

From Data to Knowledge through Grailog Visualization

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

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

Ontology, Rules, and Logic Programming for Reasoning and Applications
(RulesReasoningLP) Ontolog Mini-Series, Session 2, 31 October 2013

Acknowledgements

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

[From Data to Knowledge through Grailog Visualization](#)

ISO 15926 and Semantic Technologies 2013 Conference, Sogndal, Norway, 5-6 September 2013

[Grailog 1.0: Graph-Logic Visualization of Ontologies and Rules](#)

The 7th International Web Rule Symposium (RuleML 2013),
University of Washington, Seattle WA, 11-13 July 2013

[The Grailog Systematics for Visual-Logic Knowledge Representation with Generalized Graphs](#)

Faculty of Computer Science Seminar Series, University of New Brunswick, Fredericton, Canada, 26 September 2012
High Performance Computing Center Stuttgart (HLRS), Stuttgart, Germany, 14 August 2012

[Grailog: Mapping Generalized Graphs to Computational Logic](#)

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

[The Grailog User Interface for Knowledge Bases of Ontologies & Rules](#)

OMG Technical Meeting, Ontology PSIG, Cambridge, MA, 21 June 2012

[Grailog: Knowledge Representation with Extended Graphs for Extended Logics](#)

SAP Enterprise Semantics Forum, 24 April 2012

[Grailog: Towards a Knowledge Visualization Standard](#)

BMIR Research Colloquium, Stanford, CA, 4 April 2012

PARC Research Talk, Palo Alto, CA, 29 March 2012

[RuleML/Grailog: The Rule Metalogic Visualized with Generalized Graphs](#)

PhiloWeb 2011, Thessaloniki, Greece, 5 October 2011

[Grailog: Graph inscribed logic](#)

Course about Logical Foundations of Cognitive Science, TU Vienna, Austria, 20 October -10 December 2008

Visualization of Data & Knowledge: Graphs Remove Entry Barrier to Logic

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

Grailog

Graph inscribed logic provides intuition for logic

Advanced **cognitively motivated** systematic

graph standard for visual-logic data & knowledge:

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

Generalized-graph framework as one uniform
2D syntax for major ([Semantic Web](#)) logics:

Pick subset for each targeted knowledge base,
map to/fro RuleML sublanguage, and exchange
& validate it, posing queries again in Grailog

Generalized Graphs to Represent and Map Logic Languages According to Grailog 1.0 Systematics

- We have used generalized graphs for representing various logic languages, where basically:
 - Graph nodes (vertices) represent individuals, classes, etc.
 - Graph arcs (edges) represent relationships
- *Next slides:*
What are the *principles* of this representation and what graph generalizations are required?
- *Later slides:*
How are these graphs *mapped* (invertibly) to logic, thus specifying Grailog as a ‘GUI’ for knowledge?
- *Final slides:*
What is the *systematics* of Grailog features?


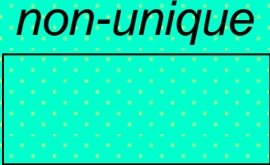
Grailog Principles

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

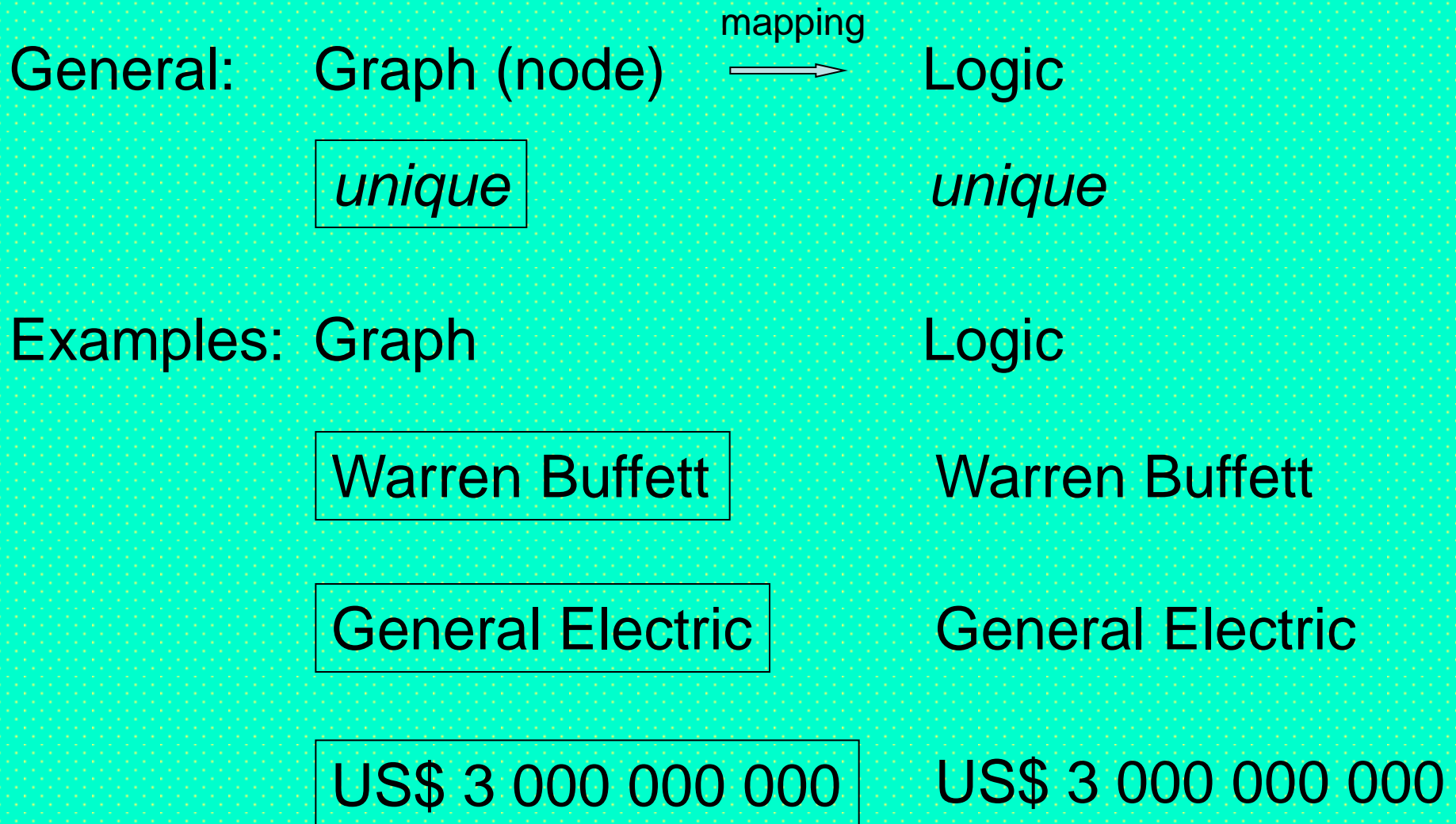
Grailog Generalizations

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

Graphical Elements: Names

- Written **into** boxes (nodes): 
Unique (canonical, distinct) names
 - Unique Name Assumption (UNA)
refined to Unique Name Specification (UNS)
- Written **onto** boxes (node labels): 
Non-unique (alternate, 'aka') names
 - Non-unique Name Assumption (NNA)
refined to Non-unique Name Specification (NNS)
- Grailog combines UNS and NNS: xNS , with $x = U$ or N

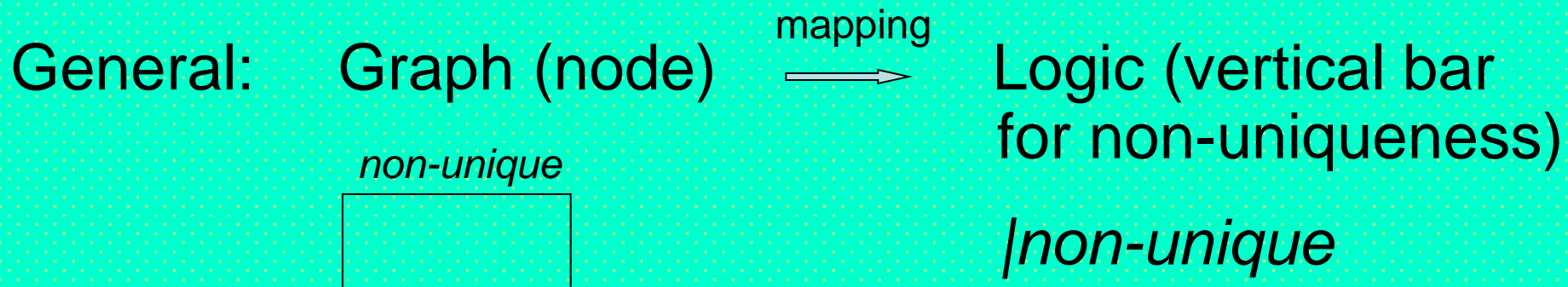
Instances: Individual Constants with Unique Name Specifications



Instances: Individual Constants with Non-unique Name Specifications

General: Graph (node) $\xrightarrow{\text{mapping}}$ Logic (vertical bar for non-uniqueness)


non-unique




/non-unique

Examples: Graph


WB



GE



US\$ 3B



Logic

/WB

/GE

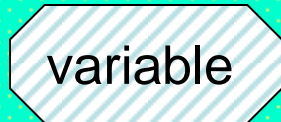
/US\$ 3B

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)



variable

Examples: Graph



Logic

X

Y

A

Predicates: Binary Relations (1)

General: Graph (*labeled arc*)

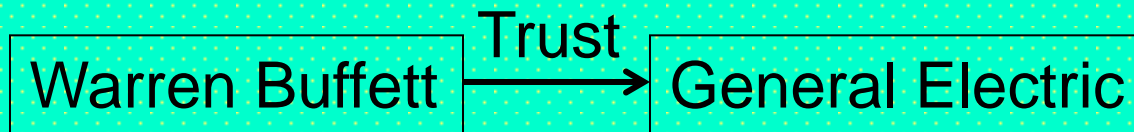
Logic



$binrel(inst_1, inst_2)$

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



$Trust(X, Y)$

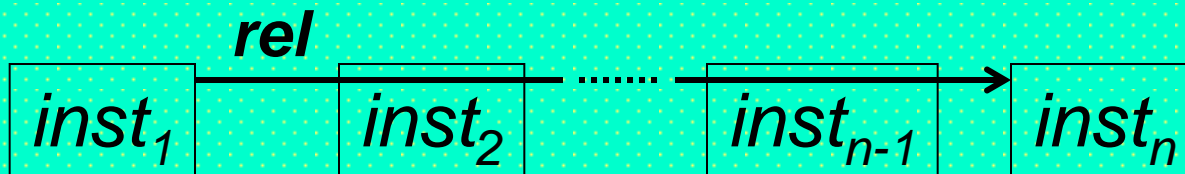
Graphical Elements: Arrows (1)

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

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

General: Graph (*hyperarc*)

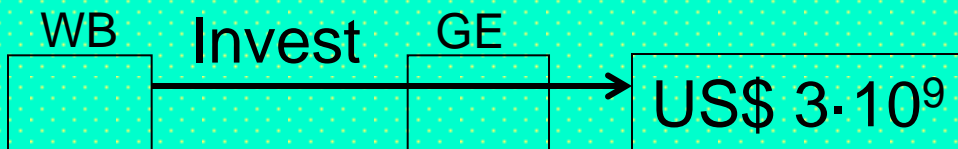
Logic



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

Example: Graph
($n=3$)

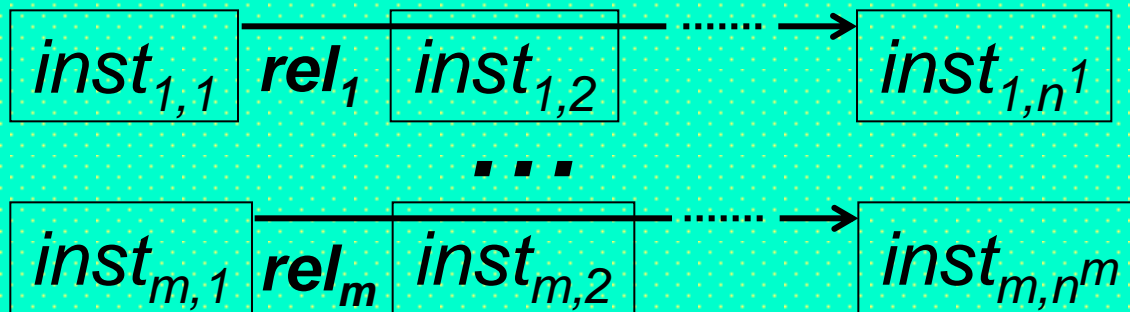
Logic



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

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

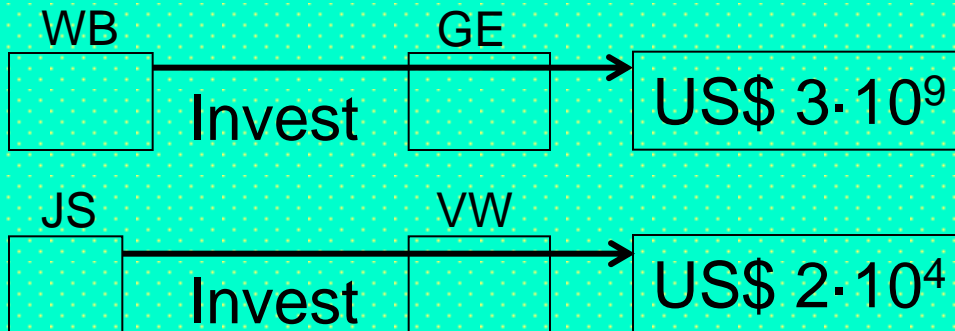
General: Graph (m hyperarcs)



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)



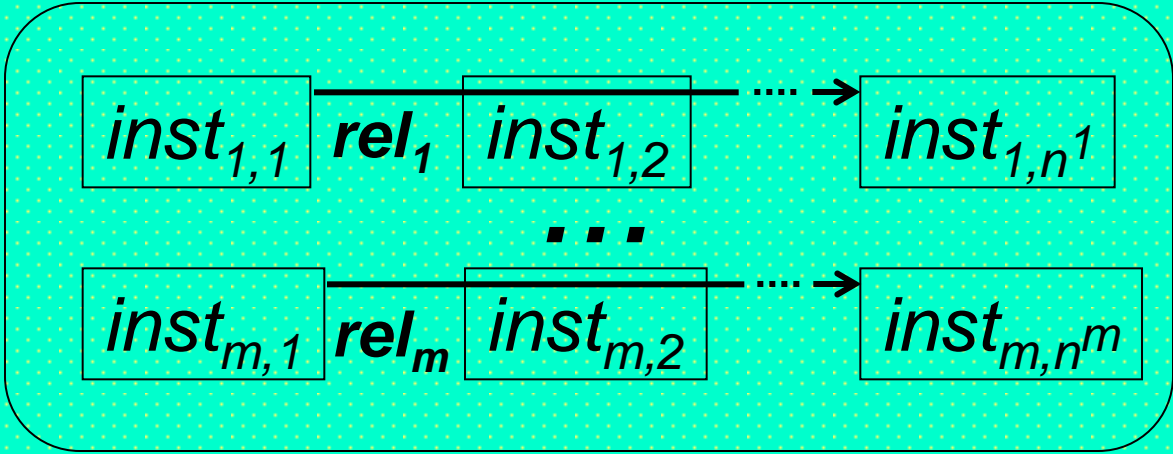
Logic

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

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

General: Graph (solid+linear)

