Concept Relationship Lattice Logic

where Conceptual Modelling and Plural Logic meet

by Gert Schmeltz Pedersen, crllsoft.dk , 21 June 2026

The web site crllsoft.dk is concerned with the theory and practice of **Concept Relationship Lattice Logic**, **CRL Logic**, or **CRLL**, pronounced 'cril logic' or just 'crill'.

You may join as registered user of CRLL Prototype.

CRL Logic is the underlying logic of the Relationship Lattice Formalism, which was defined and explored in a ph.d. project [1993a] [1993b] [1993c] [1994a] [1994b] [1994c] [1995], on a background of lattice theory [1990], concept algebra [1992b] [1993e], conceptual graphs [1976b] [1984a], mereology [1940] [1987] [1991], entity-relationship modelling [1976a], object-oriented software development and databases [1982] [1993d], relational databases [1970] [1975], and bibliographic databases [1992a].

The ph.d. project was directed and supervised by Jens Langeland-Knudsen, COO at Computer Resources International A/S, by Professor Hans Siggaard Jensen, Copenhagen Business School, by Professor Jørgen Fischer Nilsson, Technical University of Denmark, and by Professor Peter Ingwersen, Royal School of Library and Information Science.

Conceptual modelling has roots in the 1960's and 1970's, where Chen's entity-relationship diagrams [1976a] and Sowa's conceptual graphs [1976b] [1984a] [2000] were developed for software and database designs and for knowledge representation. CRL Logic owes much to Sowa's oeuvre. Ontological conceptual modelling [2008a] [2022a] is an important recent development.

Conceptual modelling is an abstraction process. It is like drawing a geographical map. Only a small fraction of the features of the real thing is expressed in the model, only those, which serve the purpose.

The use of the term "conceptual modelling" became common in computing science during the 1970's alongside such terms as "semantic data modelling" in the database area, "knowledge representation" in the artificial intelligence area, and "abstraction" in the programming language area [1982]. The contributions from these three areas were assembled by Brodie et al. [1984b]. In their view, conceptual modelling needs can be met only by a leap to a higher, more abstract level of system descriptions, like the leap from assembly languages to high level programming languages. The fundamental characteristic of the new level of system descriptions is that it is closer to the human conceptualization of a problem domain. Descriptions at this level can enhance

communication between system designers, domain experts and ultimately, system end-users. CRL Logic offers a style of conceptual modelling, which is applicable in all three areas.

Plural Logic [2016a] [2018] [2020] [2021a] [2022b] is First Order Logic with plural terms, plural variables, plural predication, and plural quantification, and it has the expressive and deductive power of monadic Second Order Logic.

Plural Logic has roots in mathematical logic and metaphysics. Recently many references discuss further aspects of plural logic, e.g. superplurals, plural higher order logic, critical plural logic, multigrade phenomena, etc. These discussions indicate that First Order Logic and Plural Logic are stepping stones to a more complete logic. We investigate CRL Logic as such a more complete logic, including considerations about Second-order and Higher-order Logic [2024d], but based on both plurals/pluralities and sets, as discussed in chapter 4 of [2021a]. Our example meta-level CRLL model shows that properties may be quantified in CRL Logic.

Concepts are all the thoughts and ideas you have about physical or immaterial things, factual or imagined, countable or measurable, propositions, situations, events, time, space, stuff, measurements, numbers, texts, data structures, URIs, predicates, functions.

Relationships are dual properties, relating two concepts, such that each concept gets a property with the other concept as the value. Relationships may be potential or actual.

Potential relationships have minimum and maximum cardinalities as in entity-relationship modelling, they are like schema declarations in a database system. CRL Logic uses **relationship modifiers** to incorporate both cardinalities and logical quantification, e.g.:

- A cardinality of (1:n) is like existential quantification
- (n:n) is like universal quantification
- (0:0) means NOT
- (2:n) or (m:n) is collective predication
- versus distributive predication, (1:n)
- generalized quantifiers [2024b] may be e.g. (1:few_units), (many_units:all_units), (one_pair_of_units:all_pairs_of_units)

Actual relationships are like the data in a database system.

CRL Logic relationships owe much to relational database systems, one of the most successful software technologies.

Lattices consist of ordering links between a superconcept and a subconcept, giving a partial order. The partial order is a "conceptual part-of", not a "physical part-of" as in physical mereology. A subconcept inherits properties from its superconcepts. So properties are like the differentiae of ontologies.

CRL Logic lattices with relationships owe much to object-oriented programming with class hierarchies, also one of the most successful software technologies.

A **Concept Relationship Lattice**, or **CRLL model**, or **CRLL model module**, is a boolean lattice, where all sums and products are implicitly present on all defined concepts, although most sums are not very meaningful, like the lattice sum of a trout and a turkey, or the lattice sum of the Eiffel Tower and president de Gaulle (examples from the mereology literature).

The meaning of a concept c is given by its **intension** and its **extension**. The intension consists of all the properties of c, including inherited properties. The extension consists of c itself and all the concepts below c in the lattice, a sublattice. Thus the intension of a superconcept is part-of the intension of its subconcepts, and the extension of a subconcept is part-of the extension of its superconcepts. So the lattice sum of a trout and a turkey has very little intension, just properties like weight and length.

Considerations of both intension and extension are necessary, when modelling what something *is*, preventing erroneous conceptual modelling, such as "the committee *is* the group of persons", or "the gold ring *is* the mass of gold".

A CRLL model represents a conception of a domain of interest. CRLL models may be merged in order to represent more complex domains. So CRLL models are modules that may build up, endlessly. Sometimes, contradictions may be detected among modules, when merged. Detecting contradictions between conceptual models may be a way to resolve disputes about misinformation. How to detect contradictions is a possible subject for scientist and student projects.

A concept called **TOP** represents everything in the domain, it has no superconcepts, nor properties.

A concept called **BOTTOM** represents nothing in the domain, it has no subconcepts.

A concept with a declared ordering link to BOTTOM is called a **unit** or **unit concept** (previously called an atom). A unit has a cardinality of 1.

A concept above units in the ordering is called a **plural** or **plural concept**. A plural has a cardinality, which is either the number of units below it in the ordering or a declared cardinality as a shorthand for the sum of unspecified units, e.g. a subconcept of Person labelled Trio with a cardinality of 3. The cardinality may be 0, while it has no units declared below it, in which case the plural has an implicit ordering link to BOTTOM.

In model-theoretic terms, the plural concepts and their potential relationships form a theory, and the unit concepts and their sums and actual relationships form a model.

A **query**, or **query concept**, is like a plural concept, except its extension is evaluated when required. Thus inferences may be performed on CRLL models in order to reason and deduce new insights, as by queries in a database system, or by formulas in First Order Logic.

Philosophy and linguistics have a distinction between **countables**, e.g. persons and physical things, **and mass terms**, e.g. water and gold. This has a long history [2024a]. CRL Logic as such does not make the distinction between countable concepts and mass concepts. In our example CRLL models of masses, mass concepts have measures to define the least value of their unit concepts. So when merging two model modules, where one measures water in liters, the other in milliliters, then an implementation shall transform to the most specific, milliliters.

CRL Logic is Plural Logic **without the distinction between an element and the singleton set consisting of that element**, whereby the Russell paradox of set theory is avoided, like in mereology. Is the expressive power of CRL Logic therefore reduced compared to Plural Logic? No, on the contrary, it brings clarity to our models, and the singleton sets reappear in plural form from chains of relationships, as we will see in many examples, e.g. Person and Committee. Or phrased differently, a unit concept has properties inherited from its plural superconcepts (e.g. Person), while related plural concepts (e.g. Committee) have their particular properties, just like elements and sets have their separate properties.

Modal logic [2023] has modal operators, where ◊ and □ represent possibility and necessity. It is likely and debatable, that they are equivalent to the potential and actual relationships of CRL Logic. Other modal operators may be included in CRL Logic after further considerations. Among our examples, we propose a CRLL model of the sentence

"Tom believes Mary wants to marry a sailor."

as discussed by John Sowa [2002]. There, we have modal aspects of beliefs and desires to be modelled.

