The Grailog Systematics for Visual-Logic Knowledge Representation with Generalized Graphs

(Long version: http://www.cs.unb.ca/~boley/talks/RuleMLGrailog.pdf)

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

Computer Science Seminar Series
Faculty of Computer Science, University of New Brunswick
Fredericton, Canada, 26 September 2012

Thanks for feedback on various versions and parts of this presentation:

Same title, Talk at the High Performance Computing Center Stuttgart (HLRS), 14 August 2012, Stuttgart, Germany

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 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 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 talk demonstrates how knowledge representation with graphs and logics can be reconciled. It proceeds 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 dimensions, which has led to the Grailog language as part of the Web-rule industry standard RuleML (http://ruleml.org/#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

Sample data visualization (http://wordle.net): Word cloud for frequency of words from BMIR abstract of this talk

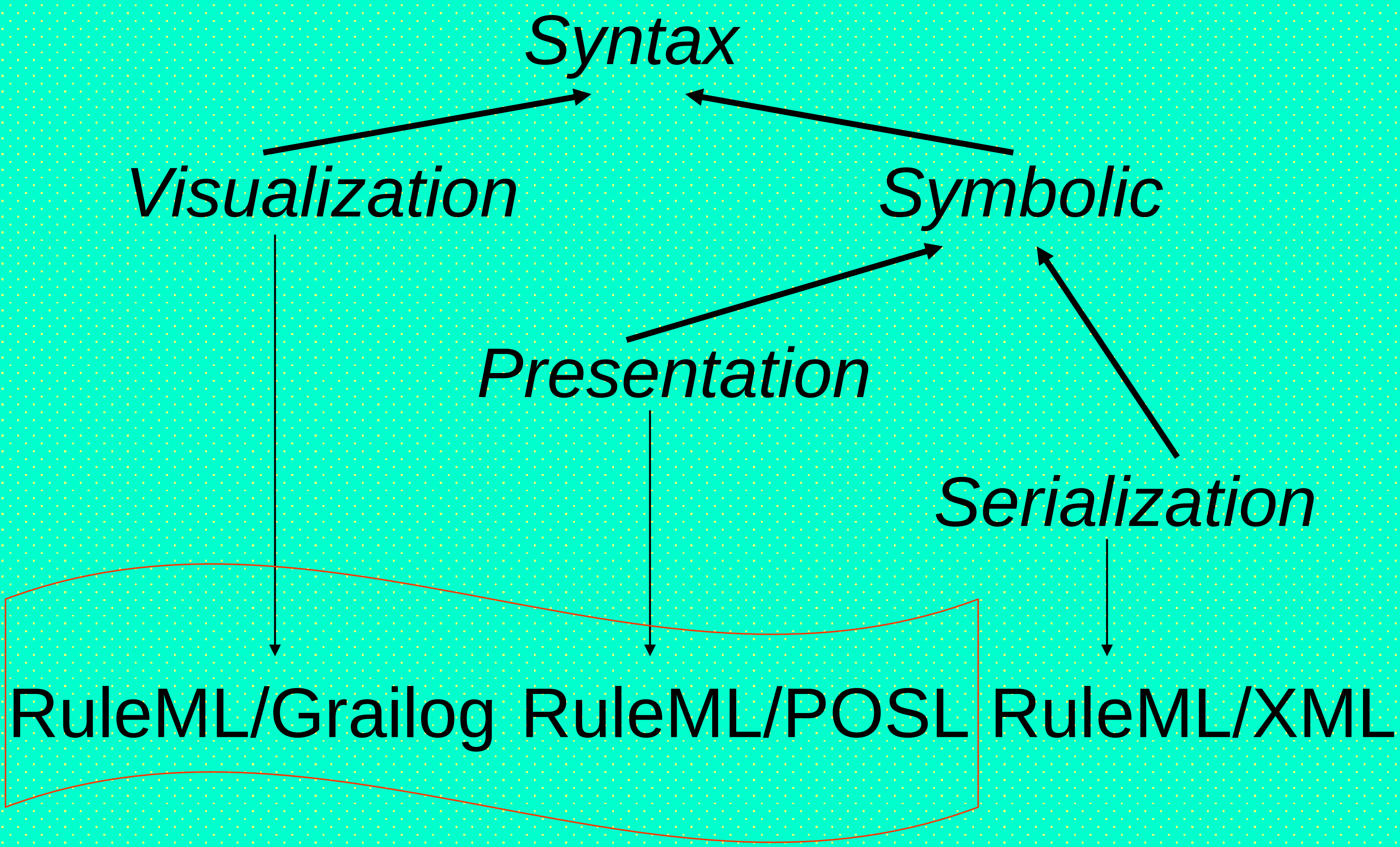
Graph Visualization of Knowledge: Remove Barrier to Entry for Logic

- From 1-dimensional *symbol-logic* knowledge **specification** to 2-dimensional *graph-logic* **visualization** in a systematic <u>2D syntax</u>
  - Supports human in the loop in knowledge **elicitation**, **validation**, as well as **processing**
- Combinable with graph transformation, (‘associative’) indexing & parallel processing for efficient **implementation** of specifications
- Move towards model-theoretic **semantics**
  - Deep names, as graph nodes, mapped directly/ injectively to elements of semantic interpretation

Rule Meta**L**ogic 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 **i**nscribed **log**ic invokes imagery for logic

Proposed cognitively motivated systematic graph standard for visual-logic knowledge:

Features orthogonal → easy to learn/remember, e.g. for (Business) Analytics/Intelligence

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

Pick subset for each elicited 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 (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 & Map Logic Languages and Grailog 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 RuleML?
- *Final slides:*
  What is the *systematics* of Grailog features?

Grailog Principles

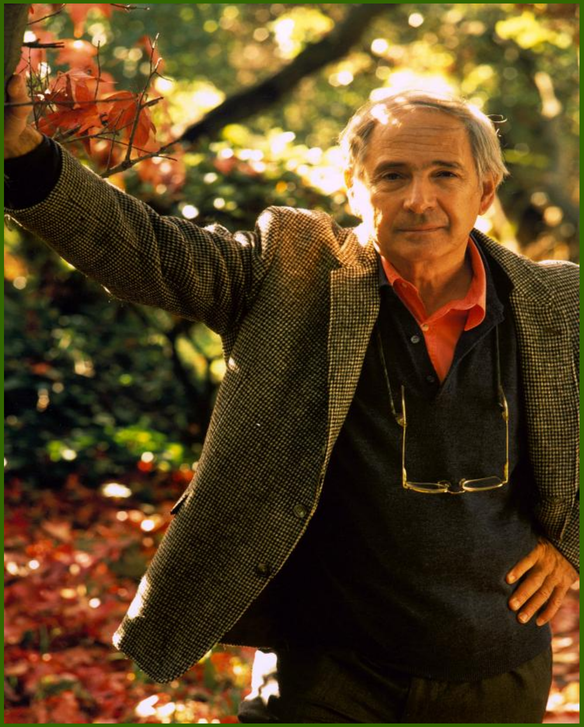
- Graphs should make it easier for humans to read and write logic constructs by exploiting a 2-dimensional representation with shorthand & normal forms, from Controlled English to logic
- Graphs should be *natural extensions* (e.g. n-ary) of Directed Labeled Graphs (DLGs), often used to represent simple semantic nets, i.e. of atomic ground formulas in function-free dyadic predicate logic (cf. binary Datalog ground facts, RDF triples, the Open Graph, and the Knowledge Graph)
- Graphs should allow *stepwise refinements* for all logic constructs: Description Logic constructors, F-logic frames, general PSOA RuleML terms, etc.
- Extensions to boxes & links should be *orthogonal*

Informal Grailog Preview: Searle's Chinese Room Argument

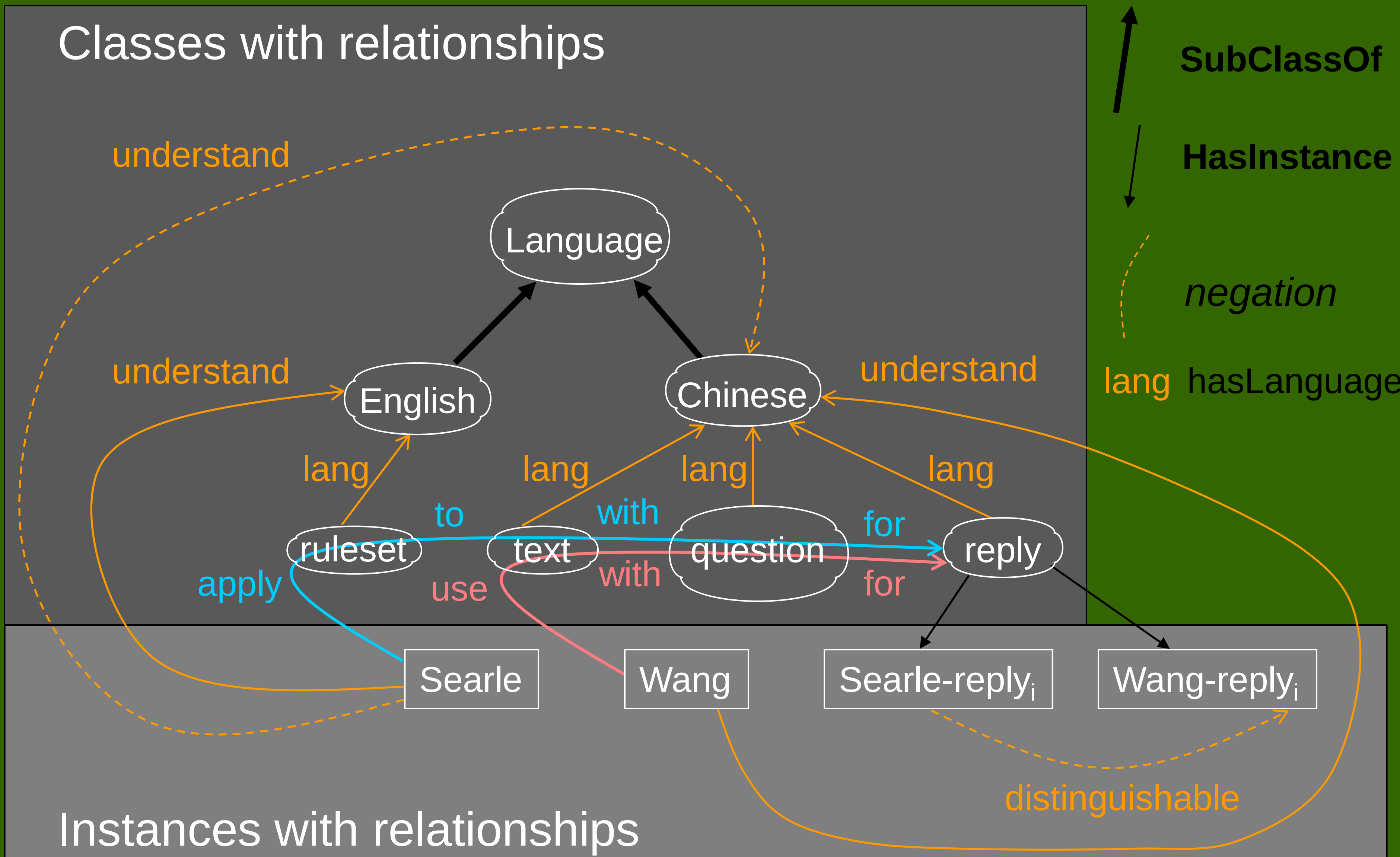
John Searle (*emphasis added*):

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

(Minds, Brains and Programs, 1980)



Searle's Chinese Room Scenario: Grailog for Visual Controlled English



Grailog Generalizations

- **Directed hypergraphs:** For n-ary relationships, directed relation-labeled (binary) arcs should 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 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 relations (predicates), arc *labels* should also become usable as *nodes*

Graphical Elements: Names

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

deepname

- Written **onto** boxes (node labels):
  Shallow (alternate, 'aka') names
  - (Occurrence-)restricted
    Non-unique Name Assumption (rNNA)
    via Shallow Name Occurrence (SNO)

shallowname

Instances: Individual Constants as Deep Name Occurrences

General:	Graph (node) $\xrightarrow{\text{mapping}}$	Logic
	$\boxed{\textit{deepname}}$	*deepname*
Examples:	Graph	Logic
	$\boxed{\text{Warren Buffett}}$	Warren Buffett
	$\boxed{\text{General Electric}}$	General Electric
	$\boxed{\text{US\\$ 3 000 000 000}}$	US$ 3 000 000 000

Instances: Individual Constants as Shallow Name Occurrences

		mapping	
General:	Graph (node)	→	Logic (vertical bar marks shallowness)
	shallowname		|*shallowname*
Examples:	Graph		Logic
	WB		|WB
	GE		|GE
	US$ 3B		|US$ 3B

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)

Graph	Logic
variable	*variable*

Examples: Graph Logic

Graph	Logic
X	*X*
Y	*Y*
A	*A*

Predicates: Binary Relations (1)

General: Graph (*labeled* arc) Logic

$\boxed{inst_1} \xrightarrow{\textbf{\textit{binrel}}} \boxed{inst_2}$

binrel($inst_1$, $inst_2$)

Example: Graph Logic

$\boxed{\text{Warren Buffett}} \xrightarrow{\text{Trust}} \boxed{\text{General Electric}}$

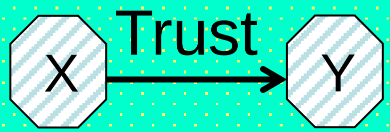
Trust(Warren Buffett,
General Electric
)

Predicates: Binary Relations (2)

General: Graph (*labeled* arc) Logic

$\textit{binrel}(\textit{var}_1, \textit{var}_2)$

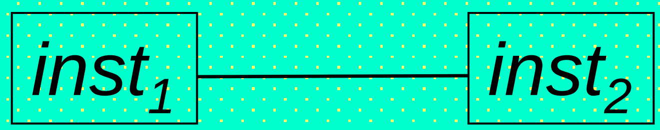
Example: Graph Logic



Trust(*X*,*Y*)

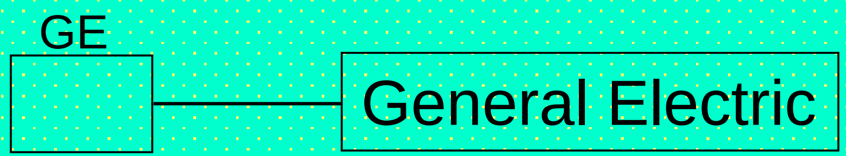
Ground Equality: Identifying Pairs of Constants

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



$inst_1 = inst_2$

Example: Graph Logic



|GE = General Electric

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)	Logic (with oriented equality)
	$\boxed{inst_1}$: ——— $\boxed{inst_2}$	$inst_1 := inst_2$
Example:	Graph	Logic
	GE $\boxed{}$: ——— $\boxed{\text{General Electric}}$	|GE := General Electric

Ground Equality: Defining Symbolic Constants as IRIs

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

inst : *IRI*

inst := IRI

Example: Graph — Logic

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

$$\boxed{\mathit{inst}_1} \xrightarrow{\textbf{\textit{binrel}}} \boxed{\mathit{inst}_2}$$

$$\neg\, \mathit{binrel}(\mathit{inst}_1, \mathit{inst}_2)$$

Example: Graph Logic

$$\boxed{\text{Joe Smallstock}} \xrightarrow{\text{Trust}} \boxed{\text{General Electric}}$$

¬ Trust(
Joe Smallstock,
General Electric
)

Negated Predicates: Binary Relations (Long Form)

General: Graph (*dashed* box) Logic

$$\boxed{inst_1} \xrightarrow{\textbf{\textit{binrel}}} \boxed{inst_2}$$

$\neg\,(binrel(inst_1, inst_2))$

Example: Graph Logic

$$\boxed{\text{Joe Smallstock}} \xrightarrow{\text{Trust}} \boxed{\text{General Electric}}$$

¬ (Trust(
Joe Smallstock,
General Electric
))

Ground Inequality: Pairwise Difference (Shorthand)

General: Graph (*dashed unlabeled undirected* arc) — Logic (with equality)

[$inst_1$] - - - - - - - - - - - - [$inst_2$] — $inst_1 \neq inst_2$

Example: Graph — Logic (with equality)

[Joe Smallstock] - - - - - - - - - [Warren Buffett] — Joe Smallstock $\neq$ Warren Buffett

Ground Inequality: Pairwise Difference (Long Form)

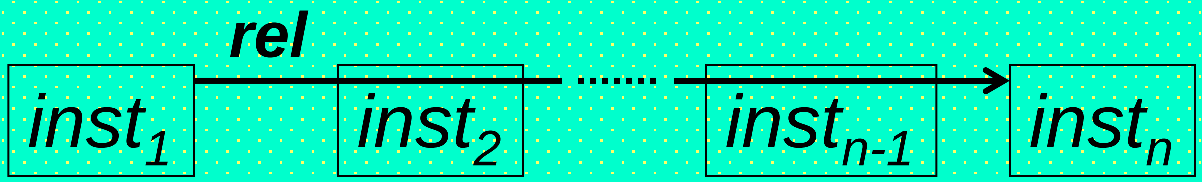
General:	Graph (*dashed* box, *unlabeled* *undirected* arc)	Logic (with equality)
	$inst_1$ —— $inst_2$	$\neg\, (inst_1 = inst_2)$
Example:	Graph	Logic (with equality)
	Joe Smallstock —— Warren Buffett	$\neg$ (Joe Smallstock = Warren Buffett)

Graphical Elements: Arrows (1)

- Labeled arrows (directed links) for arcs and hyperarcs (where hyperarcs ‘cut through’ nodes intermediate between first and last)

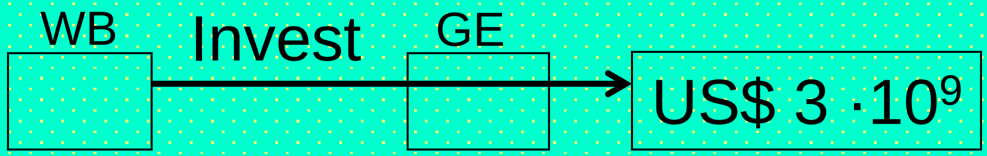
Predicates: n-ary Relations (n>1)

General: Graph (*hyper*arc) Logic



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

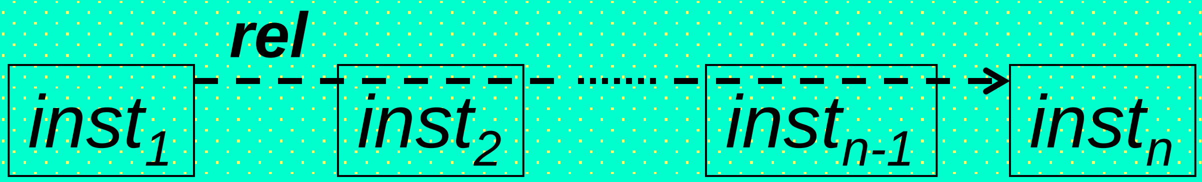
Example: (n=3) Graph Logic



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

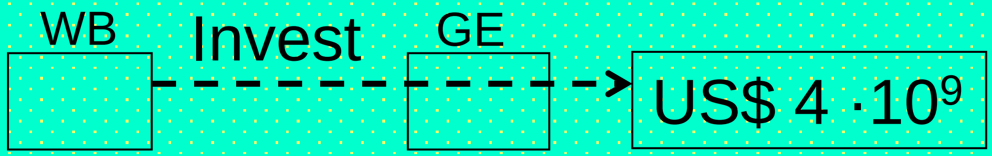
Negated Predicates: n-ary Relations (Shorthand)

General: Graph (*dashed* hyperarc) Logic



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

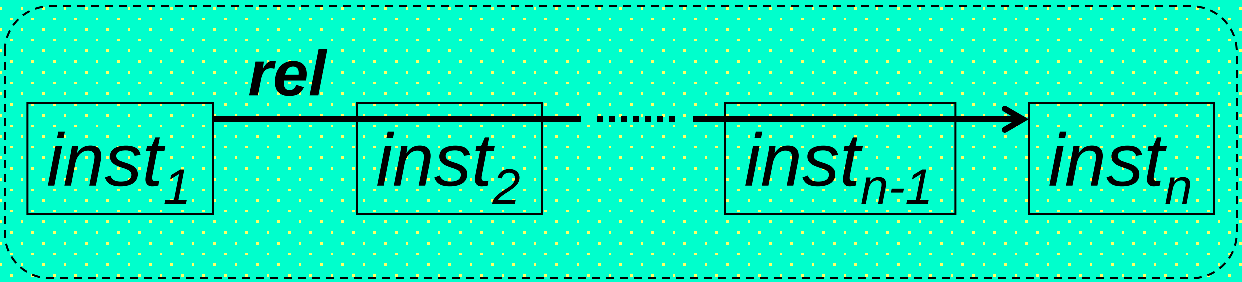
Example: (n=3) Graph Logic



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

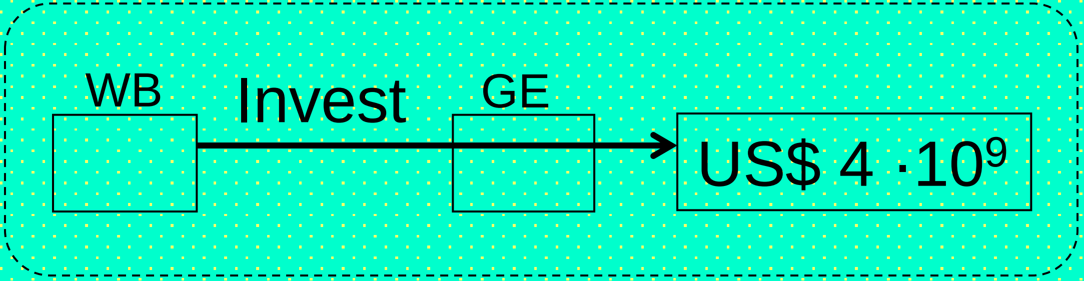
Negated Predicates: n-ary Relations (Long Form)

General: Graph (*dashed* box) Logic



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

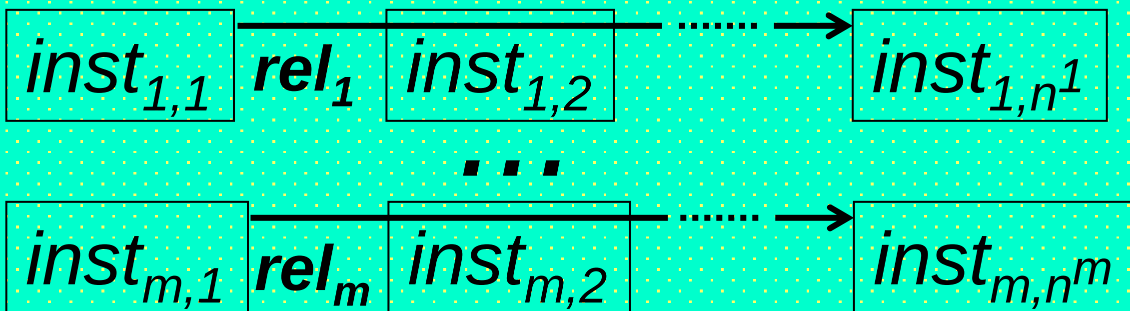
Example: (n=3) Graph Logic



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

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

General: Graph (m hyperarcs) Logic

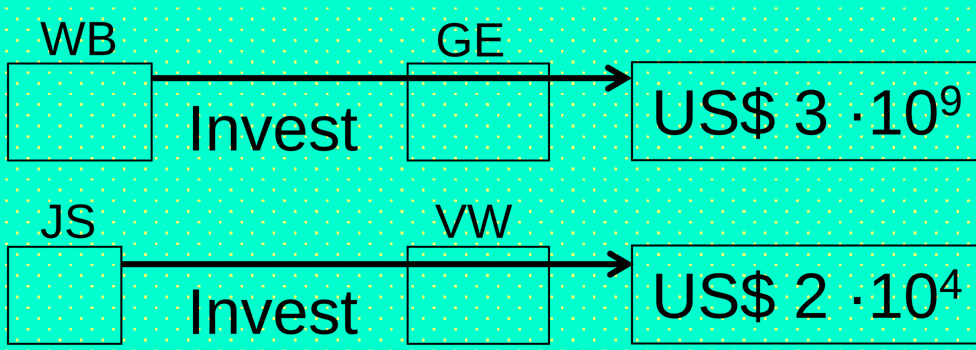


$rel_1(inst_{1,1}, inst_{1,2}, \ldots, inst_{1,n^1}) \;\wedge$

$\ldots \;\wedge$

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

Example: Graph (2 hyperarcs) Logic

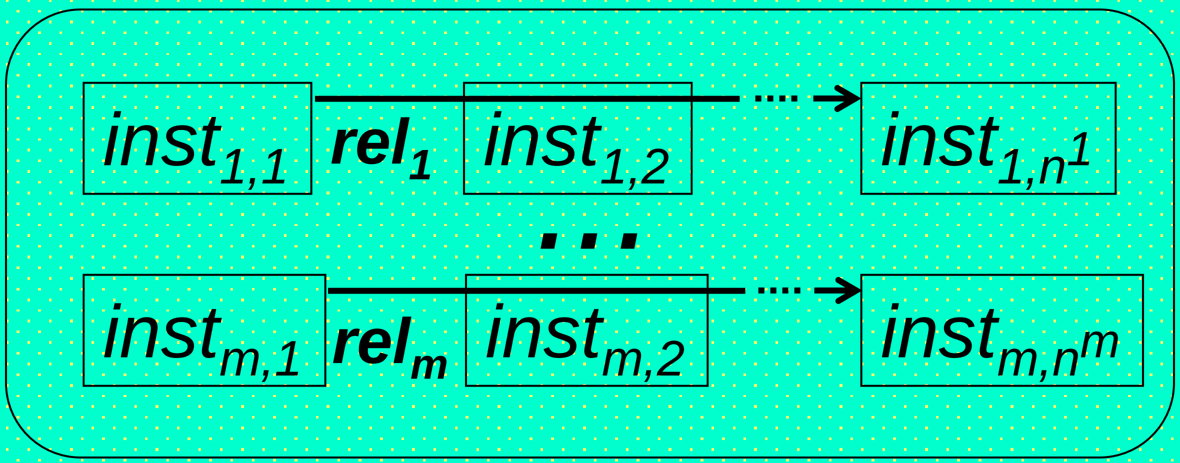


Invest(|WB, |GE, US\$ $3 \cdot 10^9$) $\wedge$

Invest(|JS, |VW, US\$ $2 \cdot 10^4$)

