

# RuleML/Grailog: The Rule Metalogic Visualized with Generalized Graphs


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

PhiloWeb 2011  
Thessaloniki, Greece, 5 October 2011

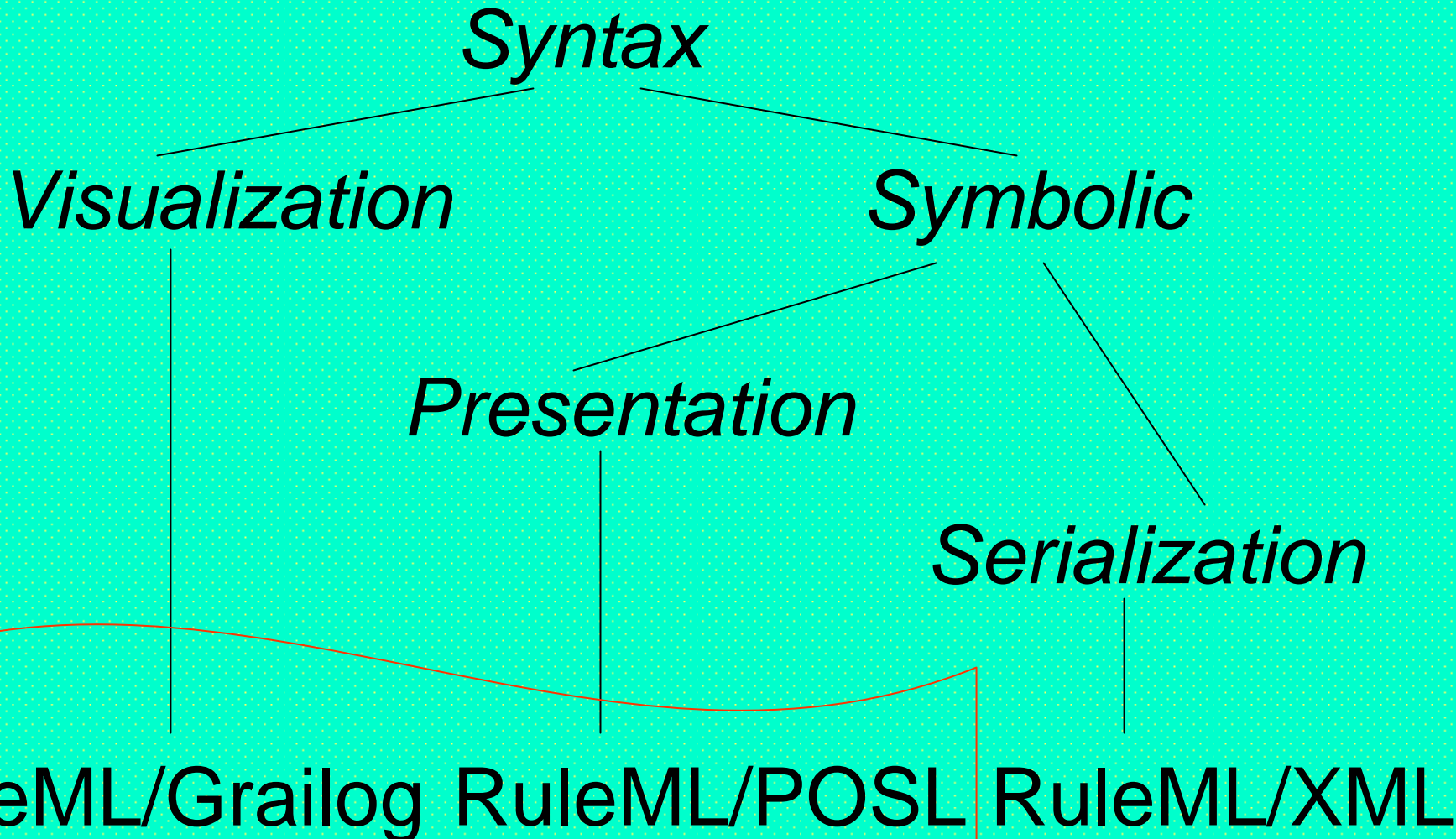
# Graph Visualization of Logic

- Enrich logical knowledge specifications with a convenient 2-dimensional syntax for logic-as-graph visualization
  - Supports **human in the loop** in knowledge **elicitation, validation, and processing**
- Complementary to the use of graphs for the efficient implementation of such specifications

# Rule MetaLogic Provides Family of Language Standards for Web 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)



# 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:

<http://ojs.academypublisher.com/index.php/jetwi/article/view/0204343353>

## *Visualization* = RuleML/Grailog:

Based on Directed Recursive Labelnode Hypergraphs (DRLHs):

<http://www.dfki.uni-kl.de/~boley/drlhops.abs.html>

# Grailog

**Graph inscribed logic** invokes imagery for logic

Proposed **cognitively adequate**

graph standard for visualized knowledge:

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

Permits logic-cognitive nets  
on the Semantic Web

# Generalized Graphs for the Representation and Mapping of Logic Languages

- We have used generalized graphs for representing various logic languages, where basically:
  - Graph nodes (vertices) represent individuals, classes, etc.
  - Graph arcs (edges) represent relations
- *Next slides:*  
What are the principles of this representation and what graph generalizations are required?
- *Later slides:*  
How are these graphs (invertibly) mapped to logic?

# Grailog Principles

- Graphs should make it easier for humans to read and write logic constructs by exploiting the 2-dimensional representation
- Graphs should be *natural extensions* of Directed Labeled Graphs (DLGs), used to represent simple semantic nets, i.e. of atomic ground formulas in function-free binary predicate logic (cf. binary Datalog ground facts and RDF triples)
- Graphs should allow *stepwise refinements* for all logic constructs, e.g. description logic TBoxes