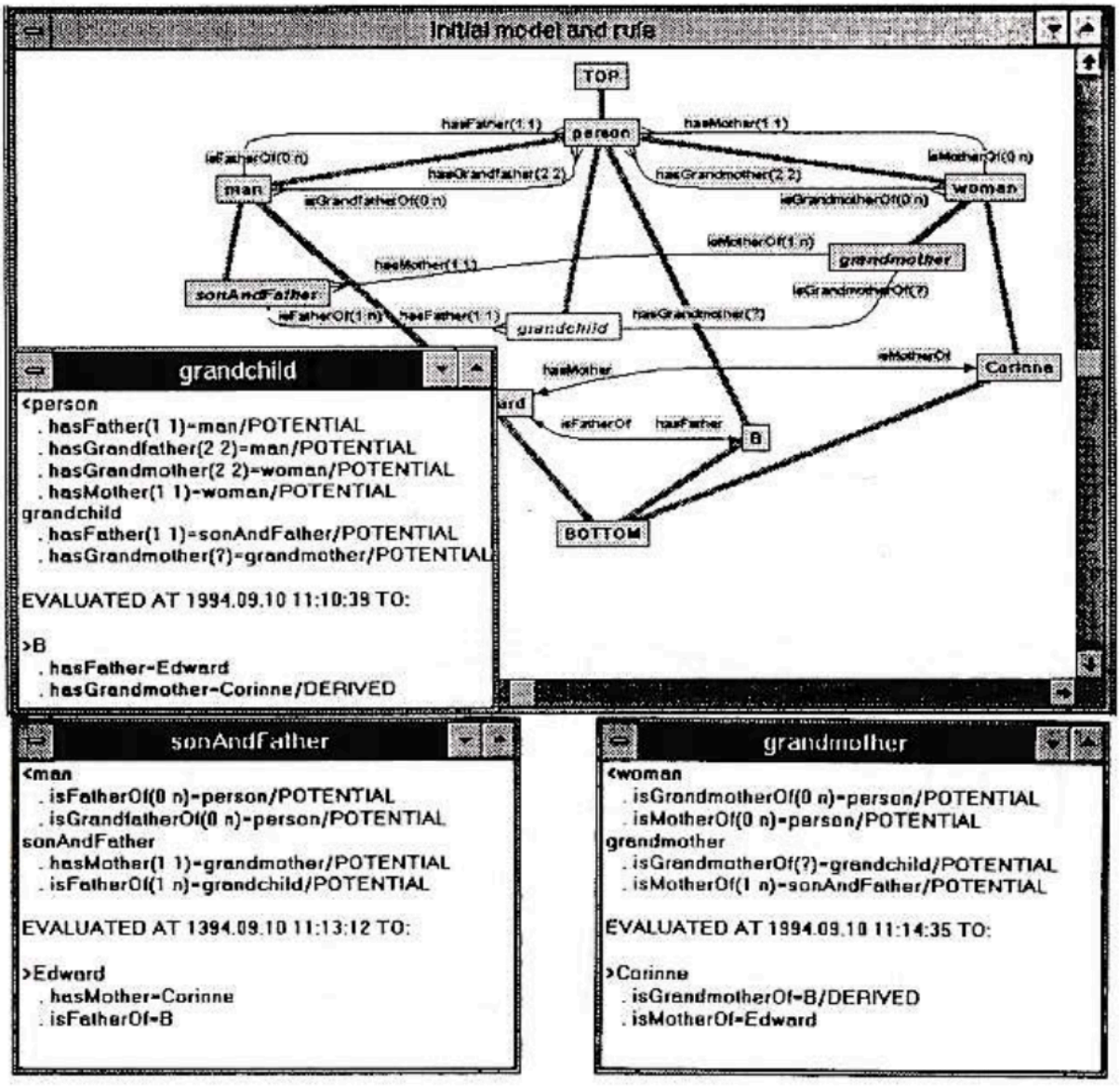
Tools and Examples

CRLL models are abstract creatures in your mind. You need software tools in order to build, visualize, communicate, merge, utilize, and manage them, and to exchange them with others.

In [1995] a very incomplete tool was developed, called RLL for Relationship Lattice Laboratory, programmed in Smalltalk, an object-oriented programming language, on a portable Mac computer.

Diagrams was used to visualize CRLL models, inspired by Hasse diagrams of lattices [1990] and by entity-relationship diagrams in a variant of the original form by Chen [1976a].

Here is an example of the use of RLL:



RLL figure 36 - Family Relationships

Here, family relationships are modelled between the plural concepts labelled person, man, and woman.

Further, three query concepts with labels in italics, *grandmother*, *sonAndFather*, and *grandchild*, model the rule that a woman is grandmother of a person, the grandchild, if she is the mother of a man, her son, who is the father of the grandchild.

When three unit concepts labelled Corinne, Edward and B, are defined, then the relationship

Corinne • isGrandmother0f ► ◄ hasGrandmother • B

may be derived.

Subwindows for each of the query concepts show their intensions and evaluated extensions.

The current tool development, CRLL Prototype, is used in the following example.

The term *regimentation* is used in philosophy for the transformation of natural language sentences into a formal language, similar to our use of the term *conceptual modelling*.

The most used example of a natural language sentence regimented into Plural Logic is the Geach-Kaplan sentence:

"Some critics admire only one another".

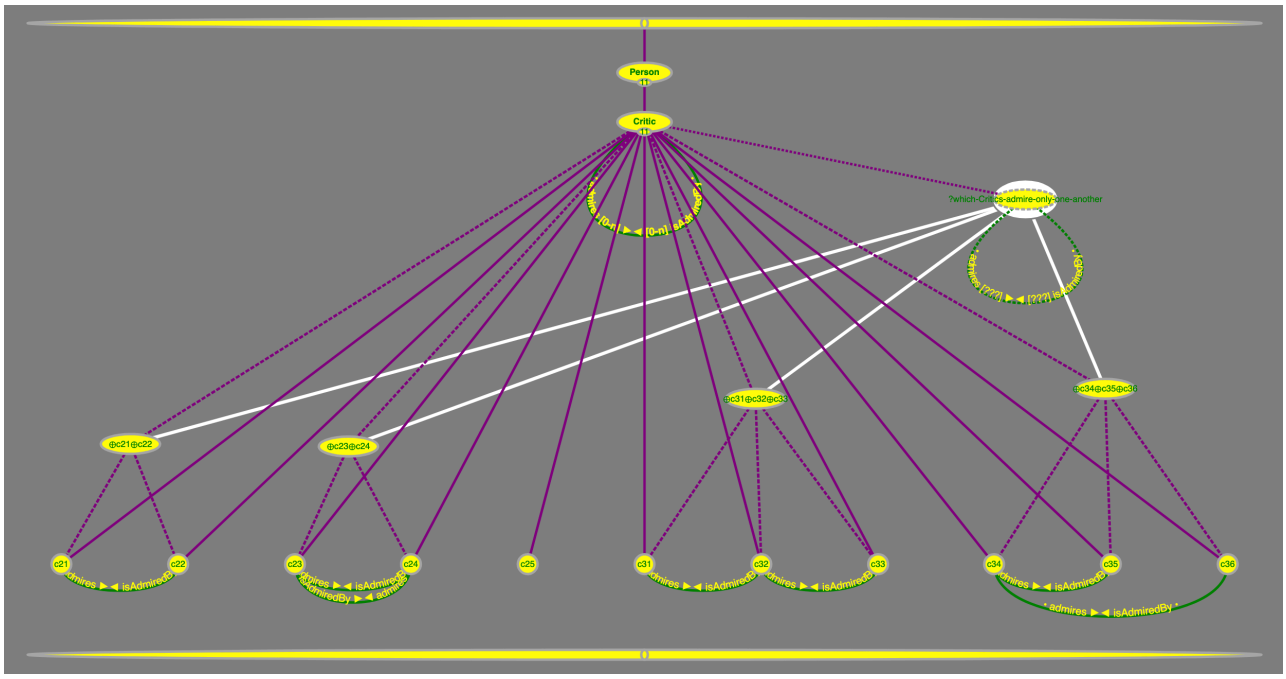
The sentence cannot be transformed into First Order Logic, but it can be transformed into Plural Logic.

According to [2021a] at point (2.27), the Plural Logic formula for the sentence is

$\exists xx\ (\forall x(x < xx \rightarrow C(x)) \wedge \forall x \forall y[(x<xx \wedge A(x,y)) \rightarrow (y<xx \wedge x \neq y)])$

meaning that the evaluated answer xx (a plural variable) shall include all the x that are critics (C(x)) and that admires (A(x,y)) a y that is also in the result and is not x.

Formulated as a query, the sentence is: "Which critics admire only one another?".



The Geach-Kaplan sentence as a CRLL model in CRLLP

The Geach-Kaplan sentence as a CRLL model in CRLLP assumes that the query concept labelled

?which-Critics-admire-only-one-another

can be evaluated as the transitive closure of the relationship • admires► ◄admiredBy •. It is open for discussion, whether this model is a correct or satisfactory interpretation/regimentation of the sentence.

The plural concept Critic is a subconcept of the plural concept Person, and it has potential relationship

• admires(0-n)► ◄admiredBy(0-n) •

to itself.

The unit concepts are examples with actual relationships • admires► ◄admiredBy •

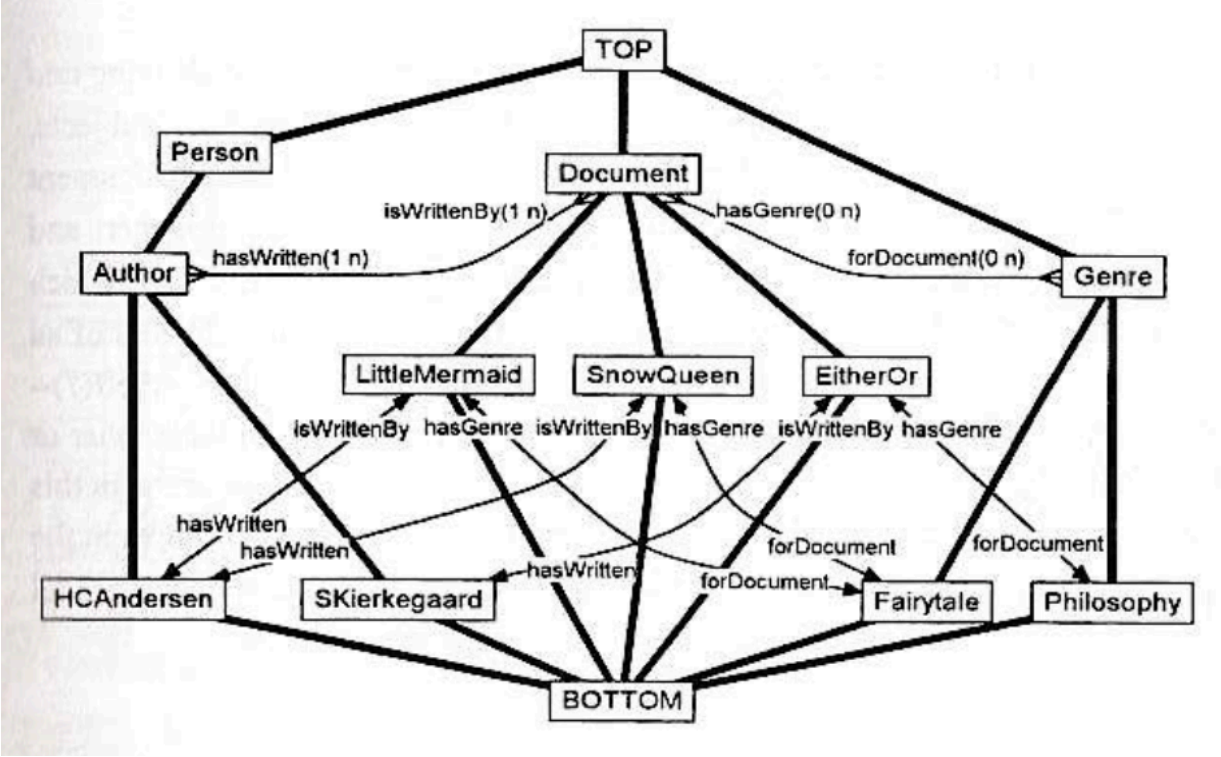
The answer evaluated as the transitive closure of the relationship is shown by the white lines from ?which-Critics-admire-only-one-another to the plural sums of Critic units that satisfy it: ⊕c21⊕c22, ⊕c23⊕c24, ⊕c31⊕c32⊕c33, and ⊕c34⊕c35⊕c36 ('⊕' is used in CRLLP for lattice sum).