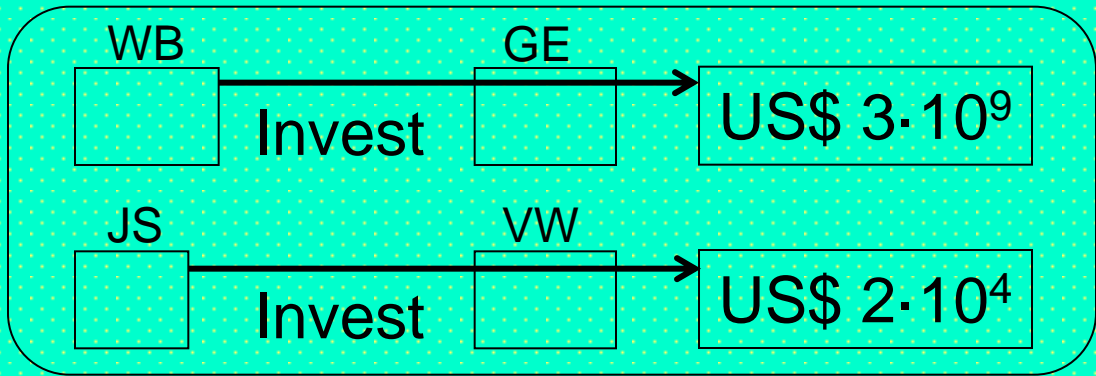
Logic



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

Example: Graph

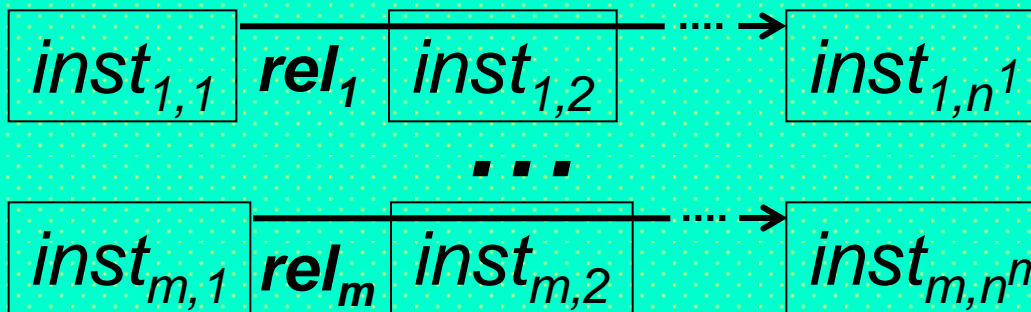
Logic



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

Disjunction of Formula Graphs: Co-Occurrence in Or Node

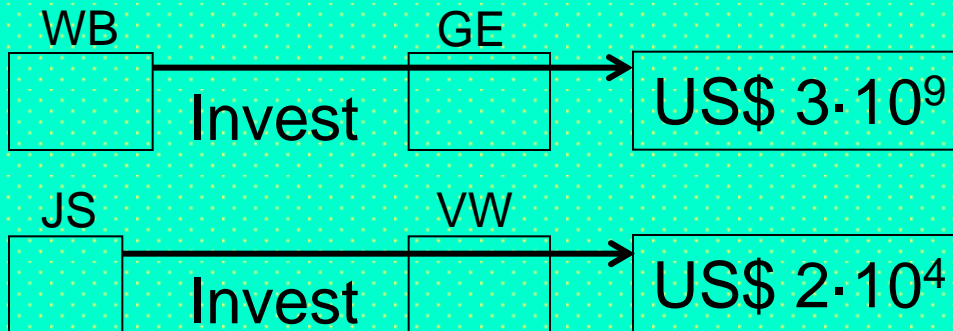
General: Graph (*solid+wavy*)



Logic

$$\left(\begin{array}{l} rel_1(inst_{1,1}, inst_{1,2}, \\ \dots, inst_{1,n^1}) \quad \vee \\ \dots \\ rel_m(inst_{m,1}, inst_{m,2}, \\ \dots, inst_{m,n^m}) \end{array} \right)$$

Example: Graph

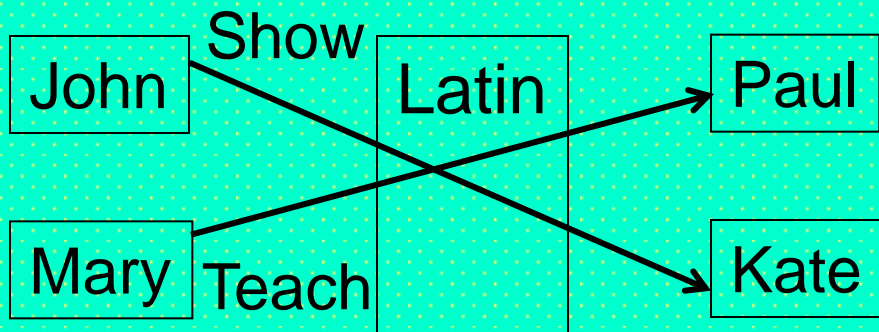


Logic

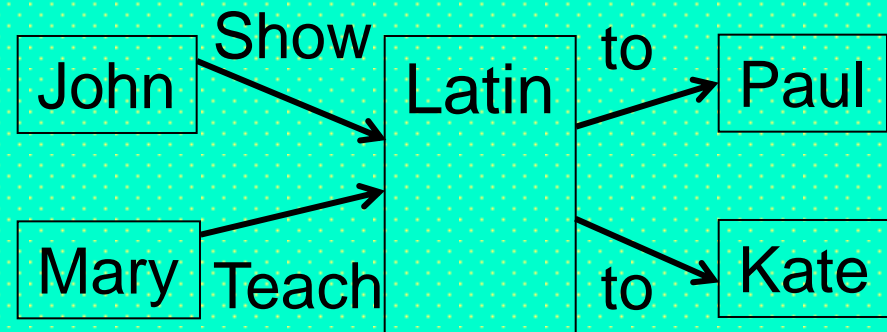
$$\left(\begin{array}{l} Invest(/WB, /GE, \\ \quad US\$ 3 \cdot 10^9) \quad \vee \\ Invest(/JS, /VW, \\ \quad US\$ 2 \cdot 10^4) \end{array} \right)$$

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

Hypergraph (2 hyperarcs, crossing inside a node)



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

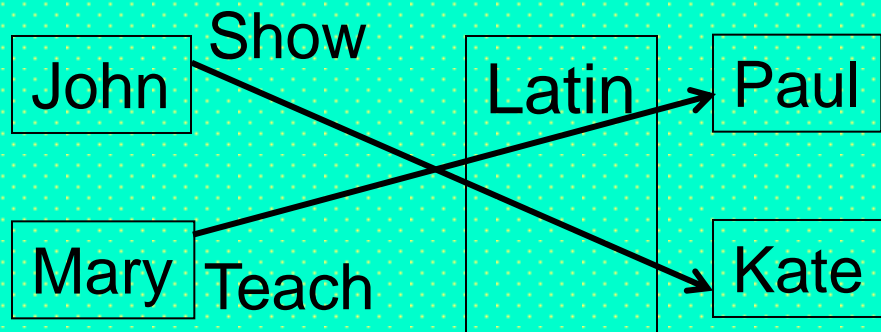


Symbolic Controlled English

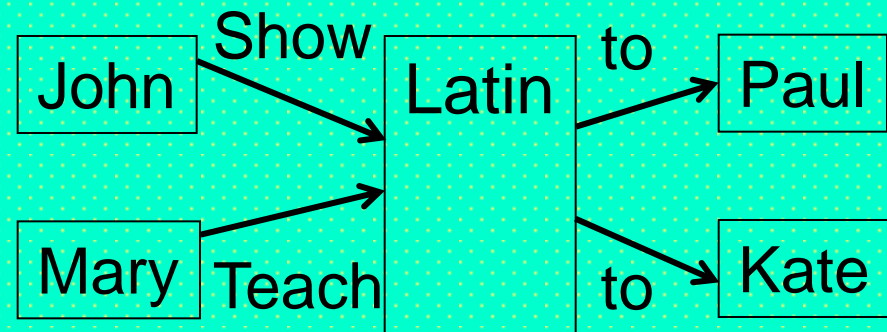
*“John shows Latin to Kate.
Mary teaches Latin to Paul.”*

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

Hypergraph (2 hyperarcs, crossing outside nodes)

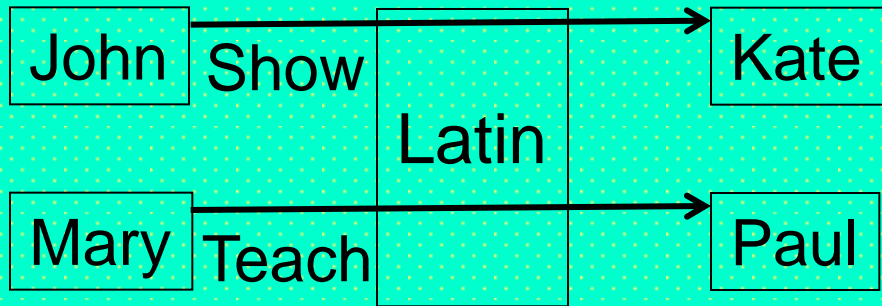


DLG (4 arcs, do not specify to whom Latin is shown or taught)

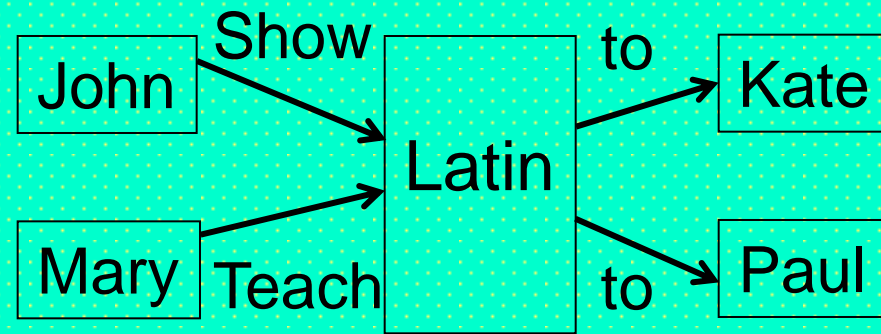


From Hyperarc Crossings to Node Copies as a Normalization Sequence (1^{**})

Hypergraph (2 hyperarcs, parallel-cutting a node)



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

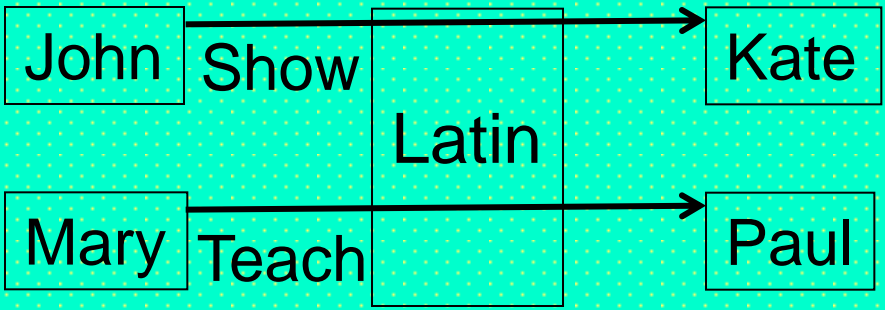


The hyperarc for, e.g., ternary $Show(John, Latin, Kate)$ can be seen as the path composition of 2 arcs for binary $Show(John, Latin)$ and binary $to(Latin, Kate)$

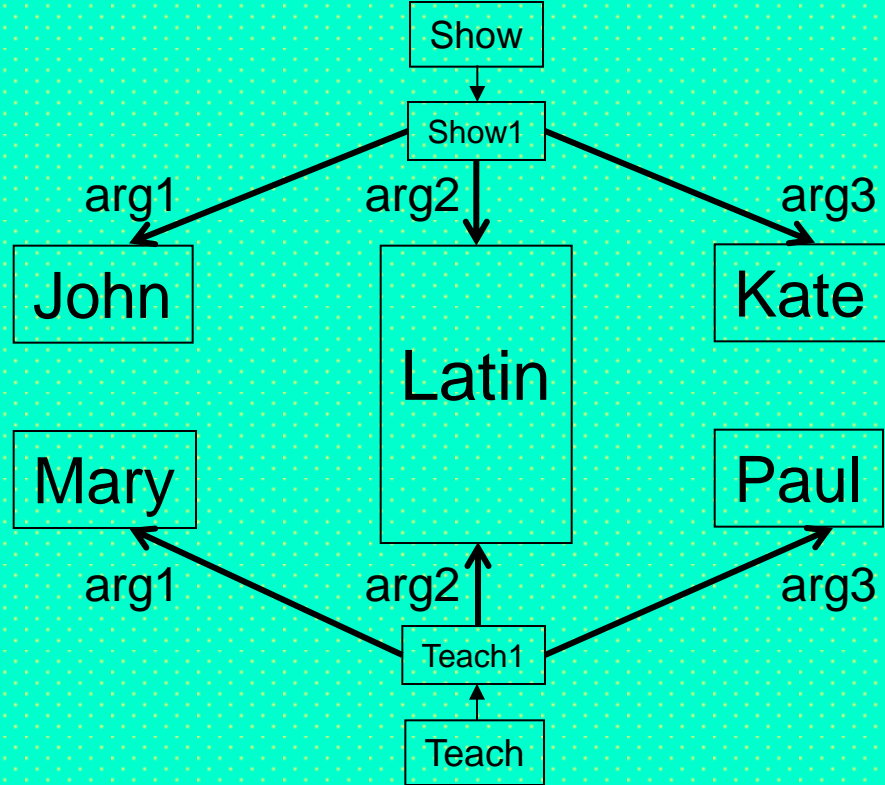
From Hyperarc Crossings to Node Copies

— Insert on Correct Binary Reduction

Hypergraph (2 hyperarcs, parallel-cutting a node)

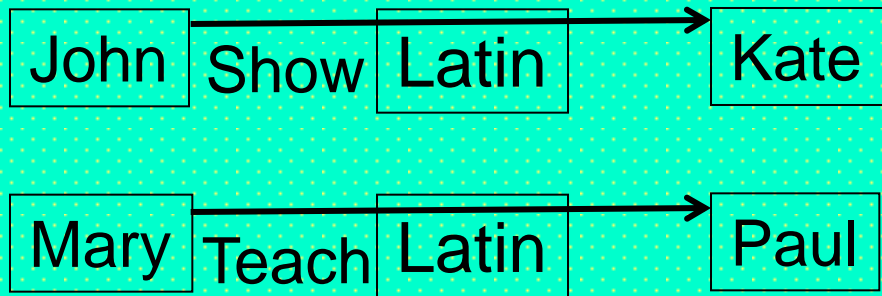


DLG (8 arcs with 4 'reified' relation/ship nodes to point to arguments)



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

Hypergraph (2 hyperarcs,
employing
a node copy)