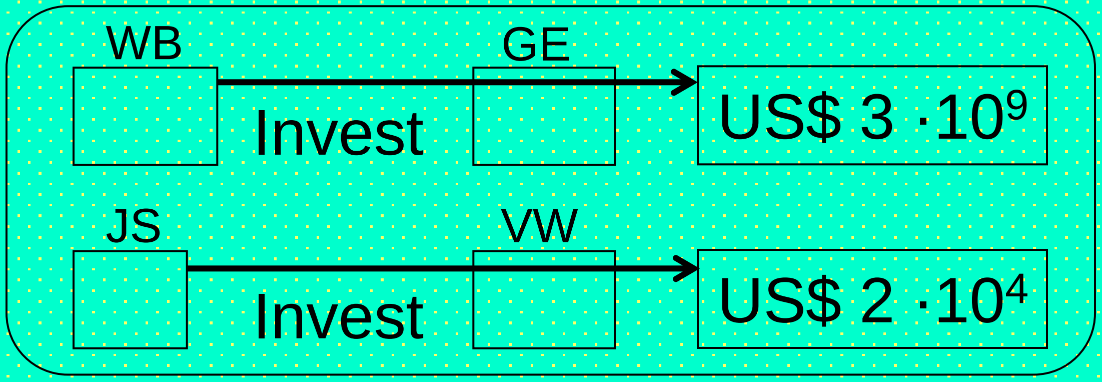
Explicit Conjunction of Formula Graphs: Co-Occurrence in (parallel-processing) And Node

General: Graph (*m* hyperarcs) Logic



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

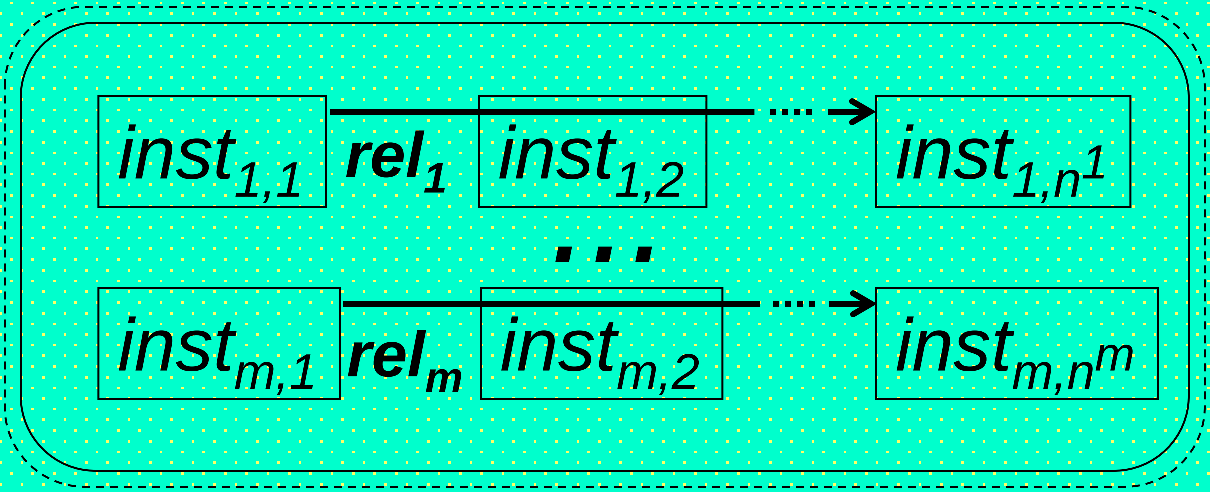
Example: Graph (2 hyperarcs) Logic



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

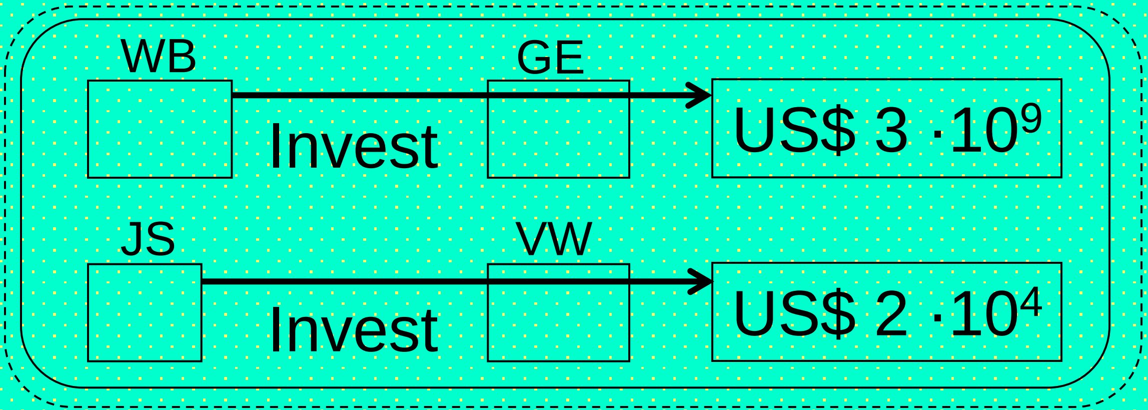
Not of And of Formula Graphs: Co-Occurrence in a Not's And Node

General: Graph (outer *dashed*) Logic



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

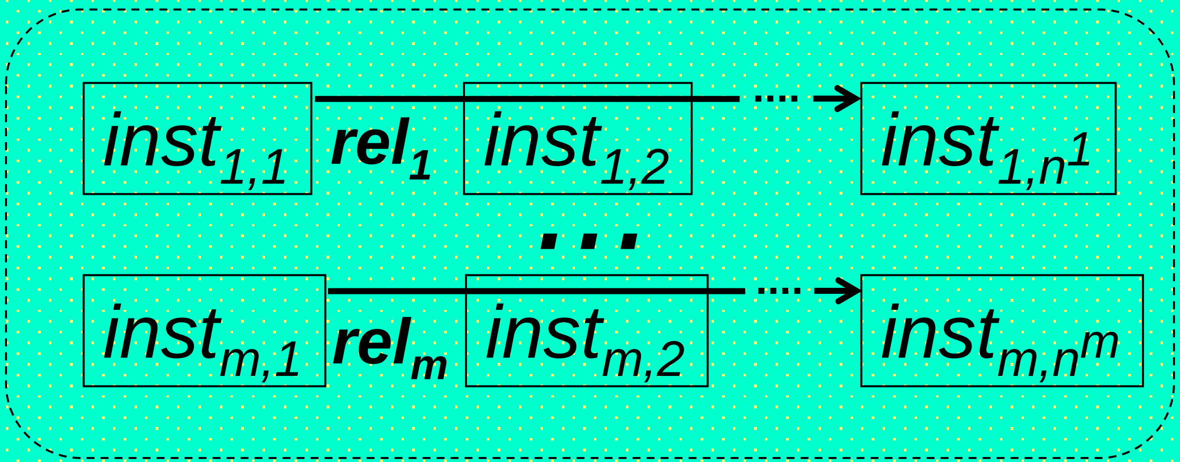
Example: Graph Logic



$$\neg(\text{Invest}(|\text{WB}, |\text{GE}, \text{US\$ } 3 \cdot 10^9) \wedge \text{Invest}(|\text{JS}, |\text{VW}, \text{US\$ } 2 \cdot 10^4))$$

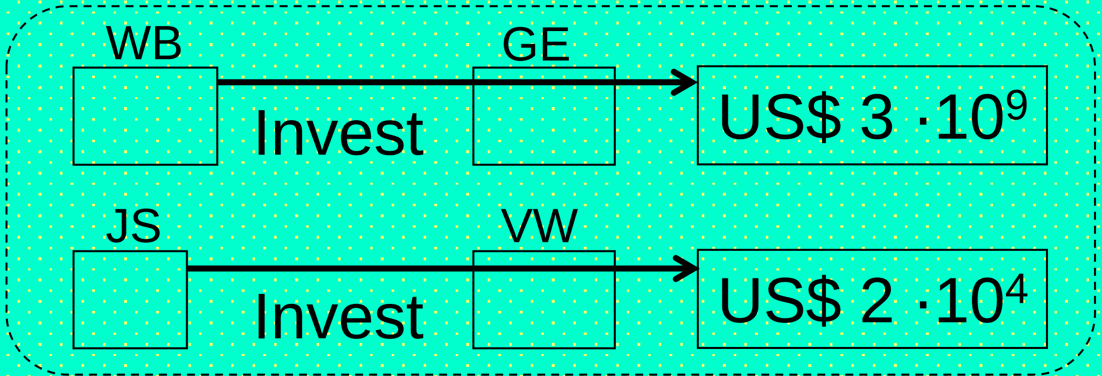
Not of And (Nand) of Formula Graphs: Co-Occurrence in Nand Node (Shorthand)

General: Graph (*dashed*) Logic



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

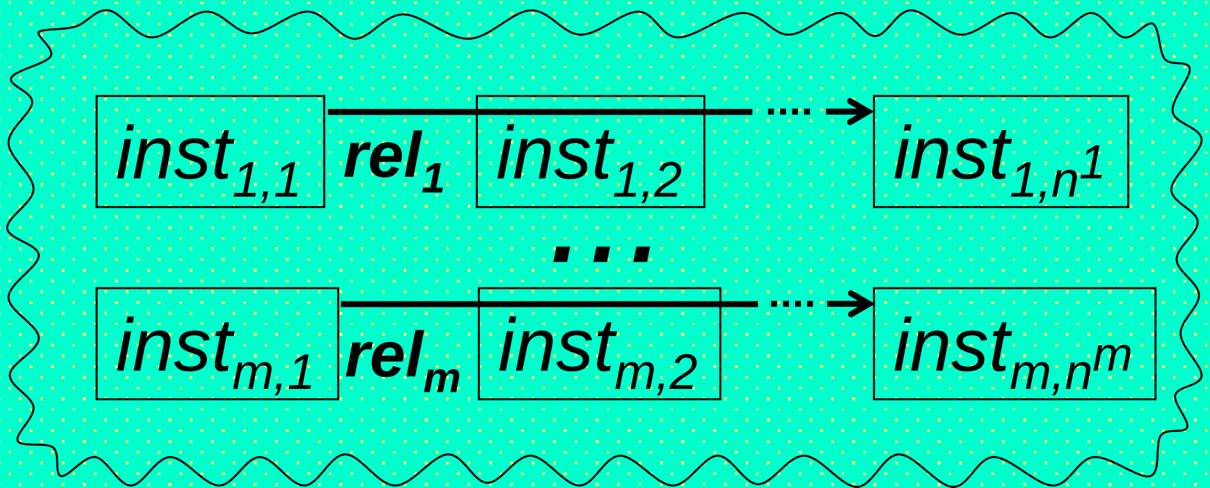
Example: Graph Logic



$$\neg(\text{Invest}(|\text{WB}, |\text{GE}, \text{US\$ } 3\cdot10^9) \wedge \text{Invest}(|\text{JS}, |\text{VW}, \text{US\$ } 2\cdot10^4))$$

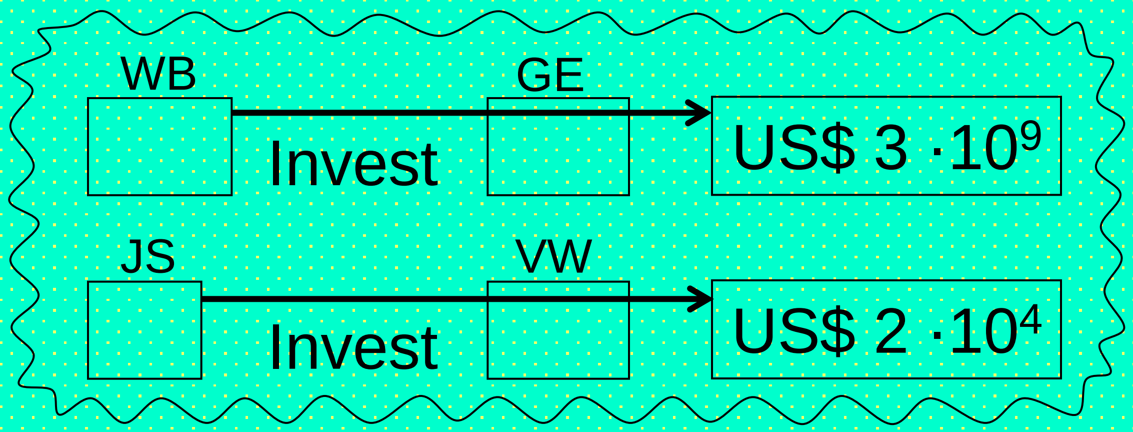
Disjunction of Formula Graphs: Co-Occurrence in Or Node

General: Graph (*wavy*) Logic



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

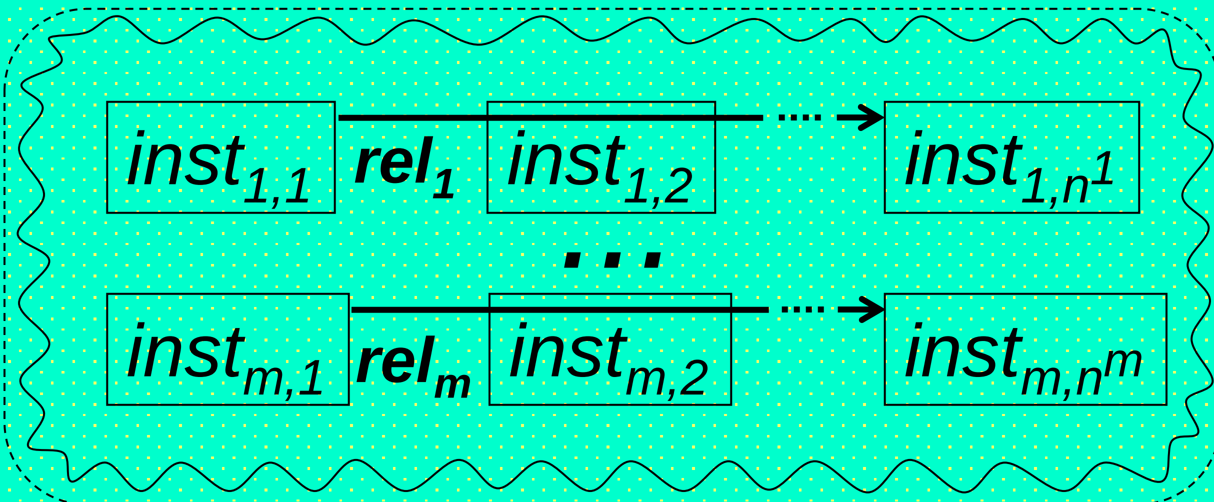
Example: Graph Logic



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

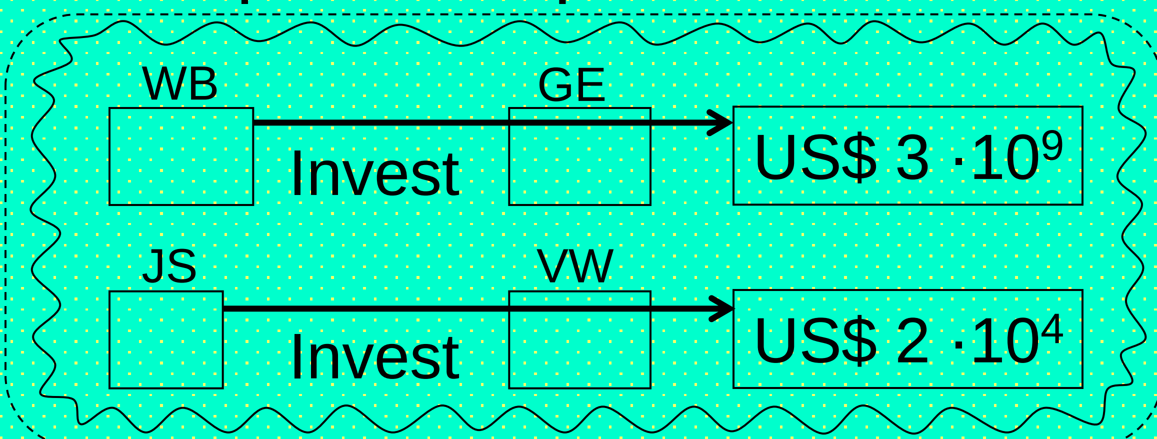
Not of Or of Formula Graphs: Co-Occurrence in a Not's Or Node

General: Graph (outer *dashed*) Logic



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

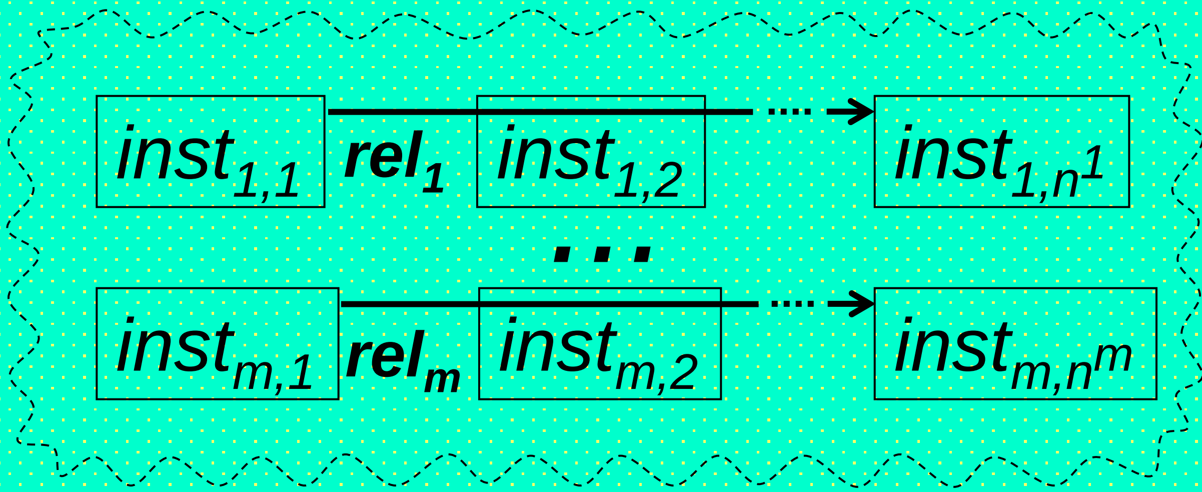
Example: Graph Logic



$$\neg(\text{Invest}(|\text{WB}, |\text{GE}, \text{US\$ } 3 \cdot 10^9) \lor \text{Invest}(|\text{JS}, |\text{VW}, \text{US\$ } 2 \cdot 10^4))$$

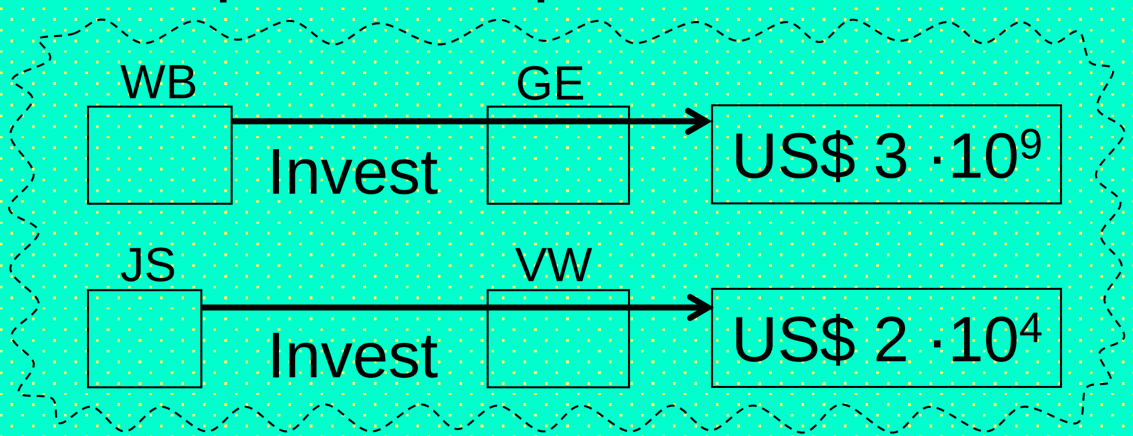
Not of Or (Nor) of Formula Graphs: Co-Occurrence in Nor Node (*Orthogonal* Shorthand)

General: Graph (*dashed*+*wavy*) Logic



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

Example: Graph Logic

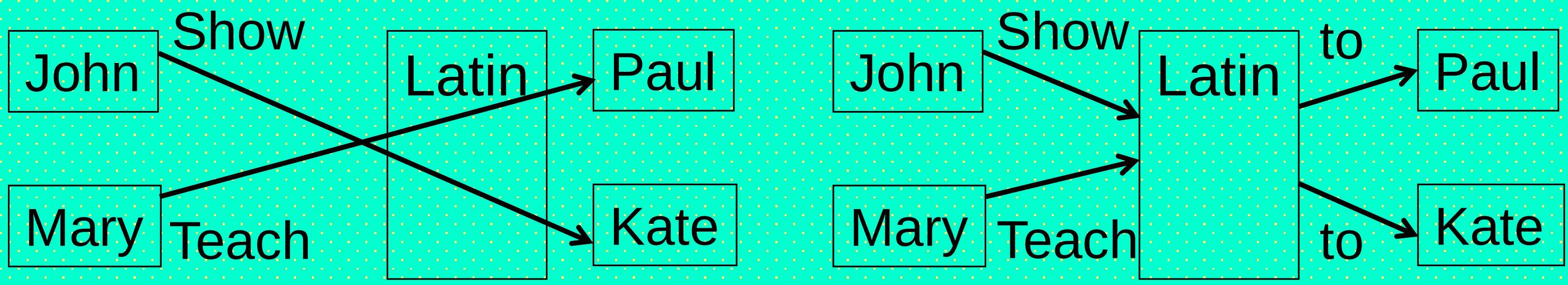


$$\neg(\text{Invest}(|\text{WB}, |\text{GE}, \text{US\$ } 3 \cdot 10^9) \vee \text{Invest}(|\text{JS}, |\text{VW}, \text{US\$ } 2 \cdot 10^4))$$

From Hyperarc Crossings to Node Copies as a Normalization Sequence (1)

Hypergraph (2 hyperarcs, crossing outside nodes)

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



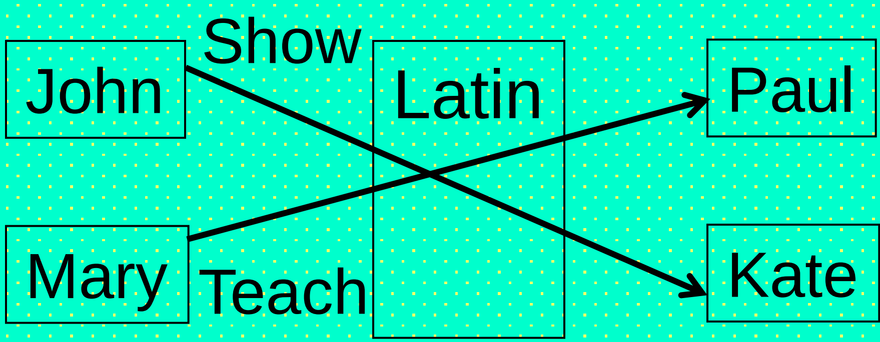
Symbolic Controlled English

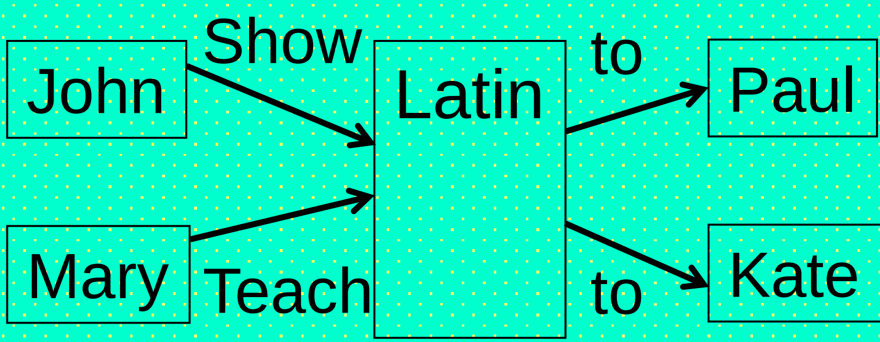
"John shows Latin to Kate.
Mary teaches Latin to Paul."