Thus, the answer is a plural of plurals, also called a superplural [2008b].

You may experiment with the model as a registered user of CRLLP.

More RLL Examples

RLL figure 10 - RLL model of the Author/Document/Genre universe



The interpretation of the example is that

- the plural concept labelled Author represents authors in the modelled universe,
- who have written one or more documents represented by the plural concept labelled Document, as given by the potential relationship labelled
  • hasWritten(1 n)► ◄isWrittenBy(1 n) •
- and that the unit concept labelled HCAndersen represents an author,
- who has written documents represented by the unit concepts labelled LittleMermaid and SnowQueen, as given by the actual relationship labelled • hasWritten► ◄isWrittenBy •
- Author has the extension
  ⊕Author⊕HCAndersen⊕SKierkegaard
- and the intension
  Author • hasWritten(1 n) ► Document.

- HCAndersen has the extension
  ⊕HCAndersen

- and the intension
  Author • hasWritten(1 n) ► Document,
  HCAndersen • hasWritten ► LittleMermaid,
  HCAndersen • hasWritten ► SnowQueen.

RLL figure 12 - Evaluating a query concept

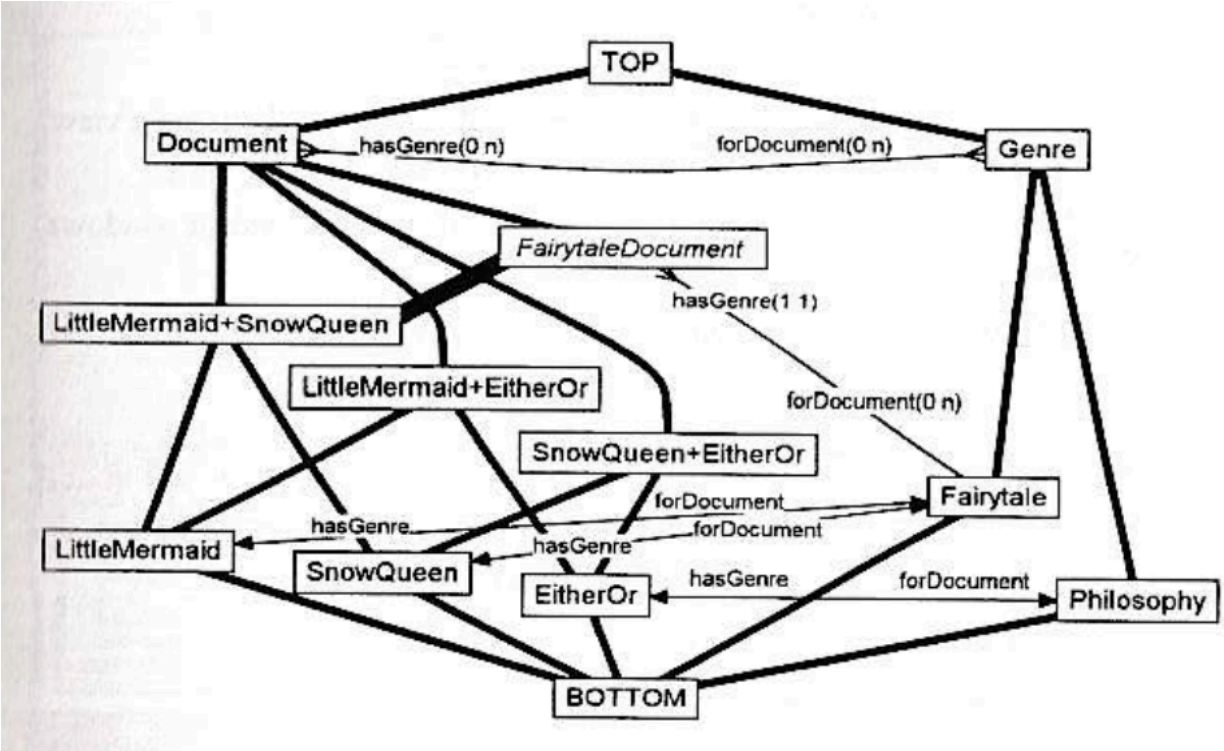


Figure 12 shows a Boolean sublattice implicitly constructed over three documents, LittleMermaid, SnowQueen, and EitherOr, showing three plurals, unit sums, of two documents.

We have a classification of documents according to their Genre, induced by the relationship between Document and Genre.

Then we may talk about the documents, which has the Genre of Fairytale labelled *FairytaleDocument*.

The query concept *FairytaleDocument* is a subconcept of Document, its extension is some of the documents, and its intension is the intension inherited from Document with the addition of Fairytale as Genre.

When evaluated, *FairytaleDocument* evaluates to one of the plurals, LittleMermaid+SnowQueen.

RLL figure 25 - Two Boolean sublattices

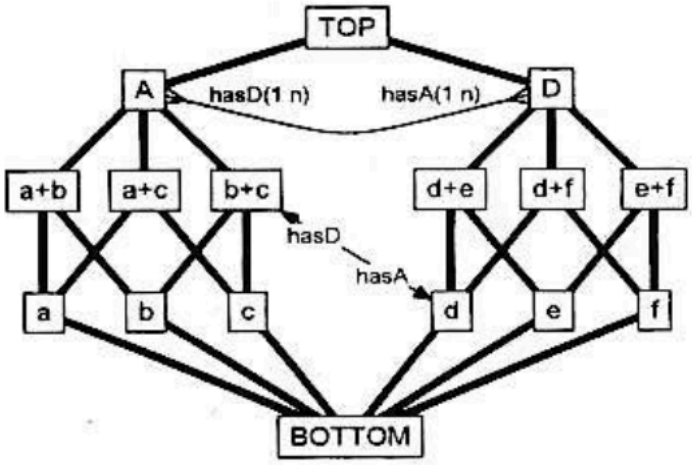


Figure 25 shows two Boolean sublattices generated by a potential relationship between A and D, whereby Boolean sublattices are constructed implicitly over their unit subconcepts.

If A represents authors and D documents, then the actual relationship

b+c • hasD ►◄ hasA • d

means that the author b togetherWith the author c have written the document d.

In a symmetric example one author could have written two documents, e.g.:

a • hasD ►◄ hasA • e
a • hasD ►◄ hasA • f

There is an important semantic difference in these two examples of the sum of two concepts involved in some relationship together.

In the first example there is only one event of writing behind the relationship, it is called collective for b+c to d, in the second example there are two writing events, it is called distributive for a to e and f.

In order to model this semantic difference, we use time and events as in figure 40.

RLL figure 40 - An event with absolute time

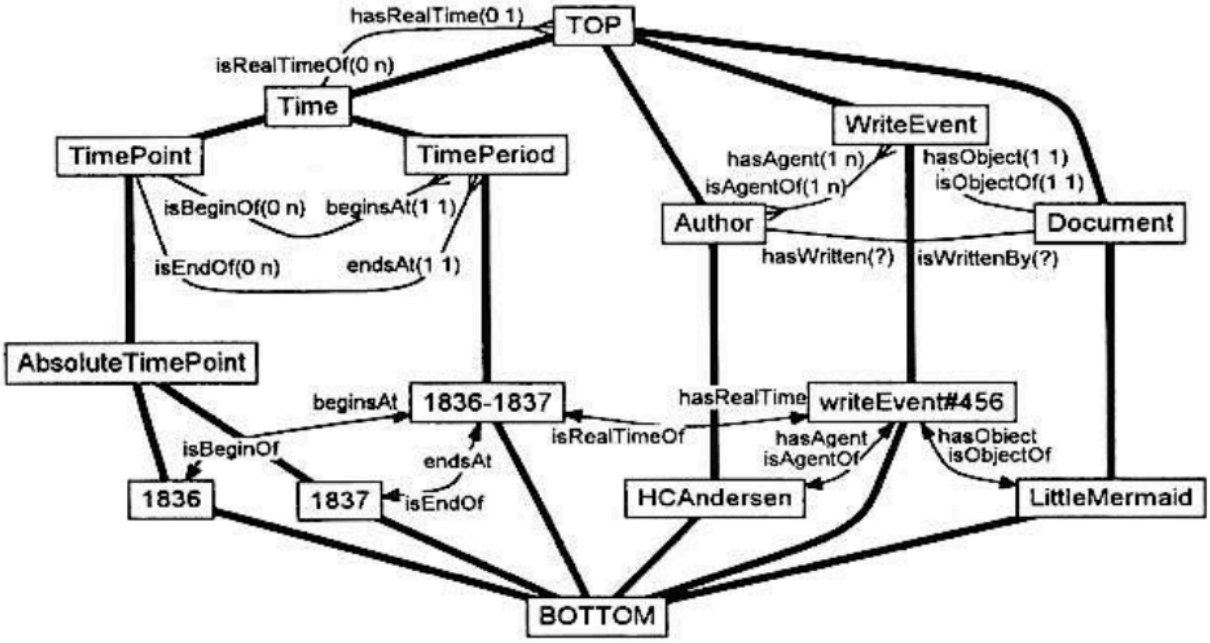


Figure 40 models “H.C. Andersen wrote The Little Mermaid in 1836-1837”.

Events are concepts of state transitions. They may be sequenced in time, telling a story.

Figure 40 states that there was a writing event in the period 1836-1837, where HCAndersen was the agent, and LittleMermaid was the object.

The figure also shows an example of derivation, namely of the relationship
• hasWritten(?) ► ◄ isWrittenBy(?) •,
saying that an author has written a document, when there is a writing event with that author as agent and that document as object.

