB, Y, A) \land
Trust(JS, Y) \implies
Invest(JS, Y, US$ 5 \cdot 10^3))

Proposing an attribute unique
with value "no" for NNS,
and "yes" for UNS as the default
Implication-Defined Predicate Odd: RuleML/XML Serialization

Datalog RuleML/XML

<Implies closure="universal">
  <And>
    <Atom>
      <Rel>Greater</Rel>
      <Var>X</Var>
      <Data>2</Data>
    </Atom>
    <Atom>
      <Rel>Prime</Rel>
      <Var>X</Var>
    </Atom>
  </And>
  <Atom>
    <Rel>Odd</Rel>
    <Var>X</Var>
  </Atom>
</Implies>

Logic

(∀ X)
Greater(X, 2) ∧ Prime(X) ⇒ Odd(X)

Graph (prenormal)

Graph ‘⇒’ arrow normalizes to RuleML-like closure="universal"
Relations Equivalent to Relations

General: Graph (inst/var terms)

Example: Graph

Logic

$$\forall var_{i,j}$$

$$(\forall var_{i,j})$$

$$rel_1(term_{1,1}, term_{1,2}, ..., term_{1,n_1}) \iff$$

$$rel_2(term_{2,1}, term_{2,2}, ..., term_{2,n_2})$$

Example: Graph

Logic

$$(\forall X, Y, A)$$

Transfer($X, Y, A$) $\iff$ 

Receive($Y, X, A$)
Equivalence-Defined Predicate Even: RuleML/XML Serialization

**FOL RuleML/XML**

\[
\text{even(X)} : \iff \text{divisible}(X, 2)
\]

**Graph (prenormal)**

```
<Equivalent oriented="yes" closure="universal">
  <Atom>
    <Rel>Even</Rel>
    <Var>X</Var>
  </Atom>
  <Atom>
    <Rel>Divisible</Rel>
    <Var>X</Var>
    <Data>2</Data>
  </Atom>
</Equivalent>
```

Graph ‘\(\iff\)’ arrow normalizes to RuleML-like closure="universal"
Equality-Defined Function Double: RuleML/XML Serialization

**Functional RuleML/XML**

```
<Equal oriented="yes" closure="universal">
  <Expr>
    <Fun per="value">Double</Fun>
    <Var>X</Var>
  </Expr>
  <Expr>
    <Fun per="value">Mult</Fun>
    <Var>X</Var>
    <Data>2</Data>
  </Expr>
</Equal>
```

**Logic**

\( (\forall X) \)  
\( \text{Double}(X) := \text{Mult}(X, 2) \)

**Graph (prenormal)**

Graph ‘:=’ arrow normalizes to RuleML-like closure="universal"
Positional-Slotted-Term Logic: Rule-defined Anonymous Family Frame (Visualized from IJCAI-2011 Presentation)

