From Data to Knowledge through Grailog Visualization


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

Computational Logic Seminar – MUGS
Stanford Logic Group, Stanford University, CA
11 February 2015

Editorial Update: 14 February 2015
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
Decision CAMP, eBay Town Hall, San Jose, CA, 4-6 November 2013
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, AISP/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): 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/on the loop of 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
- Symbolic
- Presentation
- Serialization

RuleML/Grailog RuleML/POSL RuleML/XML
Three RuleML Syntaxes (2)

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

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

**Visualization .INSTANCE 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
Grailog’s Integrated Representations and API4KB

• Grailog covers the main *data & knowledge* representation paradigms, needed in API4KB:
  – Integrated Directed Labeled Graph (e.g. RDF) and Relational (Datalog-fact-like) *data*
  – Integrated Ontology (e.g. RDFS and description-logic) and Rule (Horn & general-logic) *knowledge*

• Besides mapping [Grailog to/fro RuleML](#), RDF and UML+OCL can be targeted, with uniform access to be provided by [API4KB](#)

• An API can be (initially) designed and tested with a human in the loop much like a GUI
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?
- **Central slides:**
  How are these graphs *mapped* (invertibly) to logic, thus specifying Grailog as a ‘GUI’ for knowledge?
- **Later slides:**
  What is the *systematics* of Grailog features?
- **Final slides:**
  How is RuleML *transformed* to Grailog in SVG?
Grailog Principles

• Graphs should make it easier for humans to read and write logic constructs via a 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 be compositional, e.g. allowing refinements for logic constructs: Description Logic constructors, F-logic frames, PSOA RuleML terms, ...

• 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)
English

Chinese

understand

ruleset

question

text

reply

Searle

Wang

Searle-reply_i

Wang-reply_i

SubClassOf

HasInstance

distinguishable

negation

hasLanguage

understand

lang

apply

to

use

with

for

for

apply

with

for

for

distinguishable

Classes with relationships

Instances with relationships
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: \(x\text{NS}\), with \(x = U\) or \(N\)
Different Unique Names — Different Referents

Names:

Venus

Mars

Referents:

V

m
Equated Different Unique Names — No Referents

Names:

Venus ——— Mars

Referents:
Different Non-Unique Names — Different Referents

Names: Morningstar  Eveningstar

Referents: V  W
Equated Different Non-Unique Names (“Synonym”) — Same Referent

Names:
- Morningstar
- Eveningstar

Referents:
- V

Merge
Equated Different Non-Unique Names ("Synonym") and Same Unique Name — Same Referent

Names:
- Morningstar
- Eveningstar
- Venus

Referents:
- V

Merge
Same Non-Unique Name Different Unique Names ("Homonym") — No Referents

Names: 

Venus

Eveningstar

Mars

Referents: 

Eveningstar

Unique Referent Principle (URP)
Instances: Individual Constants with Unique Name Specifications

General: Graph (node) \[\text{mapping} \rightarrow\] Logic

**unique**

Examples: Graph

**unique**

Warren Buffett

US$ $3 \cdot 10^9$

Warren Buffett

General Electric

US$ $3 \cdot 10^9$
Instances: Individual Constants with Non-unique Name Specifications

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

Examples: Graph

| WB | GE | US$ 3B |

/non-unique

| /WB | /GE | /US$ 3B |

/non-unique
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

Warren Buffett Trust General Electric

binrel(inst₁, inst₂)

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

General: Graph (labeled arc) Logic

Example: Graph Logic

\[ \text{binrel} \]

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

\[ \text{Trust} \]

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

General: Graph (unlabeled undirected arc) Logic (with equality)

Example: Graph Logic

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)

Example: Graph

Logic (with oriented equality)