RLL figure 41 - Events and relative time

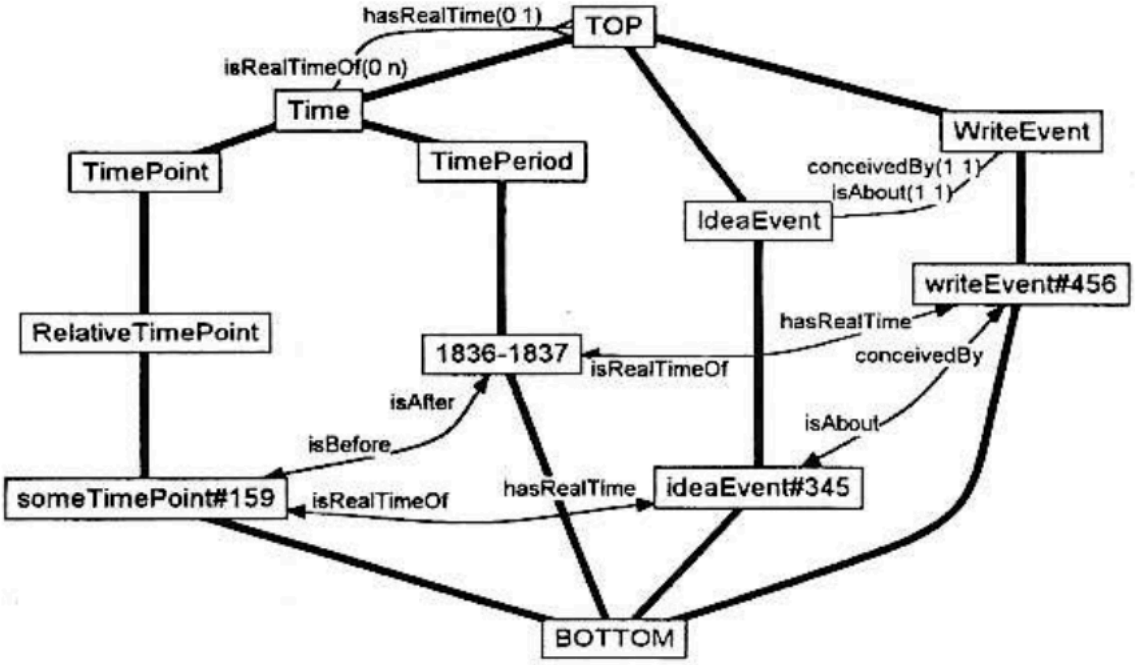
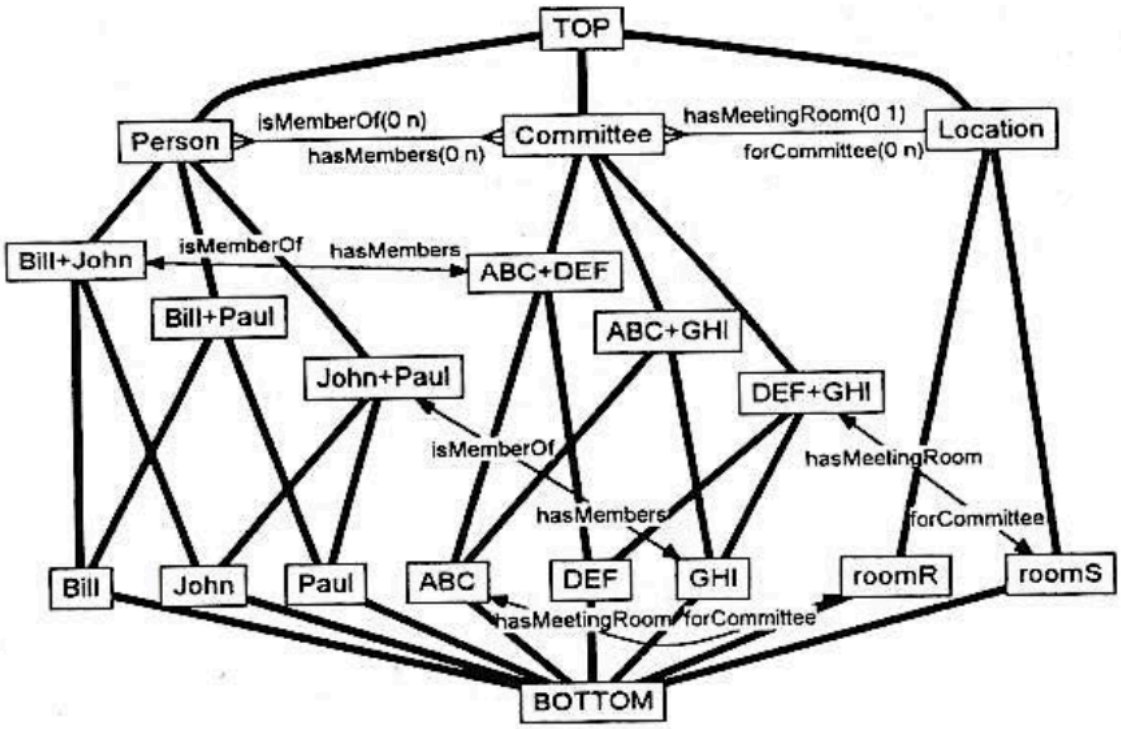


Figure 41 models “The idea came some time before 1836-1837”.

Figure 41 shows two events related by relative time.

The example states that there was an idea event with a relative real time point, which was before the real time of the writing event.

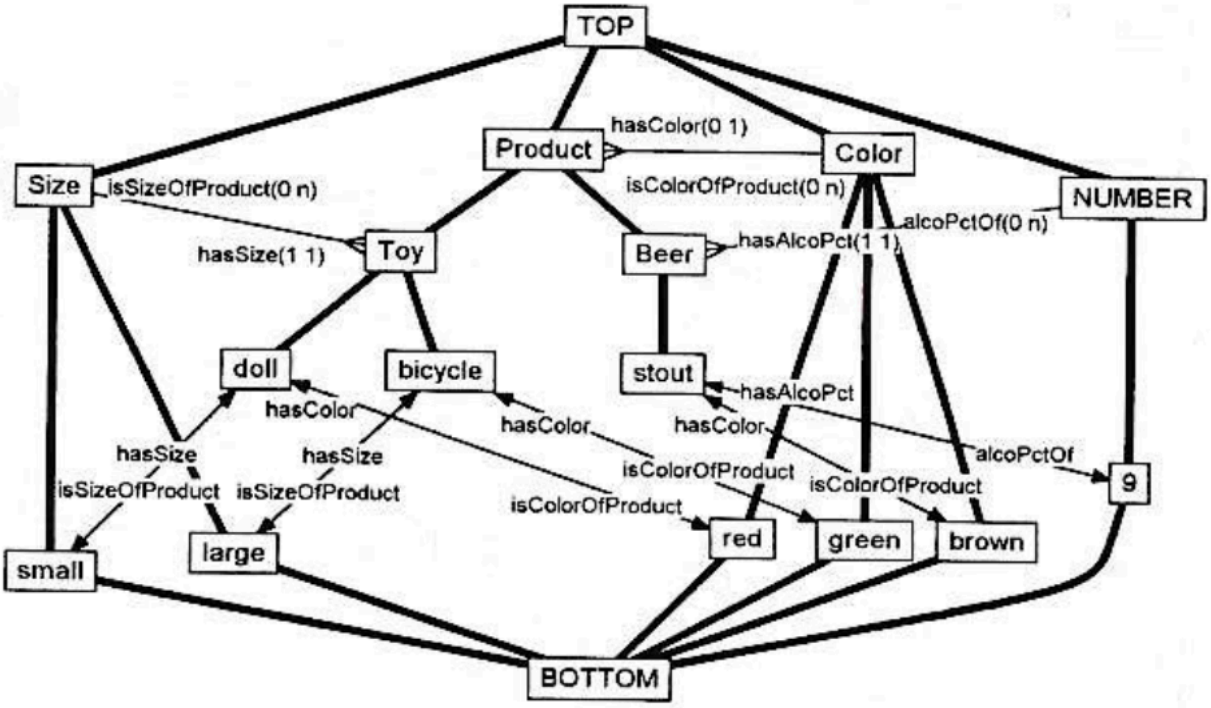
RLL figure 43 - Person, Committee, and Location



Committees may have many Persons as members, and Persons may be members of many Committees.

Committee and Person have their specific properties, so their intensions are different, therefore they cannot conceptually be part-of each other, as opposed to physical mereology, where persons can be part-of committees.

RLL figure 46 - Products with various properties



Model of products with various properties.

Dolls and bicycles are toy products that can have size and color.

Stout is a beer product that can have color and alcopct.

RLL figures 76-77 - Mass concepts Water and Gold

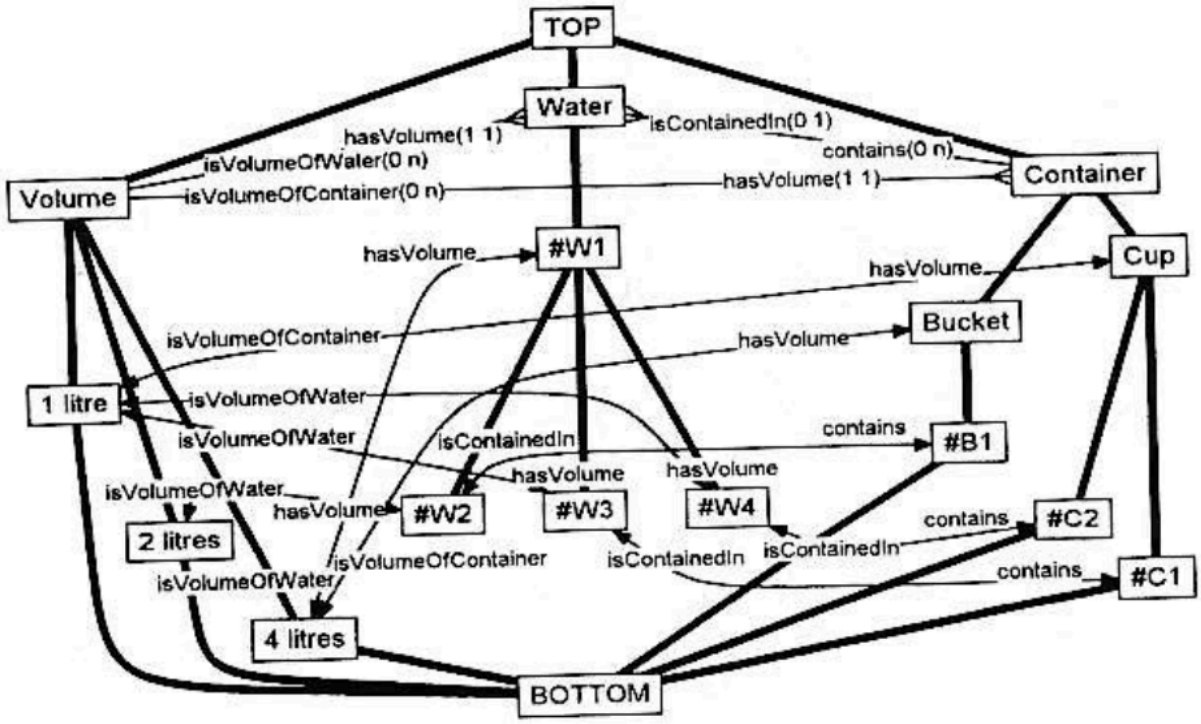


Figure 76 models Water in containers with volumes.
Water may be contained in containers, and both water parts and containers may be qualified by a volume. All buckets are containers holding 4 liters, all cups hold 1 liter. The bucket #B1 contains actually a mass of water, which is 2 liters in volume, and the two cups each contain 1 liter of water.