From Hyperarc Crossings to Node Copies as a Normalization Sequence (1*)

Hypergraph (2 hyperarcs, crossing inside a node)

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

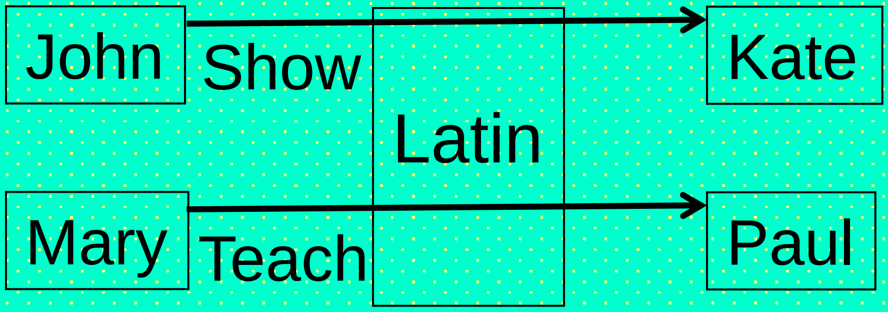


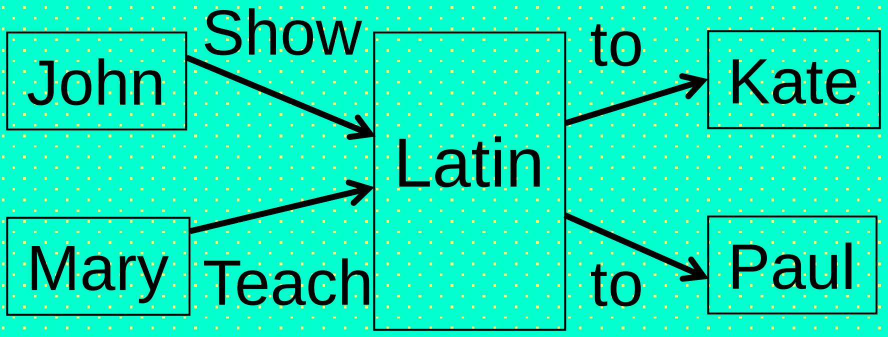


From Hyperarc Crossings to Node Copies as a Normalization Sequence (1**)

Hypergraph (2 hyperarcs, parallel-cutting a node)

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

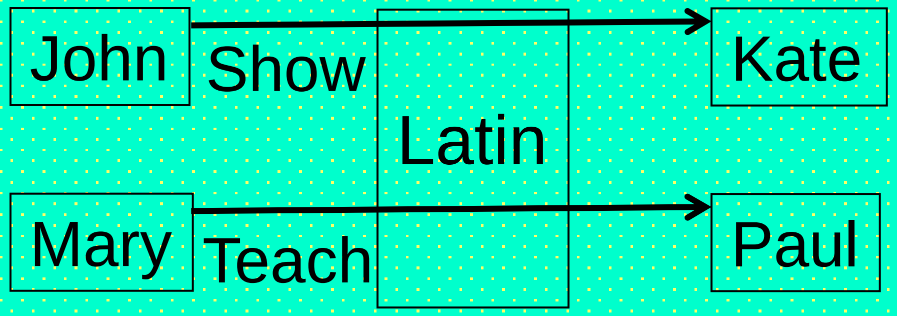




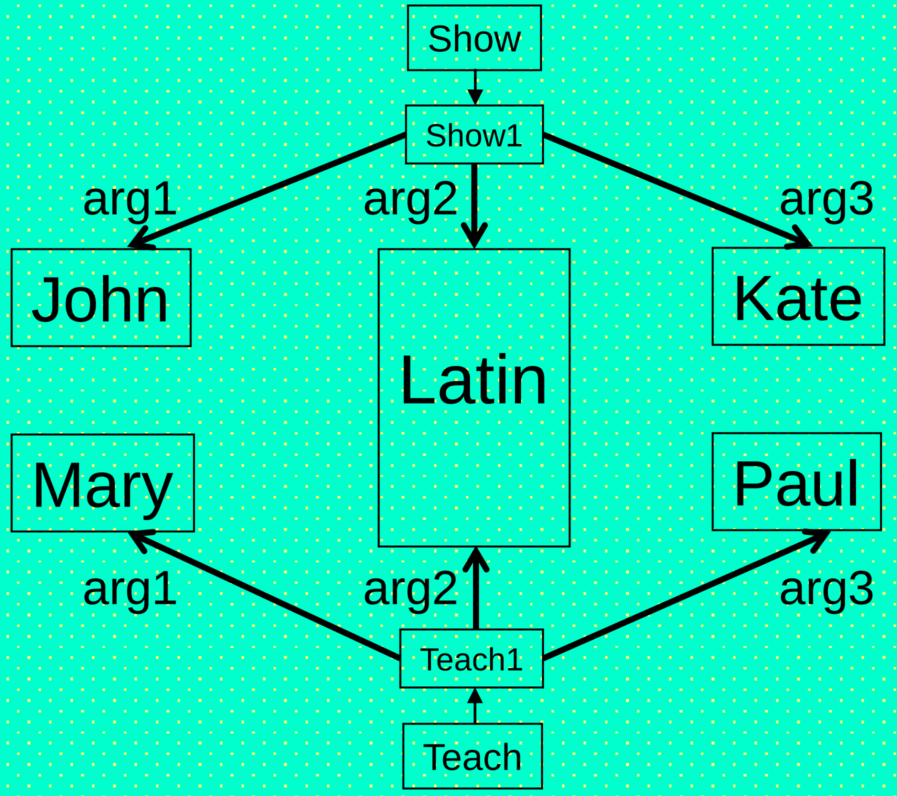
The hyperarc for, e.g., ternary Show(John,Latin,Kate) can be seen as the path composition of 2 arcs for binary Show(John,Latin) and binary to(Latin,Kate).

From Hyperarc Crossings to Node Copies — Insert on Correct Binary Reduction

Hypergraph (2 hyperarcs, parallel-cutting a node)

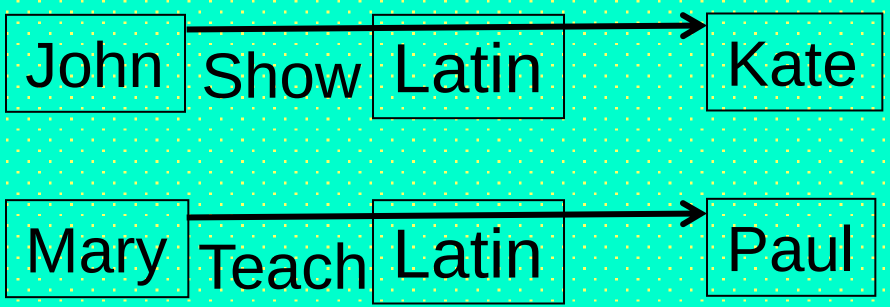


DLG (8 arcs with 4 'reified' relation/ship nodes to point to arguments)



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)

$$\text{Show(John, Latin, Kate)} \wedge \text{Teach(Mary, Latin, Paul)}$$

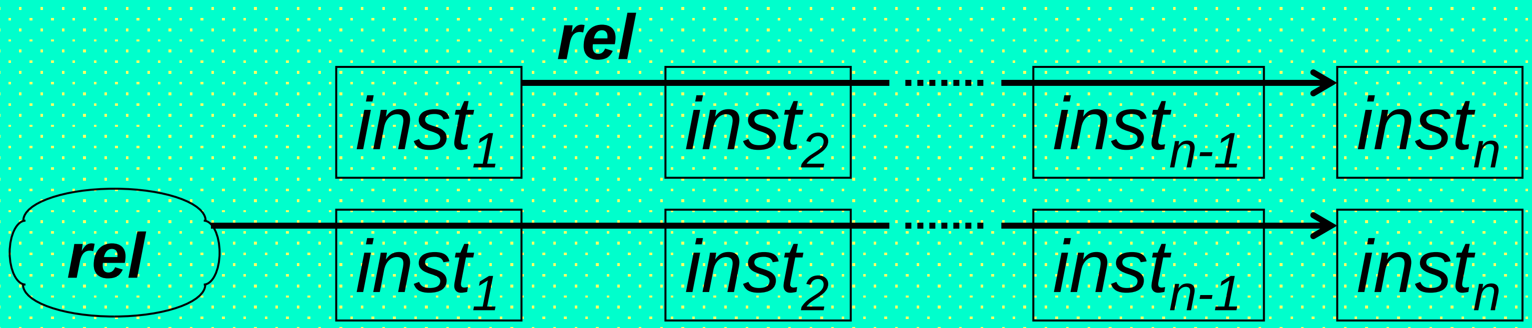
Both 'Latin' occurrences remain one node even when copied for easier layout:
As a deep name, 'Latin' will remain unique

From Predicate Labels on Hyperarcs to Labelnodes Starting Hyperarcs

General: Graph (*hyper*arc with *rectoval*-shaped labelnode)

Logic

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



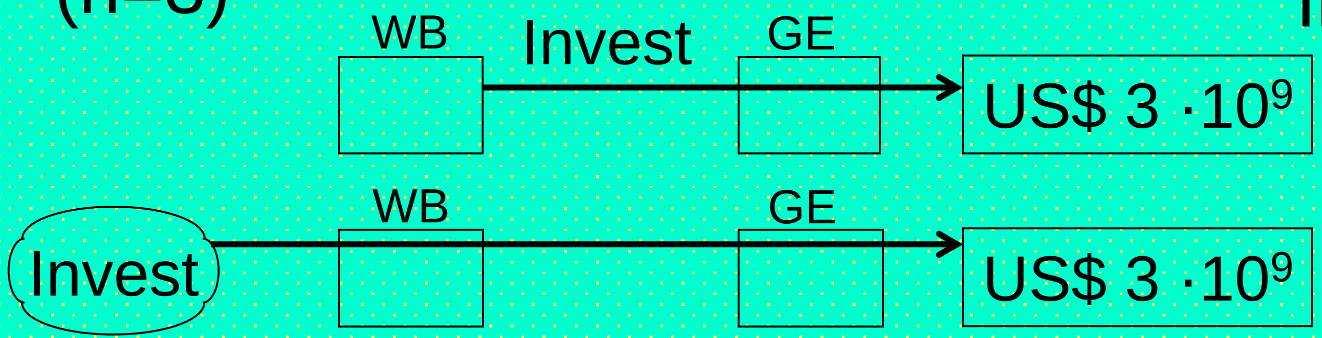
(Shorthand)

(Normal Form)

Example: Graph
(n=3)

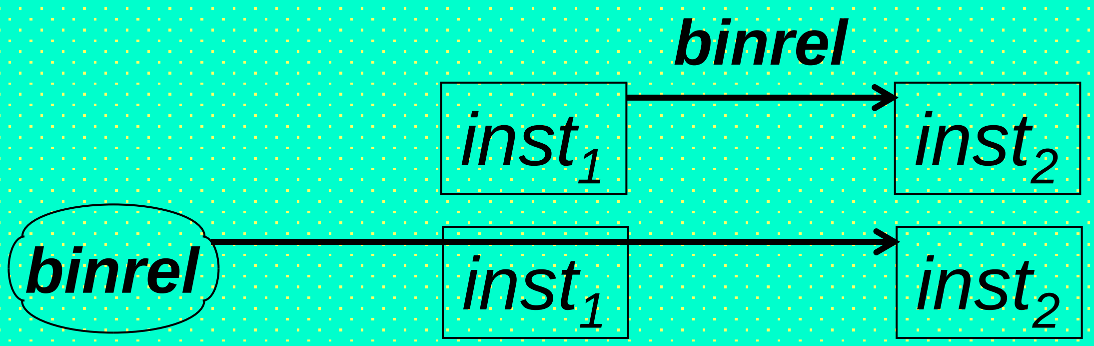
Logic

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



From Predicate Labels on Arcs to Labelnodes Starting Binary Hyperarcs

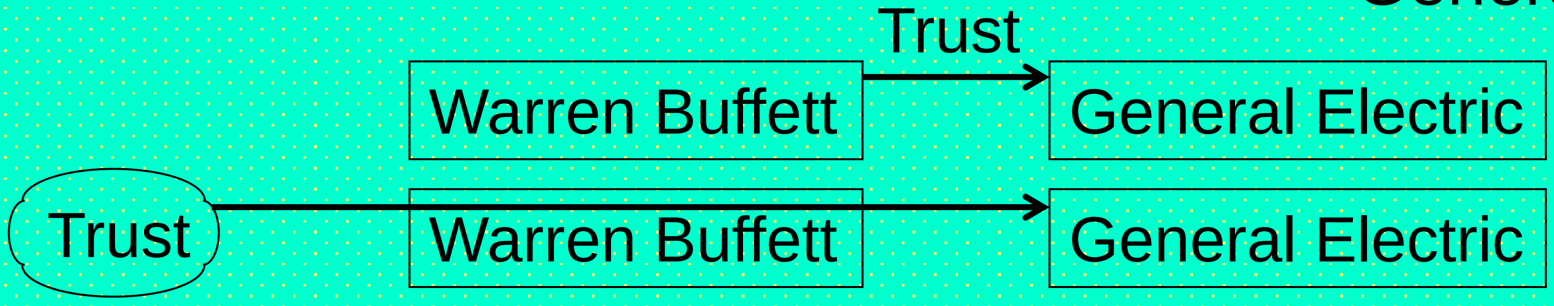
General: Graph (arc) Logic



binrel($inst_1$, $inst_2$)

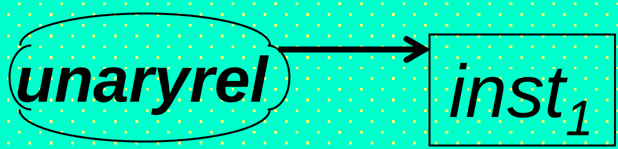
Example: Graph Logic

Trust(Warren Buffett,
General Electric)



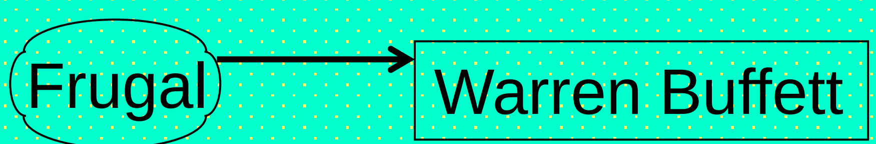
Arities Smaller than in Binary DLGs: Labelnodes Starting Unary Hyperarcs (cf. Slide on Classes, Concepts, Types)

General: Graph Logic



unaryrel(inst$_1$*)*

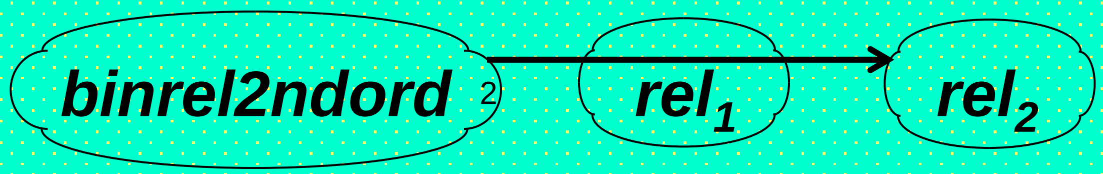
Example: Graph Logic



Frugal(Warren Buffett)

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

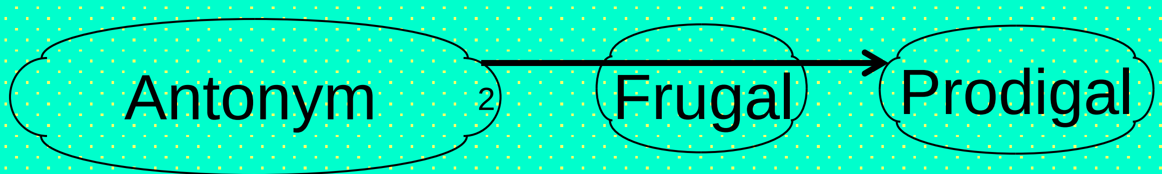
General: Graph (*2-indexed rect-oval*: 2nd order)



Logic

binrel2ndord(rel_1, rel_2)

Example: Graph



Logic

Antonym(Frugal, Prodigal)

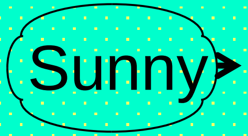
Arities Smaller than in Binary DLGs: Labelnodes for Nullary Hyperarcs (cf. Propositional Logic)

General: Graph

Logic

nullaryrel()

Example: Graph



Logic

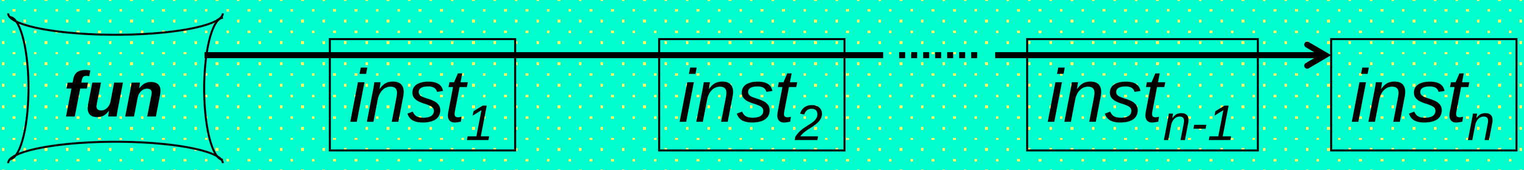
Sunny()

Functions (n-ary)

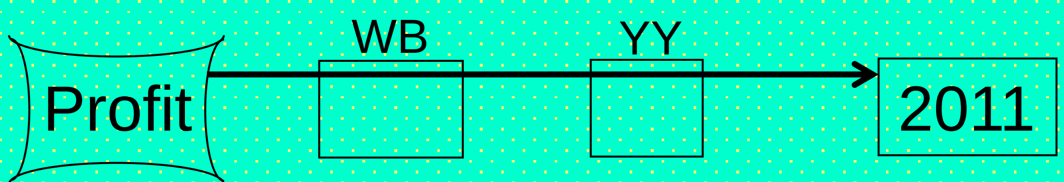
General: Graph (*hyper*arc with *rectstar*-shaped labelnode)

Logic

$fun(inst_1, inst_2, \ldots, inst_{n-1}, inst_n)$



Example: Graph
(n=3)



Logic

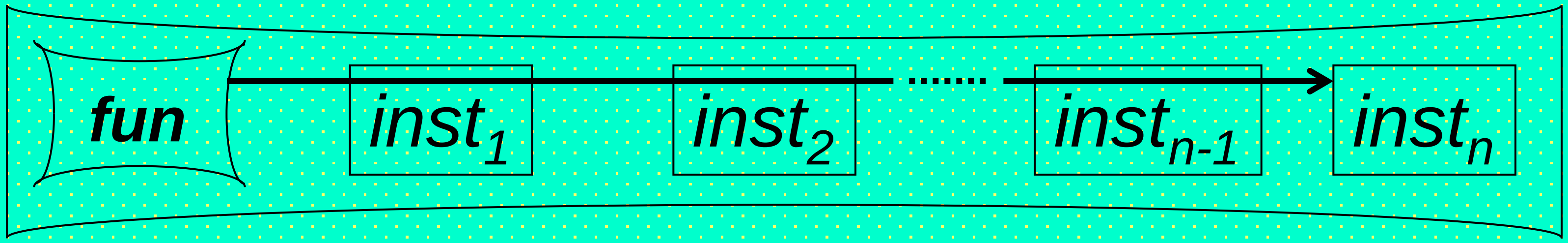
Profit(|WB,
|YY,
2011)

Functions (n-ary) — Value-Denoting: ‘Passive’ Data Construction

General: Graph (*recttie*-shaped enclosing box)

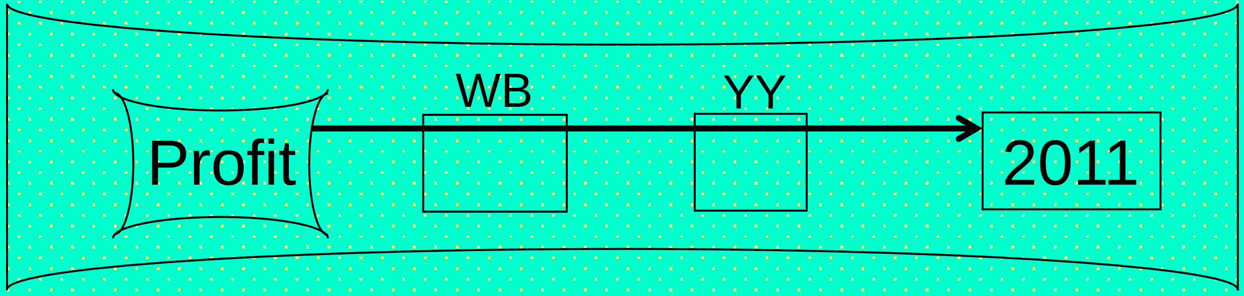
Logic (POSL)

fun[inst$_1$*, inst*$_2$*, …, inst*$_{n-1}$*, inst*$_n$*]*



Example: (n=3)

Graph



Logic

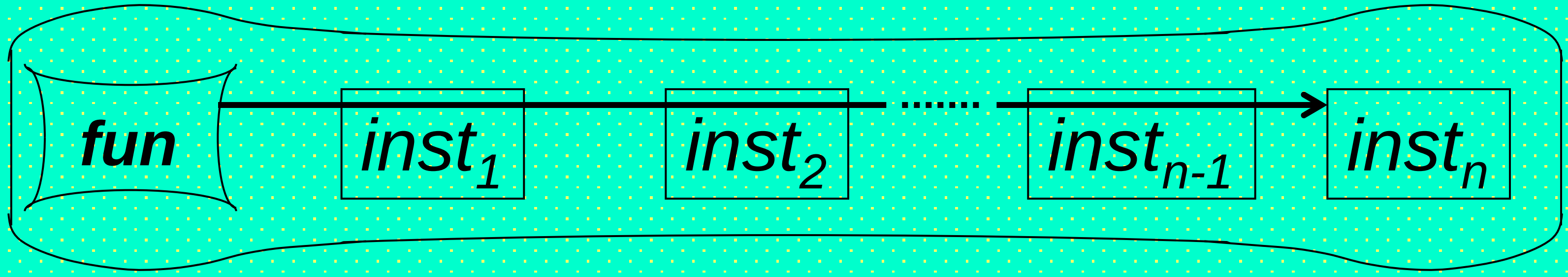
Profit[|WB,
|YY,
2011]

Functions (n-ary) — Value-Returning: 'Active' Call/Query (Content-Addressable)

General: Graph (*roundtie*-shaped enclosing box)

Logic

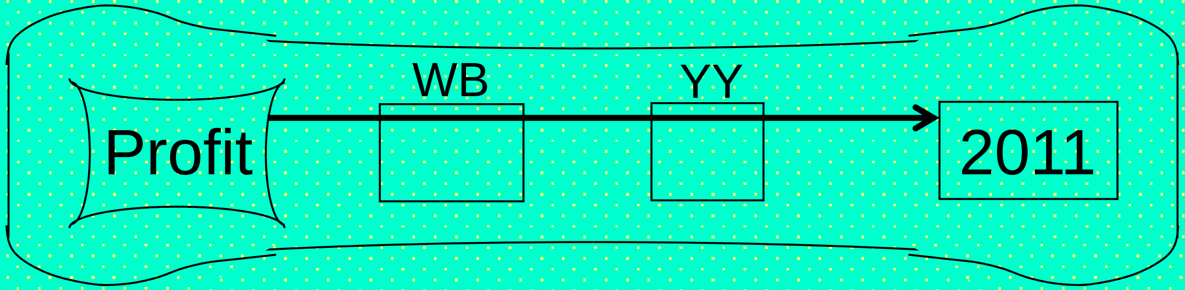
$fun(inst_1, inst_2, \ldots, inst_{n-1}, inst_n)$



Example: (n=3) Graph

Logic

Profit(|WB,
|YY,
2011)



Functions (n-ary) — Value-Returning: Result for Definition of Next Slide

General: Graph

val

Logic

val

Example: Graph
(n=3)