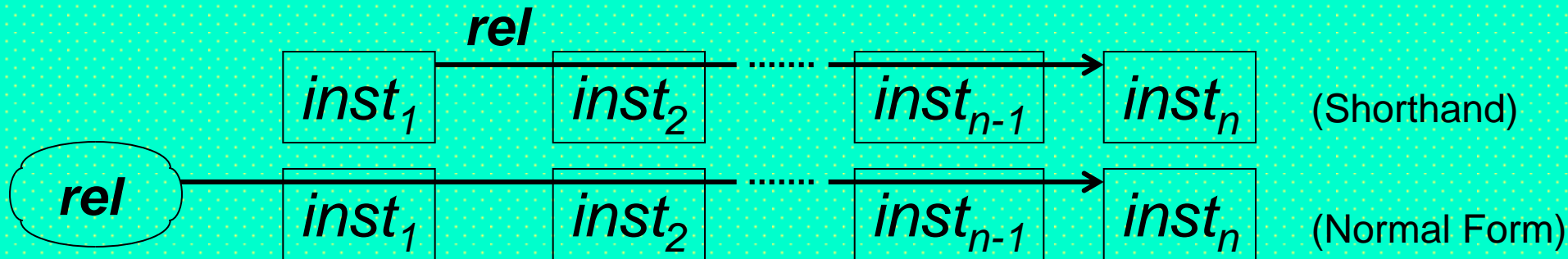
Logic (2 relations,
employing
a symbol copy)

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

*Both 'Latin' occurrences remain one node even when copied for easier layout:
Having a unique name, 'Latin' copies can be merged again.
This "fully node copied" normal form can help to learn the symbolic form, is
implemented by [Grailog KS Viz](#), and demoed in the Loan Processor [test suite](#)*

From Predicate Labels on Hyperarcs to Labelnodes Starting Hyperarcs

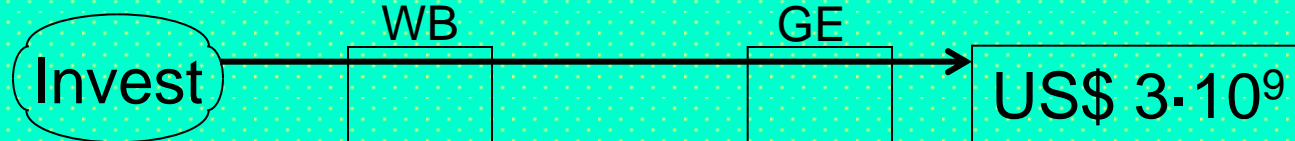
General: Graph (*hyperarc with $rect4vex$ -shaped labelnode*) Logic
 $rel(inst_1, inst_2, \dots, inst_{n-1}, inst_n)$



Example: Graph
($n=3$)

Logic

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



Predicates: Unary Relations (Classes, Concepts, Types)

General: Graph (class applied to instance node)

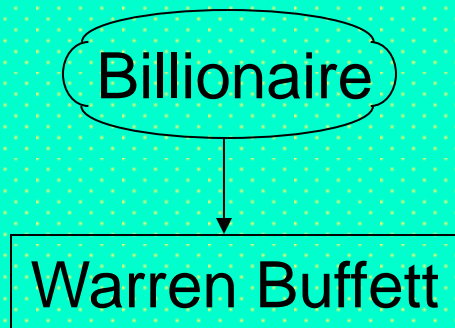
Logic



$class(inst_1)$

Example: Graph

Logic



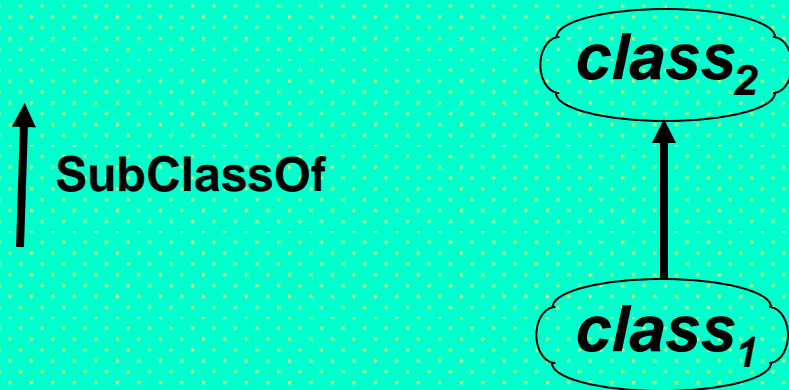
Billionaire(
Warren Buffett)

Graphical Elements: Arrows (2)

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

Class Hierarchies (Taxonomies): Subclass Relation

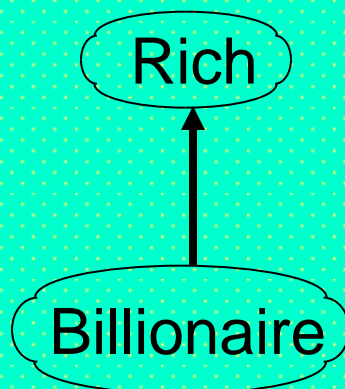
General: Graph (two nodes)



(Description)
Logic

$class_1 \sqsubseteq class_2$

Example: Graph

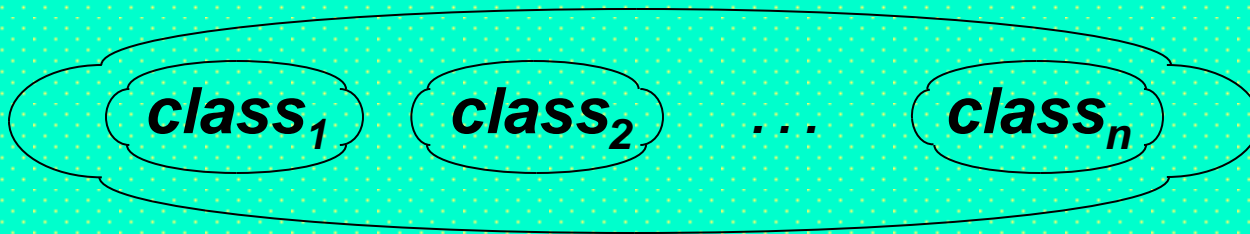


(Description)
Logic

$Billionaire \sqsubseteq Rich$

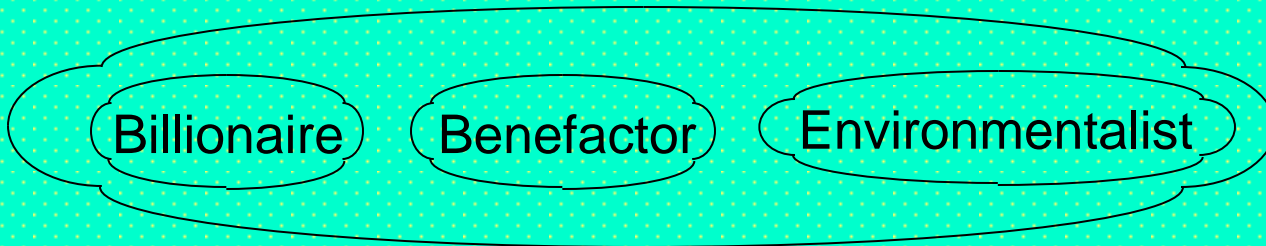
Intensional-Class Constructions (Ontologies): Class Intersection

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



$class_1 \sqcap$
 $class_2 \sqcap$
 $\dots \sqcap$
 $class_n$

Example: Graph (Description)

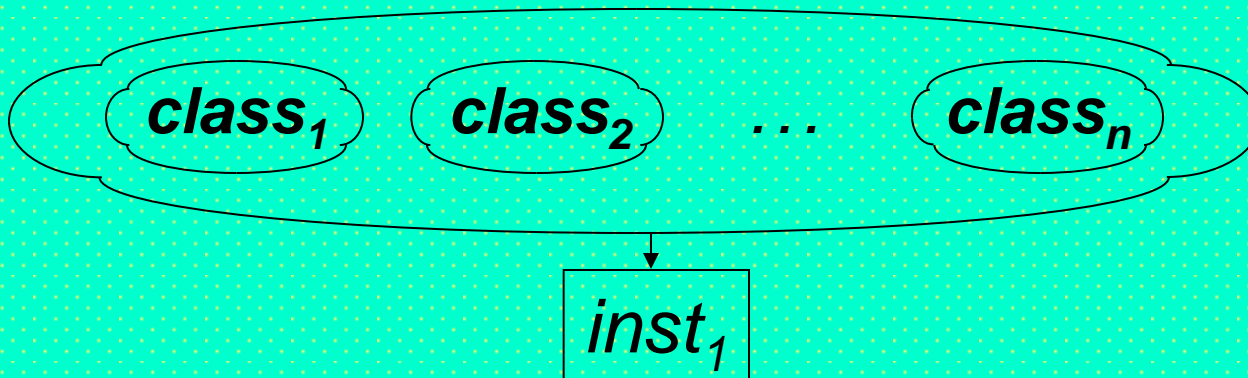


Logic
Billionaire \sqcap
Benefactor \sqcap
Environmentalist

Intensional-Class Applications: Class Intersection

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

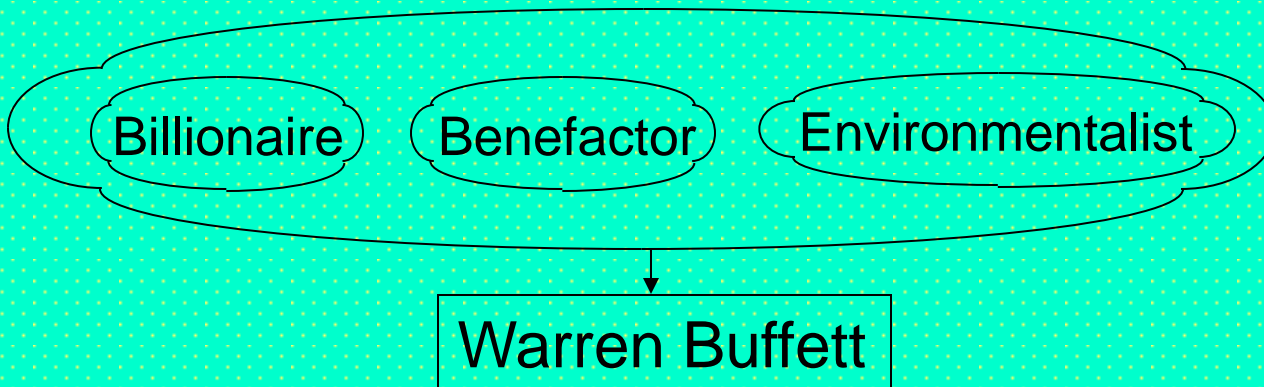
(xNS-Description)
Logic



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

Example: Graph

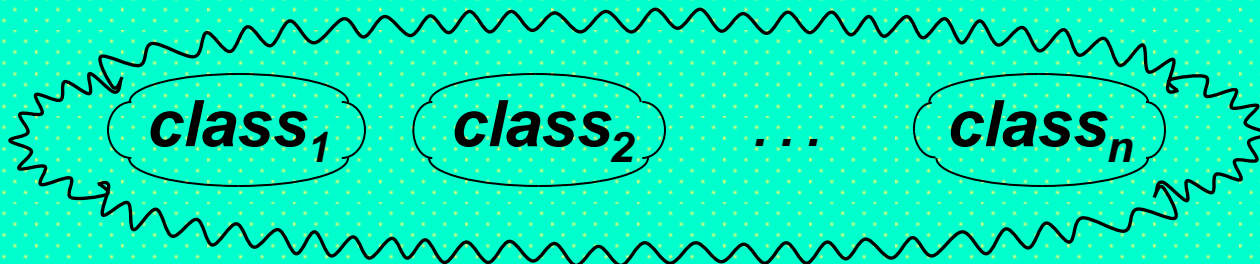
(xNS-Description)
Logic



$(Billionaire \sqcap$
 $Benefactor \sqcap$
 $Environmentalist)$
 $(Warren Buffett)$

Intensional-Class Constructions (Ontologies): Class Union

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

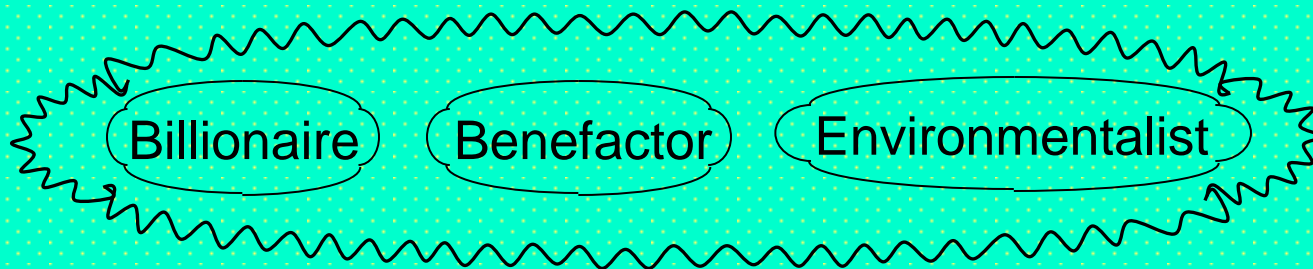


(Description)

Logic

class₁ \sqcup
class₂ \sqcup
 ... \sqcup
class_n

Example: Graph



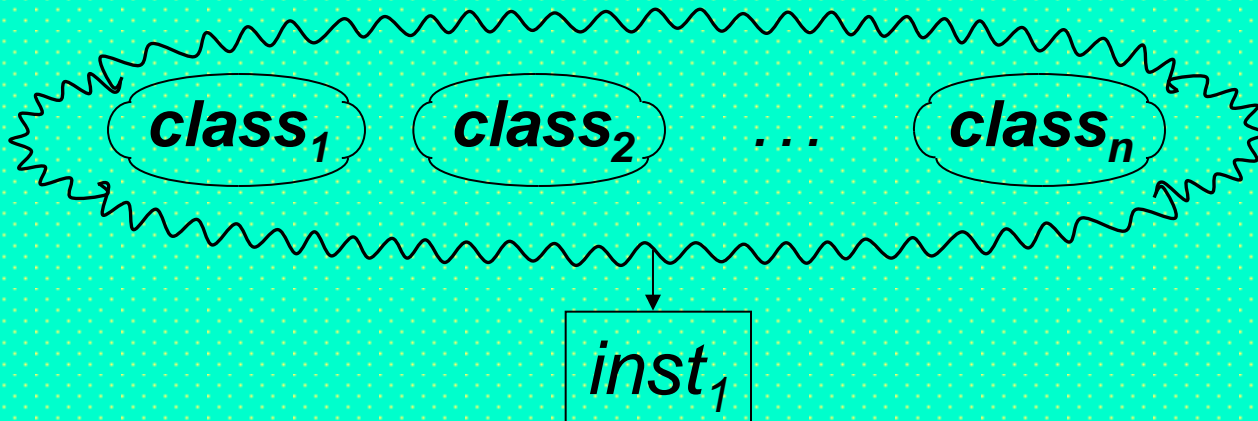
(Description)