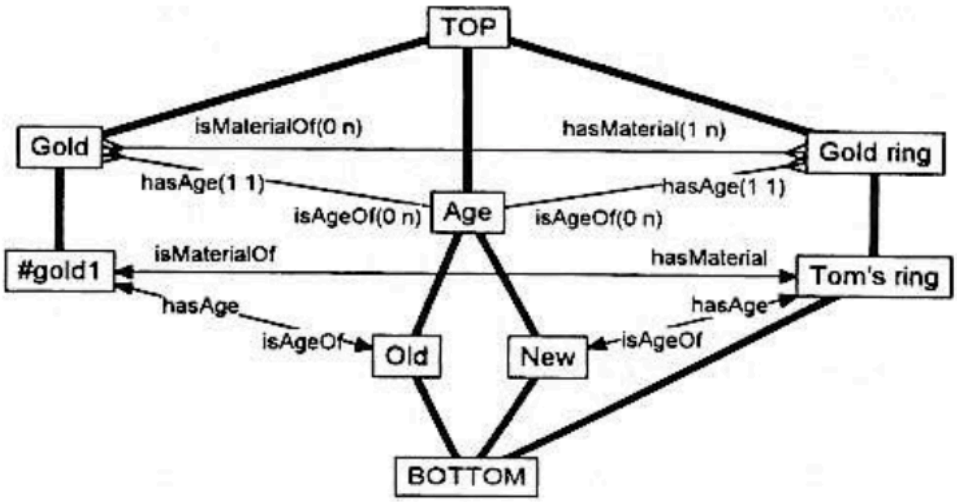


Figure 77 models "The gold in Tom's ring is old, but the ring is new"

This model distinguishes between the ring itself and the mass of gold and their different properties.

The CRL Logic Prototype (CRLLP)

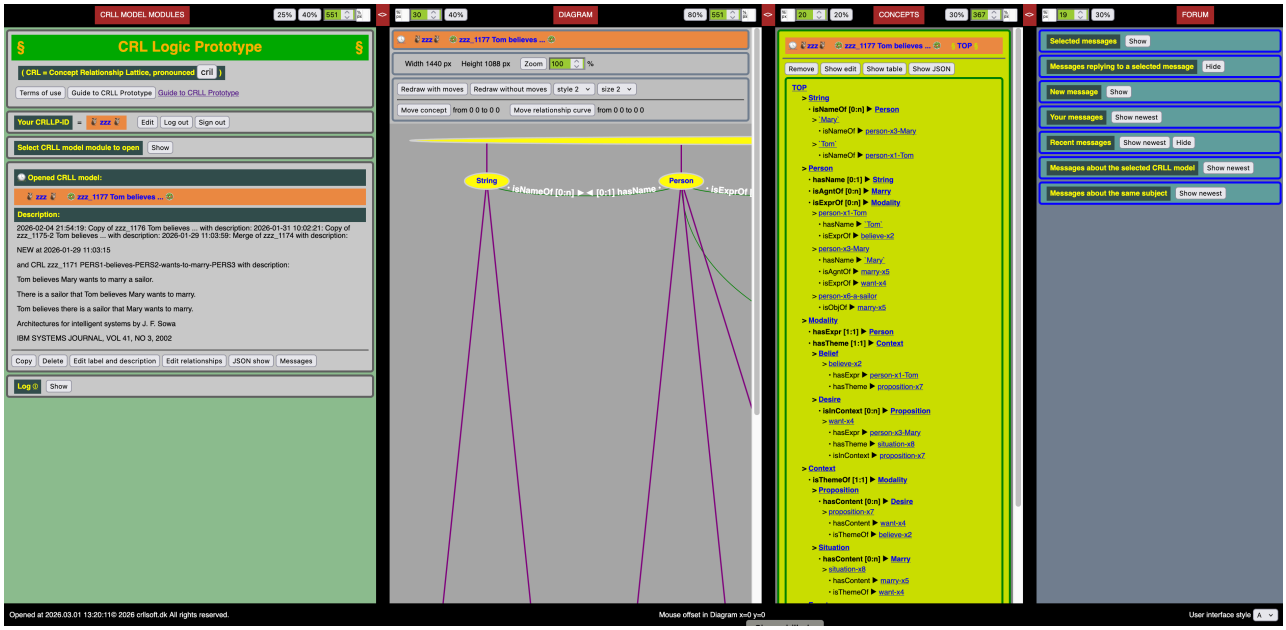
The CRL Logic Prototype (CRLLP) enables registered users to experiment with CRL Logic.

[You are invited to register as a user of the CRL Logic Prototype here.](#) We appreciate your feedback, advice, and insight in the discussion forum.

See a brief guide to CRLLP here.

We hope that scientists and students will be inspired to develop systematic presentations of CRL Logic and its uses.

The screenshot shows the selected CRLL model module in the graphical form inspired by Hasse diagrams and entity-relationship diagrams.



Screenshot of the CRL Logic Prototype

The selected CRL model module is also displayed in a table format and in a tree-structured text format, showing the intension and extension of concepts.

Users may cooperate, copying from each other and merging CRLL model modules to give more comprehensive models.

The implementation has a web interface to the core logic module that opens for application dependent user interfaces.

Investors with teams of relevant experts are invited to join the CRL Logic software community, where the CRL Logic Prototype will develop into production-ready CRL Logic Management Systems, CRLLMSs. We encourage innovation, cooperation and competition.

CRLLMSs will act like Relational DBMSs and Knowlege Base Management Systems, KBMSs. There is potential to take over the DBMS and KBMS markets.

Implementations may lead to other product types, e.g. CRL Logic spreadsheets, where users may set up connected sheets ("crllsheets") with interface to CRLL modules in interoperating CRLLMSs with clickable query concepts for dynamic evaluation of queries. A CRL Logic software product will have CRL Logic in a backend core module and will have frontend interaction modules for users and web services.

Qualified investor teams with certified legal identity may acquire a licence to the source code of the CRLL Prototype for their own further development and for reimplementations on other platforms. The teams will report regularly on their plans and progress.

Licenses will be free of charge for a limited period of time. For more information on licenses, please write secure email to info@crllsoft.dk.

The implementation of CRLLMSs shall be evaluated by test sets, so that all implementations may be evaluated by the same criteria:

- Correctness of CRL Logic operations is mandatory for an implementation.
- Completeness means that an implementation can carry out all of CRL Logic.
- Usability is to be demonstrated. We think that CRL Logic is potentially easier and more powerful than current DBMS, KBMS, logic programming, first order logic, web semantic languages, triple stores and triple query languages, etc.
- Efficiency is a goal for each implementation.

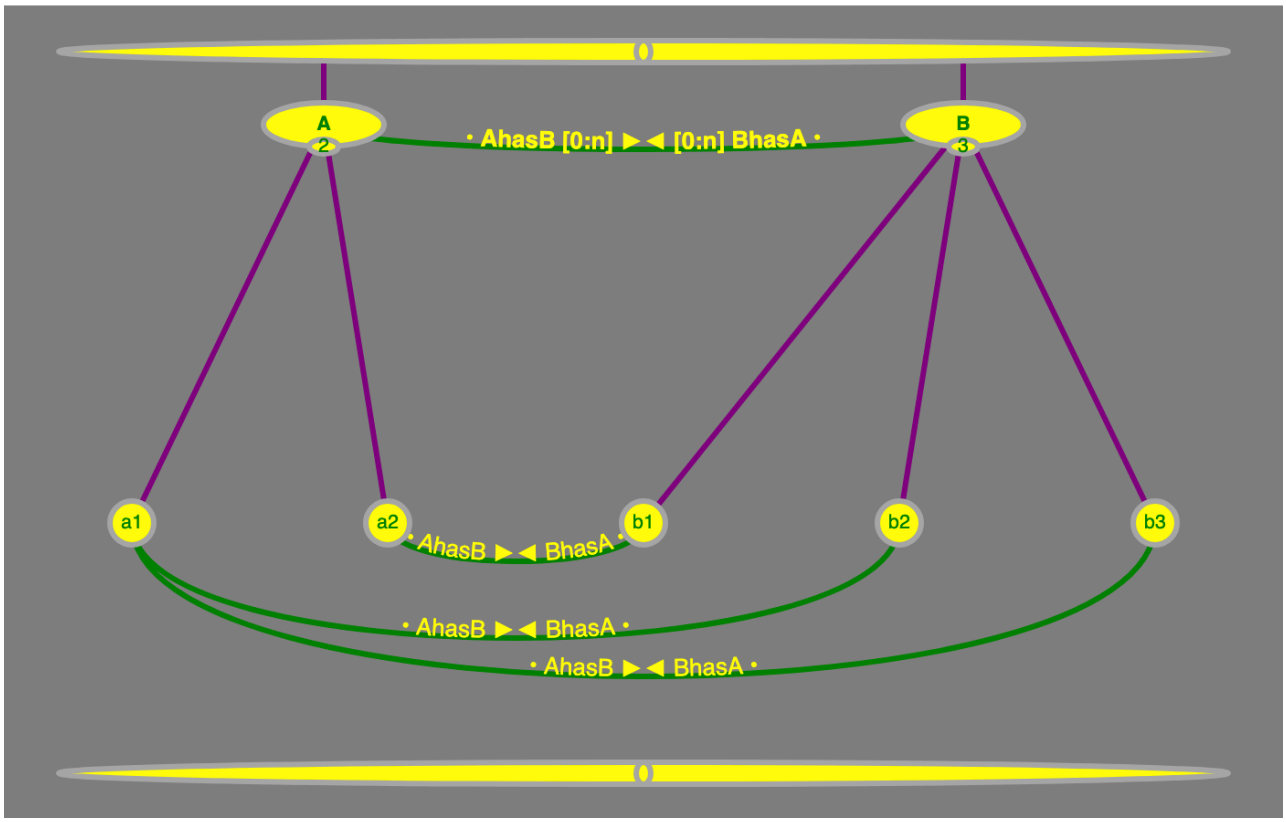
Example CRLL model modules in CRLLP

Some decisions taken for the implementation of CRLLP:

- Concepts have unique labels within the CRLL model:
    - Plural concept labels begin with A-Z (e.g. 'Person')
    - Query concept labels begin with '?' (e.g. '?AuthorsOfClassics')
    - Unit concept labels begin with a-z (e.g. 'hcandersen')
- Relationships have two property labels, unique within the CRLL model, each begin with A-Z or a-z

Abstract model modules

Example CRLLP--1 Two related plural concepts, 5 unit concepts.