Example: Graph

Logic

GE := General Electric
Ground Equality: Defining Symbolic Constants as IRIs

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

Example: Graph Logic

```
inst := IRI

GenElec := http://www.ge.com/

/GenElec := 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

Example: Graph Logic

Joe Smallstock $\rightarrow$ General Electric $\rightarrow$ Trust (Joe Smallstock, General Electric)
Negated Predicates: Binary Relations (Long Form)

General:  
Graph (dashed box)  
Logic

Example:  
Graph  
Logic

Joe Smallstock  
Trust  
General Electric

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

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

General: Graph (dashed unlabeled undirected arc)

Example: Graph

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

General: Graph (*dashed box*, *unlabeled* *undirected arc*)

Example: Graph

Logic (with equality)

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

\[ \neg (\text{Joe Smallstock} = \text{Warren Buffett}) \]
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

rel(inst_1, inst_2, ..., inst_{n-1}, inst_n)

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

General: Graph (*dashed* hyperarc) Logic

Example: Graph (n=3)

Logic

![Diagram](image-url)
Negated Predicates: n-ary Relations (Long Form)

**General:** Graph (*dashed* box)

- **Logic:** $\neg (\text{rel}(\text{inst}_1, \text{inst}_2, \ldots, \text{inst}_{n-1}, \text{inst}_n))$

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

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

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

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

Logic

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

Example: Graph (2 hyperarcs)

\[
\text{Invest} \rightarrow \text{WB} \rightarrow \text{GE} \rightarrow \text{US$3 \cdot 10^9$} \]
\[
\text{Invest} \rightarrow \text{JS} \rightarrow \text{VW} \rightarrow \text{US$2 \cdot 10^4$} \]

Logic

Invest(\text{/WB, /GE, US$3 \cdot 10^9$}) \land \\
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

\((\text{rel}_1(\text{inst}_{1,1}, \text{inst}_{1,2}, ..., \text{inst}_{1,n}) \land \text{inst}_{1,2} \land \text{inst}_{1,n}) \land \text{rel}_m(\text{inst}_{m,1}, \text{inst}_{m,2}, ..., \text{inst}_{m,n}))\)

Example: Graph

Logic

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

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

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

Example: Graph Logic

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

General: Graph (dashed+linear) Logic

Example: Graph Logic

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

General: Graph (solid+wavy)

Logic

\[
(rel_1(inst_{1,1}, inst_{1,2}, ..., inst_{1,n_1}) \lor \ldots \lor rel_m(inst_{m,1}, inst_{m,2}, ..., inst_{m,n_m}))
\]

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-Occurrence in a Not’s Or Node

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

Logic:

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

Example: Graph

Logic:

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

General: Graph (dashed+wavy) Logic

Example: Graph Logic

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

\[ \neg \left( \text{Invest}(/\text{WB}, /\text{GE}, \text{US}\$\ 3 \cdot 10^9) \lor \text{Invest}(/\text{JS}, /\text{VW}, \text{US}\$\ 2 \cdot 10^4) \right) \]
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
- Latin → Paul
- Latin → Kate

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

- John → Show → Latin
- John → Teach → Latin
- Mary → Show → Latin
- Mary → Teach → Latin
- Latin → Paul
- Latin → Kate
The hyperarc for, e.g., ternary Show(John,Latin,Kate) can be seen as the path composing the 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)
\land
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)

Example: Graph (n=3)

Logic

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

(Shorthand)

(Normal Form)

Invest(WB, GE, US$ 3·10⁹)

Invest(WB, GE, US$ 3·10⁹)
From Predicate Labels on Arcs to Labelnodes Starting Binary Hyperarcs

General: Graph (arc)

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

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)

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

General: Graph (hyperarc with rect4cave-shaped labelnode)

Logic

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