Example: Graph

```
married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

married

ma
Positional-Slotted-Term Logic: Ground Facts, incl. Deduced Frame, Model Family Semantics

Example: Graph

Previous slide’s existential variable ?1 in rule head becomes new OID constant o in frame fact, deduced from relational facts

For reference implementation of PSOA querying see PSOATransRun
Positional-Slotted-Term Logic: Conversely, Given Facts, Rule Can Be Inductively Learned

Example: Graph

Abstracting OID constants $o_1, \ldots, on$ to regain existential variable $?1$ of previous rule, now induced from matching relational and frame facts: Knowledge from data
Modally Embedded Propositions

General: Graph (Modal) Logic
(complex \textit{snip2vex} node, used to ‘encapsulate’ what another agent believes, wants, etc.)

Example: Graph (Modal) Logic

\[ \text{believe}_{agent}(\text{graph}) \]

\[ \text{believe}_{GE}(\text{Invest}(/\text{WB}, /\text{GE}, \text{US$ \ 4 \cdot 10^9})) \]

Can be serialized in the evolving Modal RuleML/XML
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.
Graphical Elements: Line Styles

- Solid lines (boxes & links): Positive
- Dashed lines (boxes & links): Negative
- Wavy lines (boxes): Disjunctive
- Light lines (unlabeled arrows): HasInstance
- Light lines (unlabeled undirected links): SameIndividual
- Heavy lines (unlabeled arrows): SubClassOf
- Heavy lines (unlabeled undirected links): EquivalentClasses
- Double lines (unlabeled arrows): Implies
- Double lines (unlabeled double-headed arrows): Equivalence
- Double lines (unlabeled undirected links): Equality
- Colon tails (unlabeled links): TailDefinedByHead
Orthogonal Graphical Features — Axes of Grailog Systematics

- Box axes:
  - Corners: *pointed* vs. *snipped* vs. *rounded*
    - To *quote/copy* vs. *reify/instantiate* vs. *evaluate* contents
      (cf. Lisp, Prolog, Relfun, Hilog, RIF, CL, and IKL)
  - Shapes (rectangle-derived): composed from sides that are straight vs. *concave* vs. *convex*
    - For neutral vs. *function* vs. *relation* contents
  - Contents: elementary vs. complex nodes

- Arrow axes:
  - Shafts: single vs. double
  - Heads: triangular vs. diamond
  - Tails: plain vs. bulleted vs. colonized

- Box & Arrow (line-style) axes:
  solid vs. dashed, linear vs. (box only) wavy
Mnemonics for Basic Box Shapes

“skateboard halfpipe”

“database barrel”

concave

function

contents

convex

relation

contents
Graphical Elements: Box Systematics — Axes of Shapes and Corners

<table>
<thead>
<tr>
<th>Shape:</th>
<th>Corner:</th>
<th>Per  ... Copy</th>
<th>... Instantiation</th>
<th>... Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral</td>
<td>-angle</td>
<td>Rect-</td>
<td>Snip-</td>
<td>Round-</td>
</tr>
<tr>
<td>Individual</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Function Application)</td>
<td>-2cave</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-4cave</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Relation Application)</td>
<td>-2vex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(incl. Class)</td>
<td>-4vex</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graphical Elements: Boxes — Function/Relation-Neutral Shape of Angles Varied w.r.t. Corner Dimension

- **Rectangle**: Neutral ‘per copy’ nodes *quote* their contents

\[
X=3: \quad \text{Mult} \xrightarrow{X} \text{2} \quad \text{Mult} \xrightarrow{X} \text{2}
\]

- **Snipangle** (octagon): Neutral ‘per instantiation’ nodes *dereference* contained variables to values from context

\[
X=3: \quad \text{Mult} \xrightarrow{X} \text{2} \quad \text{Mult} \xrightarrow{3} \text{2}
\]

- **Roundangle** (rounded angles): Neutral ‘per value’ nodes *evaluate* their contents through instantiation of variables and activation of function/relation applications

\[
X=3: \quad \text{Mult} \xrightarrow{X} \text{2} \quad \text{Mult} \xrightarrow{X} \text{2} \quad 6
\]

Assuming Mult built-in function
Graphical Elements: Boxes — Concave

- **Rect2cave** (rectangle with 2 concave - top/bottom - sides): Elementary nodes for individuals (instances). Complex nodes for quoted instance-denoting terms (constructor-function applications)