A and B are plural concepts with cardinalities 2 and 3 and with the potential relationship

A • AhasB [0:n] ▶ ◀ [0:n] BhasA • B

a1, a2, b1, b2, and b3 are unit concepts with actual relationships

a1 • AhasB ▶ ◀ BhasA • b2
a1 • AhasB ▶ ◀ BhasA • b3
a2 • AhasB ▶ ◀ BhasA • b1

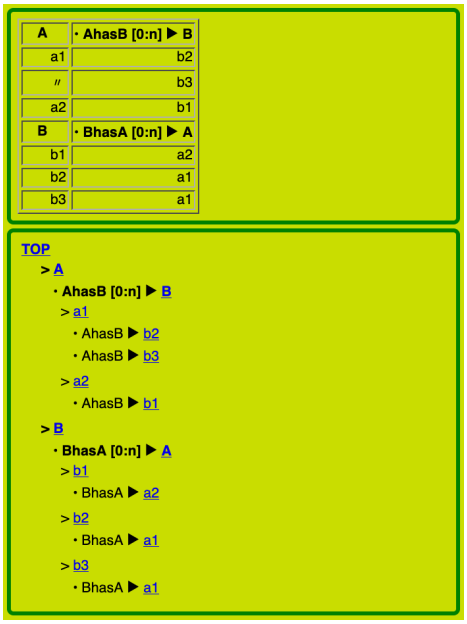
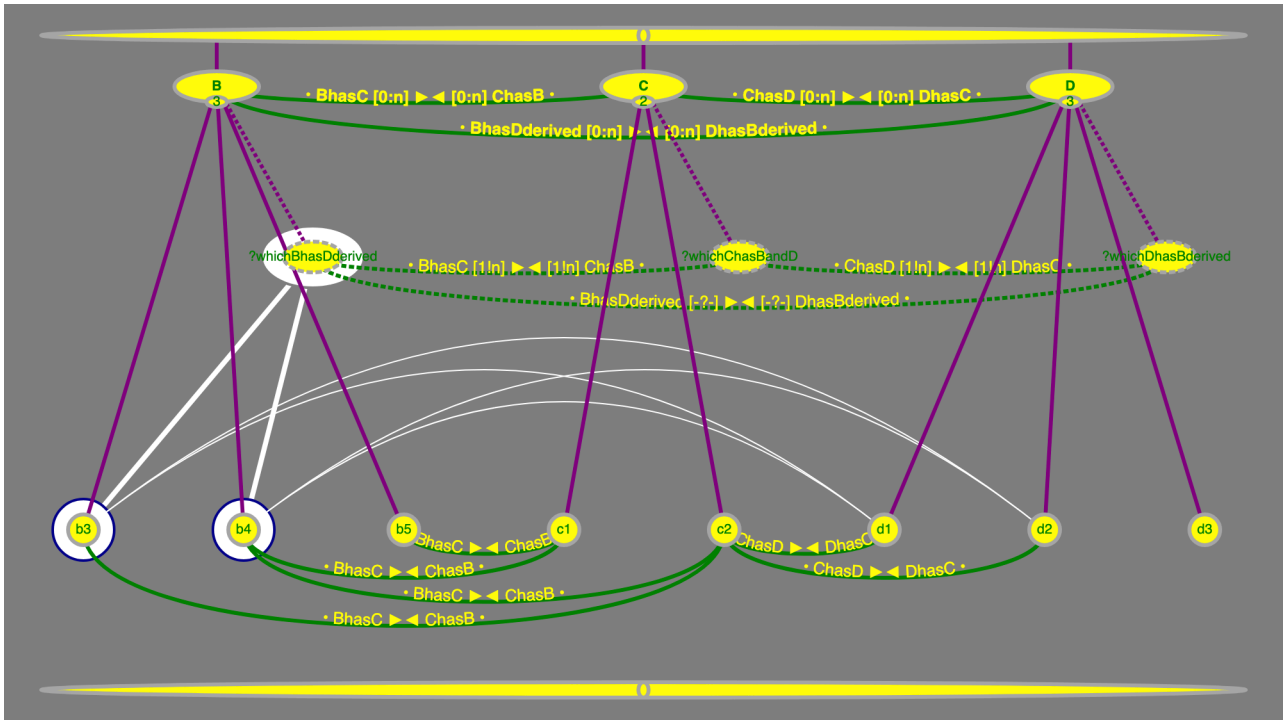


Table display and structured-list display

Example CRLLP--2 Three related plural concepts and a deriving relationship

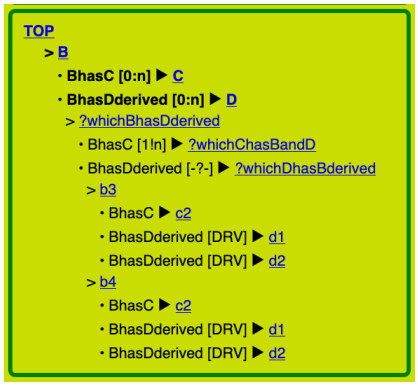


A deriving relationship is like a rule

B(x,y) and C(y,z) implies D(x,z)

B, C, and D are **plural concepts**
with cardinalities 3, 2 and 3
and with the potential relationships

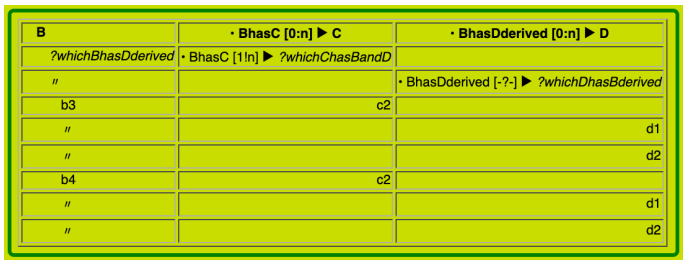
B • BhasC [0:n] ▶ ◀ [0:n] ChasB • C
C • ChasD [0:n] ▶ ◀ [0:n] DhasC • D
B • BhasDderived [0:n] ▶ ◀ [0:n] DhasBderived • D



Structured-list display

?whichBhasDderived, ?whichChasBandD, and ?whichDhasBderived
are **query concepts**
with the subquery relationships

B • BhasC [1?n] ▶ ◀ [1?n] ChasB • C
C • ChasD [1?n] ▶ ◀ [1?n] DhasC • D



B	• BhasC [0:n] ▶ C	• BhasDderived [0:n] ▶ D
?whichBhasDderived	• BhasC [1!n] ▶ ?whichChasBandD	
〃		• BhasDderived [-?-] ▶ ?whichDhasBderived
b3	c2	
〃		d1
〃		d2
b4	c2	
〃		d1
〃		d2