Example: Graph (n=3)

Logic

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

General: Graph (round2cave-shaped Logic enclosing box)

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

Example: Graph (n=3)

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

General:  Graph

Logic

Example:  Graph
(n=3)

Logic

US$ 2 \cdot 10^6

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

General: Graph (ground)

Logic

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

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

(∀ var_i)

fun(term_1, term_2, ..., term_n) := val

Example: Graph (n=1)  Logic

(∀ X)

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

Graph

Logic

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

Graph

Logic

Mult

\[
\text{Mult}(3, 2)
\]

\[
\text{Call/query of Mult assumed to be computed by a built-in definition (3 \times 2)}
\]
Double Function Sample Call: Rewriting Trace (1’’)

Graph

Logic

6

More in slides about Functional-Logic Programming with (oriented) equations, which can be visualized and animated as above, for students from Code Kids to graduate-level CS courses (cf. eLearning in “Conclusions”)
Function Apps — Value-Denoting: ‘Passive’ Data Construction

General: Graph (rect2cave-shaped enclosing box) Logic (POSL)

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

Example: Graph

\[ \text{Profit} \quad \text{WB} \quad \text{YY} \quad 2011 \]

\[ \text{fun} \quad \text{inst}_1 \quad \text{inst}_2 \quad \ldots \quad \text{inst}_{n-1} \quad \text{inst}_n \]
Predicates: Unary Relations (Classes, Concepts, Types)

General: Graph (class applied to instance node)

Example: Graph

Logic

Billionaire

Warren Buffett

HasInstance

class

class(inst_1)

Billionaire(Warren Buffett)
Negated Predicates: Unary Relations

General: Graph (class *dash*-applied Logic 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) (Description)
Logic

Example: Graph (Description)
Logic
Billionaire \sqsubseteq \text{Rich}

\begin{center}
\begin{tikzpicture}
  \node (class1) at (0,0) {class_1};
  \node (class2) at (0,2) {class_2};
  \node (rich) at (0,-2) {Rich};
  \node (billionaire) at (0,-4) {Billionaire};

  \draw[->,thick] (class1) -- (class2) node [midway, above] {SubClassOf};
  \draw[->,thick] (rich) -- (billionaire) node [midway, above] {};

  \node at (1,1.5) {};
  \node at (-1,1.5) {};
\end{tikzpicture}
\end{center}
Class Hierarchies (Taxonomies): Negated Subclass Relation

General: Graph (two nodes)

Example: Graph

---

$\text{class}_1 \not\subseteq \text{class}_2$

$\text{Poor} \not\subseteq \text{Billionaire}$
Intensional-Class Constructions (Ontologies): Class Intersection

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

Example: Graph

(Billionaire) ∩ (Benefactor) ∩ (Environmentalist)
Intensional-Class Applications: Class Intersection

General: Graph \((\text{complex class applied to instance node})\)

Example: Graph

Logic \((xNS\text{-Description})\)

\((\text{class}_1 \sqcap \text{class}_2 \sqcap \ldots \sqcap \text{class}_n)(\text{inst}_1)\)
Intensional-Class Constructions (Ontologies): Class Union

General: Graph (solid+wavy node, as for disjunction)

Example: Graph

(Billionaire ∪ Benefactor ∪ Environmentalist)
Intensional-Class Applications: Class Union

General: Graph (complex class applied to instance node)

Example: Graph

(xNS-Description) Logic

(Billionaire ∪ Benefactor ∪ Environmentalist) (Warren Buffett)
Intensional Class Constructions (Ontologies): Class Complement

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

Example: Graph

Arbitrary class

Atomic class (shorthand)

$\neg$ class

$\neg$ Billionaire
Class Hierarchies (Taxonomy DAGs): Top and Bottom

General: Top (*special node*)

![Diagram of a top node in a class hierarchy]

General: Bottom (*special node*)

![Diagram of a bottom node in a class hierarchy]
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.
Instance Assertions (Populated Ontologies): Using Restriction for ABox—Existential (1)

General: Graph (shorthand) (xNS-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 (xNS-Description)
Logic
\[ \exists \text{Substance.Physical}(\text{Socrates}) \land \text{Physical}(\text{P1}) \land \text{Substance(Substance(Socrates, P1))} \]
Instance Assertions (Populated Ontologies): Using Restriction for ABox—Existential (1*)

General: Graph (normal) (xNS-Description)