Logic

Billionaire \sqcup
Benefactor \sqcup
Environmentalist

Intensional-Class Applications: Class Union

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

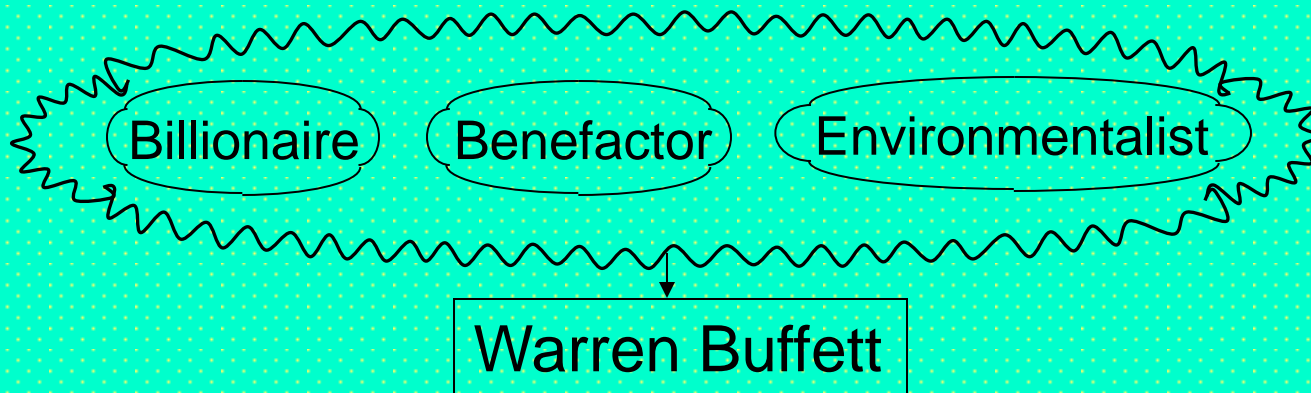


(xNS-Description)

Logic

$$\left(\begin{array}{l} class_1 \sqcup \\ class_2 \sqcup \\ \dots \sqcup \\ class_n \end{array} \right) (inst_1)$$

Example: Graph



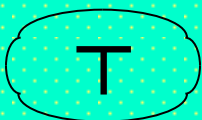
(xNS-Description)

Logic

$$\left(\begin{array}{l} Billionaire \sqcup \\ Benefactor \sqcup \\ Environmentalist \end{array} \right) (\text{Warren Buffett})$$

Class Hierarchies (Taxonomy DAGs): Top and Bottom

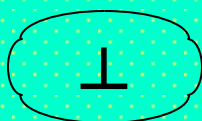
General: Top (*special* node) (Description Logic)



T

(owl:Thing)

General: Bottom (*special* node) (Description Logic)



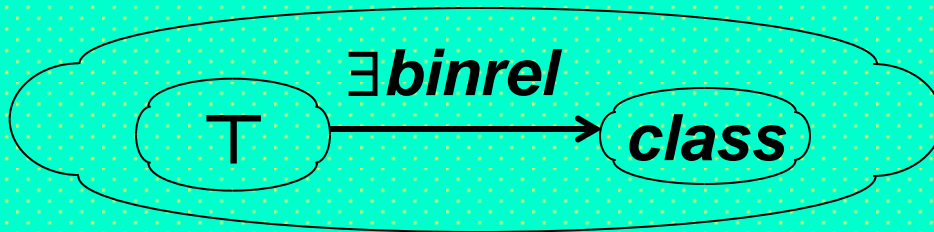
⊥

(owl:Nothing)

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

General: Graph (normal)

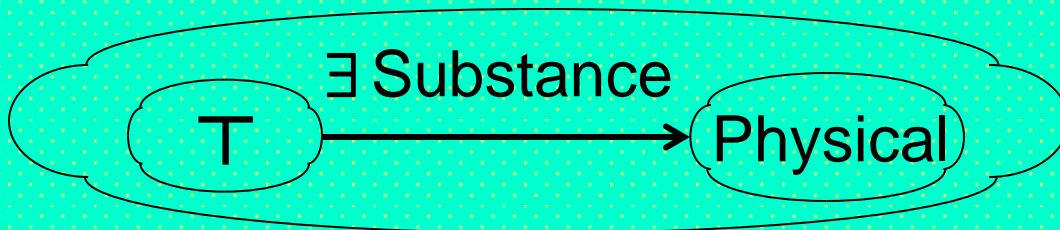
(Description)
Logic



$\exists binrel . class$

Example: Graph

(Description)
Logic

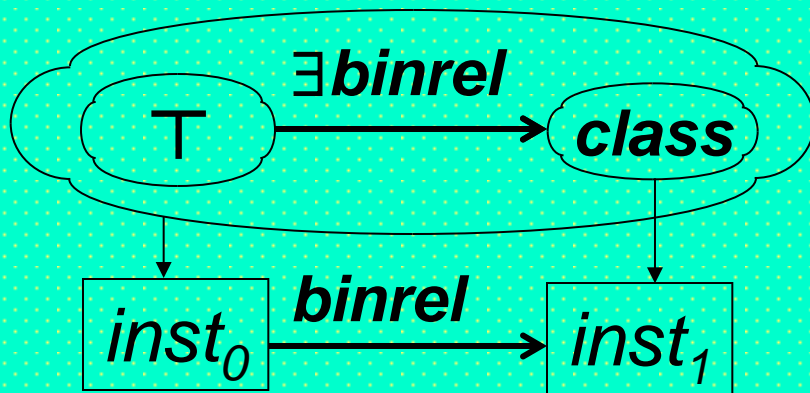


$\exists Substance . Physical$

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

Instance Assertions (Populated Ontologies): Using Restriction for ABox—Existential (1*)

General: Graph (normal)



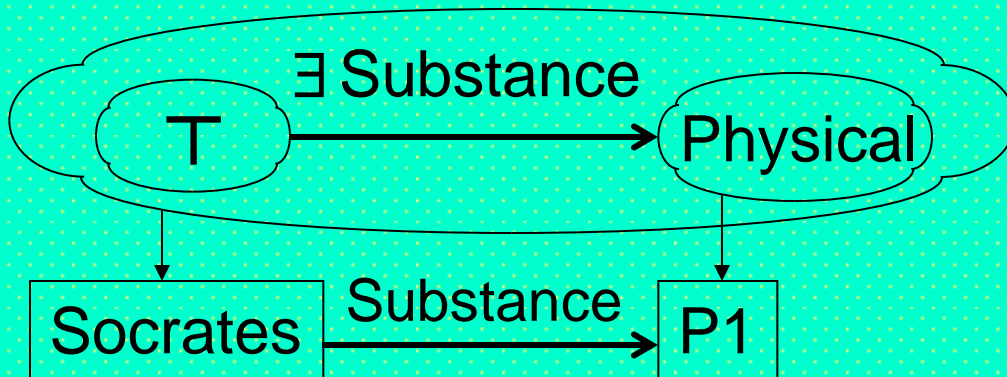
(xNS-Description)
Logic

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

$$class(inst_1) \wedge$$

$$binrel(inst_0, inst_1)$$

Example: Graph



(xNS-Description)
Logic

$$\exists Substance.Physical$$

$$(Socrates) \wedge$$

$$Physical(P1) \wedge$$

$$Substance(Socrates, P1)$$

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

General: Graph (normal)

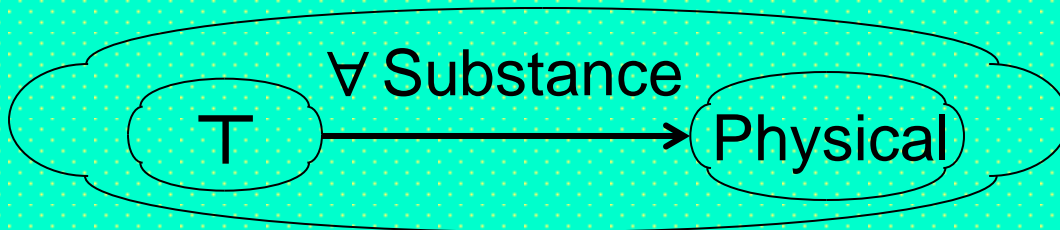
(Description)
Logic



$\forall binrel . class$

Example: Graph

(Description)
Logic



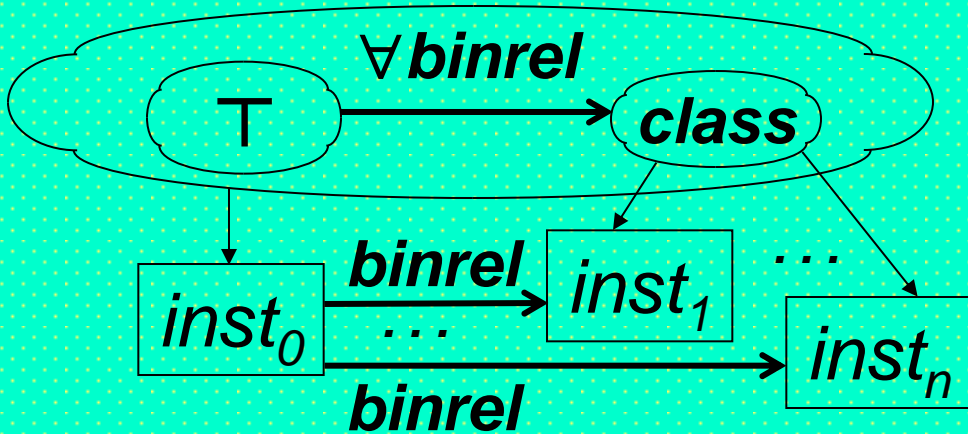
$\forall Substance . Physical$

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

Instance Assertions (Populated Ontologies):⁷⁷

Using Restriction for ABox—Universal (1*)

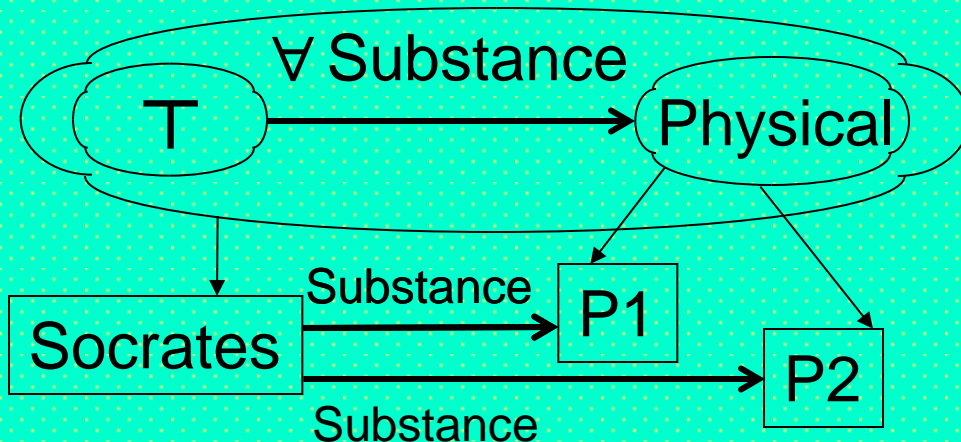
General: Graph (normal)



(xNS-Description) Logic

$$\forall \text{binrel}. \text{class}(inst_0) \wedge \text{class}(inst_1) \wedge \dots \wedge \text{class}(inst_n) \wedge \text{binrel}(inst_0, inst_1) \wedge \dots \wedge \text{binrel}(inst_0, inst_n)$$

Example: Graph



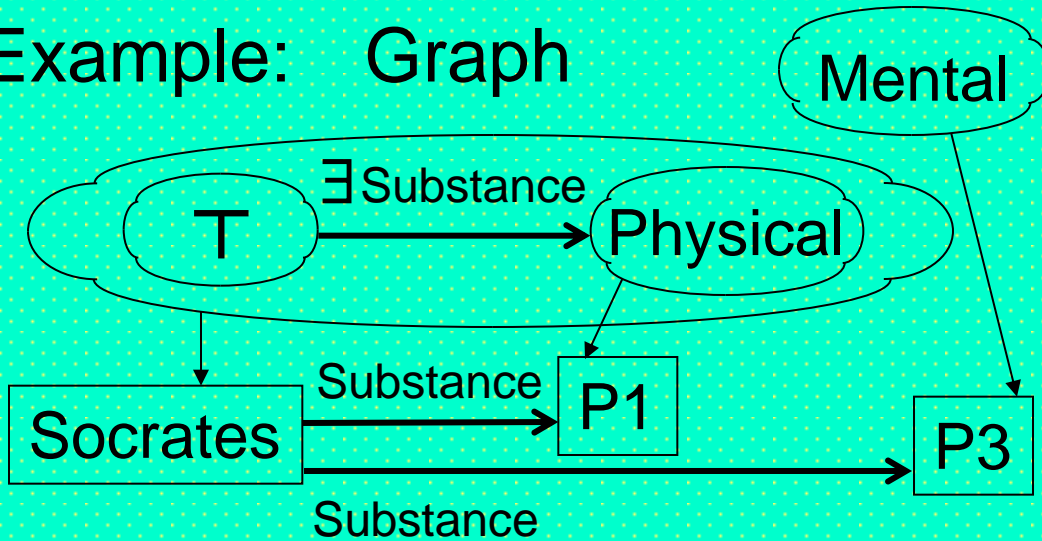
(xNS-Description) Logic

$$\forall \text{Substance}. \text{Physical}(\text{Socrates}) \wedge \text{Physical}(\text{P1}) \wedge \text{Physical}(\text{P2}) \wedge \text{Substance}(\text{Socrates}, \text{P1}) \wedge \text{Substance}(\text{Socrates}, \text{P2})$$

Existential vs. Universal Restriction

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

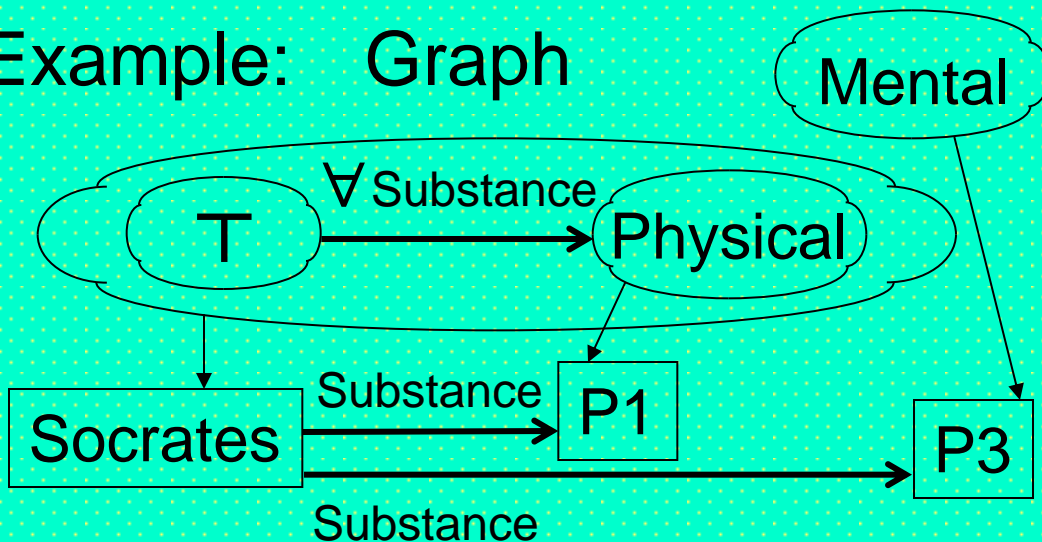
Example: Graph



(xNS-Description)
Logic

$$\begin{aligned} & \exists \text{Substance. Physical} \\ & \quad (\text{Socrates}) \wedge \\ & \text{Physical(P1)} \wedge \\ & \text{Mental(P3)} \wedge \\ & \text{Substance(Socrates, P1)} \wedge \\ & \text{Substance(Socrates, P3)} \end{aligned}$$