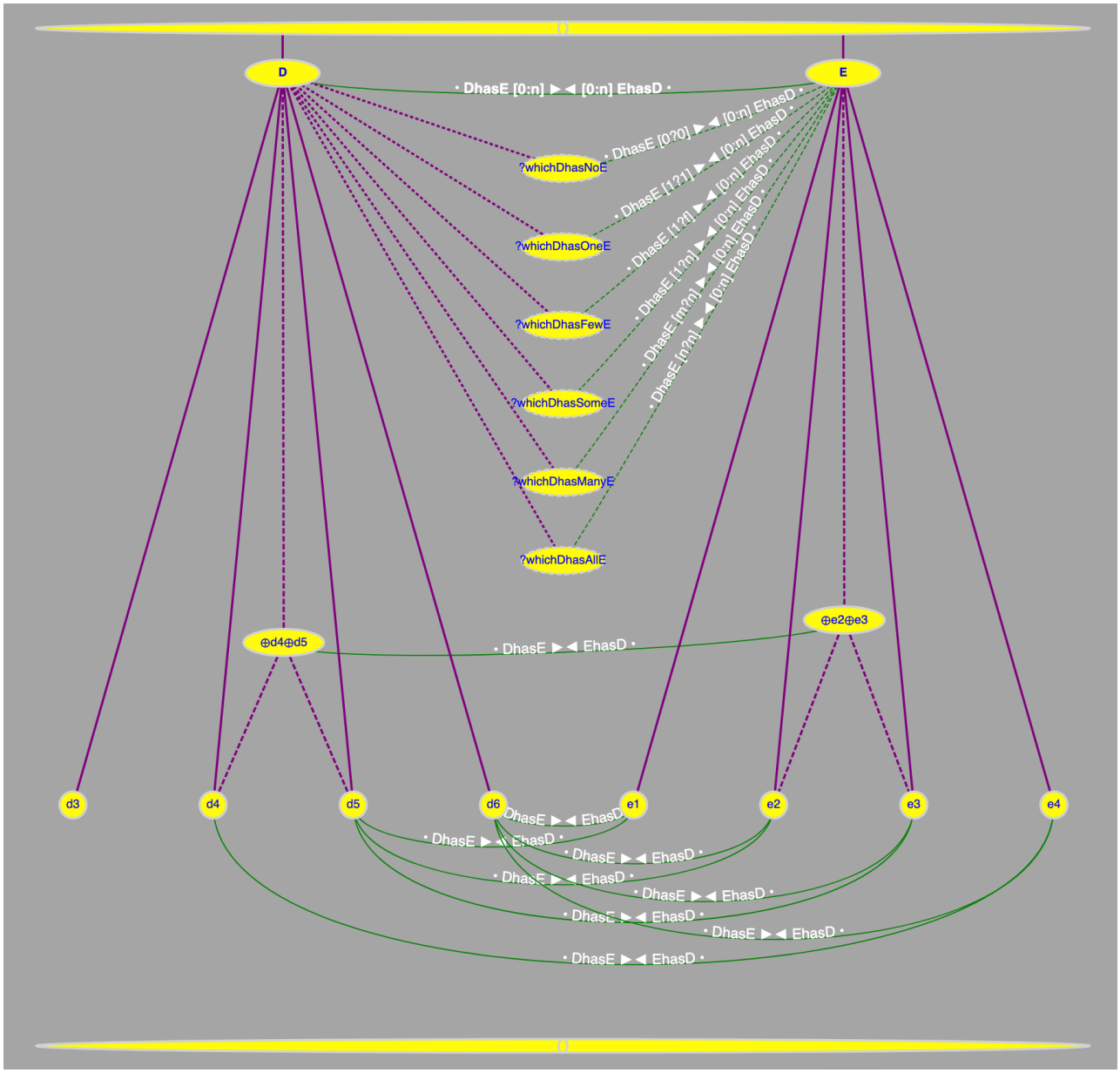
Table display

and with the deriving relationship

B • BhasDderived [-?-] ▶ ◀ [-?-] DhasBderived • D

When ?whichBhasDderived is evaluated, the diagram shows the result in white, while the structured list and the table show the result as derived properties of the unit concepts.

Example CRLLP--3 Two related plural concepts and quantifiers



This model has 6 query concepts defined by relationships with various modifiers:

- ?whichDhasNoE where the modifier is 0?0 so the qualifying D units have no E units related. This is shown by the white line in the diagram and in the structured list.

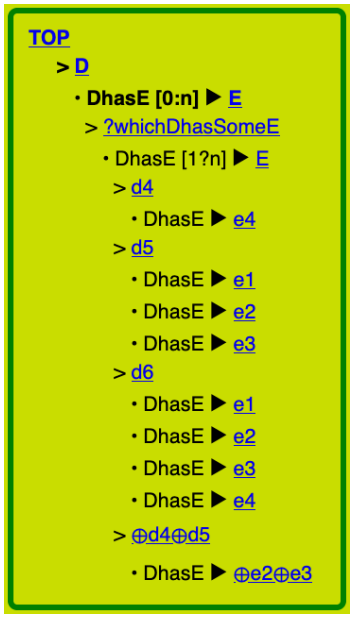
TOP
> D
• **DhasE [0:n] ▶ E**
> ?whichDhasNoE
• DhasE [0?0] ▶ E
> d3

- ?whichDhasOneE where the modifier is 1?1 so the qualifying D units have one E unit related. This is shown in the structured list.

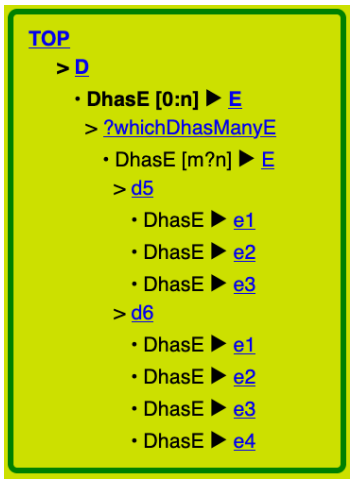
TOP
> D
• **DhasE [0:n] ▶ E**
> ?whichDhasOneE
• DhasE [1?1] ▶ E
> d4
• DhasE ▶ e4
> ⊕d4⊕d5
• DhasE ▶ ⊕e2⊕e3

- ?whichDhasFewE where the modifier is 1?f so the qualifying D units have one and up to 33% of E units related. This is shown in the structured list.

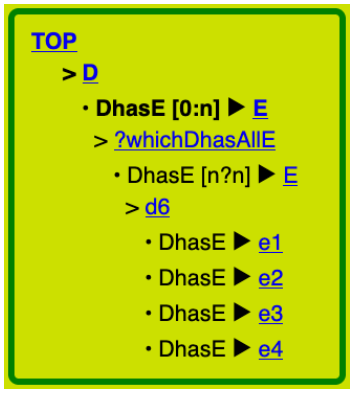
- ?whichDhasSomeE where the modifier is 1?n so the qualifying D units have at least one E unit related. This is shown in the structured list.



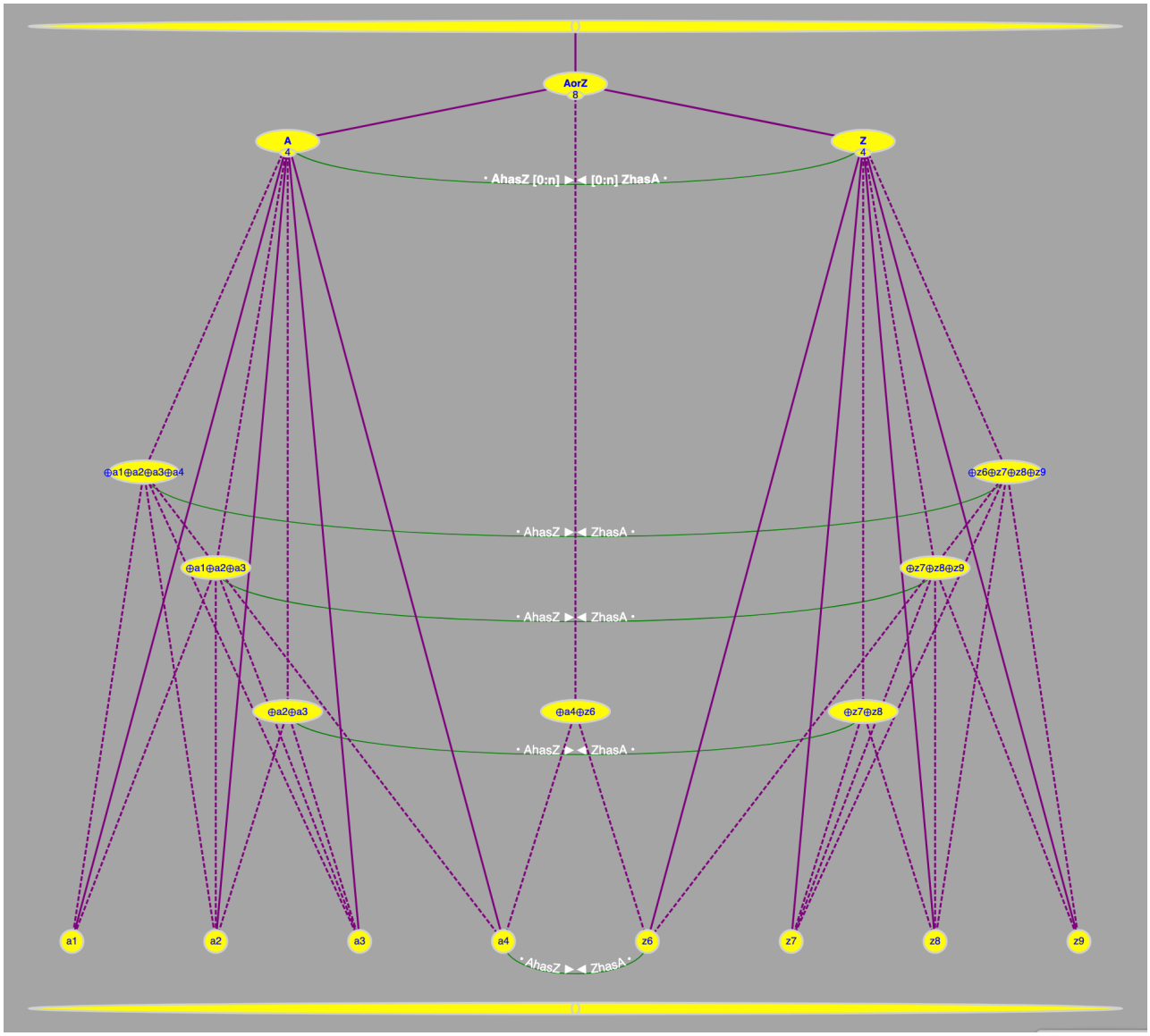
- ?whichDhasManyE where the modifier is m?n so the qualifying D units have at least 67% of E units related. This is shown in the structured list.



- ?whichDhasAllE where the modifier is n?n so the qualifying D units have all E units related. This is shown in the structured list.