```
\exists \text{binrel} \\
\top \rightarrow \text{class} \\
\text{inst}_0 \rightarrow \text{binrel} \rightarrow \text{inst}_1 \\
```

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 (xNS-Description)

```
\exists \text{Substance} \\
\top \rightarrow \text{Physical} \\
\text{Socrates} \rightarrow \text{Substance} \rightarrow \text{P1} \\
```

Logic

```
\exists \text{Substance.Physical (Socrates)} \land \\
\text{Physical(P1)} \land \\
\text{Substance(Socrates, P1)} \\
```
Intensional Class Constructions (Ontologies): Class-Property Restriction—Universal (1)

General: Graph (shorthand) (Description) Logic

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

Example: Graph (Description) Logic

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

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) (Description)

Example: Graph (Description)

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

(xNS-Description) Logic

∀ substance.Physical (Socrates) ∧
Physical(P1) ∧
Physical(P2) ∧
Substance(Socrates, P1) ∧
Substance(Socrates, P2)
Instance Assertions (Populated Ontologies): Using Restriction for ABox—Universal (1*)

General: Graph (normal)

(\( xNS\)-Description)

Logic

\( \forall binrel.class(inst_0) \land \)

\( \land class(inst_1) \land \)

\( \land \ldots class(inst_n) \land \)

\( \land binrel(inst_0, inst_1) \land \)

\( \land \ldots binrel(inst_0, inst_n) \)

Example: Graph

(\( xNS\)-Description)

Logic

\( \forall Substance\).Physical (Socrates) \land

Physical(P1) \land

Physical(P2) \land

Substance(Socrates, P1) \land

Substance(Socrates, P2)
Existential vs. Universal Restriction
(Physical/Mental Assumed Disjoint: Can Be Explicated via Bottom Intersection)

Example: Graph

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

Example: Graph

Logic
\( \forall \text{Substance}. \text{Physical} \quad \text{(Socrates)} \land \text{Physical(P1)} \land \text{Mental(P3)} \land \text{Substance(Socrates, P1)} \land \text{Substance(Socrates, P3)} \)
LuckyParent EquivalentClasses (1)

LuckyParent ≡ Person ∃ Spouse. Person ∃ ∀ Child. (¬Poor ∪ ∃ Child. Doctor)
LuckyParent Example (1*)

LuckyParent ≡ Person ≤∧∃Spouse. Person ≤∧∀Child. (¬Poor ≤∪∃Child. Doctor)
LuckyParent Example (1**)
LuckyParent Example (1**)
Object-Centered Logic:
Grouping Binary Relations Around Instance

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

\begin{align*}
\text{class} & \quad \text{inst}_0 \quad \text{binrel}_1 \quad \text{inst}_1 \\
\text{inst}_0 \quad \ldots \quad \text{binrel}_n \quad \text{inst}_n \\
\end{align*}

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

\begin{align*}
\text{Philosopher} & \quad \text{Socrates} \\
\text{Substance} & \quad \text{P1} \\
\text{Teaching} & \quad \text{T1} \\
\end{align*}

\begin{align*}
\text{(Object-Centered) Logic} \\
\text{class}(\text{inst}_0) & \land \\
\text{binrel}_1(\text{inst}_0, \text{inst}_1) & \land \\
\ldots & \\
\text{binrel}_n(\text{inst}_0, \text{inst}_n) & \\
\end{align*}

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

**General:**

Graph

\(\text{inst}_0\)-centered

\(\text{class}\)

\(\text{property}_1\)

\(\text{inst}_1\)

\(\cdots\)

\(\text{property}_n\)

\(\text{inst}_n\)

\(\text{inst}_0 \text{ } \text{property}_1 \text{ } \text{inst}_1 \text{ } \cdots \text{ } \text{property}_n \text{ } \text{inst}_n\)

\(\text{inst}_0 \text{ } \text{rdf:type} \text{ } \text{class}\)

\(\text{inst}_0 \text{ } \text{property}_1 \text{ } \text{inst}_1 \text{ } \cdots \text{ } \text{property}_n \text{ } \text{inst}_n\)

\((\text{inst}_0, \text{property}_1, \text{inst}_1),\)

\((\text{inst}_0, \text{property}_n, \text{inst}_n)\)}

**Example:**

Graph

(Socrates-centered)

Philosopher

Socrates

Substance

P1

Teaching

T1

(Socrates, rdf:type, Philosopher)

(Socrates, Substance, P1)

(Socrates, Teaching, T1)
Logic of Frames (‘Records’): Associating Slots with OID-Distinguished Instance

General: Graph (bulleted arcs)

Example: Graph

(PSOA Frame) Logic

inst₀#class( inst₁, ...
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

Logic

(PSOA Shelf)

inst₀#class(inst₁, ..., instₘ)

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

**General:**
Graph