# Searle's Chinese Room Argument

Classes with relations

understand

subsumes

hasInstance

*negation*

lang haveLanguage

understand

Language

English

Chinese

rules

texts

questions

replies

apply

use

with

for

for

Searle

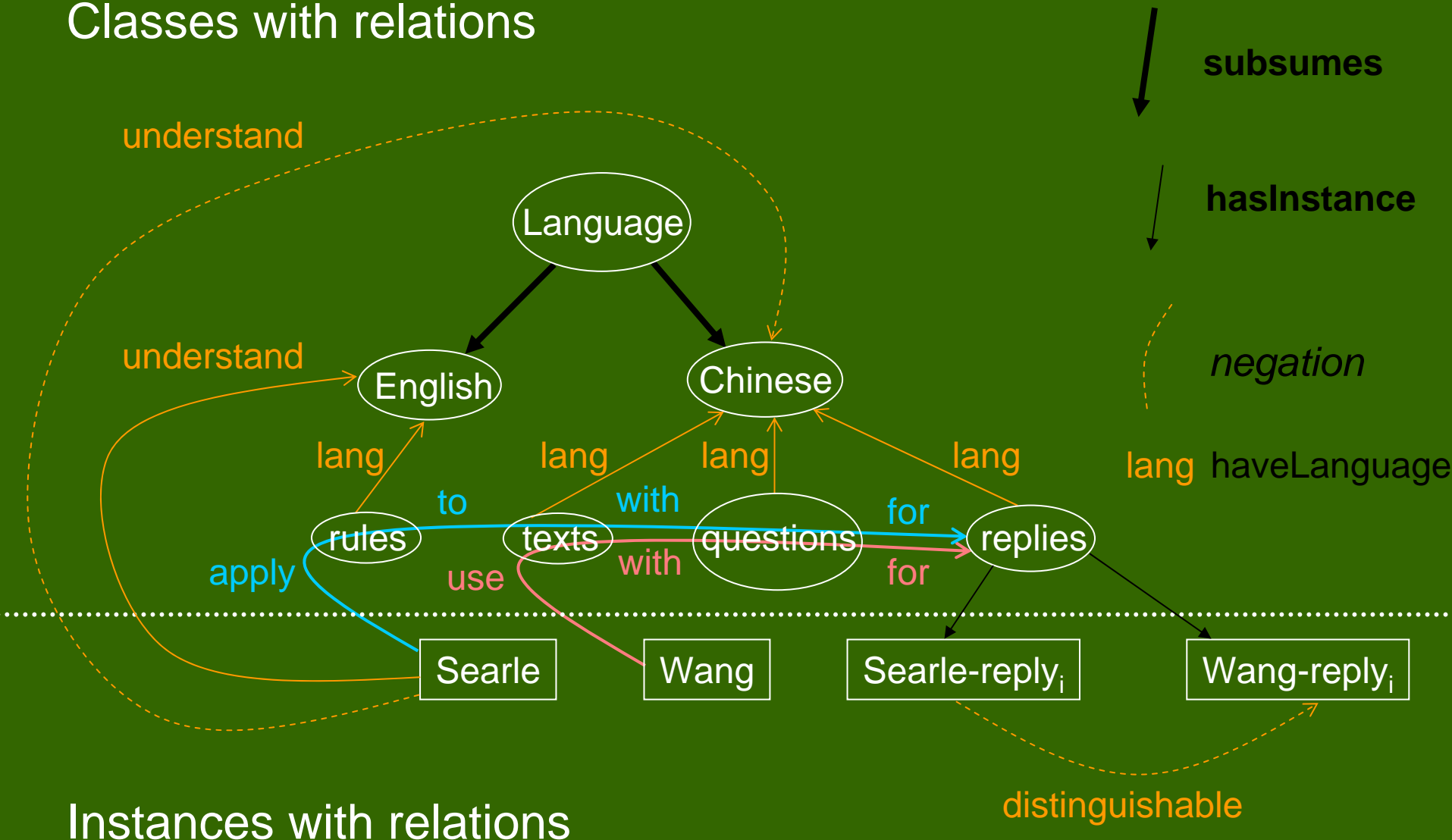
Wang

Searle-reply<sub>i</sub>

Wang-reply<sub>i</sub>

Instances with relations

distinguishable



# Grailog Generalizations

- **Directed hypergraphs:** For n-ary relations, directed (binary) arcs should be generalized to directed (n-ary) *hyperarcs*, e.g. representing relational tuples
- **Recursive (hierarchical) graphs:** For nested terms and formulas, modal logics, and modularization, ‘flat’ graphs should be generalized to allow other graphs as *complex nodes* to any level of ‘depth’
- **Labelnode graphs:** For allowing hybrid logics describing both instances and predicates, arc *labels* should also become usable as *nodes*

# Graphical Elements: Basic Shapes (1)

- Boxes for atomic and complex nodes
  - Oval: Classes, as labelnodes, for unary relations
  - Rectangle: Atomic for instances. Complex for ('passive') instance-denoting constructor-function applications
  - Roundangle (Rounded rectangle): ('active') function, predicate, and connective applications
  - Octagon: Embedded propositions and modules
- Labeled arrows (directed links) for arcs and hyperarcs (where hyperarcs 'cut through' nodes intermediate between first and last)

# Graphical Elements: Basic Shapes (2)

- Arrows for special arcs and hyperarcs
  - subsumes: Connects superclass, unlabeled, with subclass (arc, i.e. of length 2)
  - hasInstance: Connects class, as labelnode, with instance (hyperarc of length 1)
    - As in DRLHs, labelnodes can also be used (instead of labels) for hyperarcs of length  $> 1$
  - Implies: Hyperarc from premise(s) to conclusion

# Graphical Elements: Line Styles

- Solid lines (boxes & links): Positive
- Dashed lines (boxes & links): Negative
- Dotted lines (boxes): Disjunctive
- Heavy single lines (unlabeled arrows): subsumes
- Light single lines (unlabeled arrows): hasInstance
- Heavy double lines (unlabeled arrows): Implies
- Light double lines (unlabeled undirected links): Equal

# Graphical Elements: Hatching Patterns

- No hatching (boxes): Constant
- Hatching (atomic boxes): Variable

# Instances: Individual Constants

General: Graph (node)  $\xrightarrow{\text{mapping}}$  Logic (and POSL)  
*instance* *instance*

Examples: Graph

Warren Buffett

General Electric

US\$ 3 000 000 000

Logic

Warren Buffett

General Electric

US\$ 3 000 000 000

# Unknowns: Individual Variables

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

*variable*

*variable*

Examples: Graph

X

Y

A

Logic

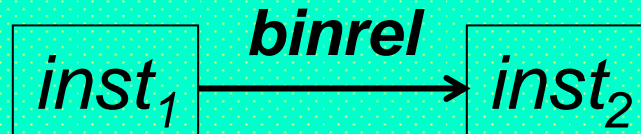
X

Y

A

# Predicates: Binary Relations (1)

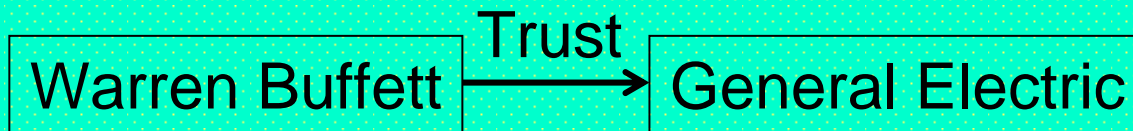
General: Graph (*labeled arc*)



Logic

*binrel(inst<sub>1</sub>, inst<sub>2</sub>)*

Example: Graph



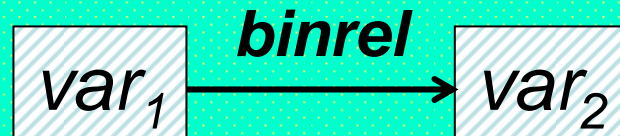
Logic

Trust(Warren Buffett,  
General Electric  
)

# Predicates: Binary Relations (2)

General: Graph (*labeled arc*)

Logic



$binrel(var_1, var_2)$

Example: Graph

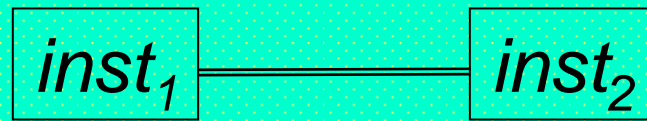
Logic



Trust(X, Y)

# Equality Predicate: Distinguished

General: Graph (unlabeled undirected *double arc*)



Logic (with equality)

$$inst_1 = inst_2$$

Example: Graph



Logic (with equality)

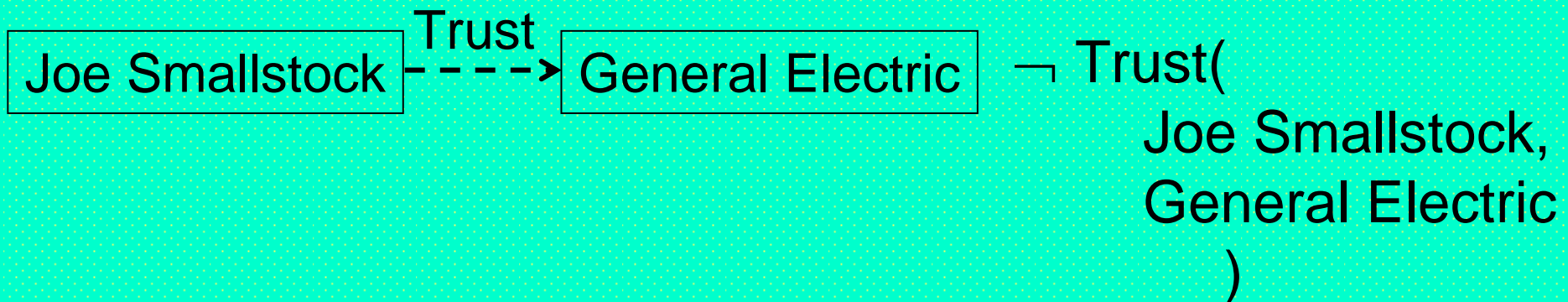
$$GE = \text{General Electric}$$

# Negated Predicates: Binary Relations

General: Graph (*dashed arc*)      Logic



Example: Graph      Logic



# Inequality Predicate: Distinguished

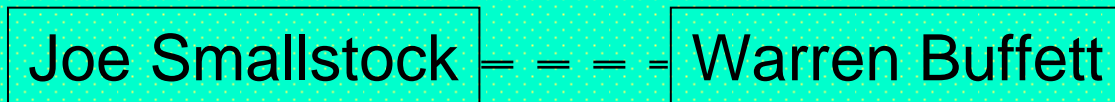
General: Graph (*dashed*  
unlabeled undirected  
*double arc*)



Logic (with equality)

$$inst_1 \neq inst_2$$

Example: Graph



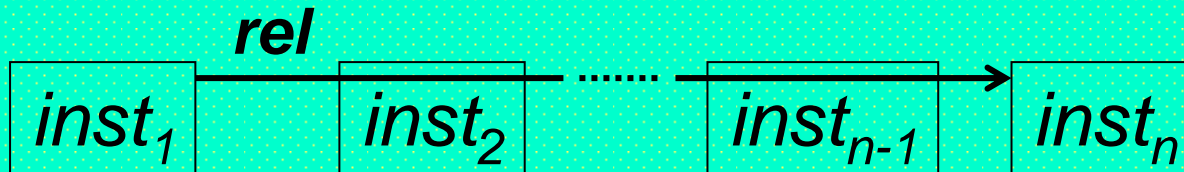
Logic (with equality)

$$\text{Joe Smallstock} \neq \text{Warren Buffett}$$

# Predicates: n-ary Relations ( $n > 1$ )

General: Graph (*hyperarc*)