$\boxed{\text{US\$ } 2 \cdot 10^6}$

Logic

US\$ $2 \cdot 10^6$

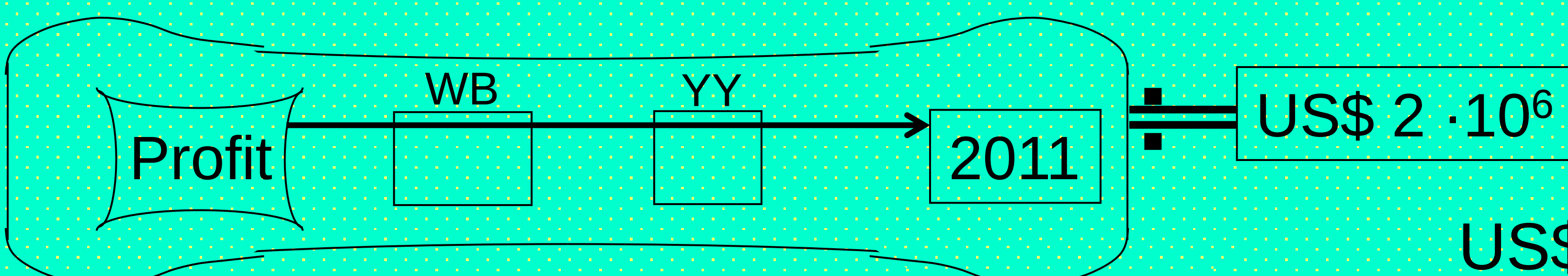
Functions (n-ary) — Value-Returning: Logic with Equality Definition (1)

General: Graph (ground) Logic

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

Example: (n=3) Graph Logic

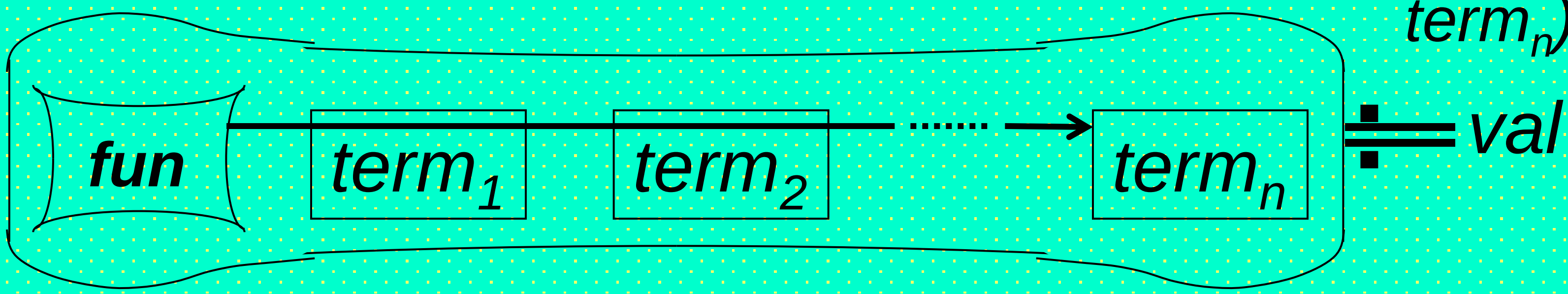
Profit(|WB, |YY, 2011) := US$ $2 \cdot 10^6$



Functions (n-ary) — Value-Returning: Logic with Equality Definition (2)

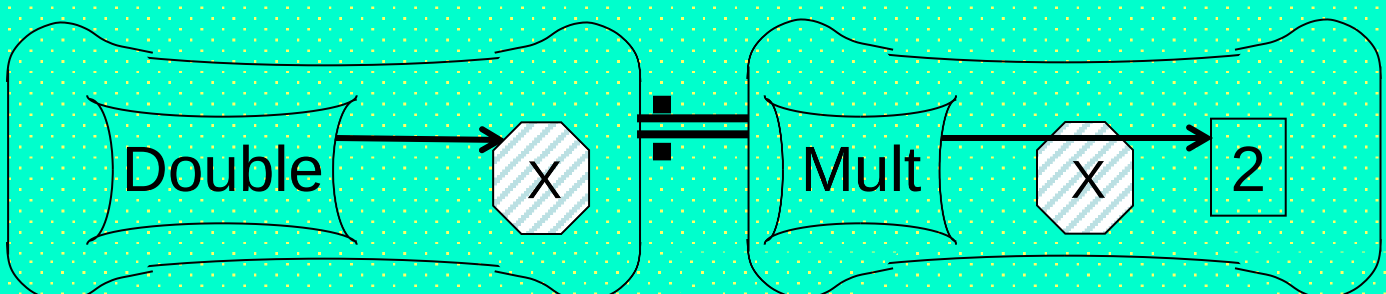
General: Graph (inst/var terms) Logic

$(\forall var_i)$
$fun(term_1, term_2, \ldots, term_n) := val$



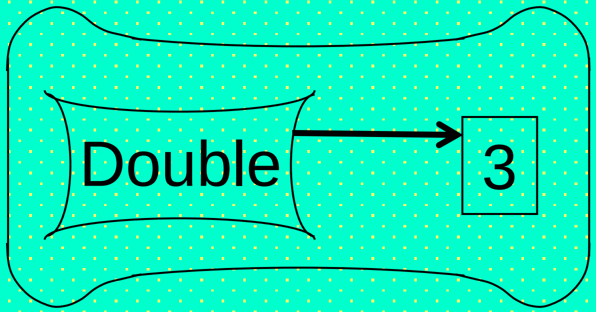
Example: (n=1) Graph Logic

$(\forall\ X)$
Double(X) := Mult(X, 2)



Double Function Sample Call/Query: Rewriting Trace (1)

Graph



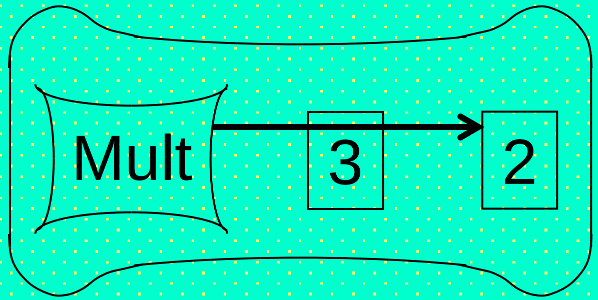
Logic

Double(3)

Call/query of Double instantiates equality definition of previous slide (X=3)

Double Function Sample Call/Query: 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/Query: Rewriting Trace (1'')

Graph	Logic
$\boxed{6}$	6

More in slides about [Functional-Logic Programming]() with (oriented) equations

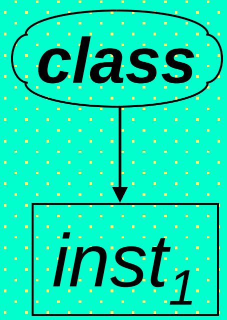
Graphical Elements: Arrows (2)

- Arrows for special arcs and hyperarcs
  - $\downarrow$ – 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
  - $\uparrow$ – SubClassOf: Connects subclass, unlabeled, with superclass (arc, i.e. of length 2)
  - $\Uparrow$ – Implies: Hyperarc from premise(s) to conclusion
  - $\bullet\!\!\nearrow$ – Object-IDentified slots and shelves: Bulleted arcs and hyperarcs

Predicates: Unary Relations (Classes, Concepts, Types)

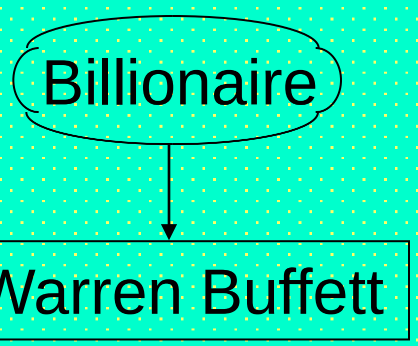
General: Graph (class applied to instance node) — Logic

HasInstance



$class(inst_1)$

Example: Graph — Logic



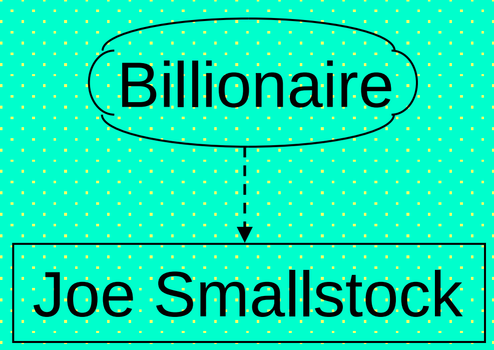
Billionaire(
Warren Buffett)

Negated Predicates: Unary Relations

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

$\neg\ class(inst_1)$

Example: Graph Logic

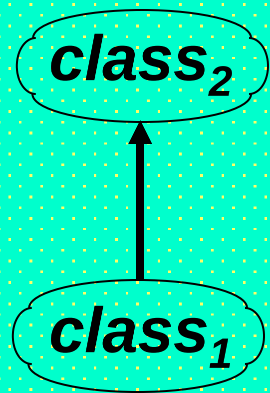


$\neg$ Billionaire(Joe Smallstock)

Class Hierarchies (Taxonomies): Subclass Relation

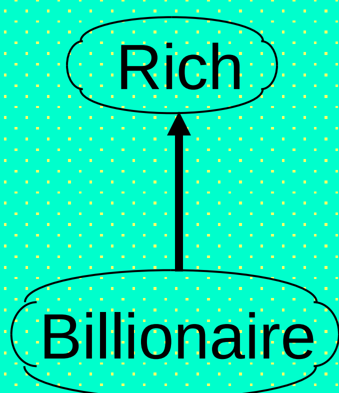
General: Graph (two nodes) (Description) Logic

↑ **SubClassOf**



$class_1 \sqsubseteq class_2$

Example: Graph (Description) Logic

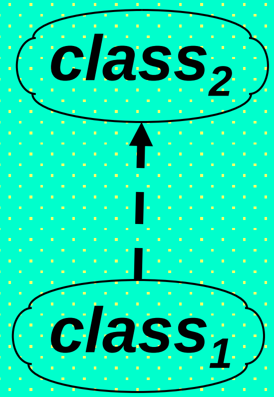


Billionaire $\sqsubseteq$ Rich

Class Hierarchies (Taxonomies): Negated Subclass Relation

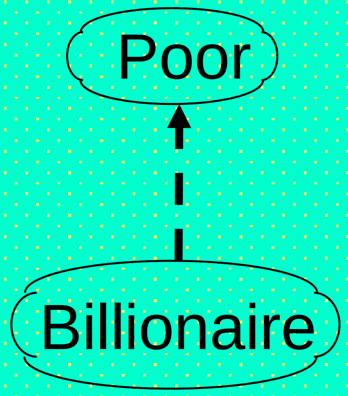
General: Graph (two nodes) (Description) Logic

not SubClassOf



$class_1 \nsubseteq class_2$

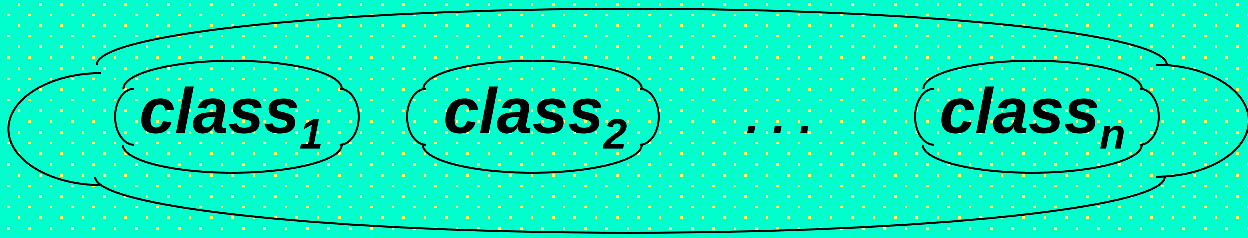
Example: Graph (Description) Logic



$\text{Billionaire} \nsubseteq \text{Poor}$

Intensional-Class Constructions (Ontologies): Class Intersection

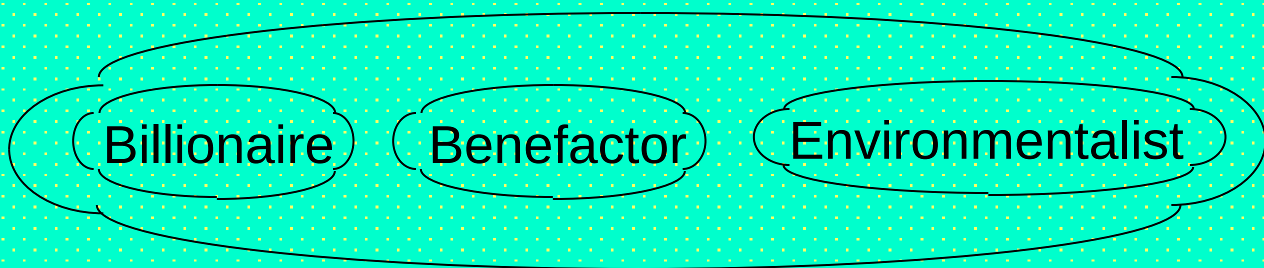
General: Graph (*solid* node, as for conjunction)



(Description) Logic

$class_1 \sqcap$
$class_2 \sqcap$
$\ldots \sqcap$
$class_n$

Example: Graph

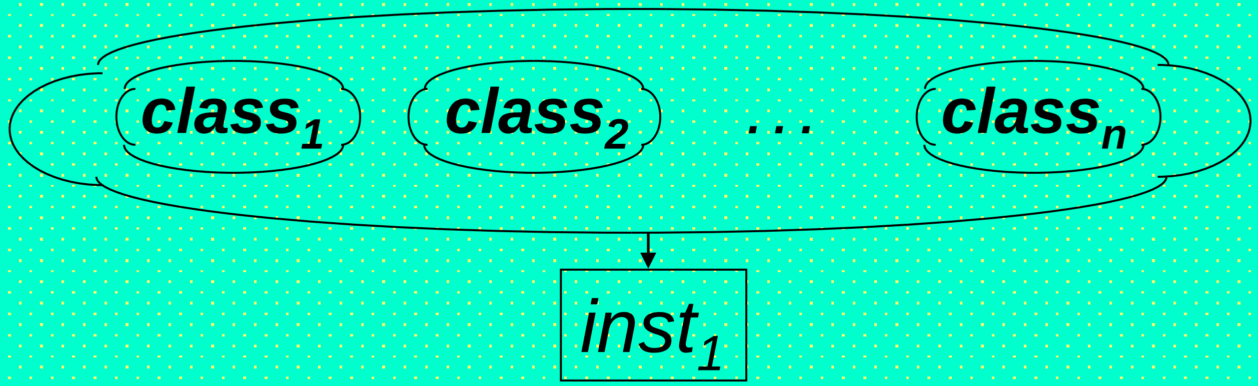


(Description) Logic

Billionaire ⊓
Benefactor ⊓
Environmentalist

Intensional-Class Applications: Class Intersection

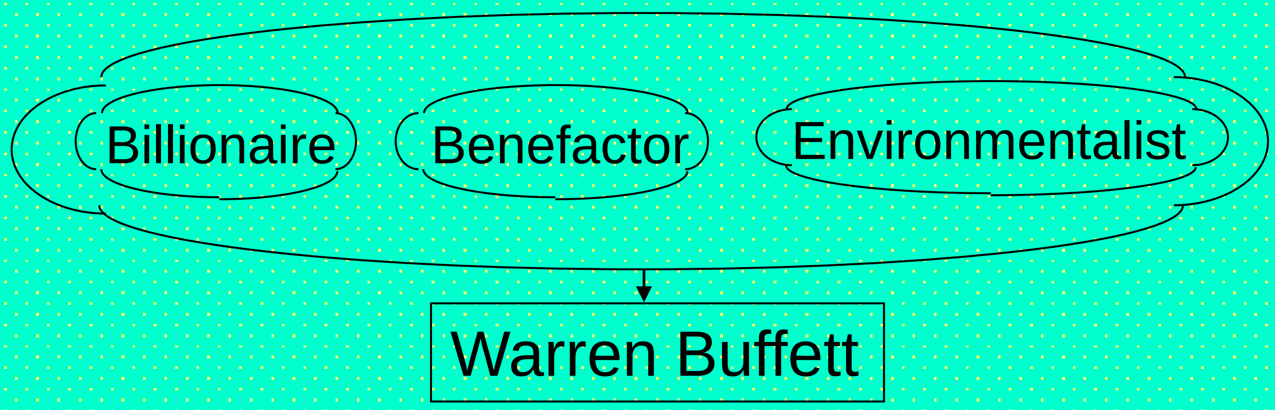
General: Graph (*complex* class applied to instance node)



(Description) Logic

$(class_1 \sqcap$
$class_2 \sqcap$
$\ldots \sqcap$
$class_n)$
$(inst_1)$

Example: Graph

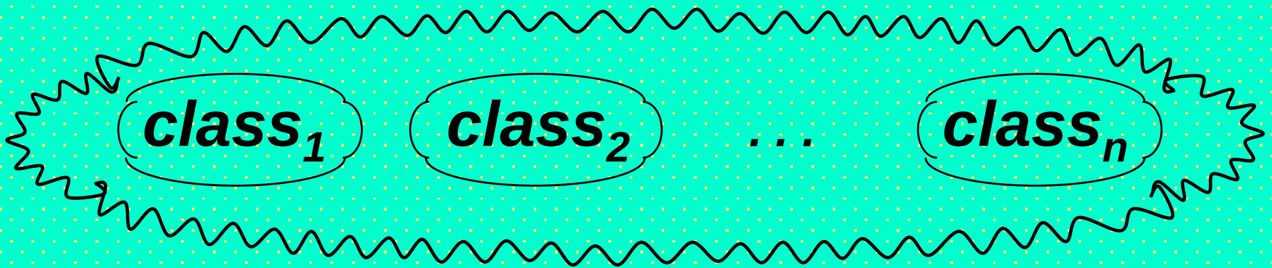


(Description) Logic

(Billionaire ⊓
Benefactor ⊓
Environmentalist)
(Warren Buffett)

Intensional-Class Constructions (Ontologies): Class Union

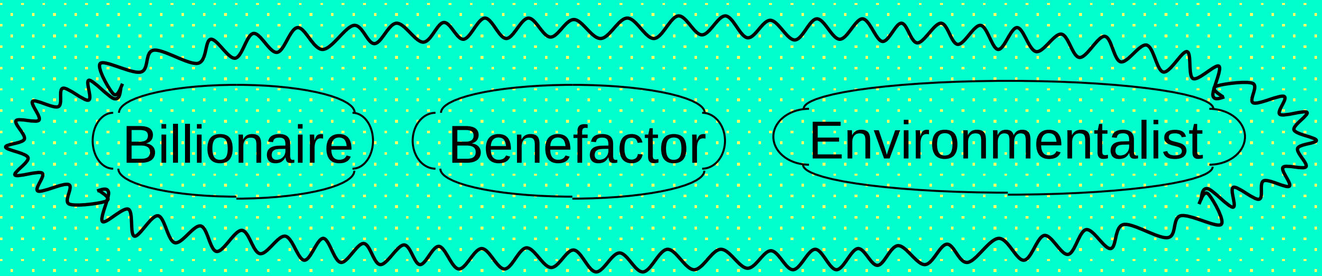
General: Graph (*wavy* node, as for disjunction)



(Description) Logic

$$class_1 \sqcup class_2 \sqcup \ldots \sqcup class_n$$

Example: Graph

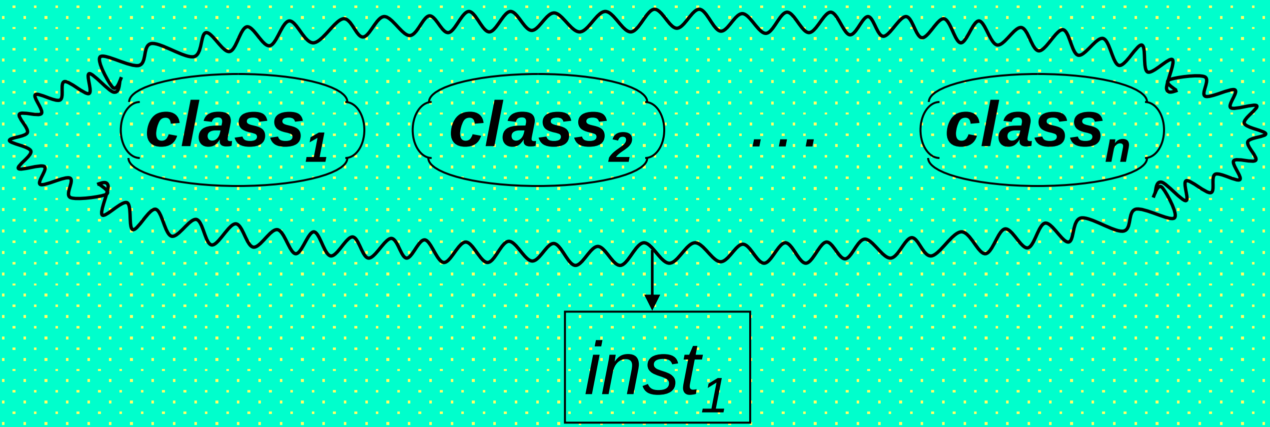


(Description) Logic

Billionaire ⊔ Benefactor ⊔ Environmentalist

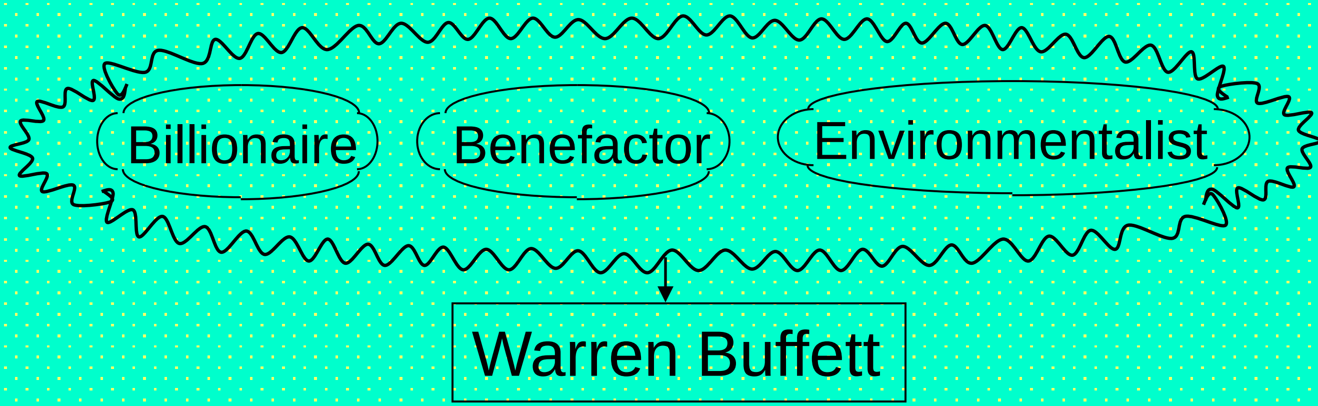
Intensional-Class Applications: Class Union

General: Graph (*complex* class applied to instance node) — (Description) Logic



$(class_1 \sqcup class_2 \sqcup \ldots \sqcup class_n)(inst_1)$

Example: Graph — (Description) Logic



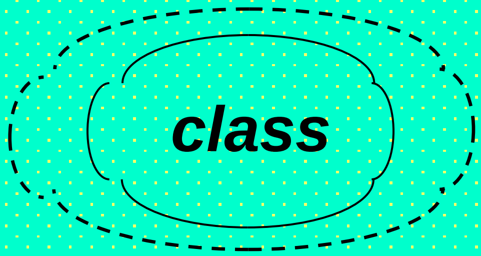
(Billionaire ⊔ Benefactor ⊔ Environmentalist)(Warren Buffett)

Intensional Class Constructions (Ontologies): Class Complement

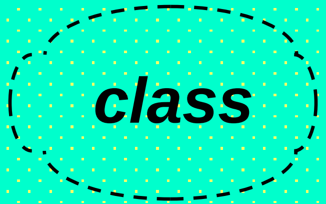
General: Graph
(*dashed* node,
as for negation contains
node to be complemented)

(Description) Logic

Arbitrary class

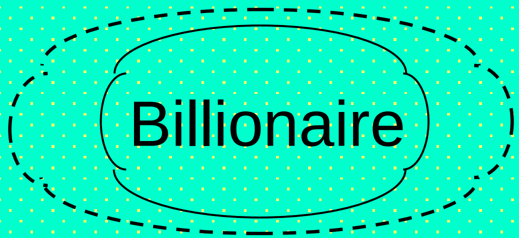


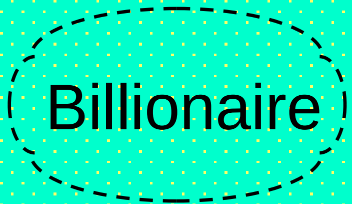
Atomic class
(shorthand)



$\neg$ *class*

Example: Graph





(Description) Logic

$\neg$ Billionaire

Class Hierarchies (Taxonomy DAGs): Top and Bottom

General: Top (*special* node)

$\top$

(Description) Logic

$\top$

(owl:Thing)

General: Bottom (*special* node)