```
<table>
<thead>
<tr>
<th>class</th>
<th>inst'_1</th>
<th>...</th>
<th>inst'_m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>slot_1</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>inst_0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>slot_n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Example:**
Graph

```
<table>
<thead>
<tr>
<th>PhD</th>
<th>inst_0</th>
<th>...</th>
<th>inst_m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>slot_1</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inst_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>slot_n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
Socrates#Philosopher(c. 469 BC, 399 BC; Substance->P1; Teaching->T1)
```

(PSOA Positional-Slotted-Term) Logic

```
inst_0#class(
inst'_1, ..., inst'_m;
slot_1->inst_1;
...;
slot_n->inst_n)
```

```
399 BC
```

```
P1
```

```
T1
```
Term and Formula Description: Associating Slots with Complex Node

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

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

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

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

General: Graph (ground, shorthand)

<table>
<thead>
<tr>
<th>inst_{1,1}</th>
<th>rel_1</th>
<th>inst_{1,2}</th>
<th>...</th>
<th>inst_{1,n^1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst_{2,1}</td>
<td>rel_2</td>
<td>inst_{2,2}</td>
<td>...</td>
<td>inst_{2,n^2}</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>WB</th>
<th>Invest</th>
<th>GE</th>
</tr>
</thead>
<tbody>
<tr>
<td>JS</td>
<td>Invest</td>
<td>VW</td>
</tr>
</tbody>
</table>

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

\[ \text{rel}_1(\text{inst}_{1,1}, \text{inst}_{1,2}, ..., \text{inst}_{1,n_1}) \Rightarrow \text{rel}_2(\text{inst}_{2,1}, \text{inst}_{2,2}, ..., \text{inst}_{2,n_2}) \]

Example: Graph

Logic

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

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

Logic

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

rel_1(var_{1,1}, var_{1,2}, ..., var_{1,n_1}) \implies rel_2(var_{2,1}, var_{2,2}, ..., var_{2,n_2})

Example: Graph

Logic

\((\forall X, Y, A, U, V, B)\)

Invest(X, Y, A) \implies Invest(U, V, B)
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

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

General: Graph (shorthand)

Logic

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

\(\text{rel}_1(\text{term}_{1,1}, \text{term}_{1,2}, \ldots, \text{term}_{1,n^1}) \land \text{rel}_2(\text{term}_{2,1}, \text{term}_{2,2}, \ldots, \text{term}_{2,n^2}) \Rightarrow \text{rel}_3(\text{term}_{3,1}, \text{term}_{3,2}, \ldots, \text{term}_{3,n^3})\)

Example: Graph

Logic

\((\forall Y, A)\)

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

General:  Graph (prenormal)

Logic

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

$\text{rel}_1(\text{term}_{1,1}, \text{term}_{1,2}, ..., \text{term}_{1,n^1}) \wedge$