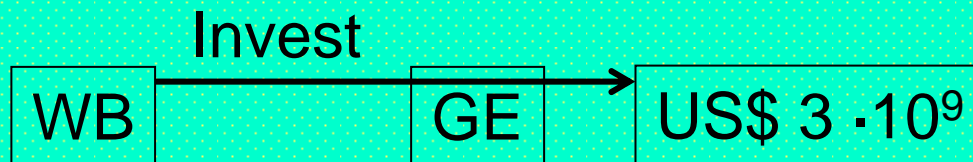
Logic



$rel(inst_1, inst_2, \dots, inst_{n-1}, inst_n)$

Example: Graph  
( $n=3$ )

Logic

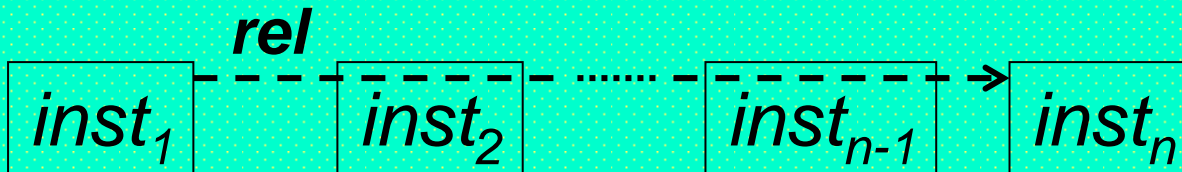


$Invest(WB, GE, US\$ 3 \cdot 10^9)$

# Negated Predicates: n-ary Relations

General: Graph (*dashed: not*)

Logic



$\neg rel(inst_1, inst_2, \dots,$   
 $inst_{n-1}, inst_n)$

Example: Graph  
( $n=3$ )

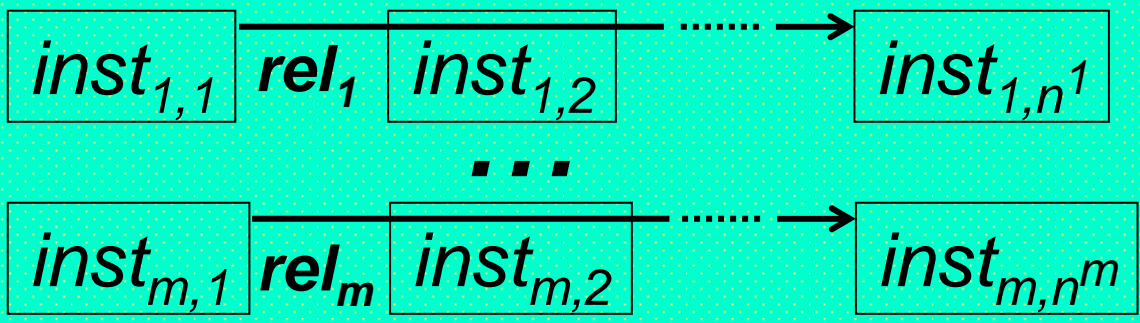
Logic



$\neg Invest(WB,$   
 $GE,$   
 $US\$ 4 \cdot 10^9)$

# Implicit Conjunction of Formula Graphs: Co-Occurrence on Top-Level

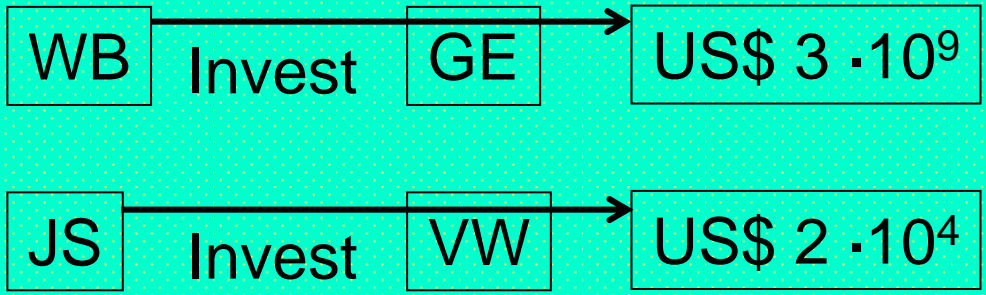
General: Graph ( $m$  hyperarcs)



Logic

$$rel_1(inst_{1,1}, inst_{1,2}, \dots, inst_{1,n^1}) \wedge \dots \wedge rel_m(inst_{m,1}, inst_{m,2}, \dots, inst_{m,n^m})$$

Example: Graph (2 hyperarcs)



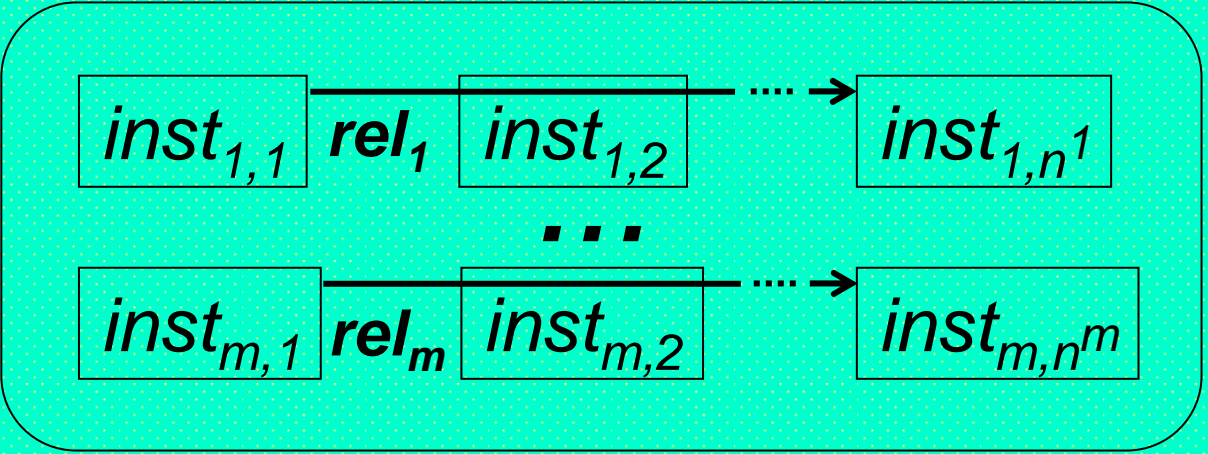
Logic

$$Invest(WB, GE, US\$ 3 \cdot 10^9) \wedge Invest(JS, VW, US\$ 2 \cdot 10^4)$$

# Explicit Conjunction of Formula Graphs: Co-Occurrence in Complex Node

General: Graph ( $m$  hyperarcs)

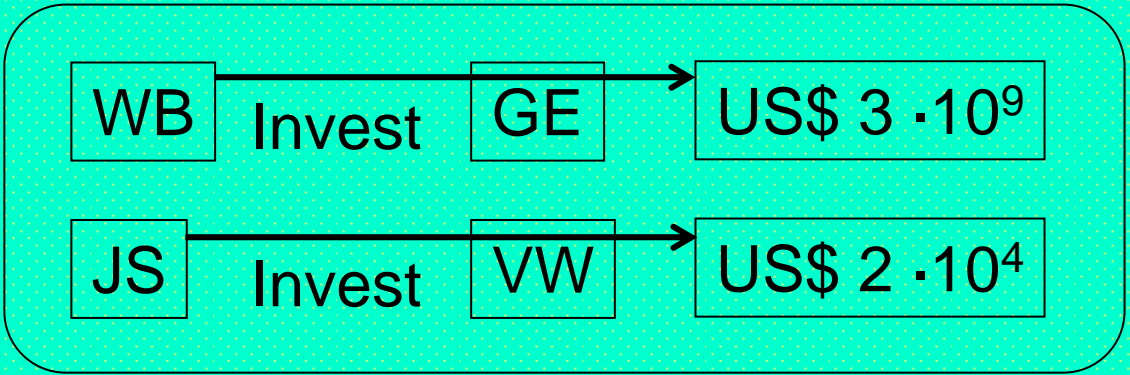
Logic



$$\begin{aligned}
 & (rel_1(inst_{1,1}, inst_{1,2}, \\
 & \quad \dots, inst_{1,n^1}) \wedge \\
 & \quad \dots \wedge \\
 & rel_m(inst_{m,1}, inst_{m,2}, \\
 & \quad \dots, inst_{m,n^m}))
 \end{aligned}$$