Example: Graph



(xNS-Description)
Logic

$$\begin{aligned} & \forall \text{Substance. Physical} \\ & \quad (\text{Socrates}) \wedge \\ & \text{Physical(P1)} \wedge \\ & \text{Mental(P3)} \wedge \\ & \text{Substance(Socrates, P1)} \wedge \\ & \text{Substance(Socrates, P3)} \end{aligned}$$

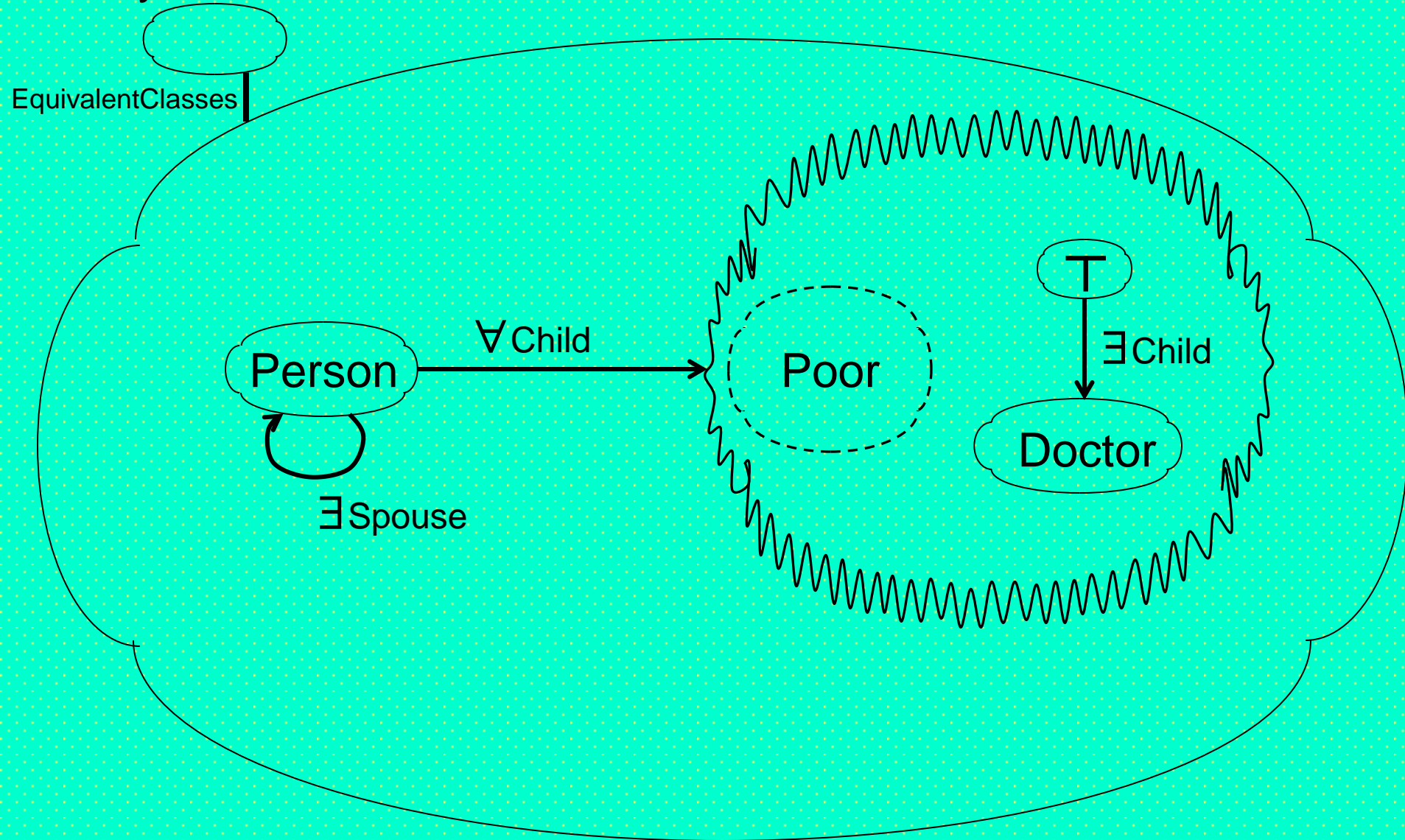
C
o
n
s
i
s
t
e
n
t

I
n
c
o
n
s
i
s
t
e
n
t

LuckyParent Example (1)

LuckyParent

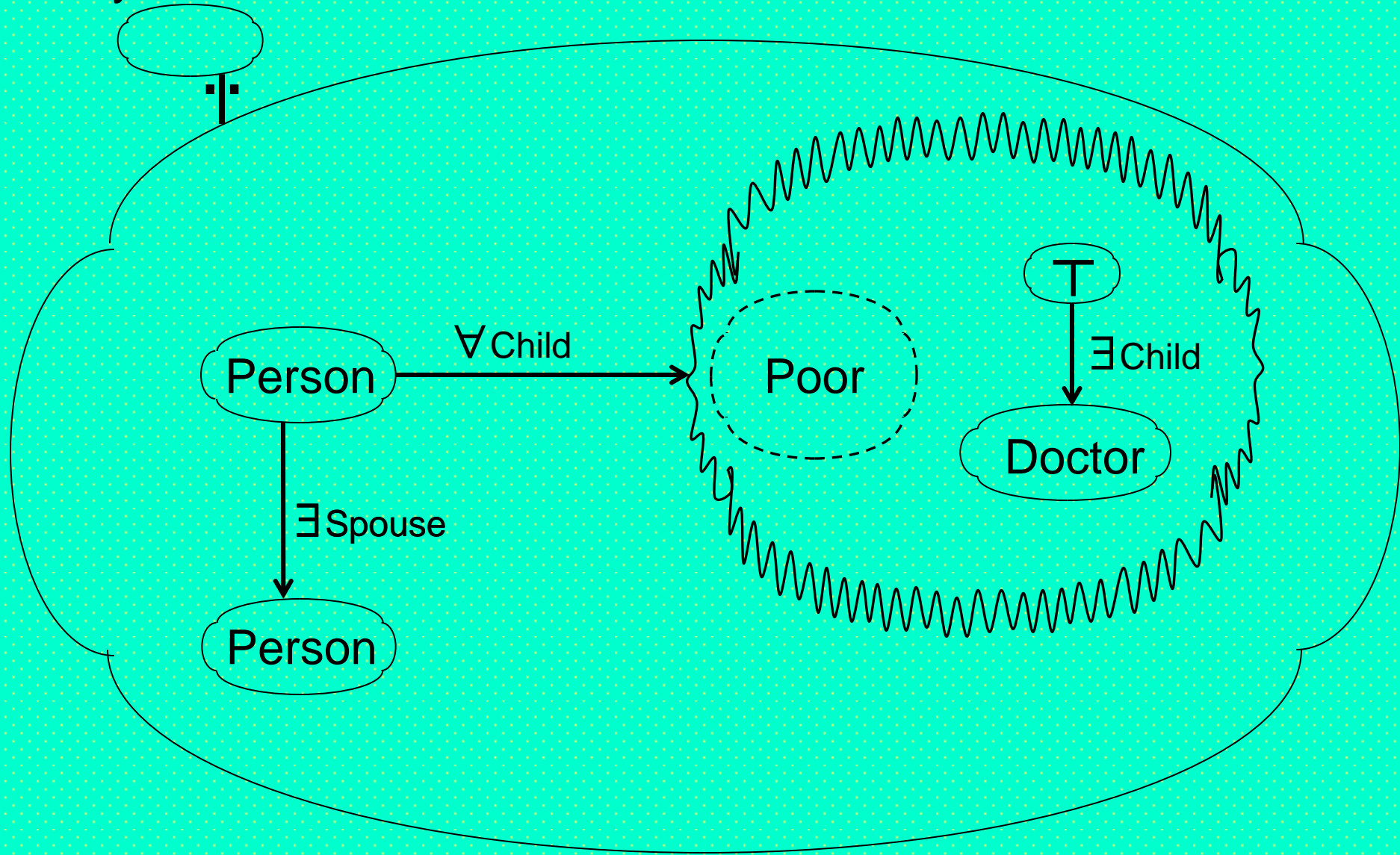
$\text{LuckyParent} \equiv \text{Person} \sqcap \exists \text{Spouse. Person} \sqcap \forall \text{Child.} (\neg \text{Poor} \sqcup \exists \text{Child. Doctor})$



LuckyParent Example (1*)

LuckyParent

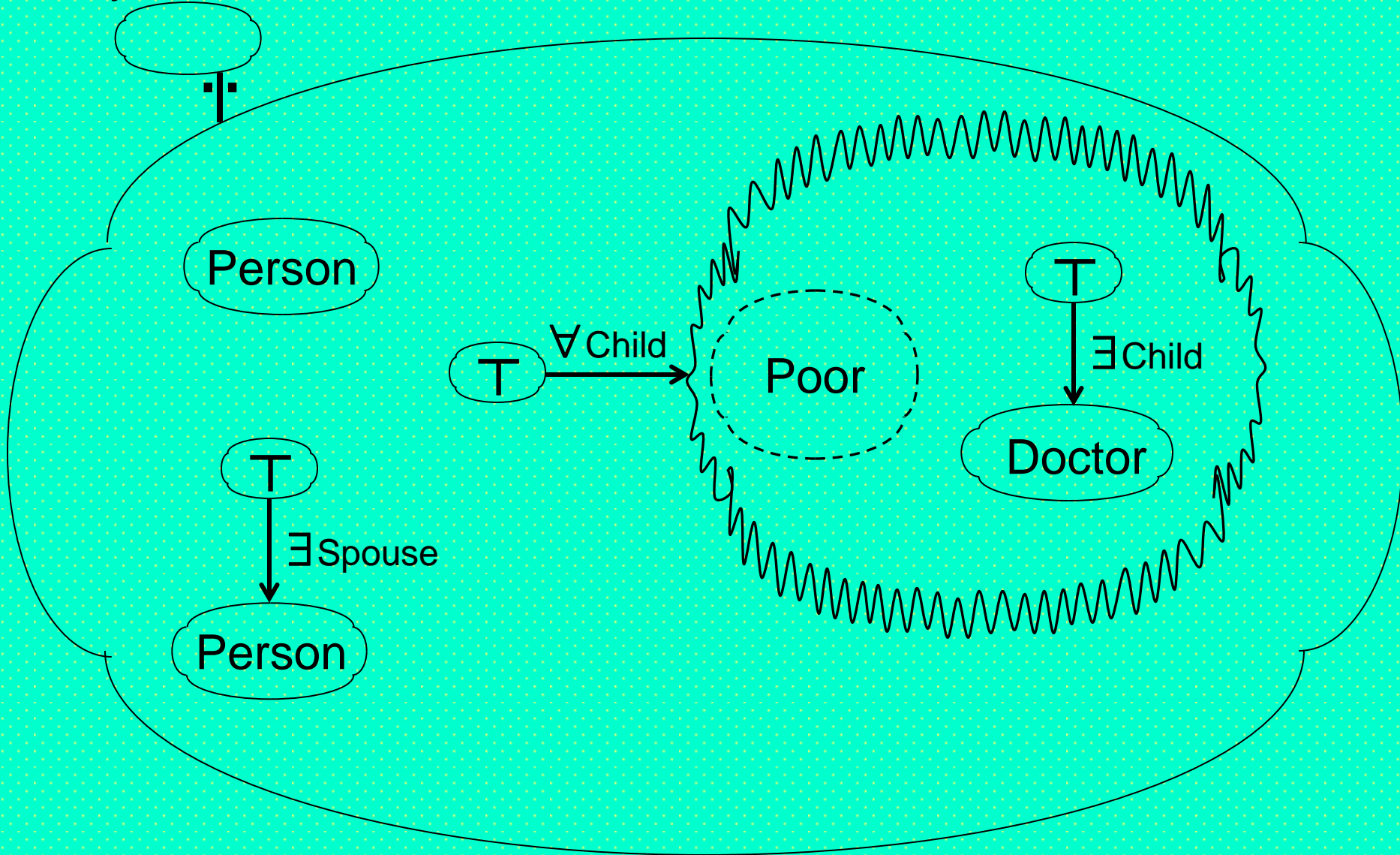
$$\text{LuckyParent} \equiv \text{Person} \sqcap \exists \text{Spouse. Person} \sqcap \forall \text{Child. } (\neg \text{Poor} \sqcup \exists \text{Child. Doctor})$$



LuckyParent Example (1**)

LuckyParent

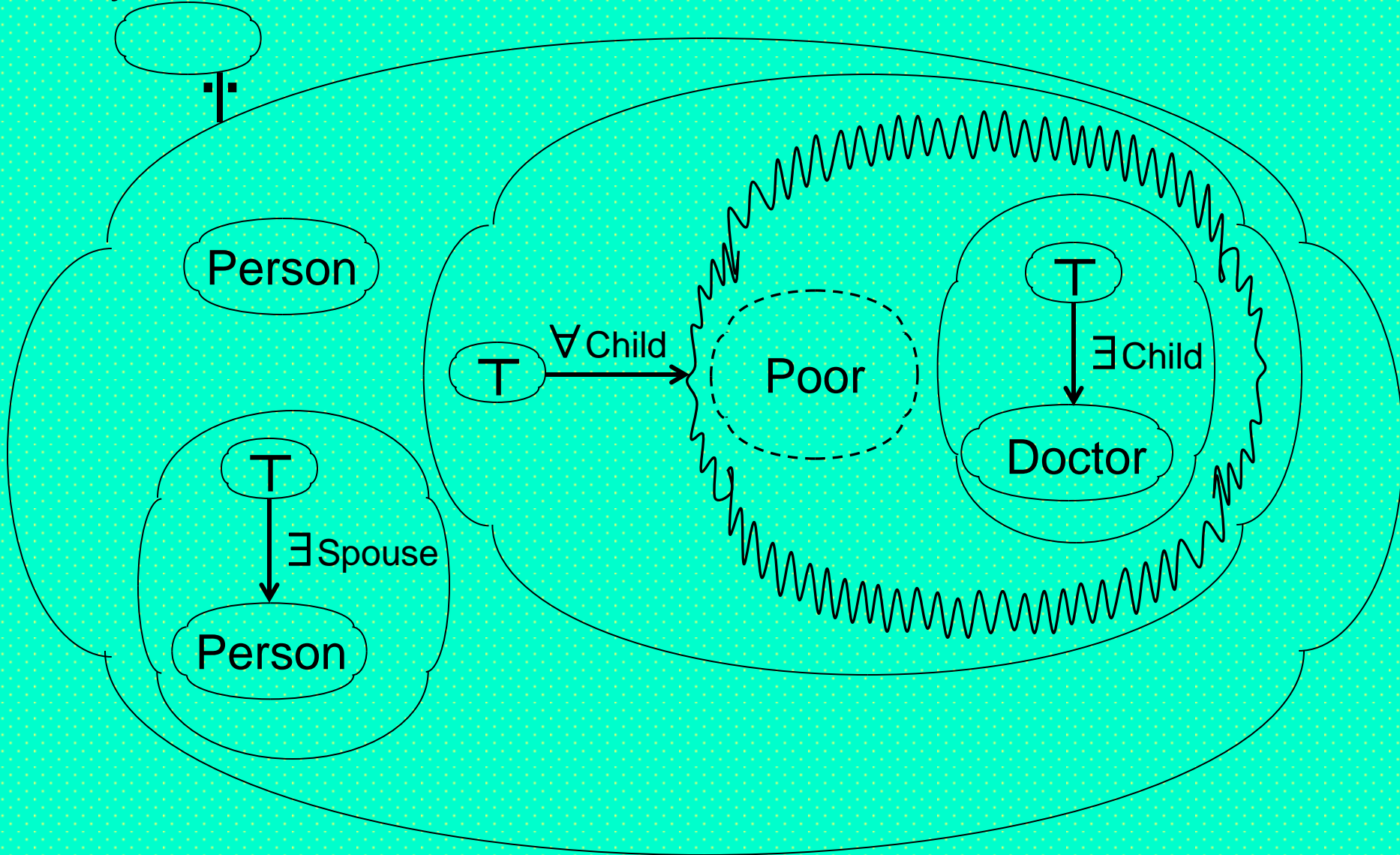
$\text{LuckyParent} \equiv \text{Person} \sqcap \exists \text{Spouse. Person} \sqcap \forall \text{Child. } (\neg \text{Poor} \sqcup \exists \text{Child. Doctor})$