$\text{rel}_2(\text{term}_{2,1}, \text{term}_{2,2}, ..., \text{term}_{2,n^2}) \Rightarrow$

$\text{rel}_3(\text{term}_{3,1}, \text{term}_{3,2}, ..., \text{term}_{3,n^3})$

Example:  Graph

Logic

$(\forall Y, A)$

Invest(/WB, Y, A) \wedge

Trust(/JS, Y) \Rightarrow

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

General: Graph (normal) 

Logic 

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

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

Example: Graph 

Logic 

$(\forall Y, A)$ 

$(\text{Invest}(/\text{WB}, Y, A) \land \text{Trust}(/\text{JS}, Y) \Rightarrow \text{Invest}(/\text{JS}, Y, \text{US$} 5 \cdot 10^3))$
Rules: Conjunctions 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>
    <Atom>
      <Rel>Invest</Rel>
      <Ind unique="no">JS</Ind>
      <Var>Y</Var>
      <Data>US$ 5·10³</Data>
    </Atom>
  </And>
</Implies>

Logic

( Y, A)
(Invest(/WB, Y, A) \&
Trust(/JS, Y) \implies
Invest(/JS, Y, US$ 5·10³))

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**

```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**

\[ (\forall X) \quad \text{Greater}(X, 2) \land \text{Prime}(X) \implies \text{Odd}(X) \]

**Graph (prenormal)**