- **Snip2cave** (snipped): Elementary nodes for variables. Complex nodes for instantiated (reified) function applications

- **Round2cave** (rounded): Complex nodes for evaluated built-in or equation-defined function applications

- **Rect4cave** (4 concave sides): Elementary nodes for fct’s. Complex nodes for quoted functional (function-denoting) terms

- **Snip4cave**: Complex nodes for instantiated funct’l terms

- **Round4cave**: Complex nodes for evaluated functional applications (active, function-returning applications)
Graphical Elements: Boxes — Convex

- **Rect2vex** (rectangle with 2 convex - top/bottom - sides):
  - Elementary nodes for truth constants (true, false, unknown).
  - Complex nodes for quoted truth-denoting propositions (embedded relation applications)

- **Snip2vex**: Complex nodes for instantiated (reified) relation applications

- **Round2vex**: Complex nodes for evaluated relation applications (e.g. as atomic formulas) and for connective uses

- **Rect4vex**: Elementary nodes for relations, e.g. unary ones (classes). Complex nodes for quoted relational (relation-denoting) terms

- **Snip4vex**: Complex nodes for instantiated relat’l terms

- **Round4vex** (oval): Complex nodes for evaluated relat’l applications (active, relation-returning applications)
Conclusions (1)

- Grailog 1.0 incorporates feedback on earlier versions
- Graphical elements for novel box & arrow systematics using orthogonal graphical features
  - Leaving color (except for variables and IRIs) for other purposes, e.g. highlighting subgraphs (for retrieval and inference)
- Introducing **Unique** vs. **Non-unique Name Specification**
- Focus on *mapping* to a family of logics as in RuleML
- Use cases from *cognition* to *technology* to *business*
- *Processing* of earlier Grailog-like DRLHs studied in Lisp, FIT, and Relfun
Conclusions (2)

• **Symbolic-to-visual** translators started as **Semantic Web Techniques Fall 2012 Projects**:
  - **Team 1** A Grailog Visualizer for Datalog RuleML via XSLT 2.0 Translation to SVG by Sven Schmidt and Martin Koch: An **Int'l Rule Challenge 2013** paper & demo introduced Grailog KS Viz
  - **Team 8** Visualizing SWRL’s Unary/Binary Datalog RuleML in Grailog by Bo Yan, Junyan Zhang, and Ismail Akbari: A **Canadian Semantic Web Symposium 2013** paper gave an overview

• Grailog invites feature **choice** or **combination**
  - E.g. **n-ary hyperarcs** or **n-slot frames** or **both**

• Grailog Initiative on open standardization calls for further feedback for future 1.x versions
Future Work (1)

- Refine/extend Grailog, e.g. along with API4KB effort
  - Compare with other graph formalisms, e.g. Conceptual Graphs (http://conceptualstructures.org) and CoGui tool
  - Define mappings to/fro UML structure diagrams + OCL, adopting UML behavior diagrams (http://www.uml.org)

- Implement further tools, e.g. as use case for (Functional) RuleML (http://ruleml.org/fun) engines
  - More mappings between graphs, logic, and RuleML/XML:
    - Grailog *generators*: Further symbolic-to-visual mappings
    - Grailog *parsers*: Initial visual-to-symbolic mappings
  - Graph indexing & querying (cf. http://www.hypergraphdb.org)
  - Graph transformations (normal form, typing homomorphism, merge, ...)
  - Advanced graph-theoretical operations (e.g., path tracing)
  - Exploit Grailog parallelism in implementation
Future Work (2)

- Develop a Grailog structure editor, e.g. supporting:
  - Auto-specialize of neutral application boxes (angles) to function apps (2caves) or relation apps (2vexes), depending on contents
  - Auto-specialize of neutral operator boxes (angles) to functions (4caves) or relations (4vexes), depending on context

- Synergize with CmapTools/COE, Protégé visualization plug-ins such as Jambalaya/OntoGraf and OWLViz for OWL ontologies and Axiomé for SWRL rules, etc.

- Proceed from the 2-dimensional (planar) Grailog to a 3-dimensional (spatial) one
  - Utilize advantages of crossing-free layout, spatial shortcuts, and analogical representation of 3D worlds
  - Mitigate disadvantages of occlusion and of harder spatial orientation and navigation

- Consider the 4\textsuperscript{th} (temporal) dimension of animations to visualize logical inferences, graph processing, etc.