LuckyParent Example (1**)

LuckyParent

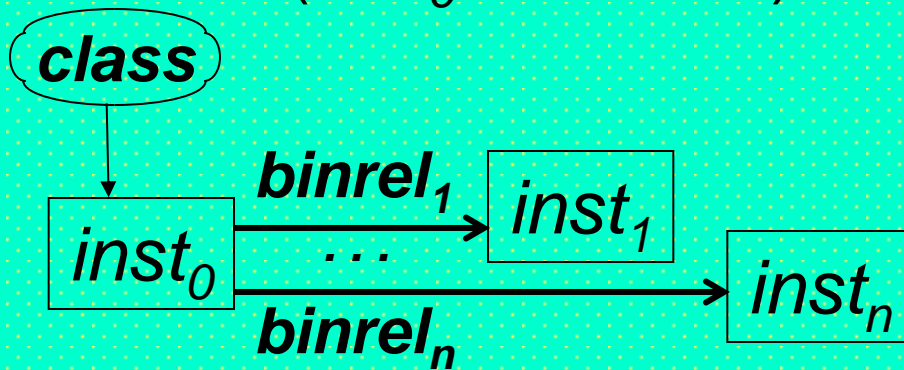
$\text{LuckyParent} \equiv \text{Person} \sqcap \exists \text{Spouse. Person} \sqcap \forall \text{Child. } (\neg \text{Poor} \sqcup \exists \text{Child. Doctor})$



Object-Centered Logic: Grouping Binary Relations Around Instance

General: Graph
($inst_0$ -centered)

(Object-Centered)
Logic



$$class(inst_0) \wedge$$

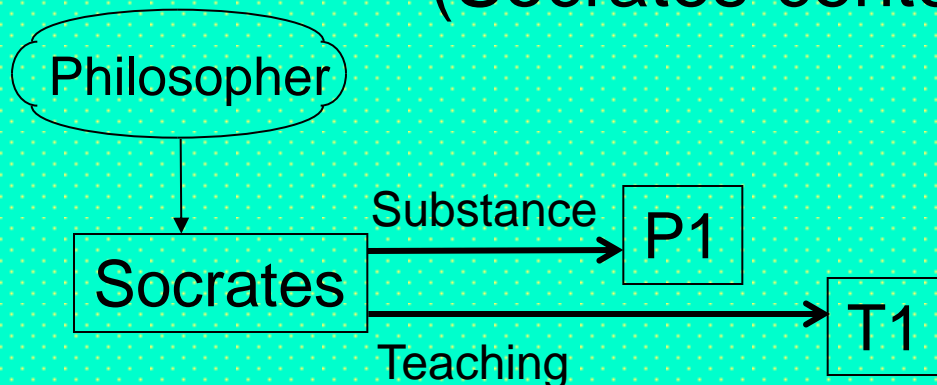
$$binrel_1(inst_0, inst_1) \wedge$$

$$\dots$$

$$binrel_n(inst_0, inst_n)$$

Example: Graph
(Socrates-centered)

(Object-Centered)
Logic



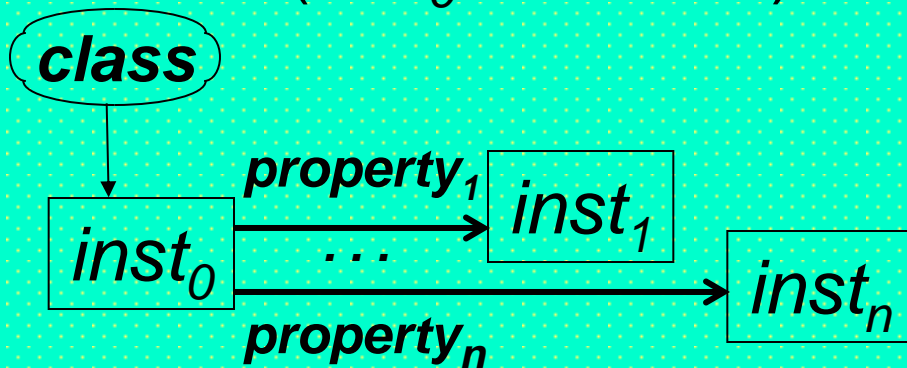
$$Philosopher(Socrates) \wedge$$

$$Substance(Socrates, P1) \wedge$$

$$Teaching(Socrates, T1)$$

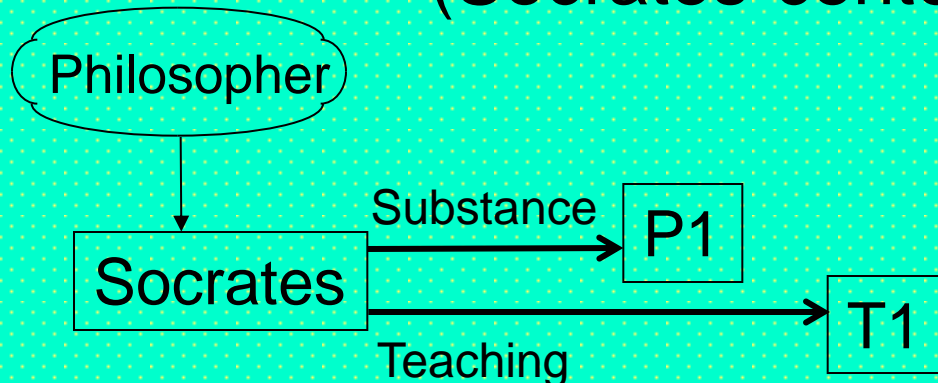
RDF-Triple ('Subject'-Centered) Logic: Grouping Properties Around Instance

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



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

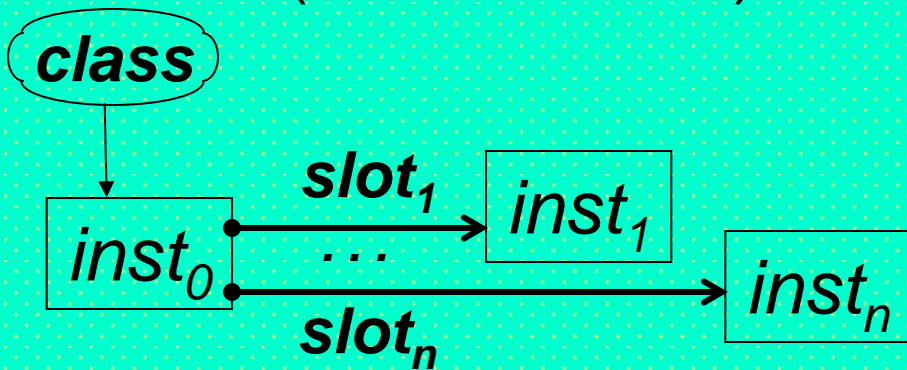
Example: Graph (Subject-Centered) Logic
(Socrates-centered)



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

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

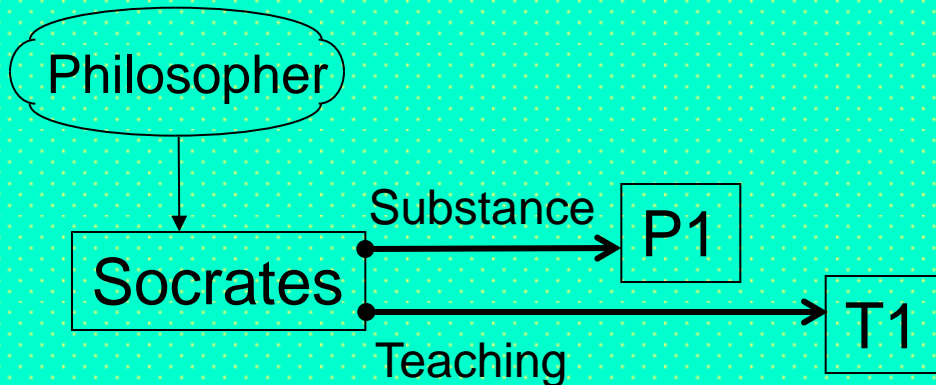
General: Graph
(*bulleted arcs*)



(PSOA Frame)
Logic

$$\begin{array}{l} inst_0 \# class(\\ \quad slot_1 \rightarrow inst_1; \\ \quad \dots \\ \quad slot_n \rightarrow inst_n) \end{array} \quad \begin{array}{l} inst_0 \in class, \\ slot_1 = inst_1, \\ \dots \\ slot_n = inst_n \end{array}$$

Example: Graph

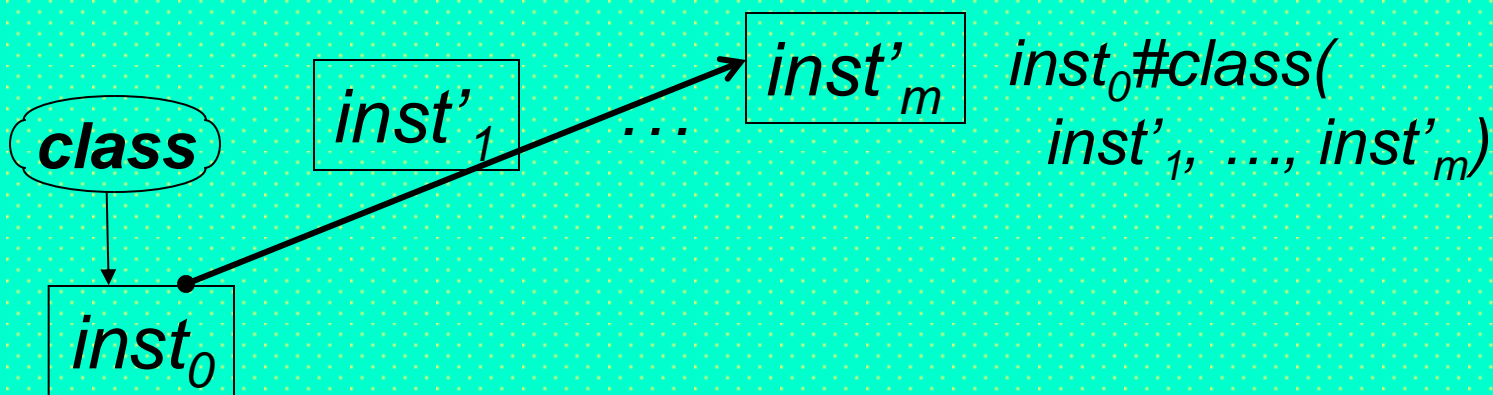


(PSOA Frame)
Logic

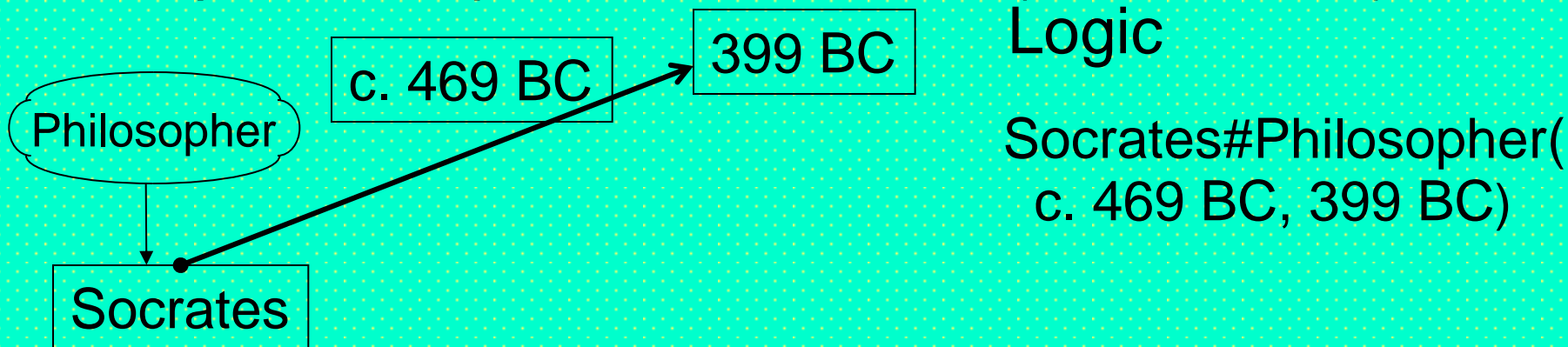
$$\begin{array}{l} Socrates \# Philosopher(\\ \quad Substance \rightarrow P1; \\ \quad Teaching \rightarrow T1) \end{array}$$