$\bot$

(Description) Logic

$\bot$

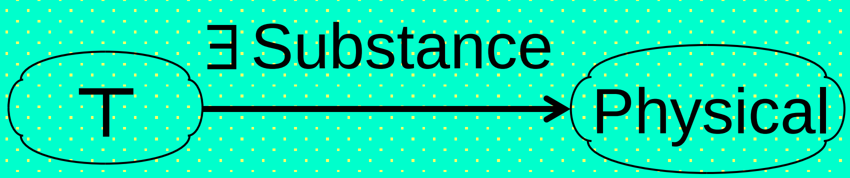
(owl:Nothing)

Intensional Class Constructions (Ontologies): Class-Property-Restricting TBox—Existential (1)

General: Graph (shorthand) (Description) Logic

$\exists$*binrel* **.** *class*

Example: Graph (Description) Logic

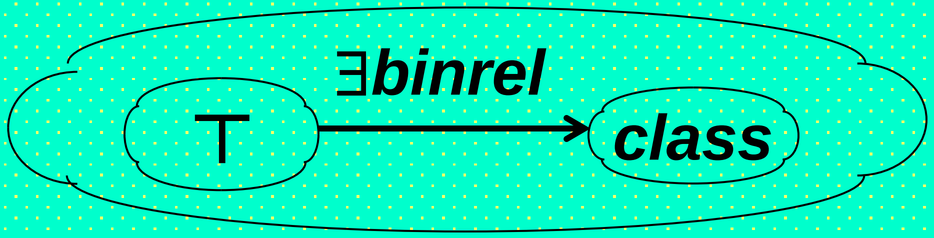


$\exists$Substance **.** Physical

A kind of schema, where Top class is specialized to have (multi-valued) attribute/property, Substance, *with <u>at least one</u> value typed by class* Physical

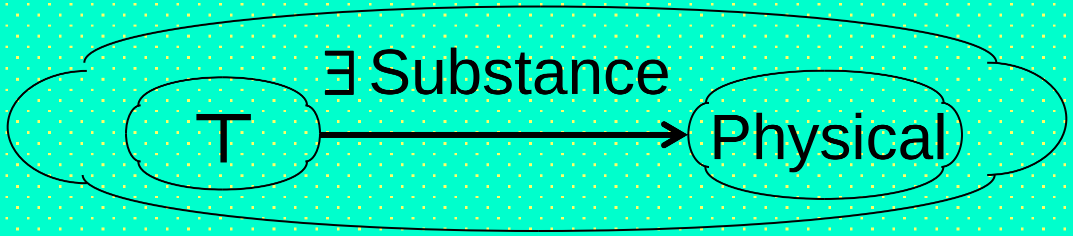
Intensional Class Constructions (Ontologies):
Class-Property-Restricting TBox—Existential (1*)

General: Graph (normal) (Description) Logic



$\exists \mathit{binrel} \, . \, \mathit{class}$

Example: Graph (Description) Logic

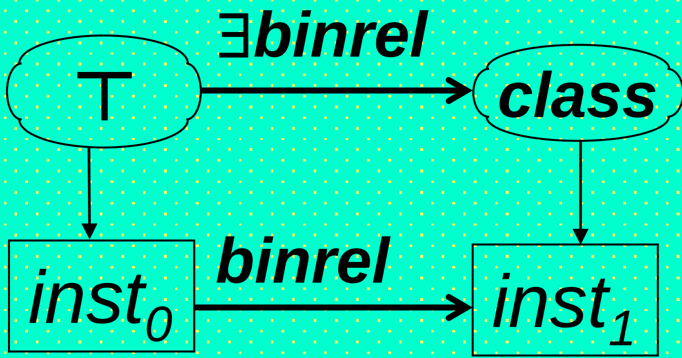


$\exists \mathrm{Substance} \, . \, \mathrm{Physical}$

A kind of schema, where Top class is specialized to have (multi-valued) attribute/property, Substance, *with <u>at least one</u> value typed by class* Physical

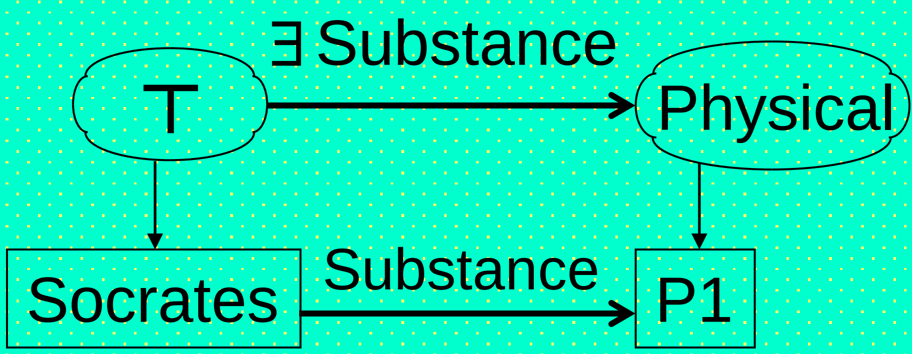
Instance Assertions (Populated Ontologies):
Adding ABox to Restriction TBox—Existential (1)

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



$\exists binrel.class(inst_0) \wedge$
$class(inst_1) \wedge$
$binrel(inst_0, inst_1)$

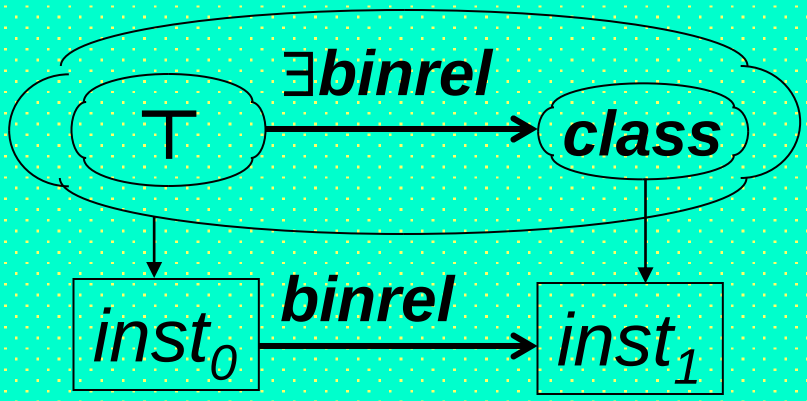
Example: Graph (rUNA-Description) Logic



∃Substance.Physical
(Socrates) ∧
Physical(P1) ∧
Substance(Socrates, P1)

Instance Assertions (Populated Ontologies): Adding ABox to Restriction TBox—Existential (1*)

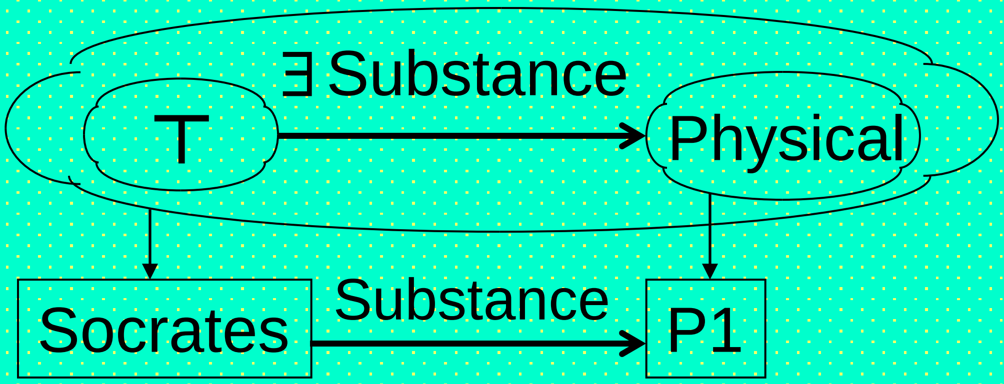
General: Graph (normal)



(rUNA-Description) Logic

$\exists binrel.class(inst_0) \wedge$
$class(inst_1) \wedge$
$binrel(inst_0, inst_1)$

Example: Graph



(rUNA-Description) Logic

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

Intensional Class Constructions (Ontologies): Class-Property-Restricting TBox—Universal (1)

General: Graph (shorthand) (Description) Logic

$$\top \xrightarrow{\forall \boldsymbol{binrel}} \boldsymbol{class} \qquad \forall \mathit{binrel} \, . \, \mathit{class}$$

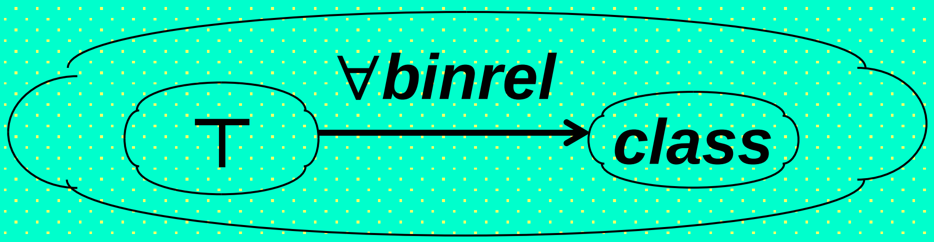
Example: Graph (Description) Logic

$$\top \xrightarrow{\forall \text{ Substance}} \text{Physical} \qquad \forall \text{Substance} \, . \, \text{Physical}$$

A kind of schema, where Top class is specialized to have (multi-valued) attribute/property, Substance, *with <u>each</u> value typed by class* Physical

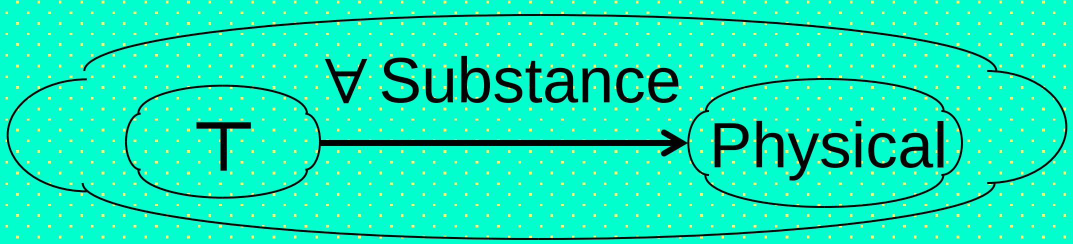
Intensional Class Constructions (Ontologies):
Class-Property-Restricting TBox—Universal (1*)

General: Graph (normal) (Description) Logic



$\forall binrel\,.\,class$

Example: Graph (Description) Logic

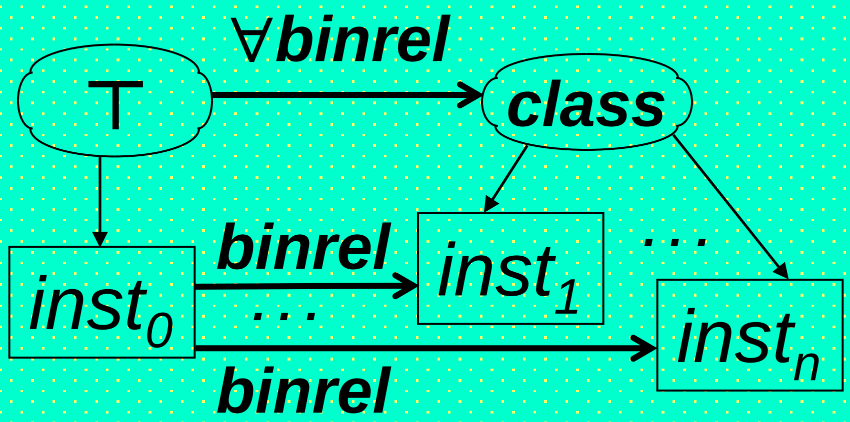


$\forall \text{Substance}\,.\,\text{Physical}$

A kind of schema, where Top class is specialized to have (multi-valued) attribute/property, Substance, *with each value typed by class* Physical

Instance Assertions (Populated Ontologies): Adding ABox to Restriction TBox—Universal (1)

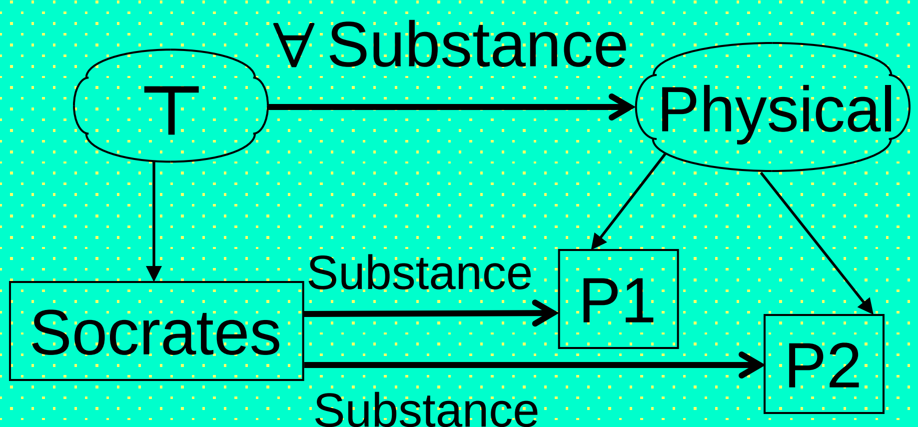
General: Graph (shorthand)



(rUNA-Description) Logic

$\forall binrel.class(inst_0) \wedge$
$class(inst_1) \wedge$
. . .
$class(inst_n) \wedge$
$binrel(inst_0, inst_1) \wedge$
. . .
$binrel(inst_0, inst_n)$

Example: Graph

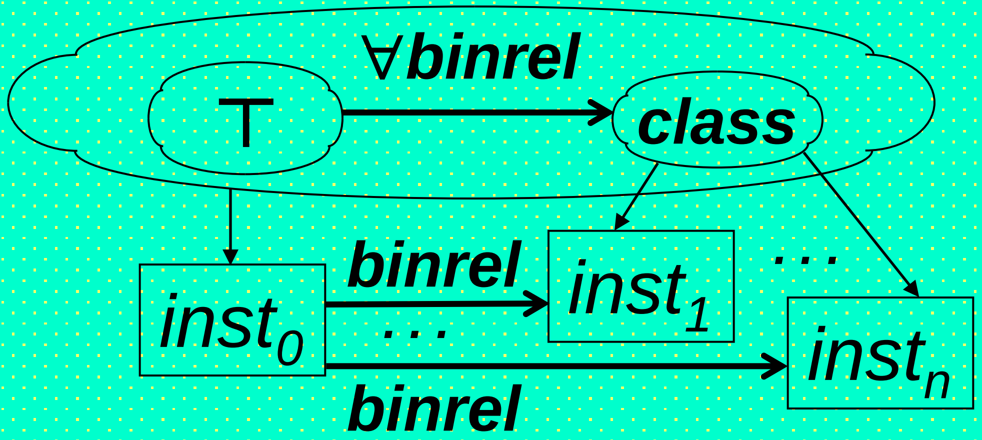


(rUNA-Description) Logic

∀Substance.Physical (Socrates) ∧
Physical(P1) ∧
Physical(P2) ∧
Substance(Socrates, P1) ∧
Substance(Socrates, P2)

Instance Assertions (Populated Ontologies): Adding ABox to Restriction TBox—Universal (1*)

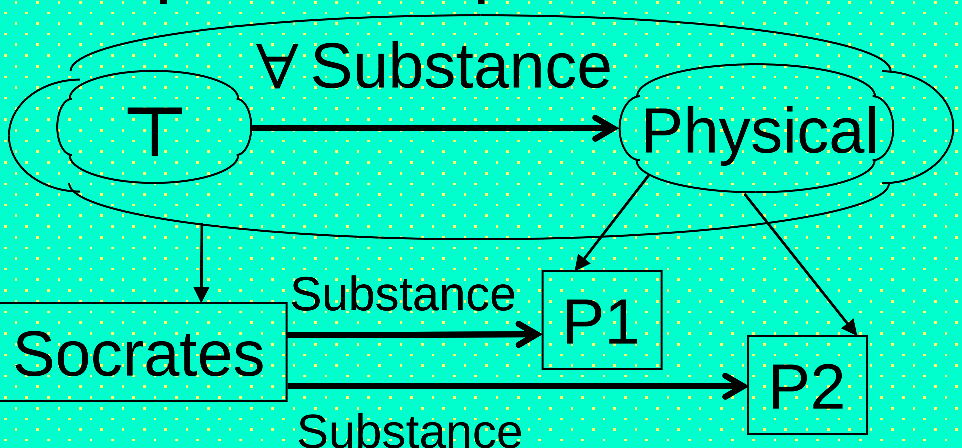
General: Graph (normal)



(rUNA-Description) Logic

$\forall binrel.class(inst_0) \land$
$class(inst_1) \land$
. . .
$class(inst_n) \land$
$binrel(inst_0, inst_1) \land$
. . .
$binrel(inst_0, inst_n)$

Example: Graph



(rUNA-Description) Logic

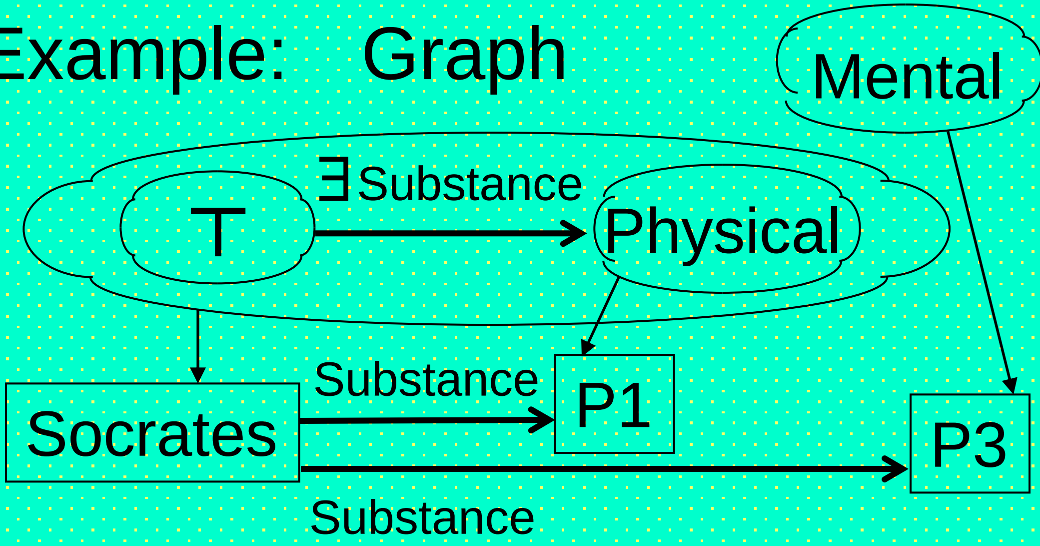
∀Substance.Physical (Socrates) $\land$
Physical(P1) $\land$
Physical(P2) $\land$
Substance(Socrates, P1) $\land$
Substance(Socrates, P2)

Existential vs. Universal Restriction

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

Consistent

Example: Graph

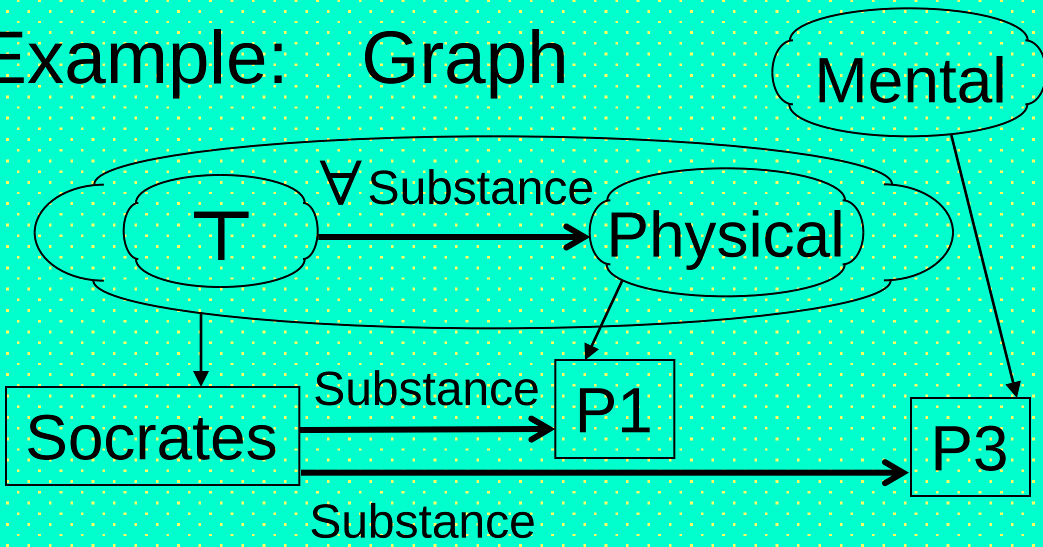


(rUNA-Description) Logic

$\exists Substance.Physical(Socrates) \wedge$
$Physical(P1) \wedge$
$Mental(P3) \wedge$
$Substance(Socrates, P1) \wedge$
$Substance(Socrates, P3)$

Inconsistent

Example: Graph

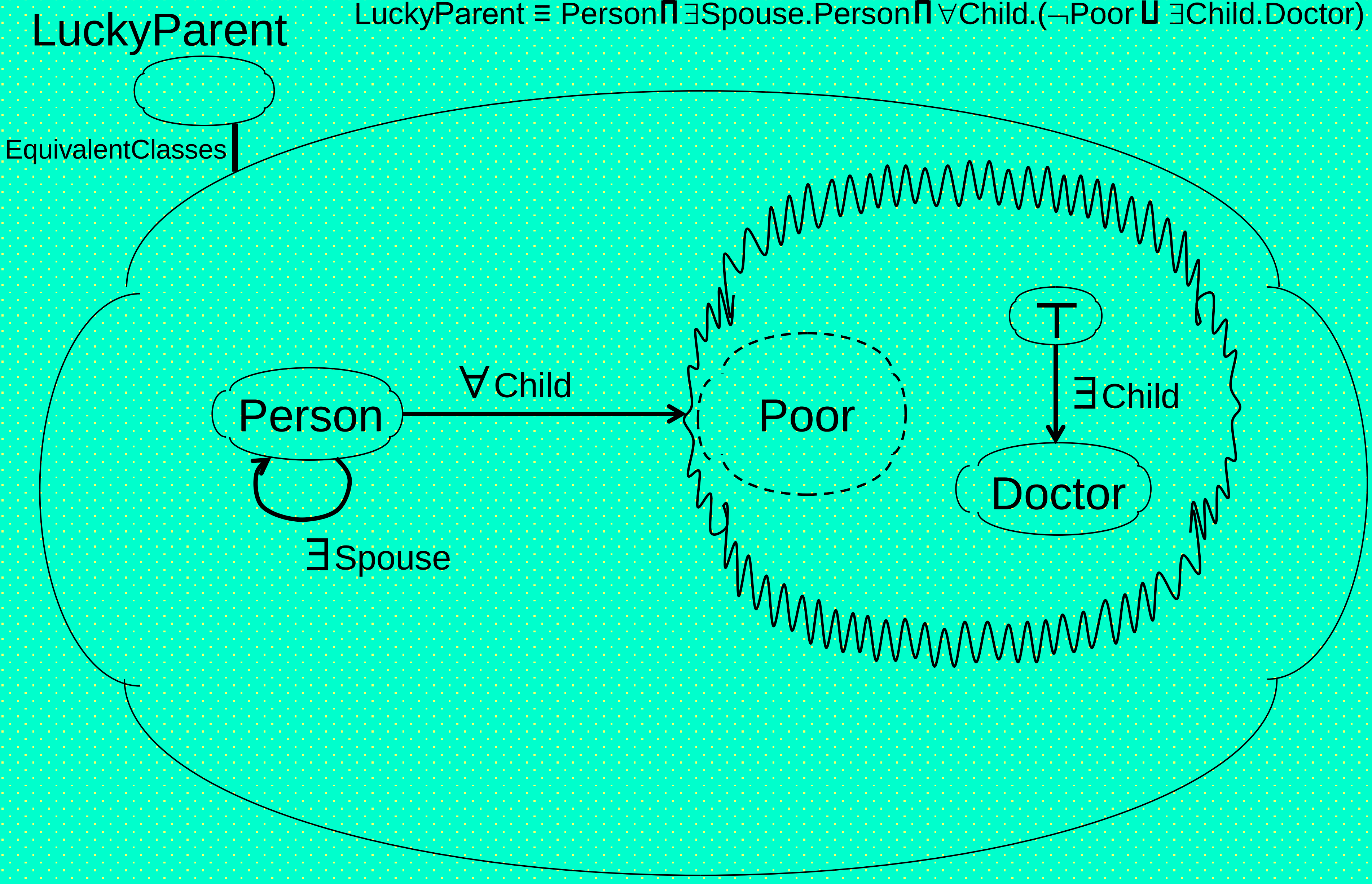


(rUNA-Description) Logic

$\forall Substance.Physical(Socrates) \wedge$
$Physical(P1) \wedge$
$Mental(P3) \wedge$
$Substance(Socrates, P1) \wedge$
$Substance(Socrates, P3)$