```
Graph '⇒' arrow normalizes to RuleML-like closure="universal"
```
Relations Equivalent to Relations

General: Graph (inst/var terms)

Logic

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

Example: Graph

Logic

\[(\forall X, Y, A) \quad \text{Transfer}(X, Y, A) \iff \text{Receive}(Y, X, A)\]
Equivalence-Defined Predicate Even: RuleML/XML Serialization

**FOL RuleML/XML**

\[
\begin{align*}
\langle \text{Equivalent oriented="yes" closure="universal"} \rangle \\
\langle \text{Atom} \rangle \\
\langle \text{Rel} \rangle \text{Even} \langle /\text{Rel} \rangle \\
\langle \text{Var} \rangle X \langle /\text{Var} \rangle \\
\langle /\text{Atom} \rangle \\
\langle \text{Atom} \rangle \\
\langle \text{Rel} \rangle \text{Divisible} \langle /\text{Rel} \rangle \\
\langle \text{Var} \rangle X \langle /\text{Var} \rangle \\
\langle \text{Data} \rangle 2 \langle /\text{Data} \rangle \\
\langle /\text{Equivalent} \rangle
\end{align*}
\]

**Logic**

\[
(\forall X) \quad \text{Even}(X) : \iff \text{Divisible}(X, 2)
\]

**Graph (prenormal)**

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

(PSOA Positional-Slotted-Term) Logic

Group (Forall ?Hu ?Wi ?Ch ( ?1#family(husb->?Hu
wife->?Wi
child->?Ch) :-
And(married(?Hu ?Wi)
Or(kid(?Hu ?Ch)
kid(?Wi ?Ch)))) )

married(Joe Sue)
kid(Sue Pete)
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 $o1, \ldots, on$ 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 snip2vex node, used to ‘encapsulate’ what another agent believes, wants, etc.)

Example: Graph (Modal) Logic
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](http://example.com), [Prolog](http://example.com), [Relfun](http://example.com), [Hilog](http://example.com), [RIF](http://example.com), [CL](http://example.com), and [IKL](http://example.com))
  - 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”

concave

function
contents

“database barrel”

convex

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

<table>
<thead>
<tr>
<th>Shape:</th>
<th>Corner:</th>
<th>Per</th>
<th>Copy</th>
<th>... Instantiation</th>
<th>... Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral</td>
<td>-angle</td>
<td>Rect-</td>
<td></td>
<td>Snip-</td>
<td>Round-</td>
</tr>
<tr>
<td>Individual</td>
<td>(Function Application)</td>
<td></td>
<td>-2cave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td></td>
<td></td>
<td>-4cave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposition</td>
<td>(Relation Application)</td>
<td></td>
<td>-2vex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relation</td>
<td>(incl. Class)</td>
<td></td>
<td>-4vex</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

  \[
  \begin{align*}
  X=3 : & \quad \text{Mult} \xrightarrow{X} 2 \\
      & \quad \text{Mult} \xrightarrow{2}
  \end{align*}
  \]

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

  \[
  \begin{align*}
  X=3 : & \quad \text{Mult} \xrightarrow{X} 2 \\
      & \quad \text{Mult} \xrightarrow{3} 2
  \end{align*}
  \]

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

  \[
  \begin{align*}
  X=3 : & \quad \text{Mult} \xrightarrow{X} 2 \\
      & \quad \text{Mult} \xrightarrow{6}
  \end{align*}
  \]

Assuming Mult built-in function
Functional Shapes: 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)
Relational Shapes: 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)
Grailog KS Viz 1.6: Subset of Horn Logic with Equality in SVG (1) — *KS Viz slides by Leah Bidlake* —

- XML documents containing hornlogeq RuleML are transformed using an XSLT stylesheet and processor into a Grailog visualization in SVG format that contains JavaScript for positioning calculations and variable naming.
- Postprocessing removes the no longer needed JavaScript for more readable code.
Grailog KS Viz 1.6: Subset of Horn Logic with Equality in SVG (2)
Supported Grailog Elements (1)

- Predicates with n-ary relations for $n \geq 1$

  \[
  \text{rel} \quad \text{inst}_1 \quad \text{inst}_2 \quad \ldots \quad \text{inst}_{n-1} \quad \text{inst}_n
  \]

- Equality (Datalog$^+$)

  \[
  \text{inst}_1 \quad \text{inst}_2
  \]
Supported Grailog Elements (2)

Single-premise rules containing:

- n-ary relations \((n \geq 1)\)

\[
\begin{align*}
&\text{rel}_1 \quad \text{inst}_{1,1} \quad \text{inst}_{1,2} \quad \cdots \quad \text{inst}_{1,n^1} \\
&\text{rel}_2 \quad \text{inst}_{2,1} \quad \text{inst}_{2,2} \quad \cdots \quad \text{inst}_{2,n^2}
\end{align*}
\]

- Equality (Datalog\(^+\))

\[
\begin{align*}
&\text{inst}_1 \quad \text{inst}_2 \\
&\text{inst}_1 \quad \text{inst}_2
\end{align*}
\]
Supported Grailog Elements (3)

Multi-premise rules containing:

- n-ary relations (n ≥ 1)

\[ \text{rel}_1 \quad \text{term}_{1,1} \quad \text{term}_{1,2} \quad \cdots \quad \text{term}_{1,n^1} \]

\[ \text{rel}_2 \quad \text{term}_{2,1} \quad \text{term}_{2,2} \quad \cdots \quad \text{term}_{2,n^2} \]

\[ \text{rel}_3 \quad \text{term}_{3,1} \quad \text{term}_{3,2} \quad \cdots \quad \text{term}_{3,n^3} \]

- Equality (Datalog\(^+\))

\[ \text{inst}_{1,1} \quad \text{inst}_{1,2} \]

\[ \text{inst}_{2,1} \quad \text{inst}_{2,2} \]

\[ \text{inst}_{3,1} \quad \text{inst}_{3,2} \]
Supported Grailog Elements (4)

- Function Applications

- Predicates with function as final element
Grailog KS Viz Structure (1)

• Set up SVG file with an initial viewBox to contain the drawings. Dimensions of viewBox are determined using JavaScript
  – Determine height using a variable that is updated with each new drawing
  – Determine width using method `maxx`

• Datalog RuleML XML files cannot contain namespace
Grailog KS Viz Structure (2)

- Distinguish between Rules and Facts:
- Search for <Implies>, <And>
  - Check for single premise rule and multi-premise rule containing <Atom>
  - Check for single premise rule and multi-premise rule containing <Equal>
Grailog KS Viz Structure (3)

• Search for <Atom>
  – Check for unary and n-ary relations
  – Check the parent of <Atom> to determine if it is contained in a single or multi-premise rule
  – For n-ary relations, check if last element is <Expr>

• Search for <Equal>
  – Check the parent of <Equal> to determine if it is contained in a single or multi-premise rule

• Search for <Expr>
Grailog KS Viz Structure (4)

• SVG
  – Drawings contain text, rectangles, rounded rectangles, polygons, straight paths, convex and concave paths using cubic Bézier curve, patterns, markers
Grailog KS Viz Structure (5)

• JavaScript
  – Calculates the coordinates used in the SVG drawings
  – Creates unique variable names used in the SVG drawings by concatenating constants (type) and position of the element
  – Updates the variable used to determine the SVG viewBox height
  – Uses method \textit{maxx} to determine the SVG viewBox width
Datalog Example: Multi-Premise Rule

“If an edge connects two vertices and the vertices are the same, then the edge is a loop”