Example: Graph (2 hyperarcs)

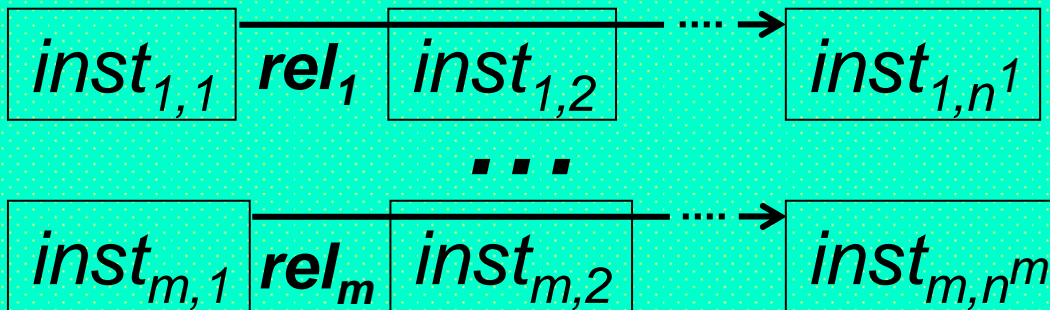
Logic



$$\begin{aligned}
 & (Invest(WB, GE, \\
 & \quad US\$ 3 \cdot 10^9) \wedge \\
 & Invest(JS, VW, \\
 & \quad US\$ 2 \cdot 10^4))
 \end{aligned}$$

# Disjunction of Formula Graphs: Co-Occurrence in Disjunctive Node

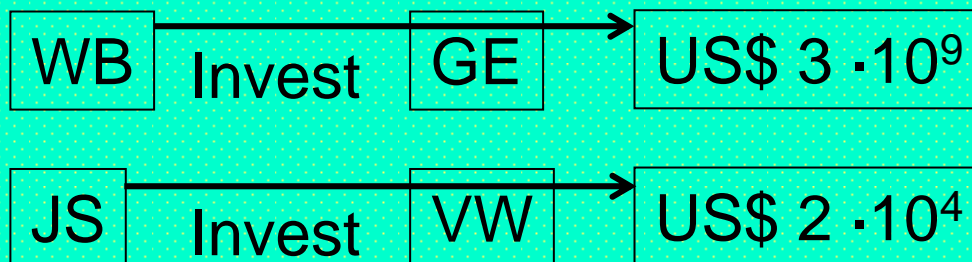
General: *Dotted Graph*



Logic

$$\begin{aligned} & (rel_1(inst_{1,1}, inst_{1,2}, \\ & \quad \dots, inst_{1,n^1}) \vee \\ & \quad \dots \vee \\ & rel_m(inst_{m,1}, inst_{m,2}, \\ & \quad \dots, inst_{m,n^m})) \end{aligned}$$

Example: *Dotted Graph*



Logic

$$\begin{aligned} & (\text{Invest}(\text{WB}, \text{GE}, \\ & \quad \text{US\$ } 3 \cdot 10^9) \vee \\ & \quad \text{Invest}(\text{JS}, \text{VW}, \\ & \quad \text{US\$ } 2 \cdot 10^4)) \end{aligned}$$

# Predicates: Unary Relations (Classes, Concepts, Types)

General: Graph (class applied to instance node)

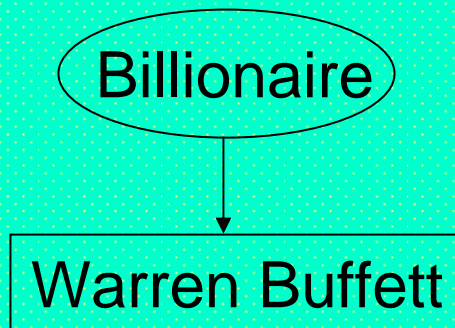
Logic



$class(inst_1)$

Example: Graph

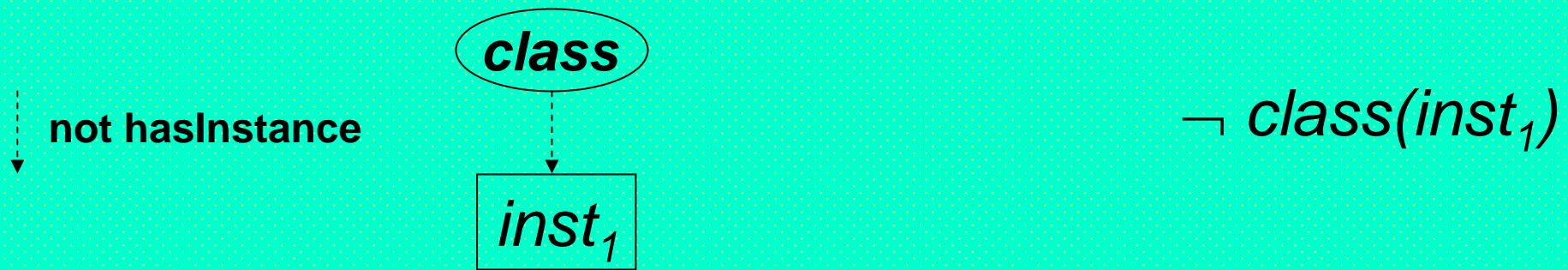
Logic



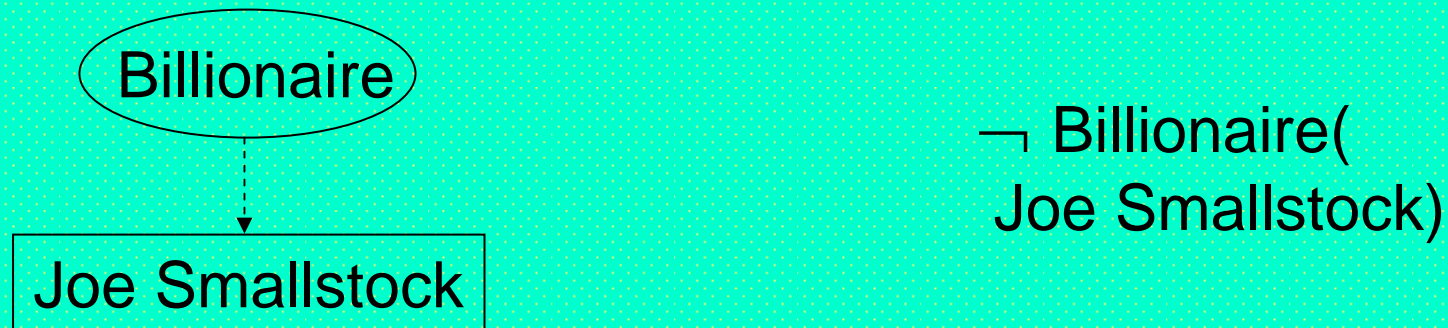
Billionaire(  
Warren Buffett)

# Negated Predicates: Unary Relations

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



Example: Graph      Logic



# Class Hierarchies (Taxonomies): Subclass Relation

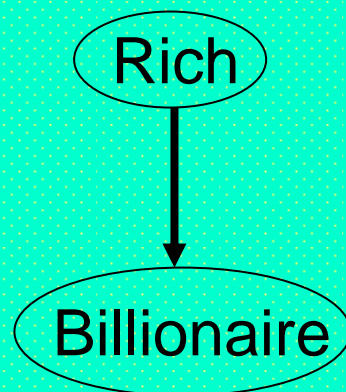
General: Graph (two nodes)