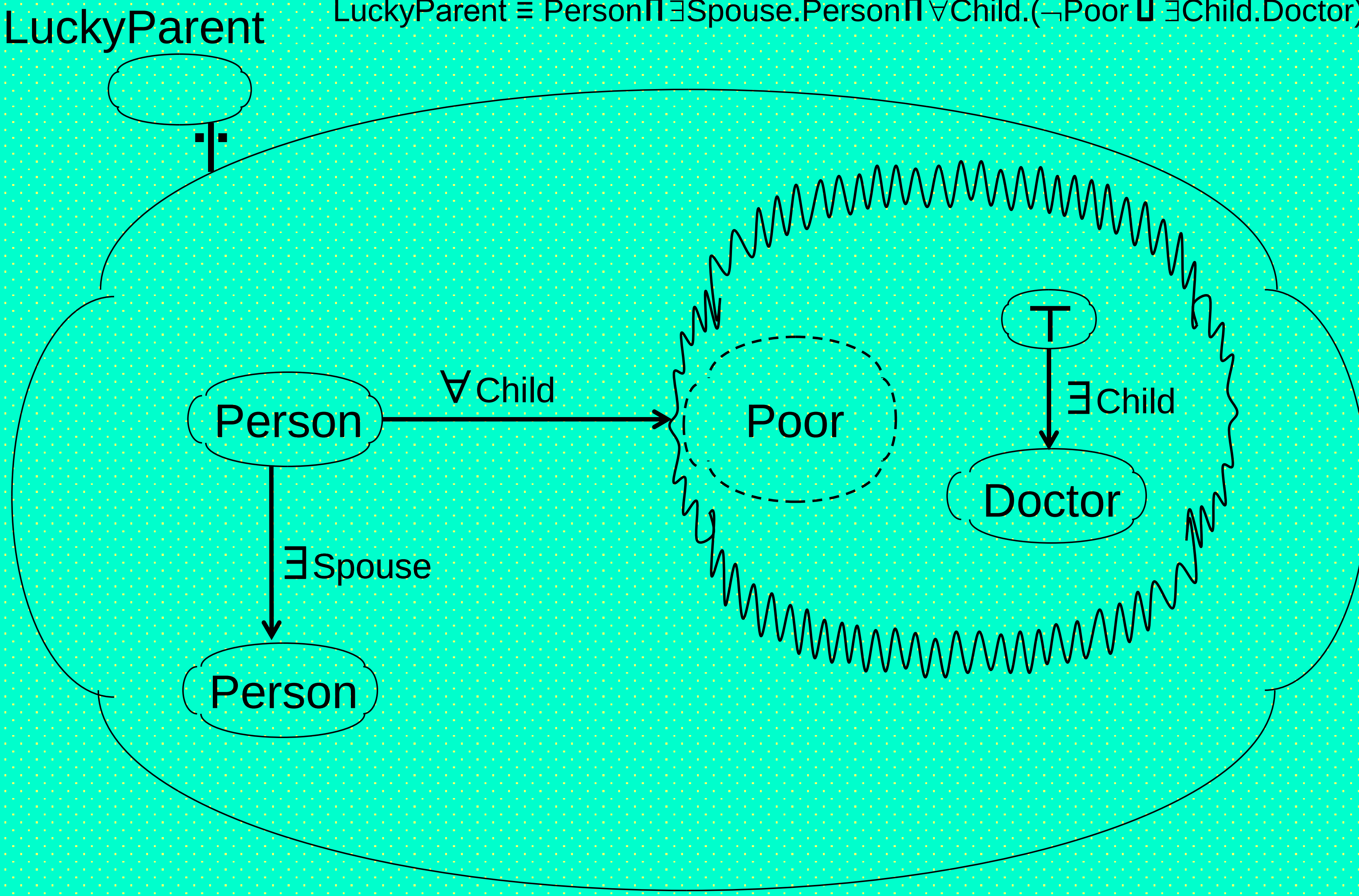
LuckyParent Example (1)

LuckyParent ≡ Person ⊓ ∃Spouse.Person ⊓ ∀Child.(¬Poor ⊔ ∃Child.Doctor)



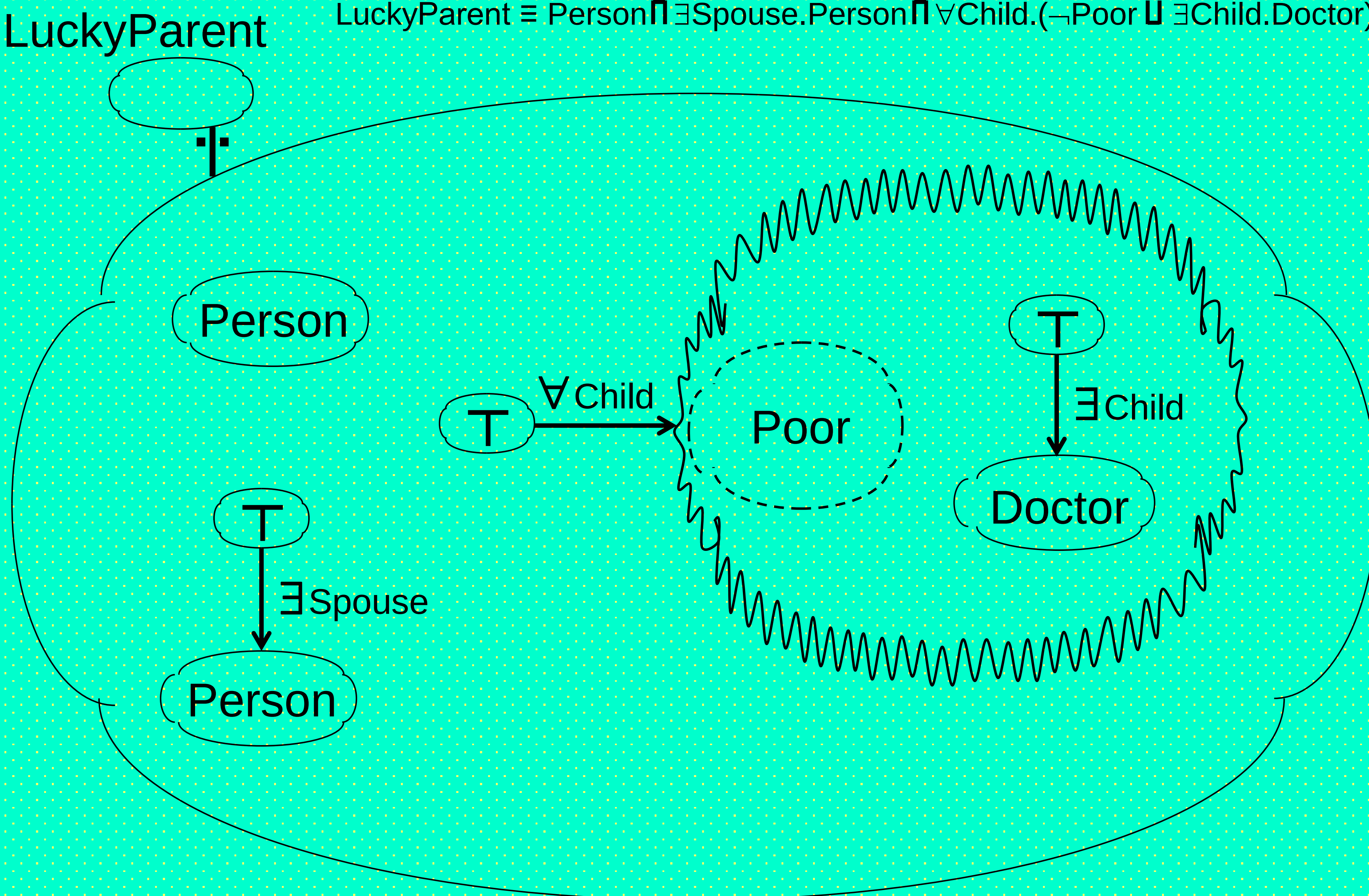
LuckyParent Example (1*)

LuckyParent ≡ Person ⊓ ∃Spouse.Person ⊓ ∀Child.(¬Poor ⊔ ∃Child.Doctor)



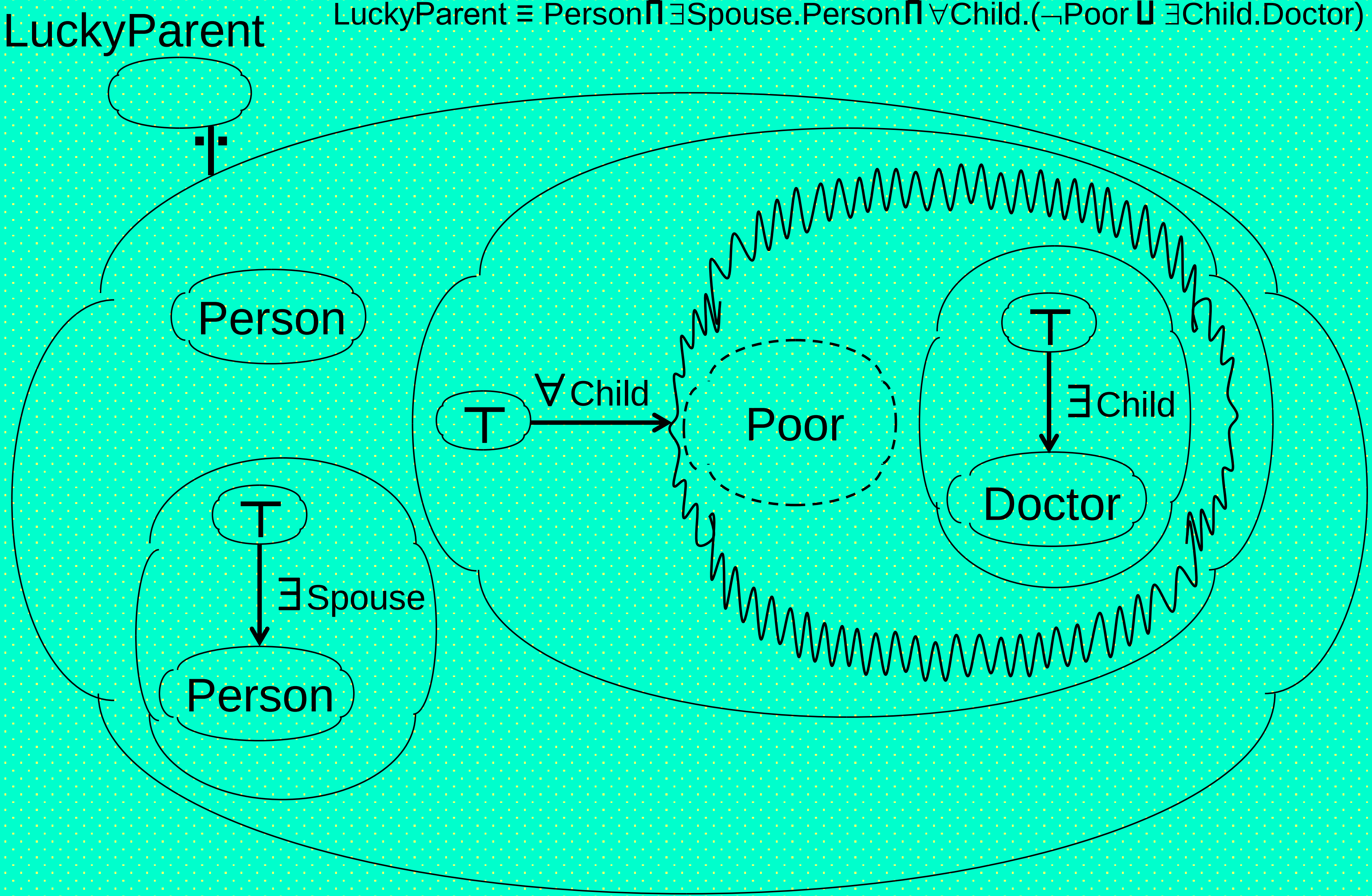
LuckyParent Example (1**)

LuckyParent ≡ Person ⊓ ∃Spouse.Person ⊓ ∀Child.(¬Poor ⊔ ∃Child.Doctor)



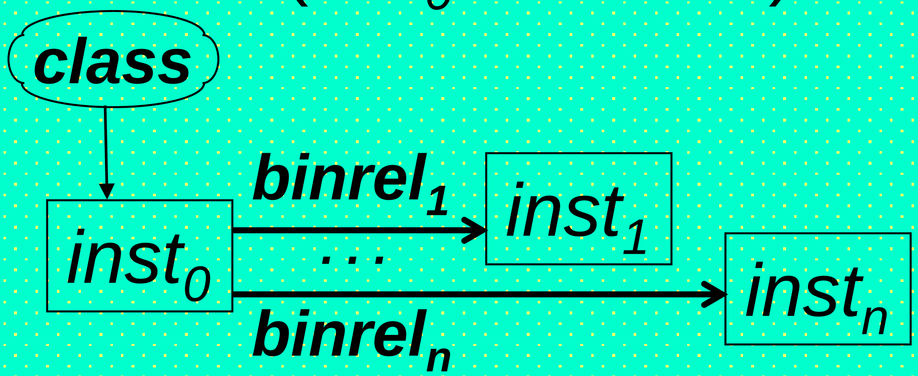
LuckyParent Example (1**)

LuckyParent ≡ Person ⊓ ∃Spouse.Person ⊓ ∀Child.(¬Poor ⊔ ∃Child.Doctor)



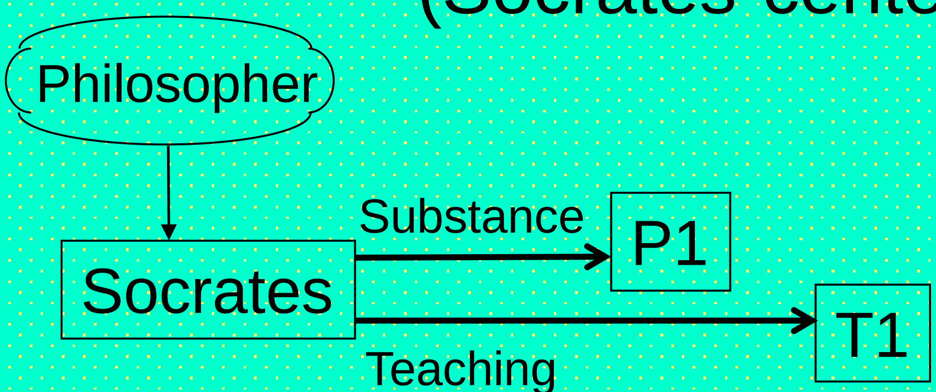
Object-Centered Logic: Grouping Binary Relations Around Instance

General: Graph ($inst_0$-centered) (Object-Centered) Logic



$class(inst_0) \wedge$
$binrel_1(inst_0, inst_1) \wedge$
$\cdots$
$binrel_n(inst_0, inst_n)$

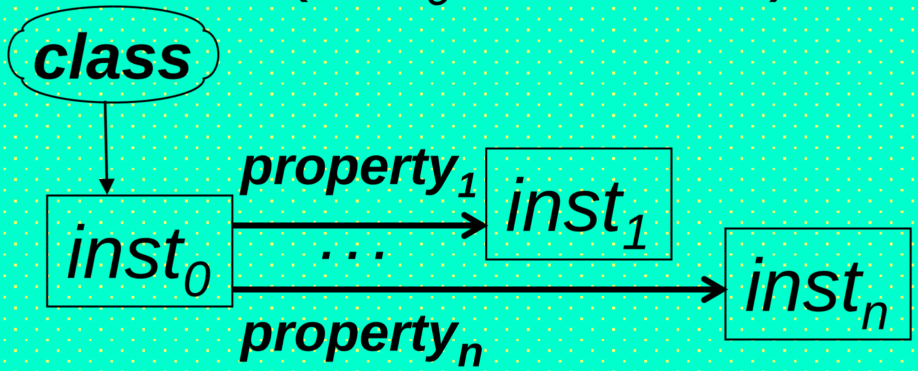
Example: Graph (Socrates-centered) (Object-Centered) Logic



Philosopher(Socrates) $\wedge$
Substance(Socrates, P1) $\wedge$
Teaching(Socrates, T1)

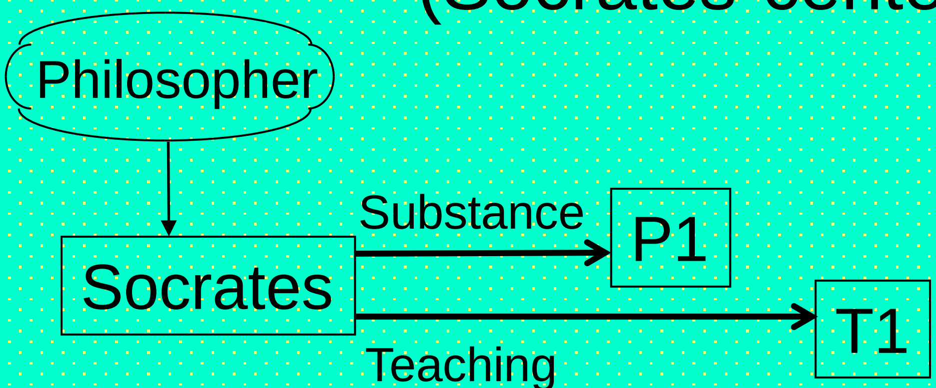
RDF-Triple ('Subject'-Centered) Logic: Grouping Properties Around Instance

General: Graph ($inst_0$-centered) (Subject-Centered) Logic



{($inst_0$, rdf:type, *class*),
($inst_0$, $property_1$, $inst_1$),
. . .
($inst_0$, $property_n$, $inst_n$)}

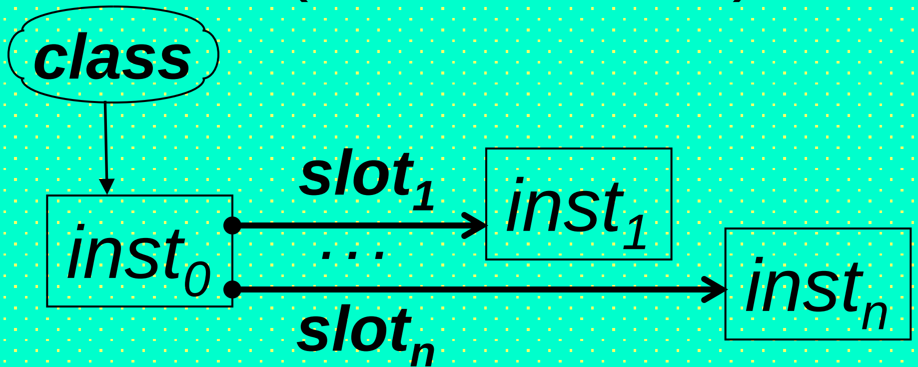
Example: Graph (Socrates-centered) (Subject-Centered) Logic



{(Socrates, rdf:type, Philosopher),
(Socrates, Substance, P1),
(Socrates, Teaching, T1)}

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

General: Graph (*bulleted* arcs)

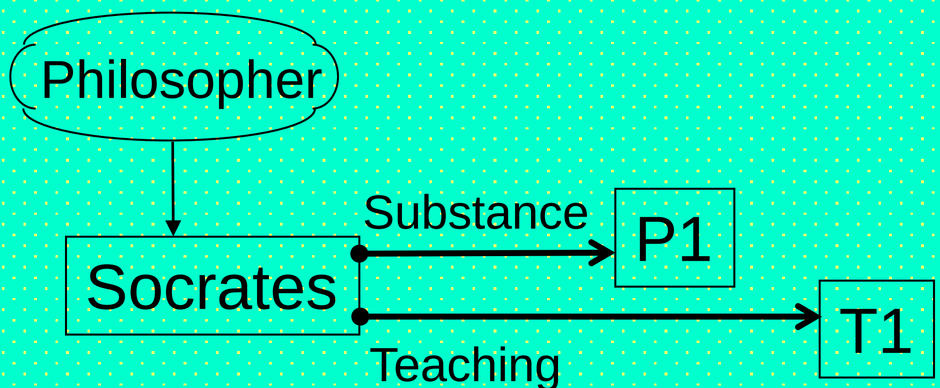


(PSOA-like Frame) Logic

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

$inst_0 \in class$,
$slot_1 = inst_1$,
. . .
$slot_n = inst_n$

Example: Graph

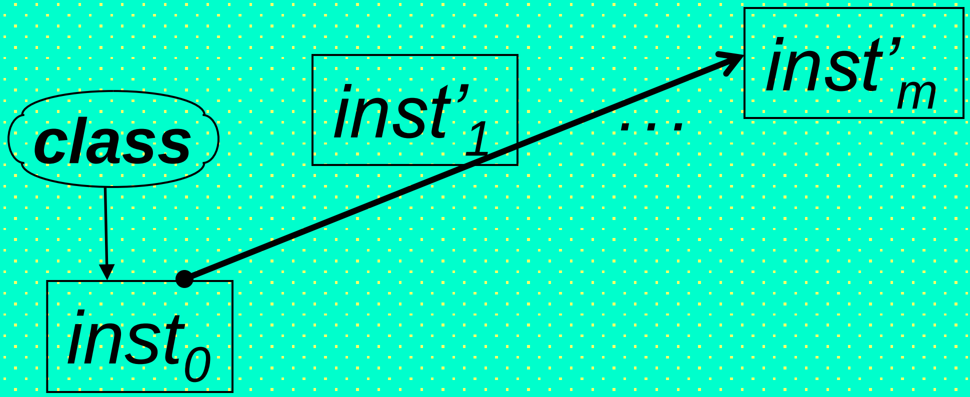


(PSOA-like Frame) Logic

Socrates#Philosopher(
Substance->P1;
Teaching->T1)

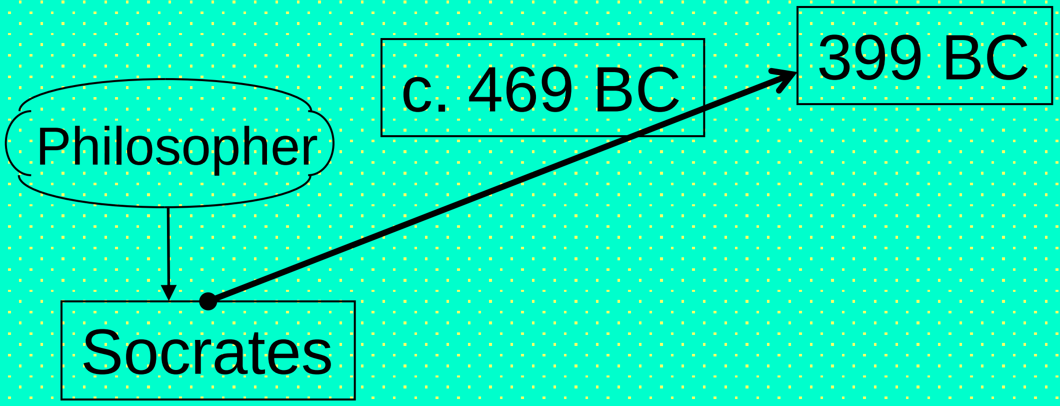
Logic of Shelves (‘Arrays’): Associating Tuple(s) with OID-Distinguished Instance

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



inst$_0$#class(
inst’$_1$, …, inst’$_m$)

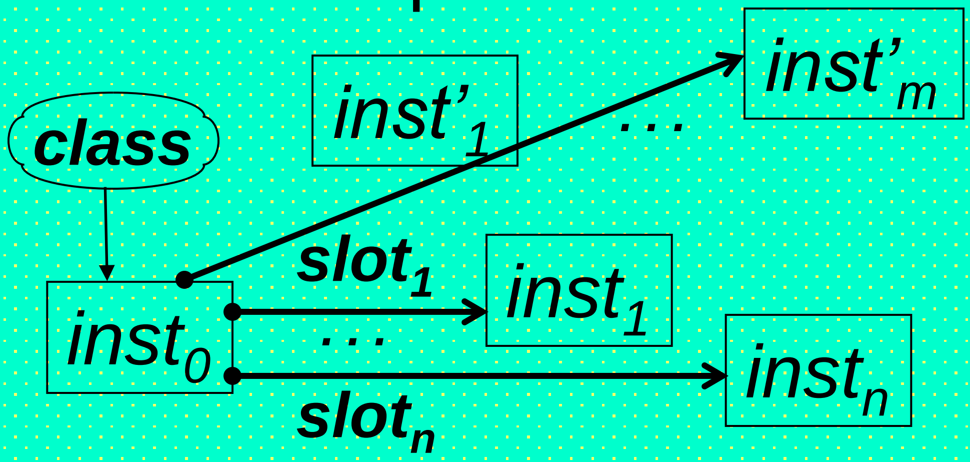
Example: Graph — (PSOA-like Shelf) Logic



Socrates#Philosopher(
c. 469 BC, 399 BC)

Positional-Slotted-Term Logic: Associating Tuple(s)+Slots with OID-Disting'ed Instance