```
<RuleML>
  <Assert>
    <Implies>
      <And>
        <Atom>
          <Rel>connects</Rel>
          <Var>edge</Var>
          <Var>v1</Var>
          <Var>v2</Var>
        </Atom>
        <Atom>
          <Rel>same_as</Rel>
          <Var>v1</Var>
          <Var>v2</Var>
        </Atom>
      </And>
      <Atom>
        <Rel>Loop</Rel>
        <Var>edge</Var>
      </Atom>
    </Implies>
  </Assert>
</RuleML>
```
Datalog⁺ Example: (Head) Equality

- Visualizes head equality as in Datalog⁺, represented as a special binary predicate

```xml
<RuleML>
  <Assert>
    <Equal>
      <Ind>point</Ind>
      <Ind>vertex</Ind>
    </Equal>
    <Equal>
      <Ind>line</Ind>
      <Ind>edge</Ind>
    </Equal>
  </Assert>
</RuleML>
```
Horn Logic Example: Function

- Currently only predicates with a function application as the final element can be visualized

```
<RuleML>
  <Assert>
    <Rel>contains</Rel>
    <Ind>NullGraph</Ind>
    <Var>set_V</Var>
    <Expr>
      <Fun per="copy">connectingEdges</Fun>
      <Data>0</Data>
    </Expr>
  </Assert>
</RuleML>
```
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
• Mapping to a family of logics, exchangeable in RuleML
• Use cases from cognition to technology to business
  – eLearning: Part of courses 384.126 LFCS and CS 6795 SWT
  – eTourism: Publication rules for multi-channel communication
• Processing of earlier Grailog-like DRLHs studied in Lisp, FIT, and Relfun
• Querying of Grailog knowledge with RuleML/POSL: http://wiki.ruleml.org/index.php/Grailog (Loan Processor test suite)
Conclusions (2)

- **Symbolic-to-visual translator implementation:**
  - 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
  - A Framework for Graph-Logic Knowledge Visualization MCS Thesis Proposal by Leah Bidlake, Dec 15, 2014 → Grailog KS Viz 1.6 (subset of Horn Logic with Equality) → …

- 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](http://www.api4kb.org) effort
  – Compare with other graph formalisms, e.g. Conceptual Graphs ([http://conceptualstructures.org](http://conceptualstructures.org)) and [CoGui](http://www.cogui.org) tool
  – Define mappings to/fro UML structure diagrams + OCL, adopting UML behavior diagrams ([http://www.uml.org](http://www.uml.org))

• Improve/implement tools
  – … → Grailog KS Viz 2.0 (full Horn Logic with Equality)
  – 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](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 Jarvis/Dexter (for data and LP rules), 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 4th (temporal) dimension of animations to visualize logical inferences, graph processing, etc.