(Description)  
Logic

$class_1 \sqsubseteq class_2$

Example: Graph

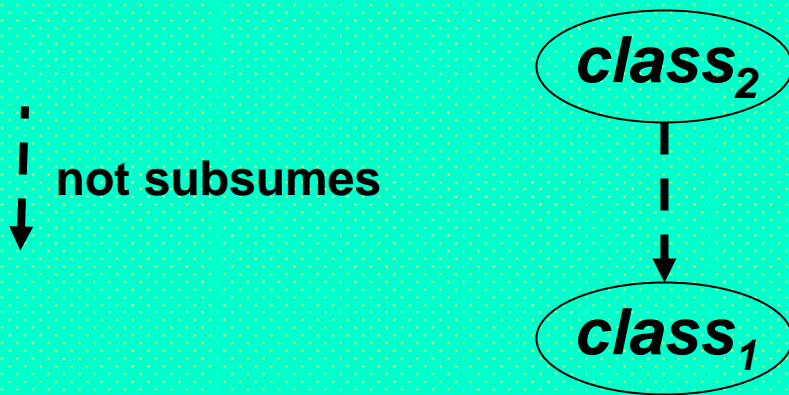


(Description)  
Logic

Billionaire  $\sqsubseteq$  Rich

# Class Hierarchies (Taxonomies): Negated Subclass Relation

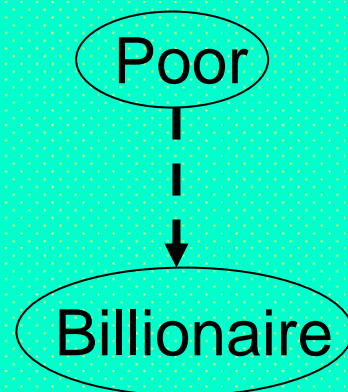
General: Graph (two nodes)



(Description)  
Logic

$class_1 \not\subseteq class_2$

Example: Graph

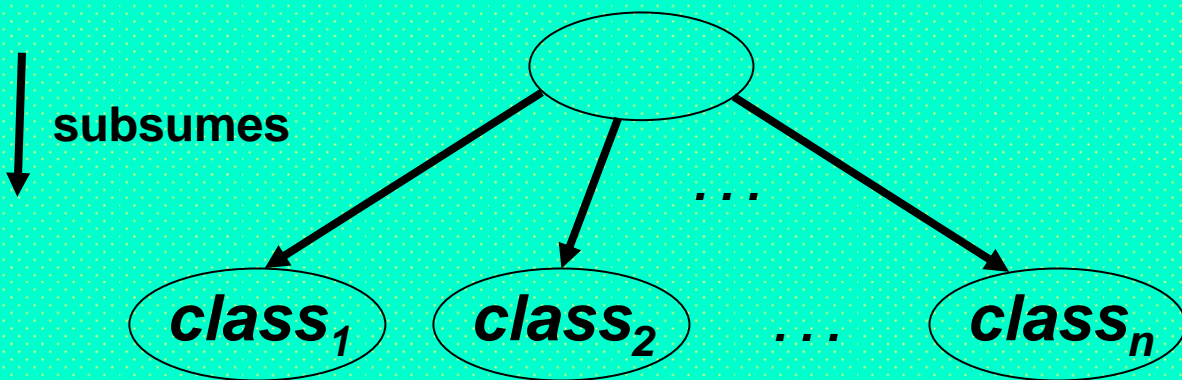


(Description)  
Logic

Billionaire  $\not\subseteq$  Poor

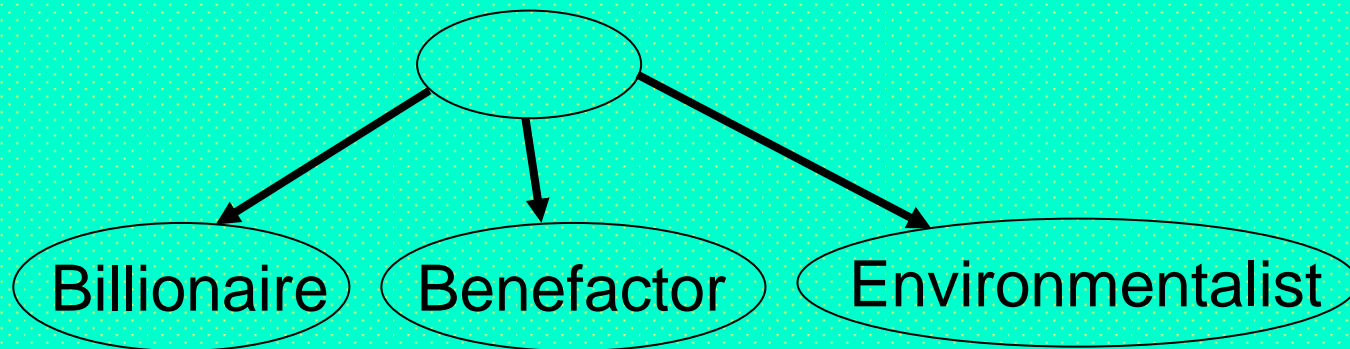
# Class Hierarchies (Taxonomy Trees): Class Union

General: Graph (blank node over  $n$ ) (Description) Logic



$class_1 \sqcup$   
 $class_2 \sqcup$   
 $\dots \sqcup$   
 $class_n$

Example: Graph (blank node over 3) (Description) Logic



Billionaire  $\sqcup$   
 Benefactor  $\sqcup$   
 Environmentalist

# Class Hierarchies (Taxonomy DAGs): Class Intersection

General: Graph (blank node *under*  $n$ ) (Description)

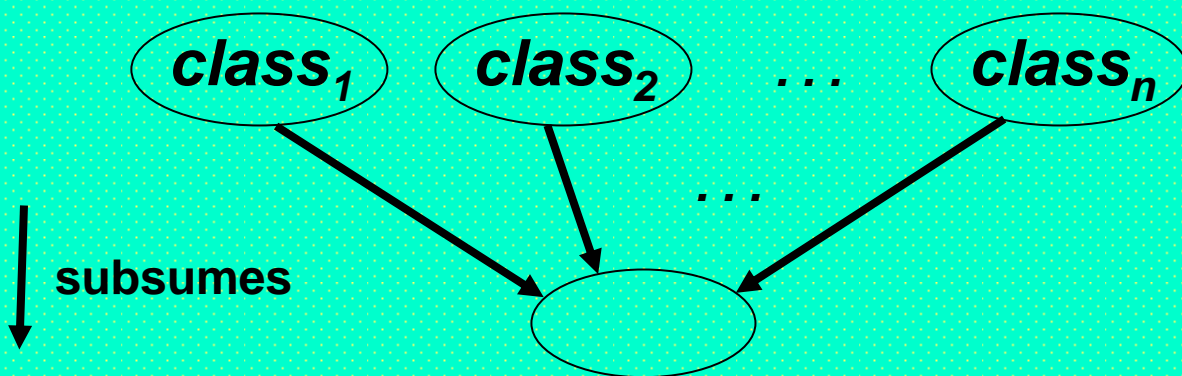
Logic

$class_1 \sqcap$

$class_2 \sqcap$

$\dots \sqcap$

$class_n$



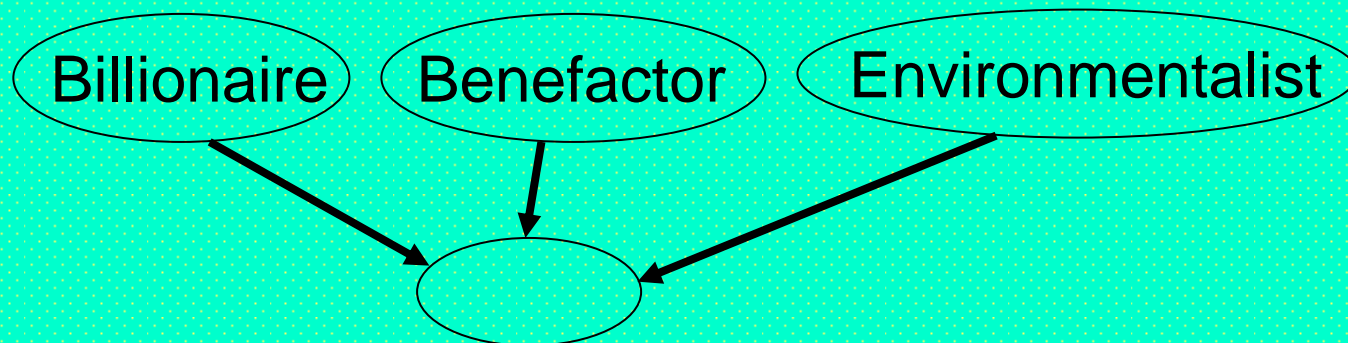
Example: Graph (blank node under 3) (Description)

Logic

Billionaire  $\sqcap$

Benefactor  $\sqcap$

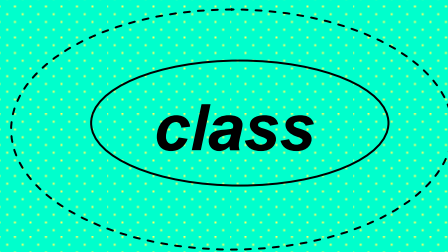
Environmentalist



# Class Hierarchies (Taxonomies): Class Complement

General: Graph (Description) Logic  
 (*dashed* node contains node to be complemented)