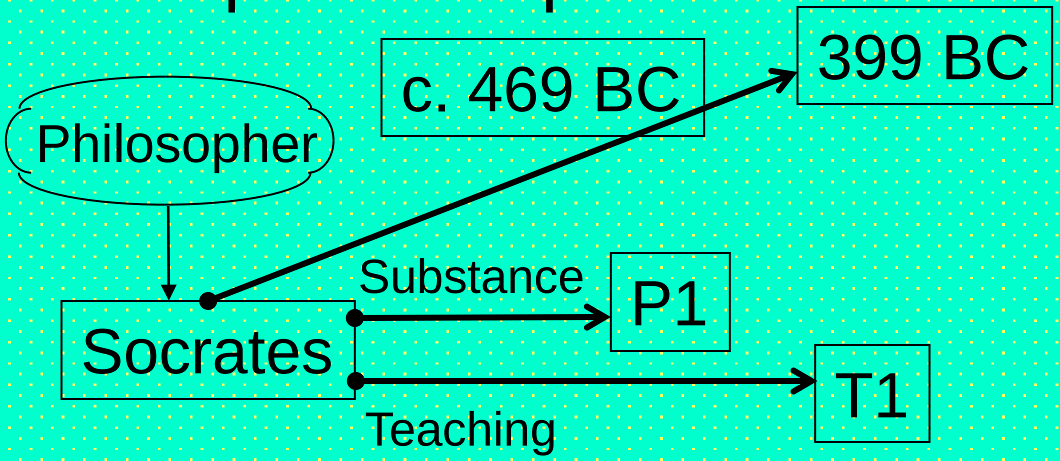
General: Graph



(PSOA-like Positional-Slotted-Term) Logic

$inst_0\#class($
$\quad inst'_1, \ldots, inst'_m;$
$\quad slot_1\text{->}inst_1;$
$\quad \ldots$
$\quad slot_n\text{->}inst_n)$

Example: Graph

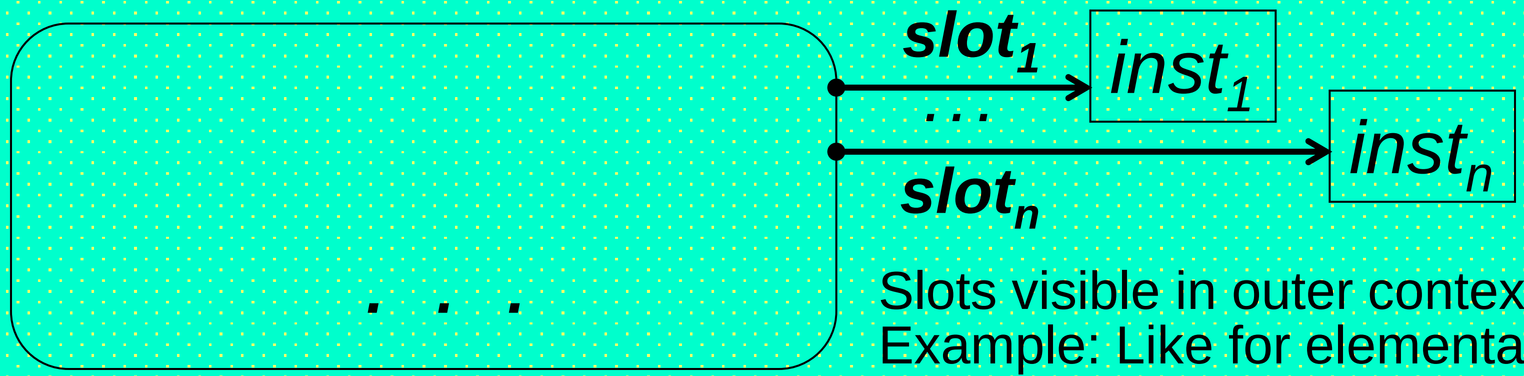


(PSOA-like Positional-Slotted-Term) Logic

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

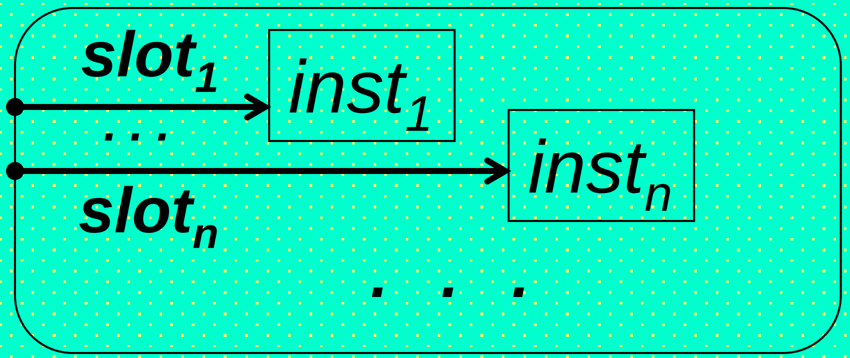
Term and Formula Description: Associating Slots with Complex Node

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



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

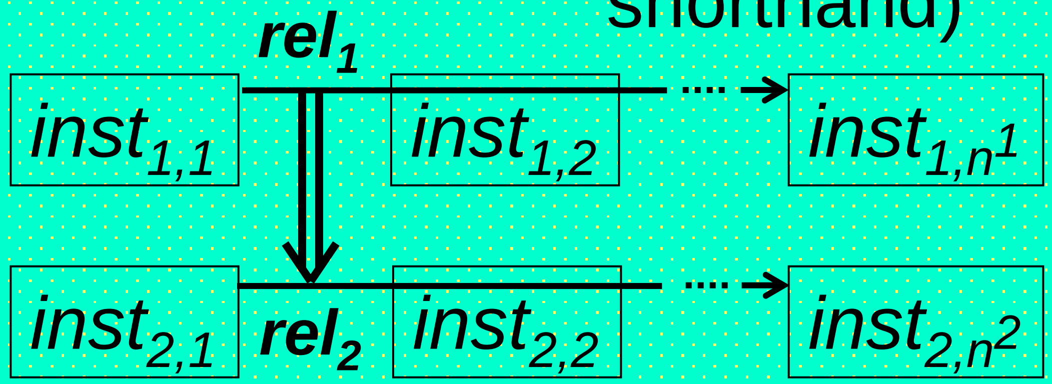
Complex Node (e.g. Roundangle) having Inward Slots



Slots visible in inner context.
Example: Later slide with closure slot

Rules: Relations Imply Relations (1)

General: Graph (ground, shorthand)

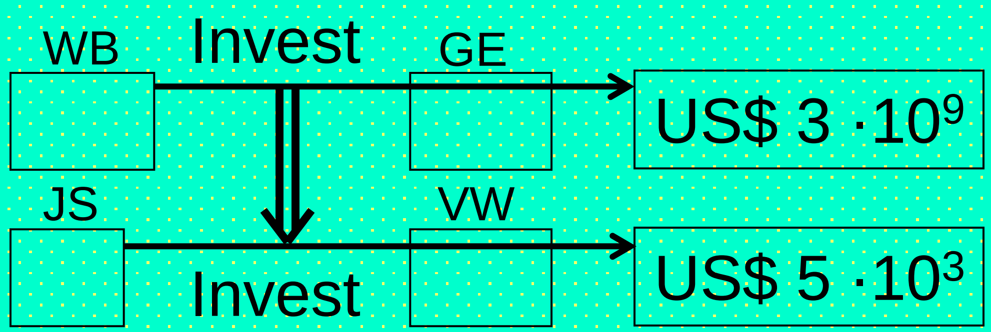


Logic

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

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

Example: Graph



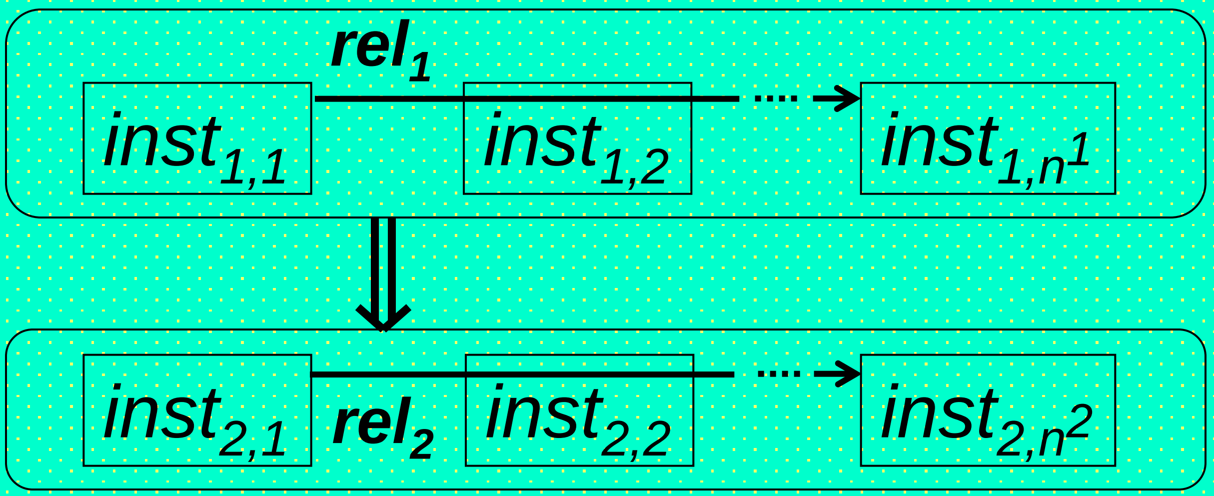
Logic

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

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

Rules: Relations Imply Relations (1*)

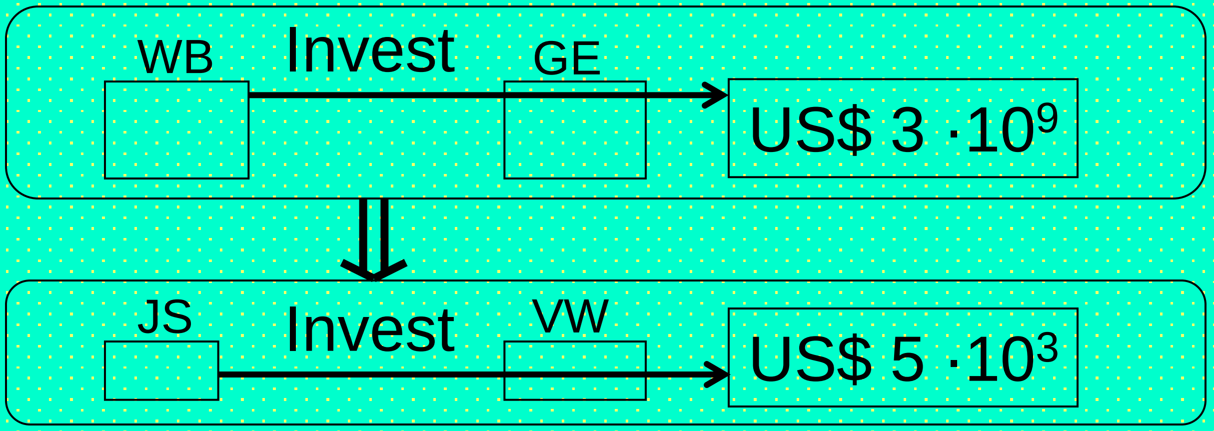
General: Graph (ground, normal) Logic



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

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

Example: Graph Logic

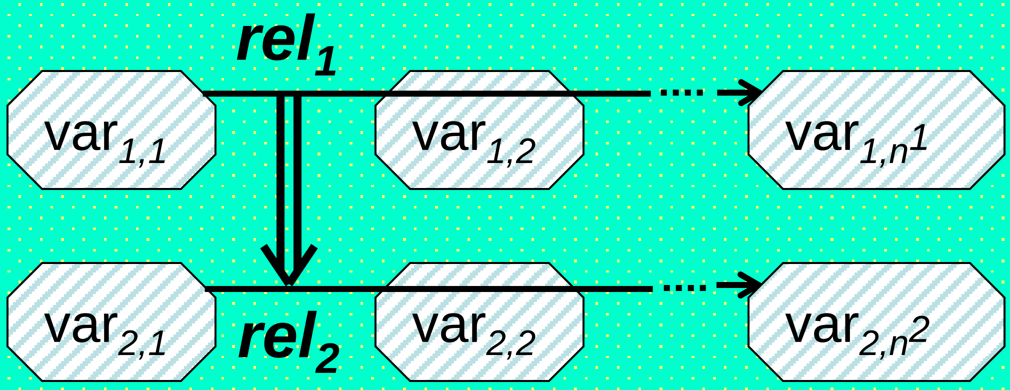


Invest(|WB, |GE, US$ $3 \cdot 10^9$) $\Rightarrow$

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

Rules: Relations Imply Relations (2)

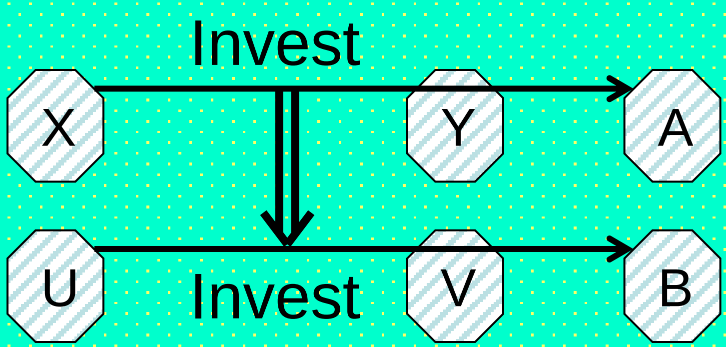
General: Graph (non-ground, where 'Implies' arrow creates universal closure)



Logic

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

Example: Graph



Logic

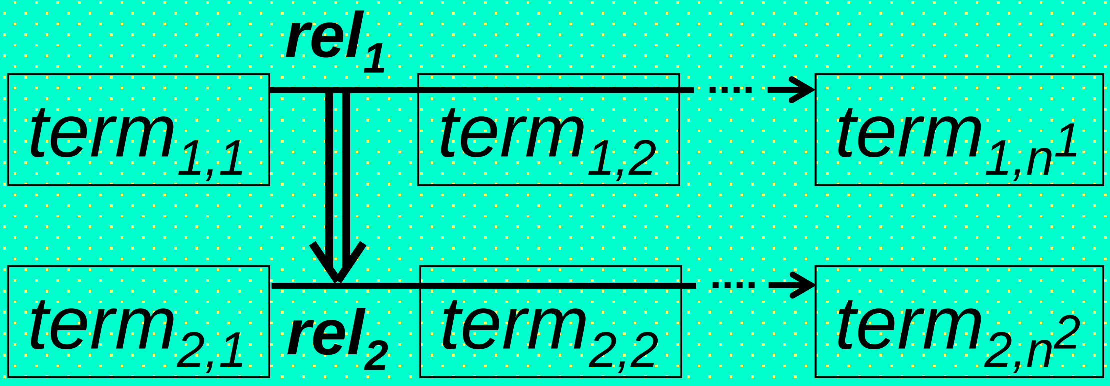
$(\forall\ X, Y, A, U, V, B)$

$\text{Invest}(X, Y, A) \Rightarrow$

$\text{Invest}(U, V, B)$

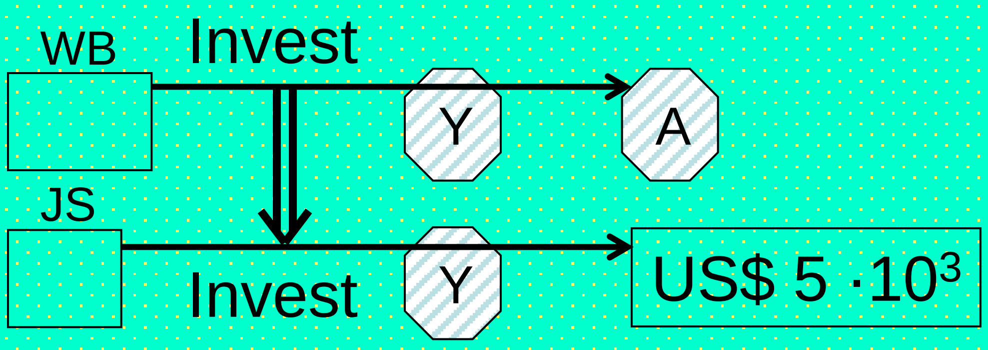
Rules: Relations Imply Relations (3)

General: Graph (inst/var terms) Logic



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

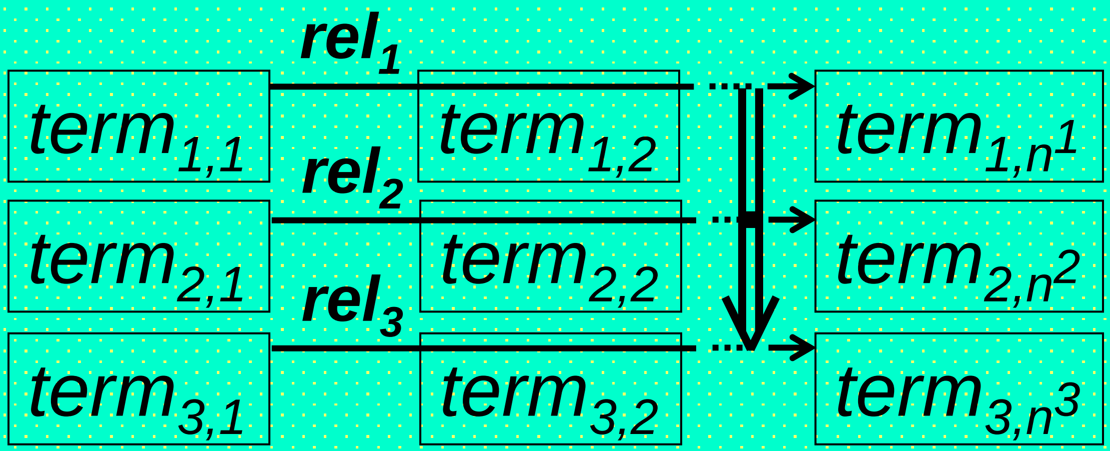
Example: Graph Logic



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

Rules: Conjuncts Imply Relations (1)

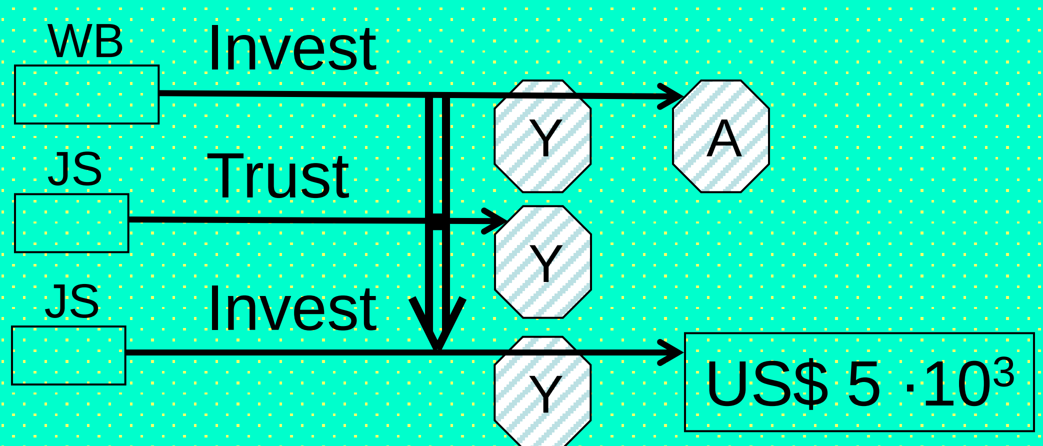
General: Graph (shorthand)



Logic

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

Example: Graph

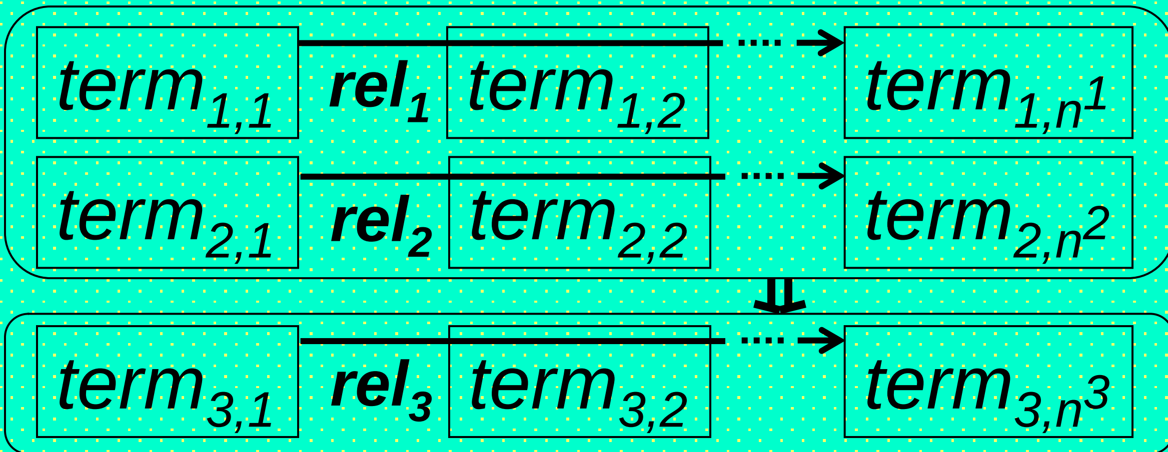


Logic

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

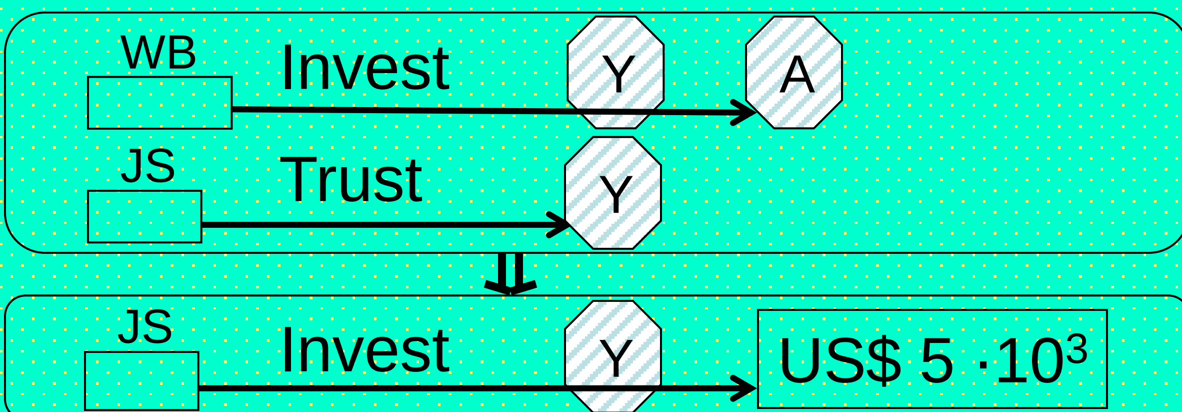
Rules: Conjuncts Imply Relations (1*)

General: Graph (prenormal) Logic



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

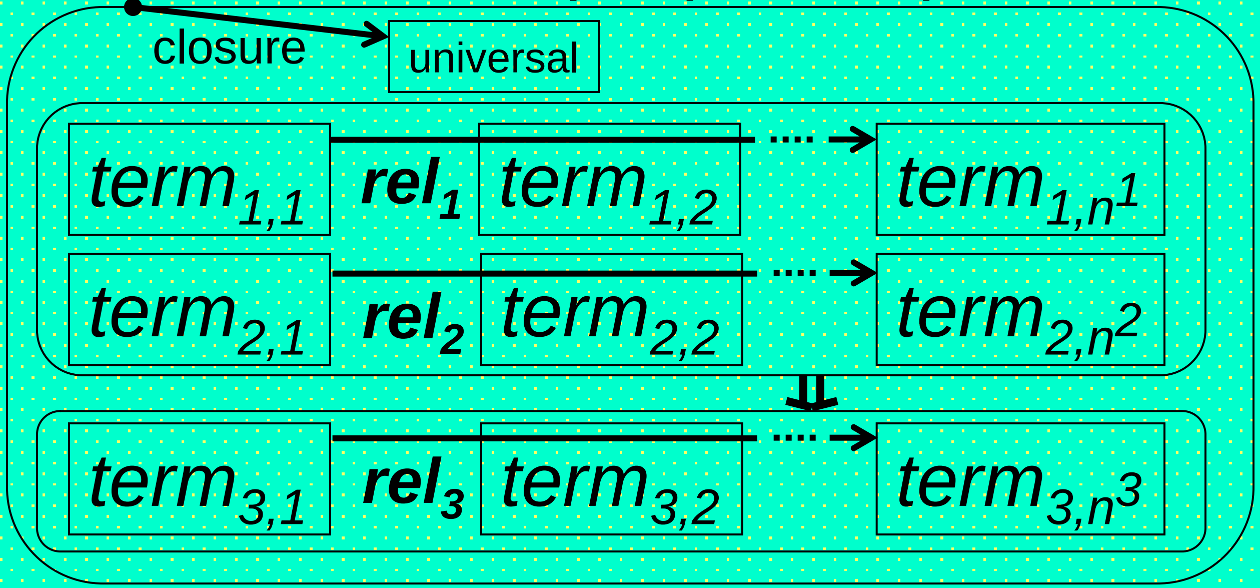
Example: Graph Logic



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

Rules: Conjuncts Imply Relations (1**)