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

General: Graph (PSOA Shelf)
 (*bulleted* hyperarc) Logic

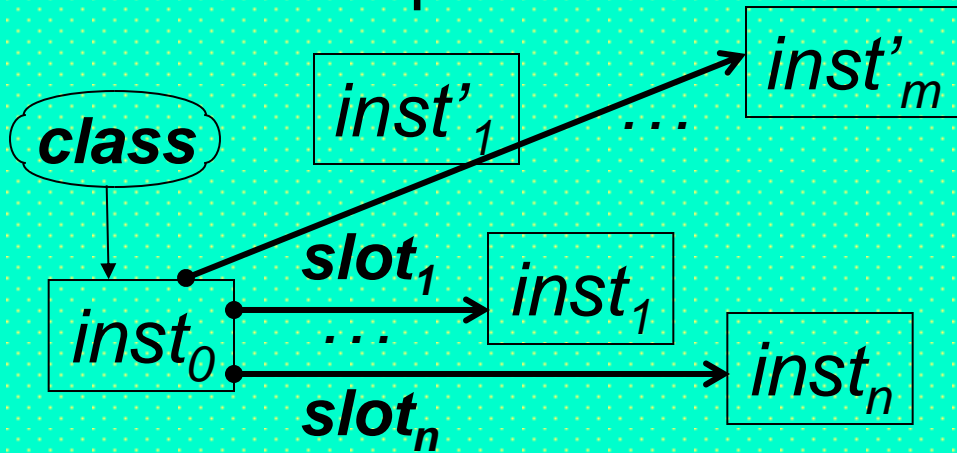


Example: Graph (PSOA Shelf)
 Logic



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

General: Graph



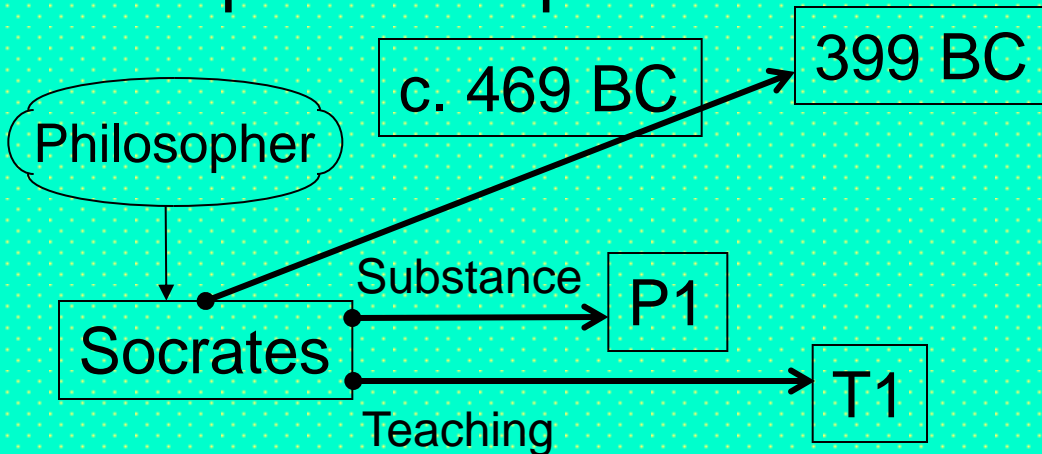
(PSOA Positional-Slotted-Term) Logic

```

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

```

Example: Graph



(PSOA Positional-Slotted-Term) Logic

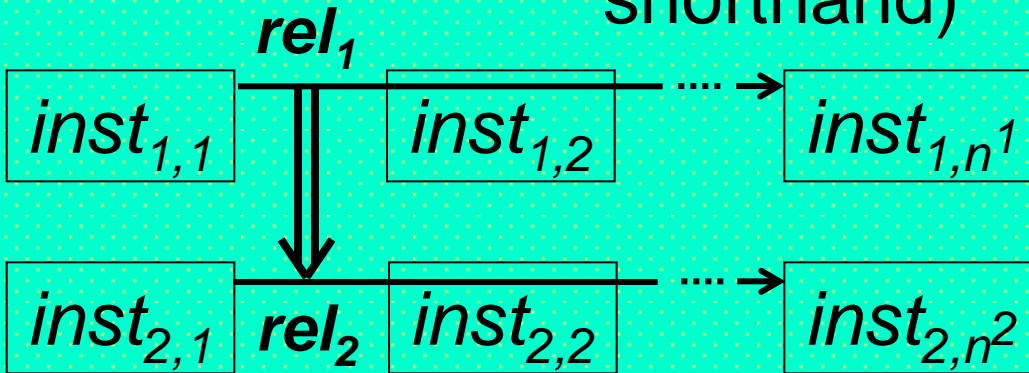
```

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

```


Rules: Relations Imply Relations (1)

General: Graph (ground, shorthand)

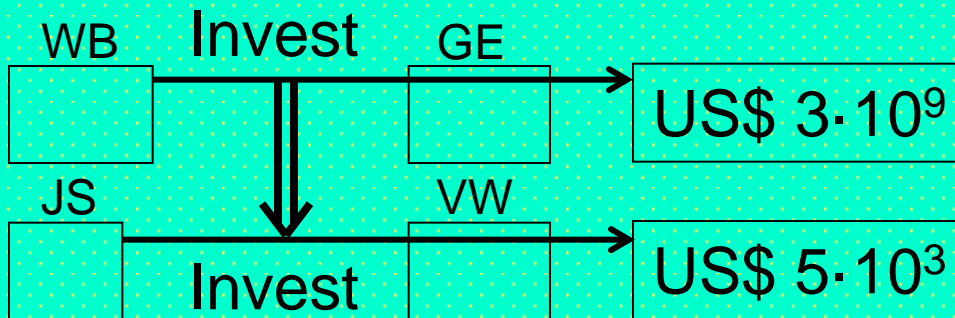


Logic

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

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

Example: Graph



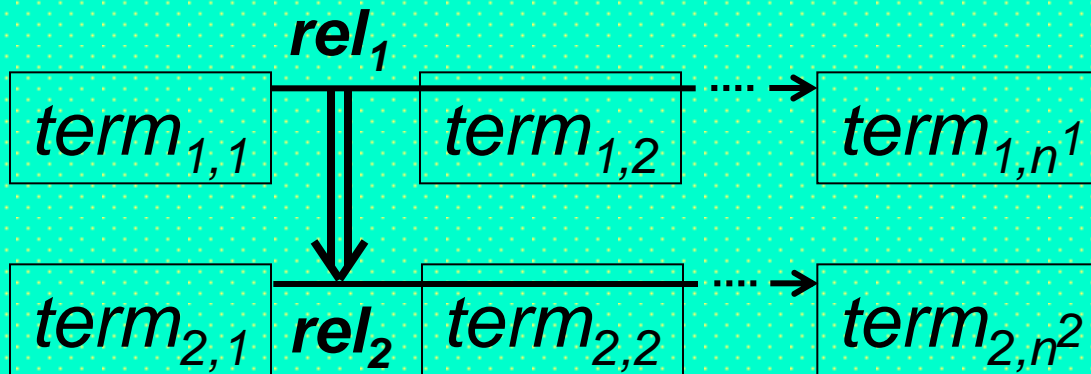
Logic

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

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

Rules: Relations Imply Relations (3)

General: Graph (inst/var terms)



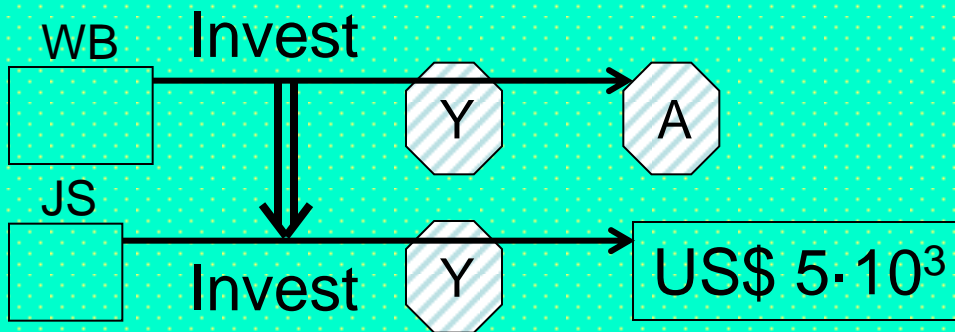
Logic

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

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

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

Example: Graph



Logic

$$(\forall Y, A)$$

$$Invest(/WB, Y, A) \Rightarrow$$

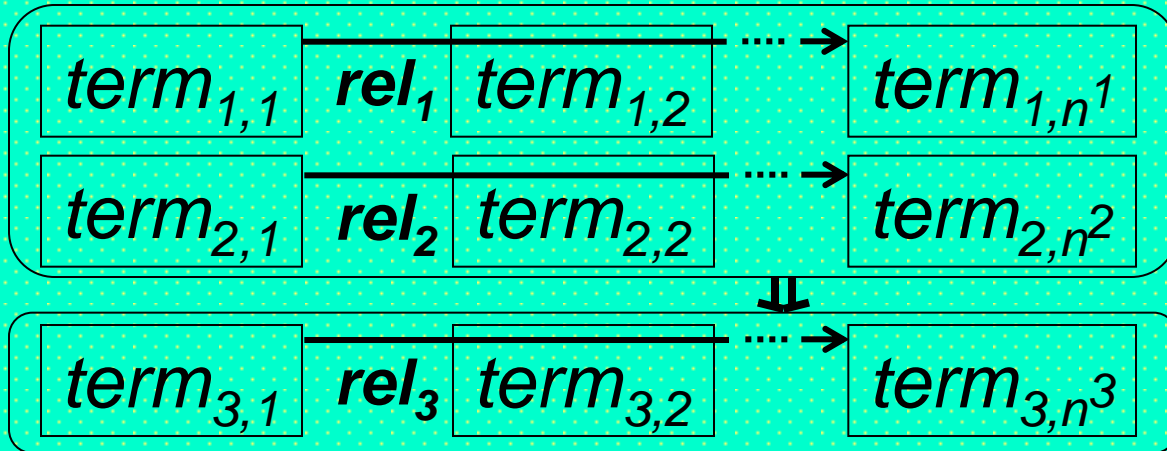
$$Invest(/JS, Y,$$

$$US\$ 5 \cdot 10^3)$$

Rules: Conjuncts Imply Relations (1*) ⁹⁴

General: Graph (prenormal)

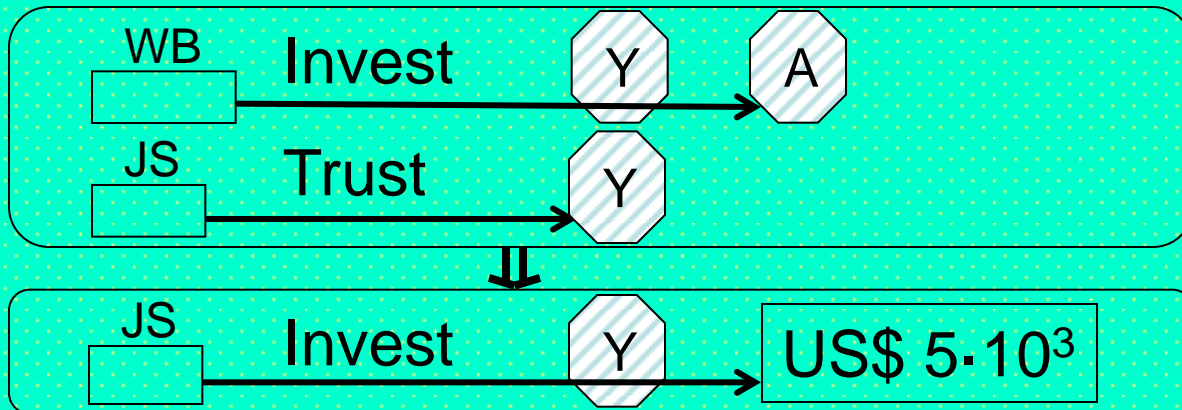
Logic



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

Example: Graph

Logic



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

Rules: Conjuncts Imply Relations (2)

Example: RuleML/XML

```

<Implies closure="universal">
  <And>
    <Atom>
      <Rel>Invest</Rel>
      <Ind unique="no">WB</Ind>
      <Var>Y</Var>
      <Var>A</Var>
    </Atom>
    <Atom>
      <Rel>Trust</Rel>
      <Ind unique="no">JS</Ind>
      <Var>Y</Var>
    </Atom>
  </And>
  <Atom>
    <Rel>Invest</Rel>
    <Ind unique="no">JS</Ind>
    <Var>Y</Var>
    <Data>US$ 5·103</Data>
  </Atom>
</Implies>

```

Logic

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

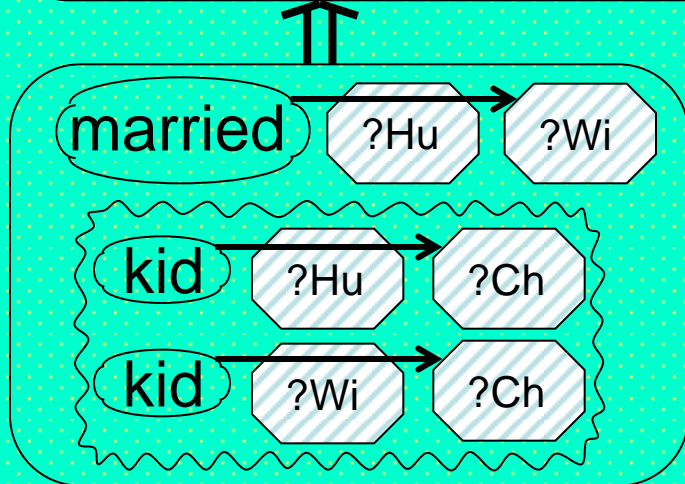
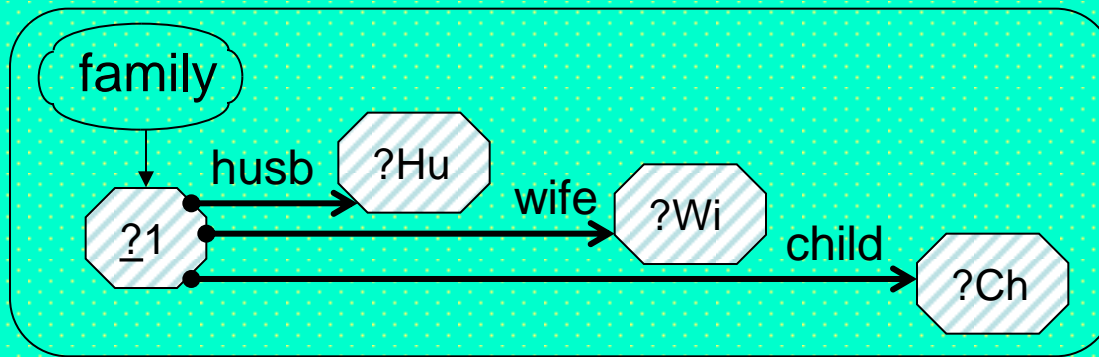
Proposing an attribute unique with value "no" for NNS, and "yes" for UNS as the default

!- superscript "3" to be parsed as Unicode U+00B3 -->

Positional-Slotted-Term Logic: Rule-defined Anonymous Family Frame

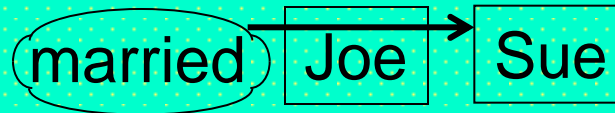
(Visualized from [IJCAI-2011 Presentation](#))

Example: Graph

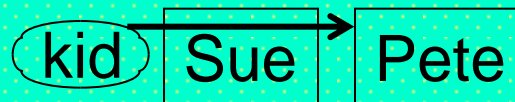


(PSOA Positional-Slotted-Term) Logic

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



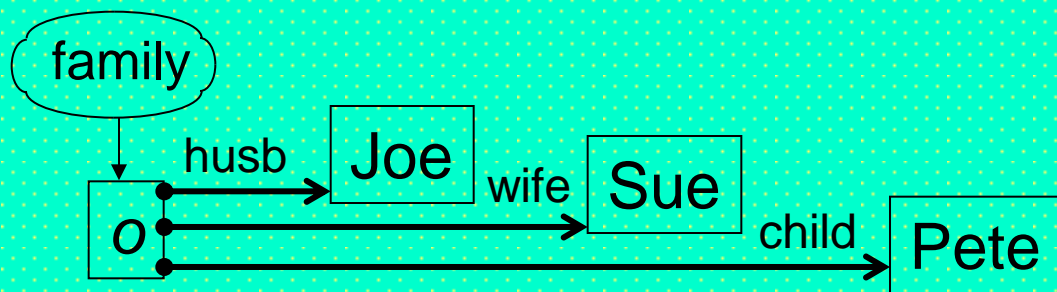
```
married(Joe Sue)
kid(Sue Pete)
```



```
)
```