Arbitrary class

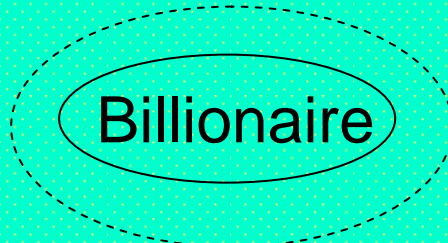



Atomic class  
(abbreviation)



$\neg$  *class*

Example: Graph (Description) Logic  
 Billionaire

$\neg$  Billionaire

# Intensional Class Constructions (Ontologies): Class-Property-Restricting TBox (Existential)

General: Graph

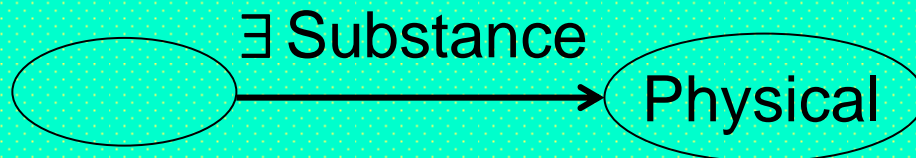
(Description)  
Logic



$\exists binrel . class$

Example: Graph

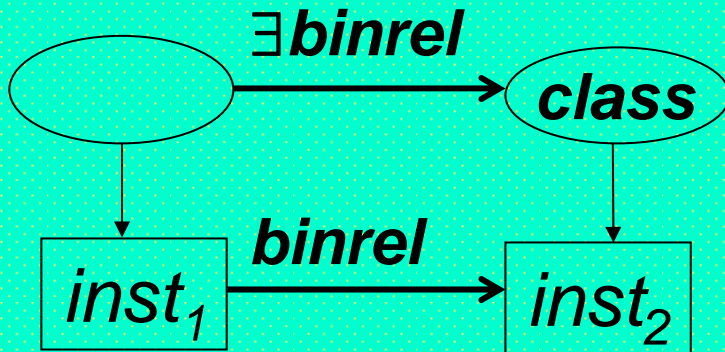
(Description)  
Logic



$\exists Substance . Physical$

# Instance Assertions (Populated Ontologies): ABox for Restriction TBox (Existential)

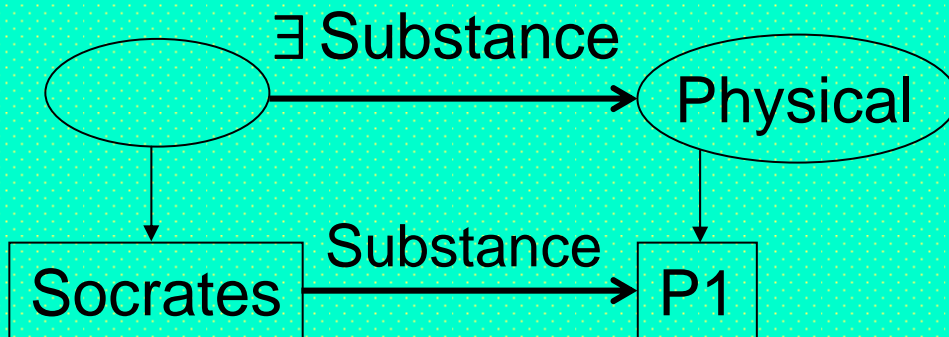
General: Graph



(Description)  
Logic

$\exists binrel.class(inst_1)$   
*class*(*inst*<sub>2</sub>)  
*binrel*(*inst*<sub>1</sub>, *inst*<sub>2</sub>)

Example: Graph



(Description)  
Logic

$\exists Substance.Physical$   
(Socrates)  
**Physical**(**P1**)  
**Substance**(**Socrates**, **P1**)

# Intensional Class Constructions (Ontologies): Class-Property-Restricting TBox (Universal)

General: Graph

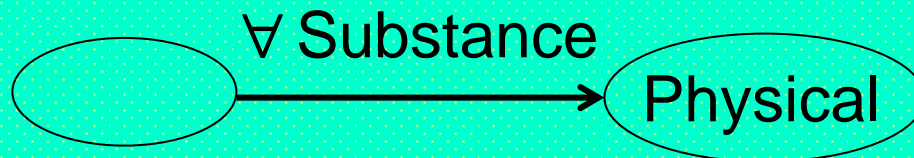
(Description)  
Logic



$\forall \textit{binrel} . \textit{class}$

Example: Graph

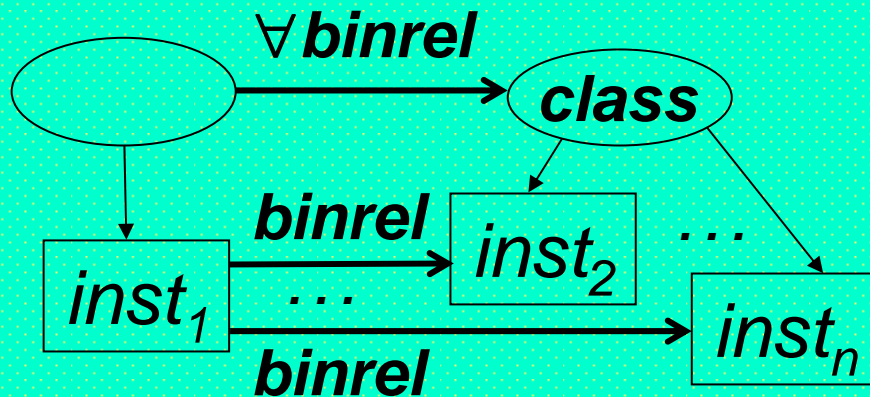
(Description)  
Logic



$\forall \textit{Substance} . \textit{Physical}$

# Instance Assertions (Populated Ontologies):<sup>36</sup> ABox for Restriction TBox (Universal)

General: Graph

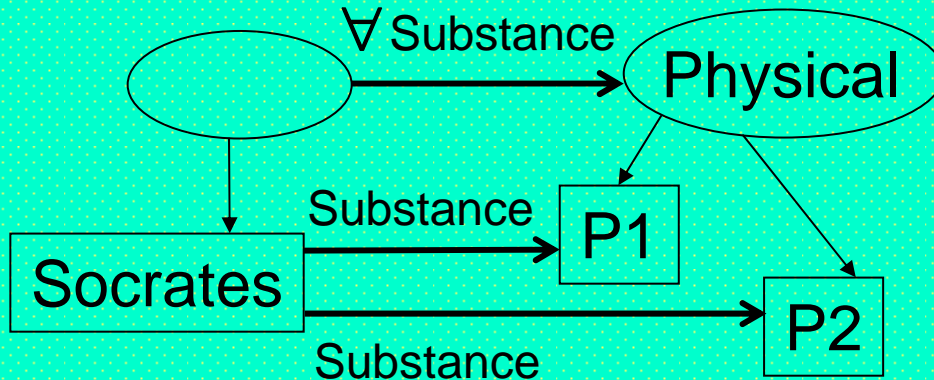


(Description)

Logic

$\forall binrel.class(inst_1)$   
 $class(inst_2)$   
 $class(inst_n)$   
 $binrel(inst_1, inst_2)$   
 $binrel(inst_1, inst_n)$

Example: Graph



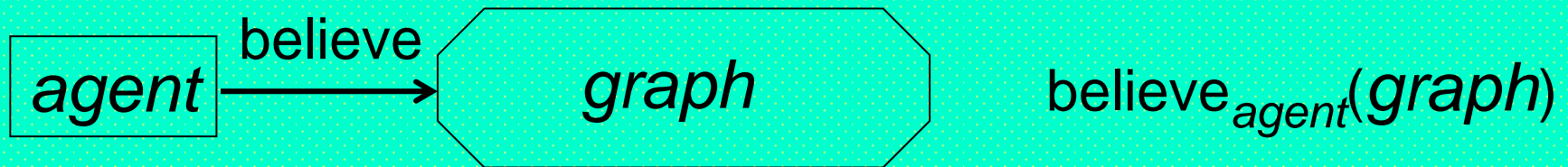
(Description)