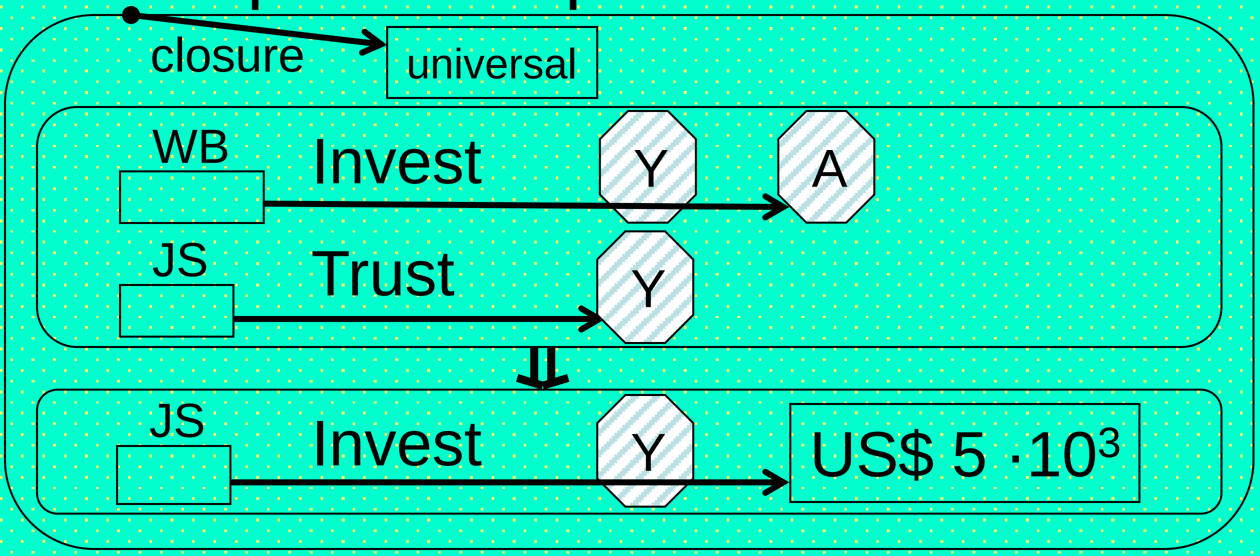
General: Graph (normal)



Logic

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

Example: Graph



Logic

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

Rules: Conjuncts Imply Relations (2)

Example: RuleML/XML

```
<Implies closure="universal">
   <And>
      <Atom>
         <Rel>Invest</Rel>
         <Ind name="shallow">WB</Ind>
         <Var>Y</Var>
         <Var>A</Var>
      </Atom>
      <Atom>
         <Rel>Trust</Rel>
         <Ind name="shallow">JS</Ind>
         <Var>Y</Var>
      </Atom>
   </And>
   <Atom>
      <Rel>Invest</Rel>
      <Ind name="shallow">JS</Ind>
      <Var>Y</Var>
      <Data>US$ 5000</Data>
   </Atom>
</Implies>
```

Logic

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

Implication-Defined Predicate Odd: RuleML/XML Serialization

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

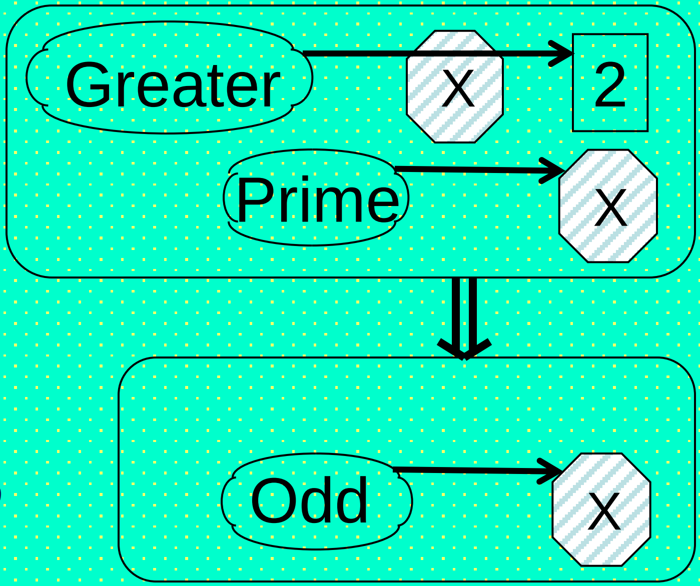
$(\forall\ X)$

$\mathrm{Greater}(X, 2) \land$

$\mathrm{Prime}(X) \ \Rightarrow$

$\mathrm{Odd}(X)$

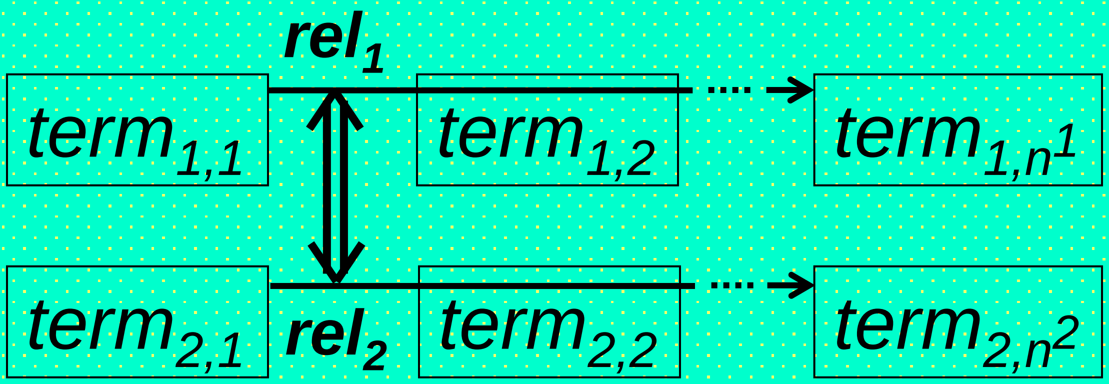
Graph (prenormal)



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

Relations Equivalent to Relations

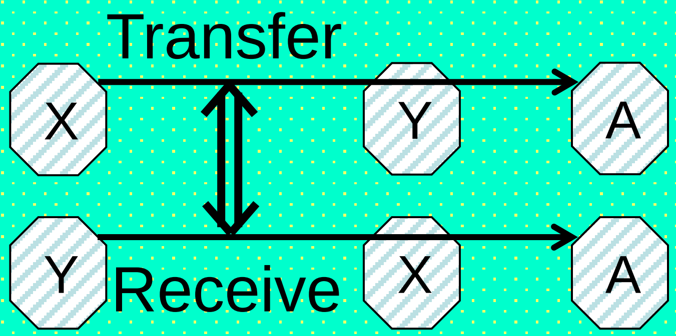
General: Graph (inst/var terms)



Logic

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

Example: Graph



Logic

$(\forall X, Y, A)$
Transfer$(X,Y,A) \Leftrightarrow$
Receive(Y,X,A)

Equivalence-Defined Predicate Even: RuleML/XML Serialization

RuleML/XML

```
<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>
```

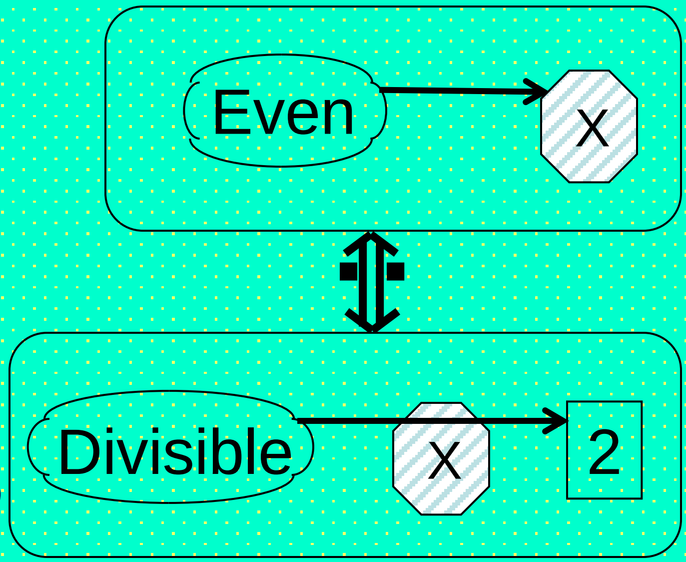
Logic

$(\forall\ X)$

$\mathrm{Even}(X)\ :\Leftrightarrow$

$\mathrm{Divisible}(X, 2)$

Graph (prenormal)



Graph ':⇔' arrow normalizes to RuleML-like closure="universal"

Equality-Defined Function Double: RuleML/XML Serialization

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>
```

Graph ':=' arrow normalizes to RuleML-like closure="universal"

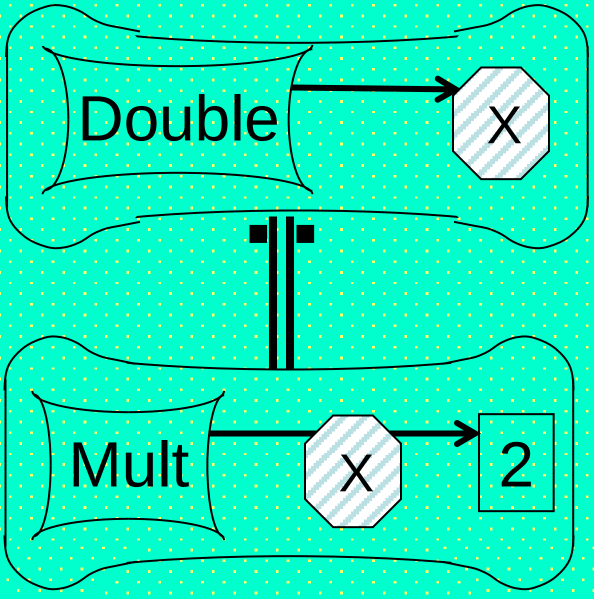
Logic

$(\forall\ X)$

$\text{Double}(X) :=$

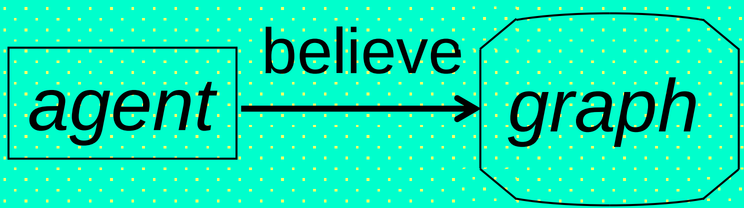
$\text{Mult}(X, 2)$

Graph (prenormal)



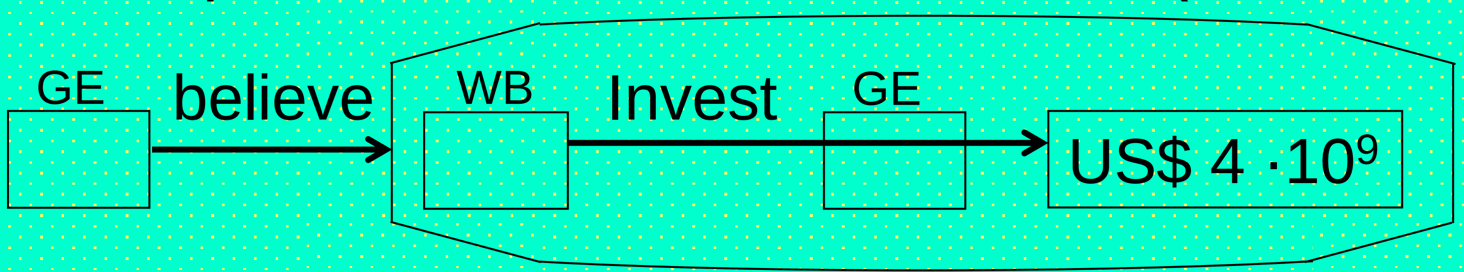
Modally Embedded Propositions

General: Graph (complex *snipkeg* node, used to ‘encapsulate’ what another agent believes, wants, etc.) — (Modal) Logic



$\text{believe}_{agent}(graph)$

Example: Graph — (Modal) Logic



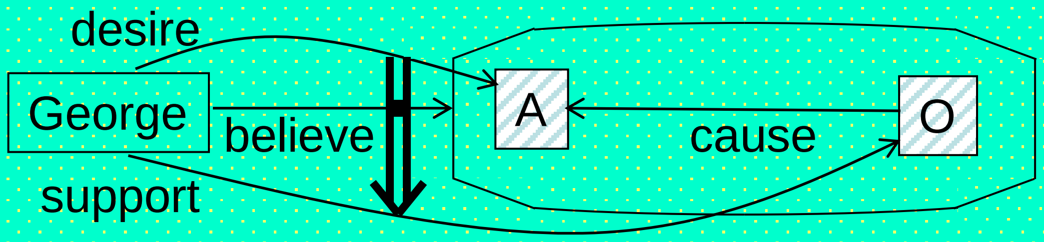
$\text{believe}_{|GE}(\text{Invest}(|WB, |GE, \text{US\$ } 4 \cdot 10^9))$

Beliefs and Desires as Propositional Attitudes (1)

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

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

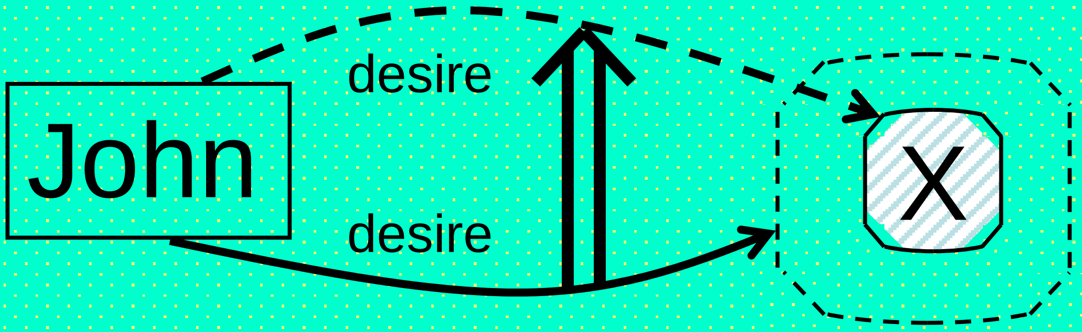
Grailog:



Beliefs and Desires as Propositional Attitudes (2)

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

Grailog:



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

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 — Dimensions of Grailog Systematics

- Box dimensions:
  - Corners: pointed vs. snipped vs. rounded
    - To quote/copy vs. reify/instantiate vs. evaluate contents (cf. Lisp, Prolog, Relfun, Hilog, RIF, 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 dimensions:
  - Shafts: single vs. double
  - Heads: triangular vs. diamond
  - Tails: plain vs. bulleted vs. colonized
- Box & Arrow (line-style) dimension: solid vs. dashed vs. wavy

Graphical Elements: Box Systematics — Dimensions of Corners and Shapes

Corner: Per ... Copy / ... Instantiation / ... Value

Shape: ***Rect-*** / ***Snip-*** / ***Round-***

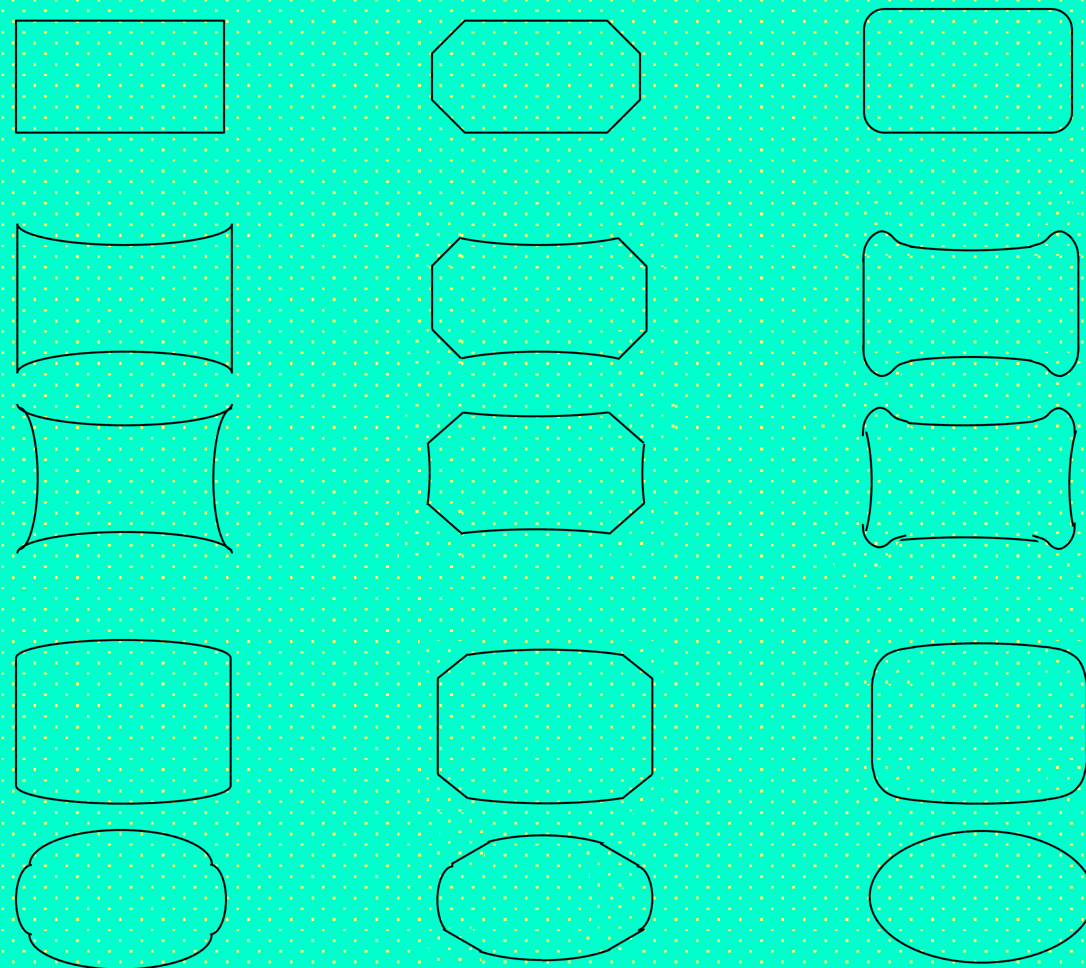
Neutral
-angle

Individual
(Function Application**)**
-tie

Function
-star

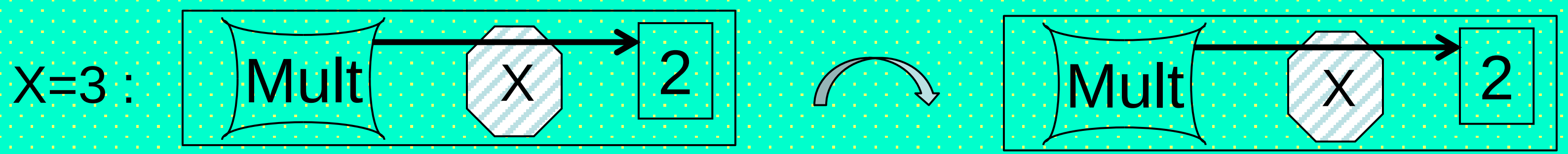
Proposition
(Relation Application)
-keg

Relation
(incl. Class)
-oval

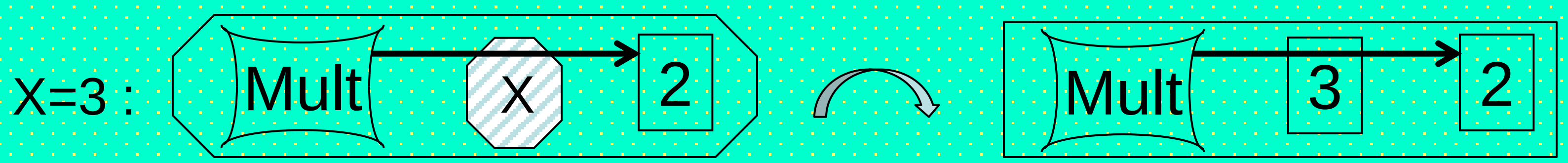


Graphical Elements: Boxes — Function/Relation-Neutral Shape of Angles Varied w.r.t. Corner Dimension

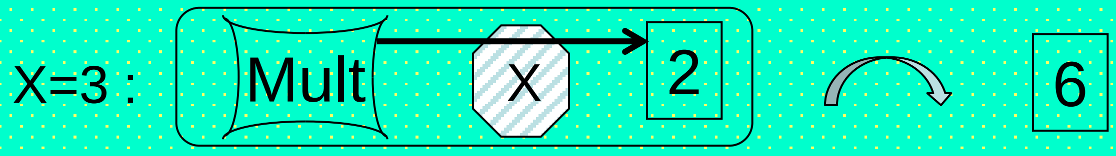
– ***Rectangle***: Neutral 'per copy' nodes *quote* their contents



– ***Snipangle*** (octagon): Neutral 'per instantiation' nodes *dereference* contained variables to values from context

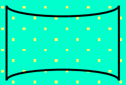


– ***Roundangle*** (rounded angles): Neutral 'per value' nodes *evaluate* their contents through instantiation of variables and activation of function/relation applications

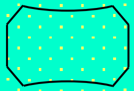


Assuming Mult built-in function

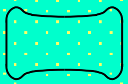
Graphical Elements: Boxes — Concave



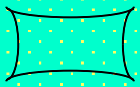
– ***Recttie*** (bowtie-like rectangle with concave top/bottom sides): Elementary nodes for individuals (instances). Complex nodes for quoted instance-denoting terms (constructor-function applications)



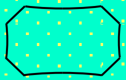
– ***Sniptie*** (snipped): Elementary nodes for variables. Complex nodes for instantiated (reified) function applications



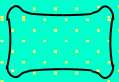
– ***Roundtie***: Complex nodes for evaluated built-in or equation-defined function applications



– ***Rectstar*** (4-point star): Elementary nodes for functions. Complex nodes for quoted functional (function-denoting) terms

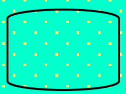


– ***Snipstar***: Complex nodes for instantiated functional terms

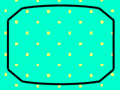


– ***Roundstar***: Complex nodes for evaluated functional applications (active, function-returning applications)

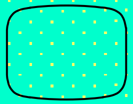
Graphical Elements: Boxes — Convex



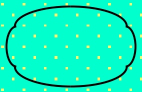
– ***Rectkeg*** (keg-like rectangle with convex top/bottom sides): Elementary nodes for truth constants (true, false, unknown). Complex nodes for quoted truth-denoting propositions (embedded relation applications)



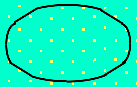
– ***Snipkeg***: Complex nodes for instantiated (reified) relation applications



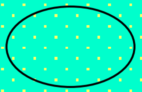
– ***Roundkeg***: Complex nodes for evaluated relation applications (e.g. as atomic formulas) and for connective uses



– ***Rectoval*** (4-point oval): Elementary nodes for relations, e.g. unary ones (classes). Complex nodes for quoted relational (relation-denoting) terms



– ***Snipoval***: Complex nodes for instantiated relational terms



– ***Roundoval***: Complex nodes for evaluated relational applications (active, relation-returning applications)

Conclusions

- Presented new edition of Grailog, including feedback
- Graphical elements for novel box & arrow systematics using orthogonal graphical features
- Leaving color (except for IRIs) for other purposes, e.g. highlighting subgraphs (for retrieval and inference)
- Introducing Deep vs. Shallow Name Specification
- Focus on *mapping* to a family of logics as in RuleML
- Use cases from *philosophy* to *technology* to *business*
  - E.g. "Logical Foundations of Cognitive Science": http://www.ict.tuwien.ac.at/lva/Boley_LFCS/index.html
- *Processing* of earlier Grailog-like DRLHs studied in Lisp, FIT, and Relfun
- Now aligned with Web-rule industry standard RuleML: http://ruleml.org/#Grailog

Future Work (1)

- Refine/extend Grailog, 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 tools, e.g. as use case for (Functional) RuleML (http://ruleml.org/fun) engines
  - More mappings between graphs, logic, and RuleML/XML
  - Graph indexing & querying (cf. http://www.hypergraphdb.org)
  - Graph transformations (normal form, typing homomorphism, merge, ...)
  - Advanced graph-theoretical operations (e.g., path tracing)
  - Exploit Grailog parallelism in implementation
- Submit for open standardization

Future Work (2)

- Develop a Grailog structure editor, e.g. supporting:
  - Auto-specialize of neutral application boxes (angles) for functions (ties) or relations (kegs), depending on contents
  - Auto-specialize of neutral operator boxes (angles) to functions (stars) or relations (ovals), depending on context
- Benefit from, and contribute to, Protégé visualization plug-ins such as Jambalaya/OntoGraf and OWLViz for OWL ontologies and Axiomé for SWRL rules
- 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.