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

Example: Graph

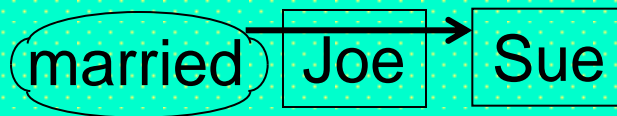


(PSOA Positional-Slotted-Term) Logic

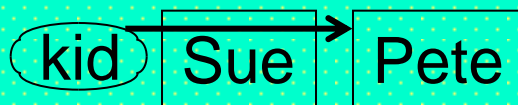
Group (

`o#family(husb->Joe
wife->Sue
child->Pete)`

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



`married(Joe Sue)
kid(Sue Pete)`

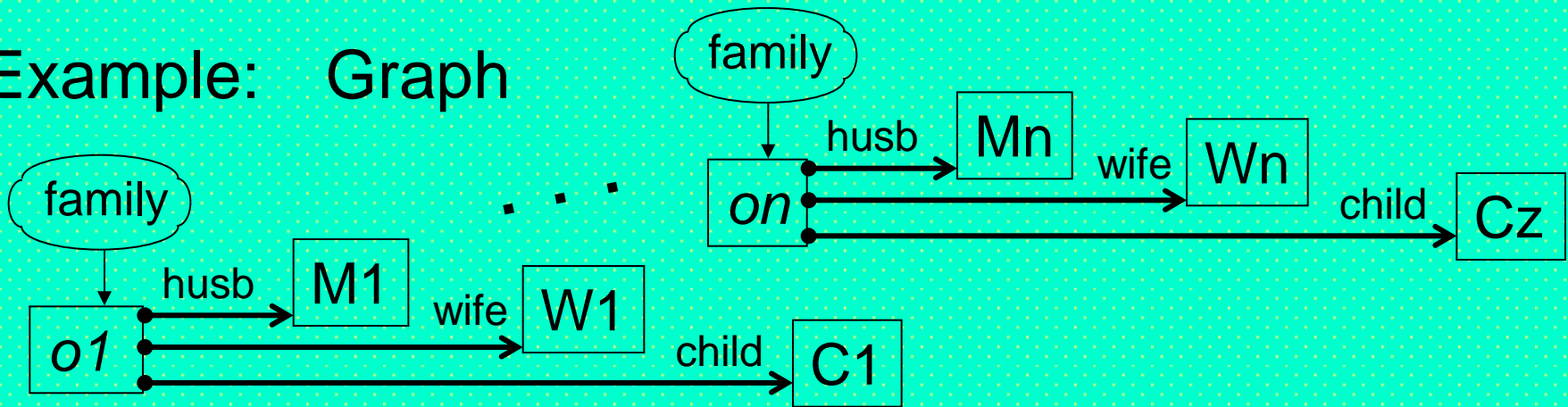


)

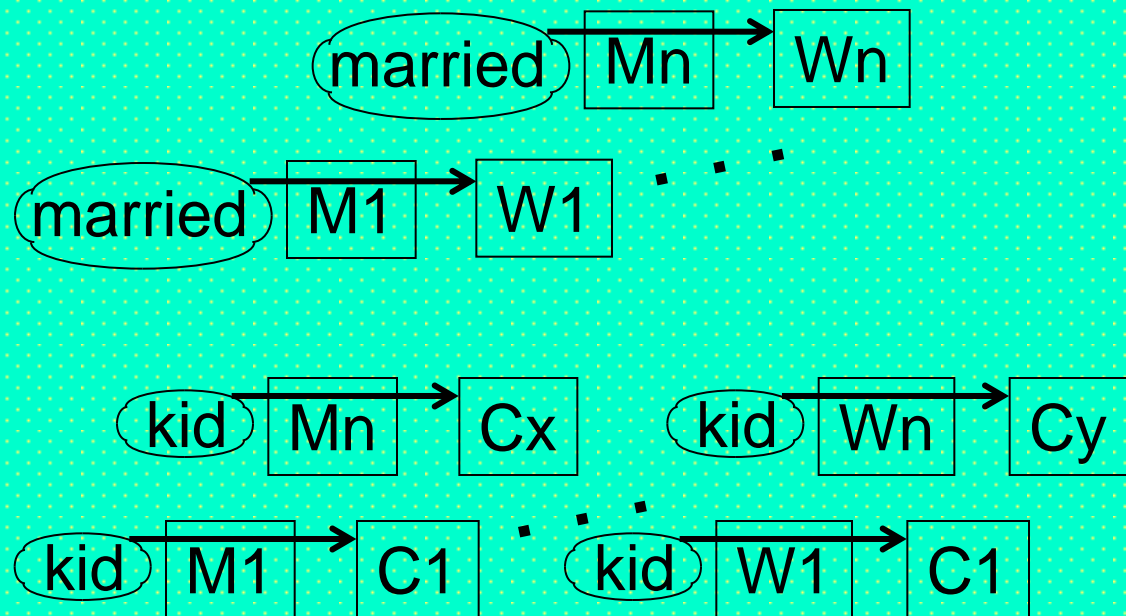
For reference implementation of PSOA querying see [PSOATransRun](#)

Positional-Slotted-Term Logic: Conversely, Given Facts, Rule Can Be Inductively Learned

Example: Graph



*Abstracting OID
constants $o1, \dots, on$
to regain existential
variable $?1$ of previous rule,
now induced from matching
relational and frame facts*


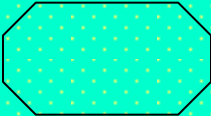
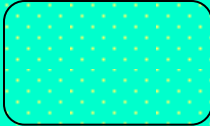
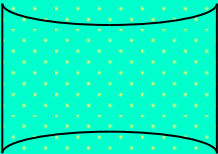
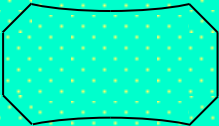
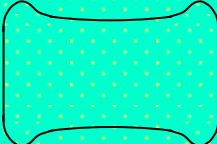
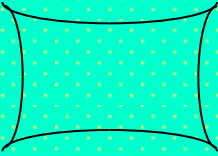
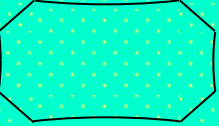
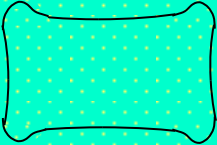
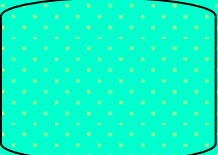
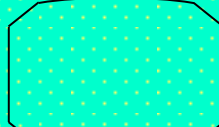

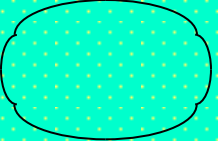
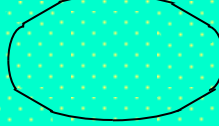
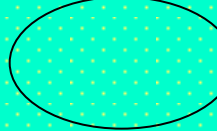


Orthogonal Graphical Features — Axes of Grailog Systematics

- Box axes:
 - Corners: pointed vs. snipped vs. rounded
 - To quote/copy vs. reify/instantiate vs. evaluate contents (cf. [Lisp](#), [Prolog](#), [Relfun](#), [Hilog](#), [RIF](#), [CL](#), and [IKL](#))
 - Shapes (rectangle-derived): composed from sides that are straight vs. concave vs. convex
 - For neutral vs. function vs. relation contents
 - Contents: elementary vs. complex nodes
- Arrow axes:
 - Shafts: single vs. double
 - Heads: triangular vs. diamond
 - Tails: plain vs. bulleted vs. colonized
- Box & Arrow (line-style) axes:
solid vs. dashed, linear vs. (box only) wavy

Graphical Elements: Box Systematics ¹⁰⁹

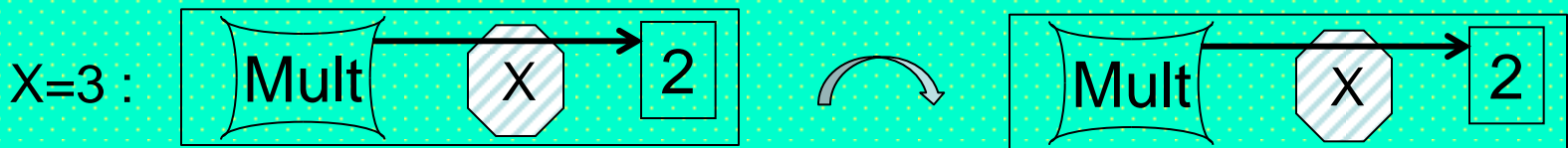
— Axes of Corners and Shapes


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

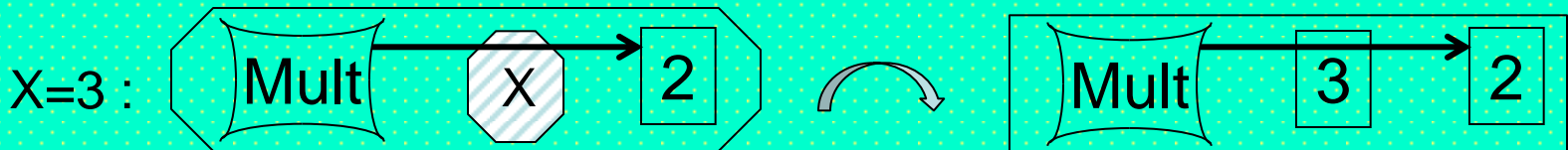
Graphical Elements: Boxes


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

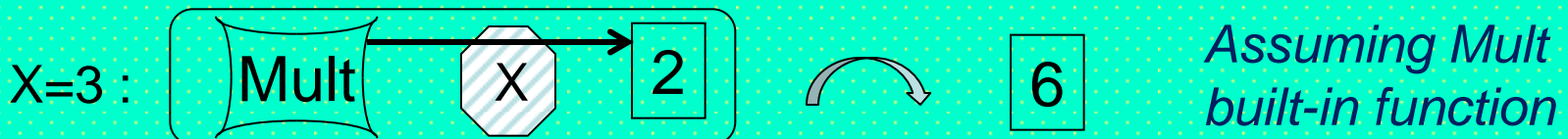
 – **Rectangle**: Neutral ‘per copy’ nodes *quote* their contents



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

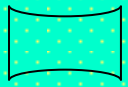


 – **Roundangle** (rounded angles): Neutral ‘per value’ nodes *evaluate* their contents through instantiation of variables and activation of function/relation applications



Graphical Elements: Boxes — Concave

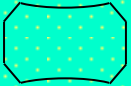
- ***Rect2cave*** (rectangle with 2 concave - top/bottom - sides):



Elementary nodes for individuals (instances).

Complex nodes for quoted instance-denoting terms
(constructor-function applications)

- ***Snip2cave*** (snipped): Elementary nodes for variables.

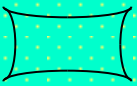


Complex nodes for instantiated (reified)
function applications

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

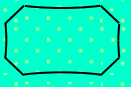


- ***Rect4cave*** (4 concave sides): Elementary nodes for fct's.

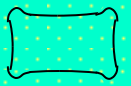


Complex nodes for quoted functional (function-denoting) terms

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

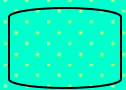


- ***Round4cave***: Complex nodes for evaluated functional
applications (active, function-returning applications)

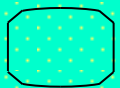


Graphical Elements: Boxes — Convex

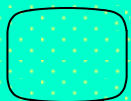
- ***Rect2vex*** (rectangle with 2 convex - top/bottom - sides):
Elementary nodes for truth constants (true, false, unknown).
Complex nodes for quoted truth-denoting propositions
(embedded relation applications)



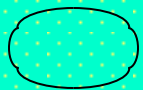
- ***Snip2vex***: Complex nodes for instantiated (reified) relation applications



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



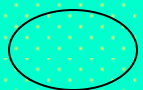
- ***Rect4vex***: Elementary nodes for relations, e.g. unary ones (classes). Complex nodes for quoted relational (relation-denoting) terms



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



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



Conclusions (1)

- Grailog 1.0 incorporates feedback on earlier versions
- Graphical elements for novel box & arrow systematics using orthogonal graphical features
 - Leaving color (except for IRIs) for other purposes, e.g. highlighting subgraphs (for retrieval and inference)
- Introducing Unique vs. Non-unique Name Specification
- Focus on *mapping* to a family of logics as in RuleML
- Use cases from *cognition* to *technology* to *business*
 - 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
- For Grailog, aligned with Web-rule standard RuleML:
<http://wiki.ruleml.org/index.php/Grailog> ([test suite](#))

Conclusions (2)

- **Symbolic-to-visual** translators started as Semantic Web Techniques Fall 2012 Projects:
 - Team 1 A Grailog Visualizer for Datalog RuleML via XSLT 2.0 Translation to SVG by Sven Schmidt and Martin Koch:
An Int'l Rule Challenge 2013 paper & demo introduced Grailog KS Viz
 - Team 8 Visualizing SWRL's Unary/Binary Datalog RuleML in Grailog by Bo Yan, Junyan Zhang, and Ismail Akbari:
A Canadian Semantic Web Symposium 2013 paper gave an overview
- Grailog invites feature **choice** or *combination*
 - E.g. **n-ary hyperarcs** or **n-slot frames** or *both*
- **Grailog Initiative** on open standardization calls for further feedback for future 1.x versions

Future Work (1)

- Refine/extend Grailog, e.g. along with [API4KB](#) effort
 - Compare with other graph formalisms, e.g. Conceptual Graphs (<http://conceptualstructures.org>) and [CoGui](#) tool
 - Define mappings to/fro UML structure diagrams + OCL, adopting UML behavior diagrams (<http://www.uml.org>)
- Implement further tools, e.g. as use case for (Functional) RuleML (<http://ruleml.org/fun>) engines
 - More mappings between graphs, logic, and RuleML/XML:
 - Grailog *generators*: Further symbolic-to-visual mappings
 - Grailog *parsers*: Initial visual-to-symbolic mappings
 - Graph indexing & querying (cf. <http://www.hypergraphdb.org>)
 - Graph transformations (normal form, [typing homomorphism](#), merge, ...)
 - Advanced graph-theoretical operations (e.g., path tracing)
 - Exploit Grailog parallelism in implementation

Future Work (2)

- Develop a Grailog structure editor, e.g. supporting:
 - Auto-specialize of neutral application boxes (angles) to function apps (2caves) or relation apps (2vexes), depending on contents
 - Auto-specialize of neutral operator boxes (angles) to functions (4caves) or relations (4vexes), depending on context
- Benefit from, and contribute to, Protégé visualization plug-ins such as [Jambalaya/OntoGraf](#) and [OWL Viz](#) 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.