Logic

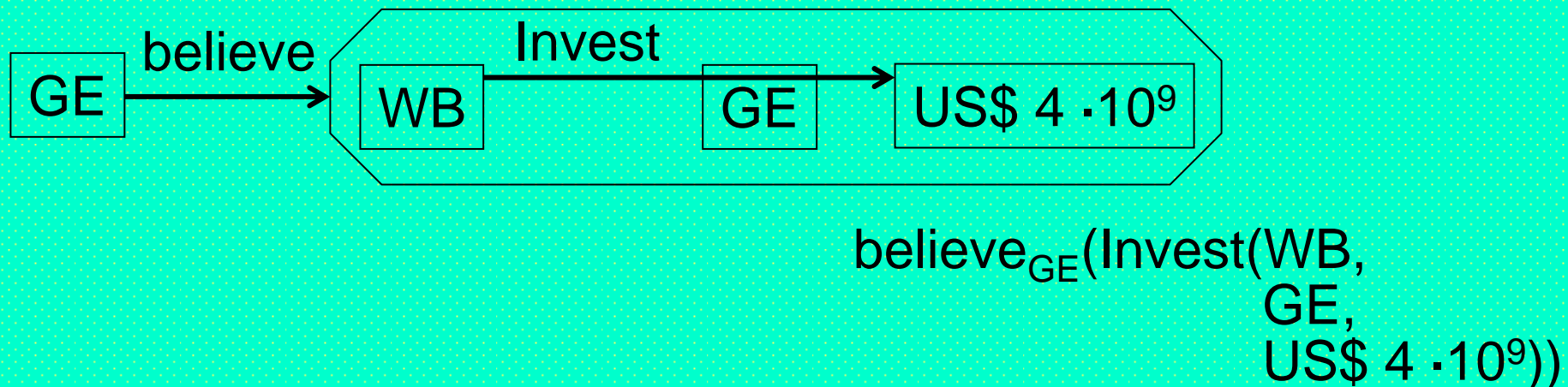
$\forall Substance.Physical$   
 (Socrates)  
 Physical(P1)  
 Physical(P2)  
 Substance(Socrates, P1)  
 Substance(Socrates, P2)

# Modally Embedded Propositions

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

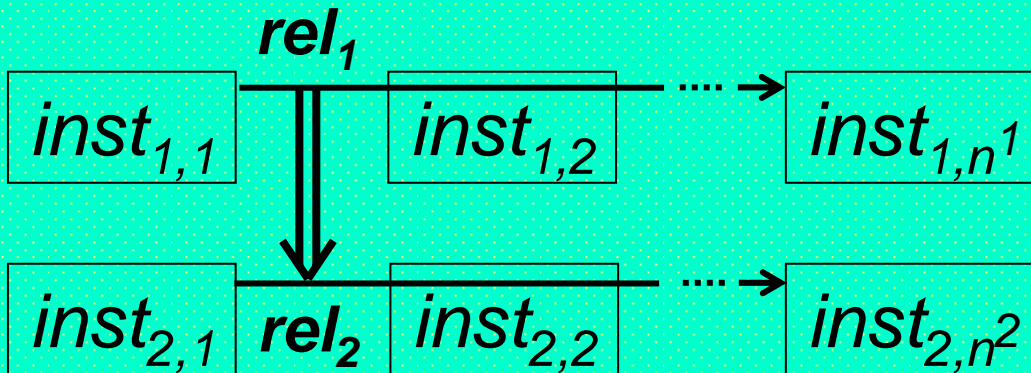


Example: Graph (Modal) Logic



# Rules: Relations Imply Relations (1)

General: Graph (ground)

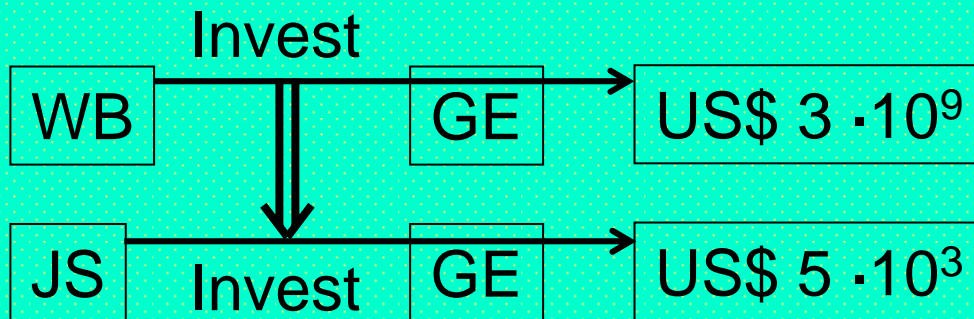


Logic

$$rel_1(inst_{1,1}, inst_{1,2}, \dots, inst_{1,n^1}) \Rightarrow$$

$$rel_2(inst_{2,1}, inst_{2,2}, \dots, inst_{2,n^2})$$

Example: Graph



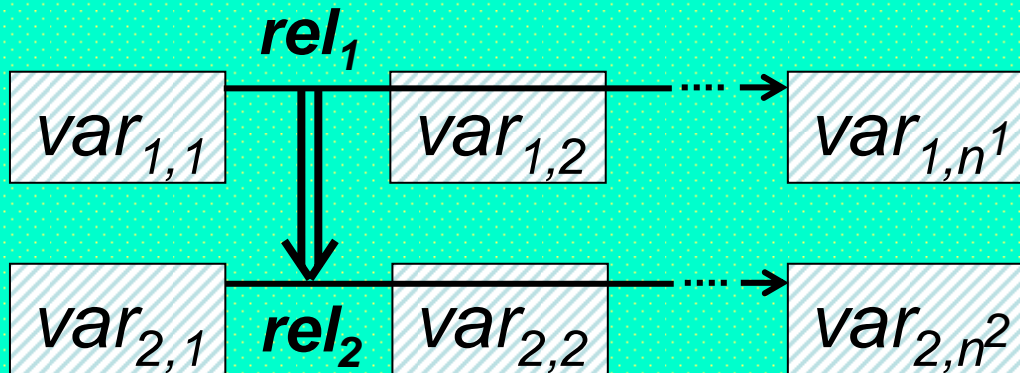
Logic

$$Invest(WB, GE, US\$ 3 \cdot 10^9) \Rightarrow$$

$$Invest(JS, GE, US\$ 5 \cdot 10^3)$$

# Rules: Relations Imply Relations (2)

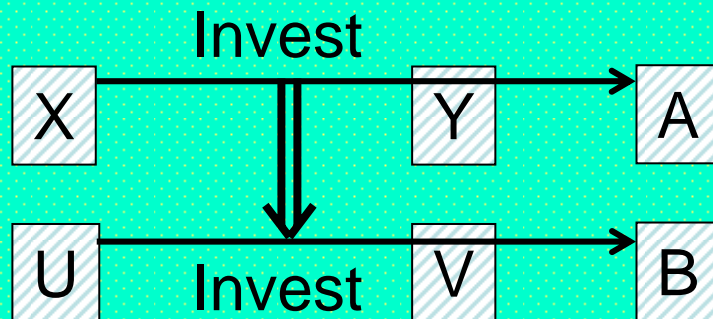
General: Graph (non-ground)



Logic

$$(\forall var_{i,j}) \\ rel_1(var_{1,1}, var_{1,2}, \dots, var_{1,n^1}) \Rightarrow \\ rel_2(var_{2,1}, var_{2,2}, \dots, var_{2,n^2})$$

Example: Graph

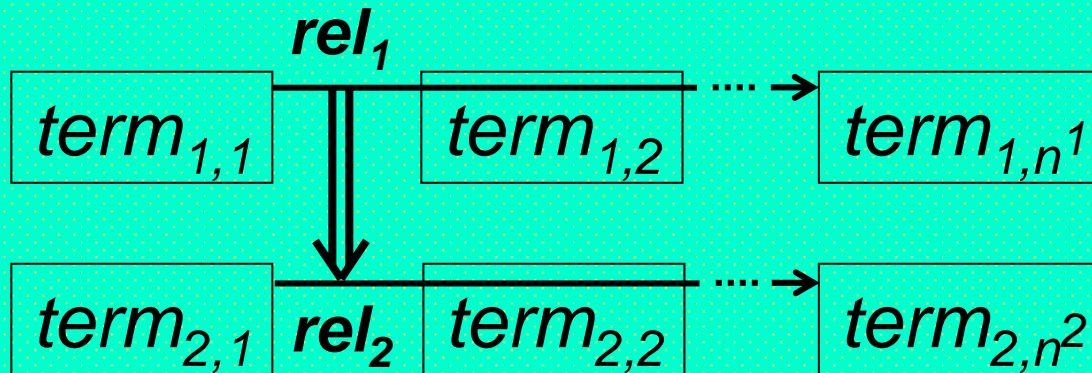


Logic

$$(\forall X, Y, A, U, V, B) \\ Invest(X, Y, A) \Rightarrow \\ Invest(U, V, B)$$

# Rules: Relations Imply Relations (3)

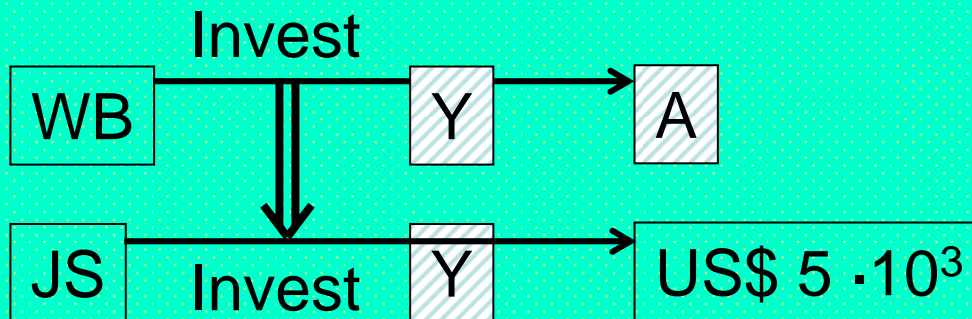
General: Graph (inst/var terms)



Logic

$$(\forall var_{i,j}) \\ rel_1(term_{1,1}, term_{1,2}, \dots, term_{1,n^1}) \Rightarrow \\ rel_2(term_{2,1}, term_{2,2}, \dots, term_{2,n^2})$$

Example: Graph

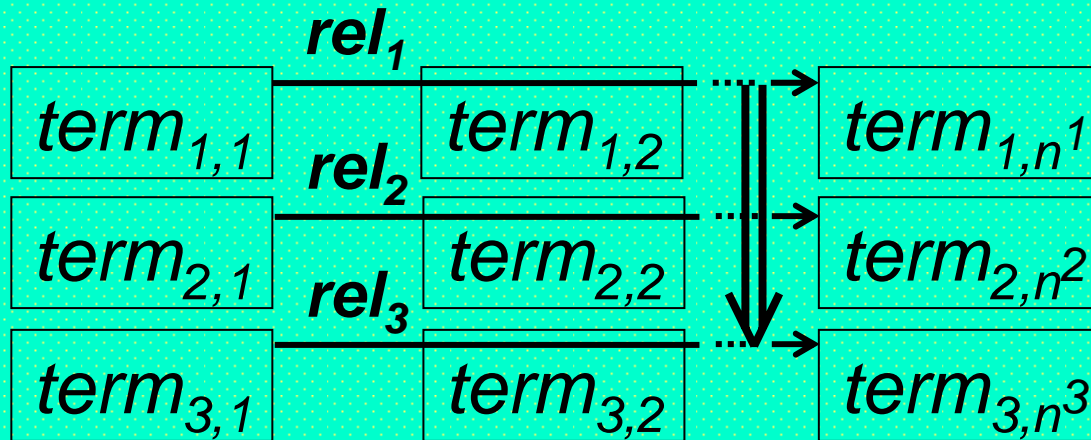


Logic

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

# Rules: Conjunctions Imply Relations

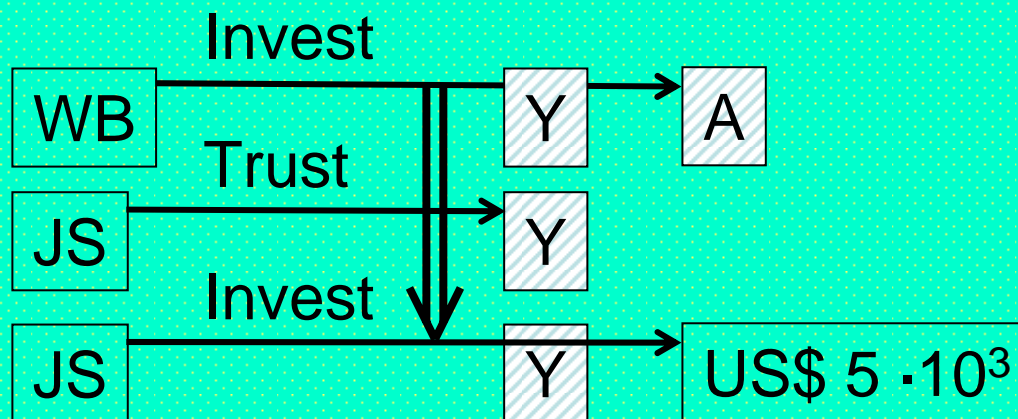
General: Graph (inst/var terms)



Logic

$$(\forall var_{i,j}) \\
 rel_1(term_{1,1}, term_{1,2}, \\
 \dots, term_{1,n^1}) \wedge \\
 rel_2(term_{2,1}, term_{2,2}, \\
 \dots, term_{2,n^2}) \Rightarrow \\
 rel_3(term_{3,1}, term_{3,2}, \\
 \dots, term_{3,n^3})$$

Example: Graph



Logic

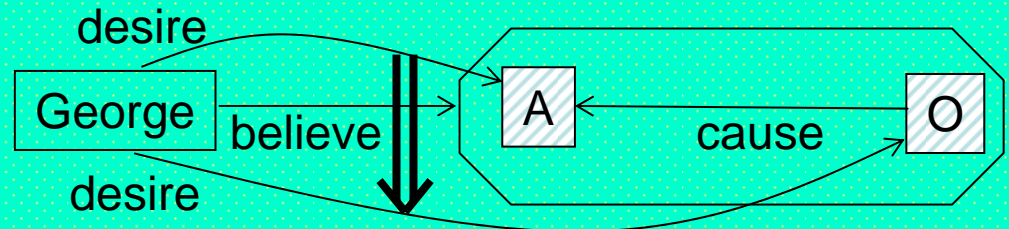
$$(\forall Y, A) \\
 Invest(WB, Y, A) \wedge \\
 Trust(JS, Y) \Rightarrow \\
 Invest(JS, Y, \\
 US\$ 5 \cdot 10^3)$$

# Beliefs and Desires as Propositional Attitudes (1)

Propositional attitude: a mental state relating a person to a proposition

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

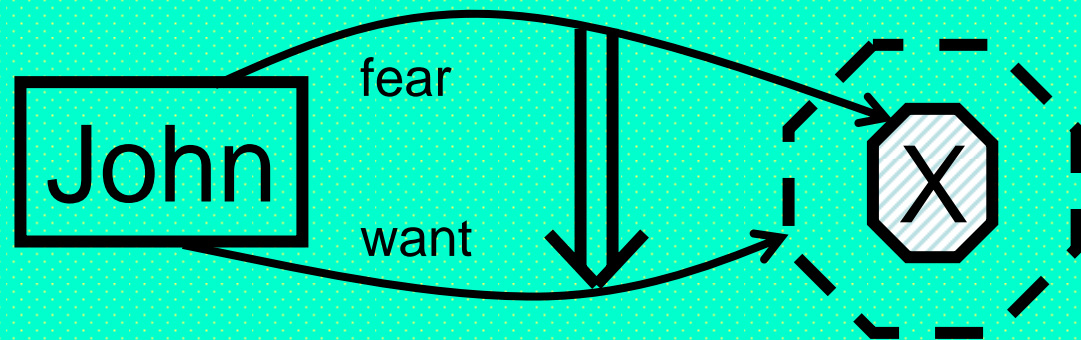
Grailog:



# Beliefs and Desires as Propositional Attitudes (2)

Example: “If John fears (state of affairs) X, then John wants that not X.”

Grailog:



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

# Conclusions (1)

- Refining/extending Grailog for the Rule Metalogic
- Comparing it with other graph formalisms
  - Conceptual Graphs: <http://conceptualstructures.org>
  - Unified Modeling Language: <http://www.uml.org>
- Use cases from philosophy to technology to business
  - E.g. “Logical Foundations of Cognitive Science”:  
[http://www.ict.tuwien.ac.at/lva/Boley\\_LFCS/index.html](http://www.ict.tuwien.ac.at/lva/Boley_LFCS/index.html)
- Implementing tools
  - Mapping between graphs, logic (as shown) & RuleML/XML
  - Graph indexing & querying (cf. <http://www.hypergraphdb.org>)
  - Graph-to-graph transformations (normal forms, merges, ...)
  - Advanced graph-theoretical operations (e.g., path tracing)
- Submitting to standards body

# Conclusions (2)

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