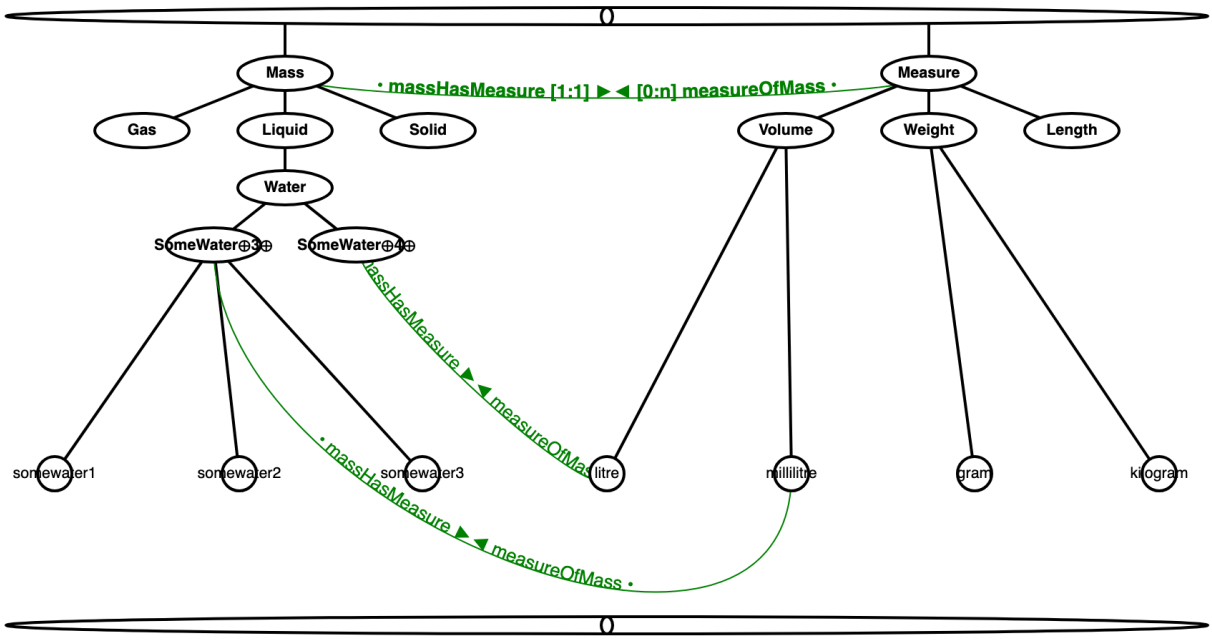


Example CRLLP--4 The Boolean Lattice



The full Boolean Lattice has all sums of unit concepts present. Here, some of the sums are defined under their respective superconcept. In particular, the sum ⊕a4⊕z6 of one subconcept of A and one subconcept of Z is under AorZ. If A were Trout and Z were Turtle, then AorZ would be TroutOrTurtle, which is a concept with very few properties, and only existing in mereological examples.

Example CRLLP--5 Mass concepts



We propose this model of mass concepts like Water.

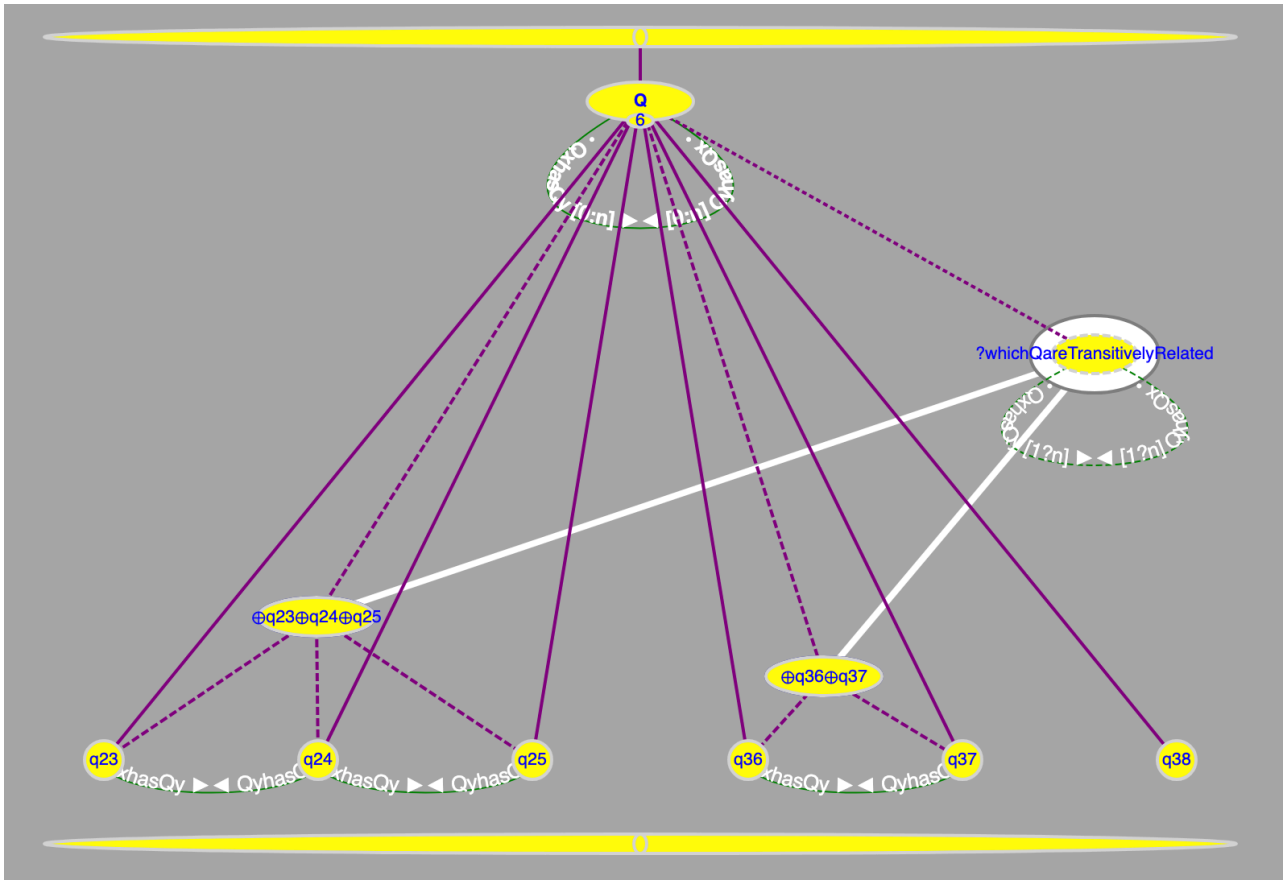
The subconcepts of Water labelled SomeWater⊕3⊕ and SomeWater⊕4⊕ are plurals with cardinalities 3 and 4, as indicated by '⊕3⊕' and '⊕4⊕'. The 3 and 4 unit subconcepts may or may not be explicitly defined.

SomeWater⊕3⊕ has the measure milliliter, and it has three units as subconcepts, explicitly, each with a cardinality of 1, that is, 1 milliliter each, 3 milliliters in all.

SomeWater⊕4⊕ has the measure liter, and it has four units as subconcepts, implicitly, each with a cardinality of 1, that is, 1 liter each, 4 liters in all.

It shall be a feature of implementations that all measures are transformable, e.g. liters to milliliters. So, SomeWater⊕4⊕ may be transformed to SomeWater⊕4000⊕ with milliliter as measure, when used with other water concepts. Thus, we may have models with mass concepts with measures of any granularity, as required for a purpose.

Example CRLLP--6 Which units are transitively related?



This is the abstract version of the model of the Geach-Kaplan sentence presented earlier.

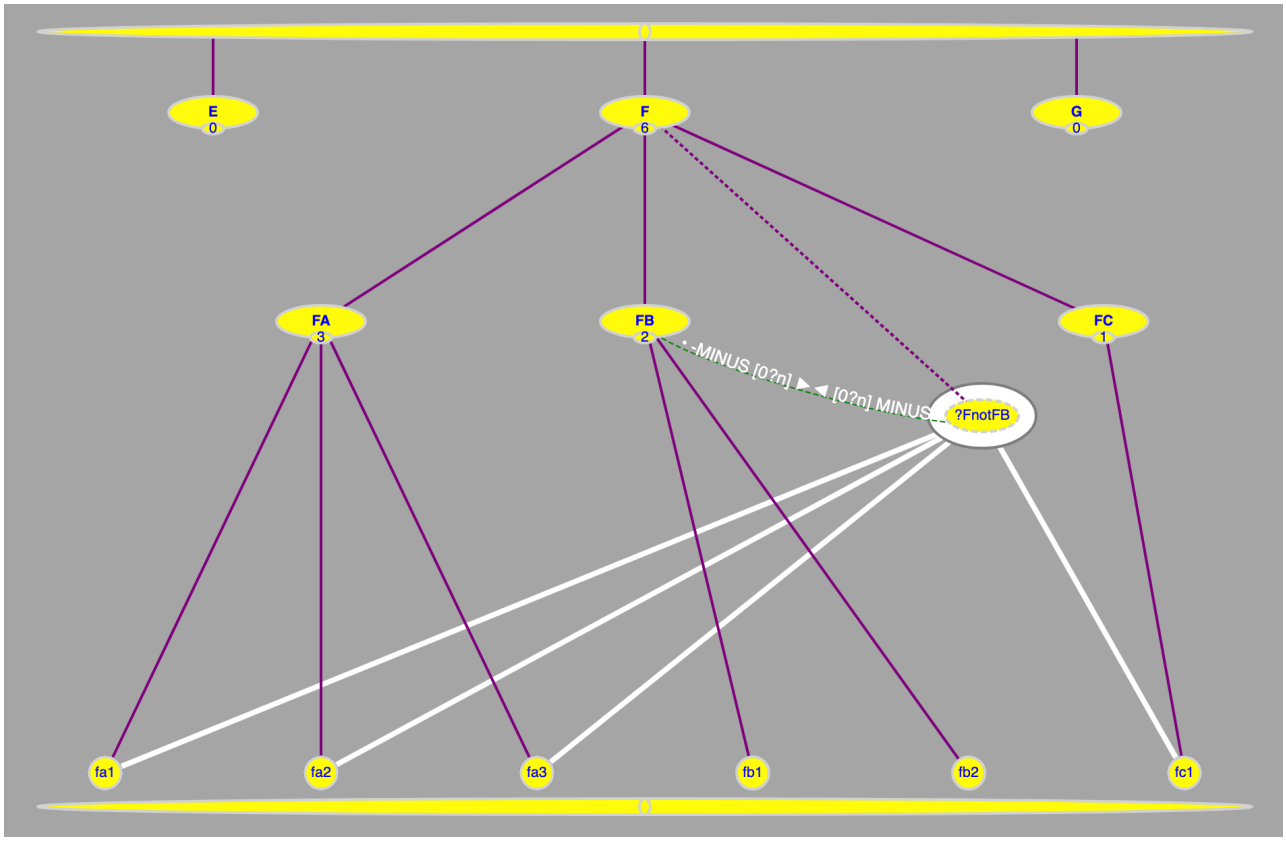
The query concept labelled

?whichQareTransitivelyRelated

is evaluated as the transitive closure of the relationship

• QxHasQy►◄QyHasQx •.

Example CRLLP--7 A superconcept F MINUS a subconcept FB of F

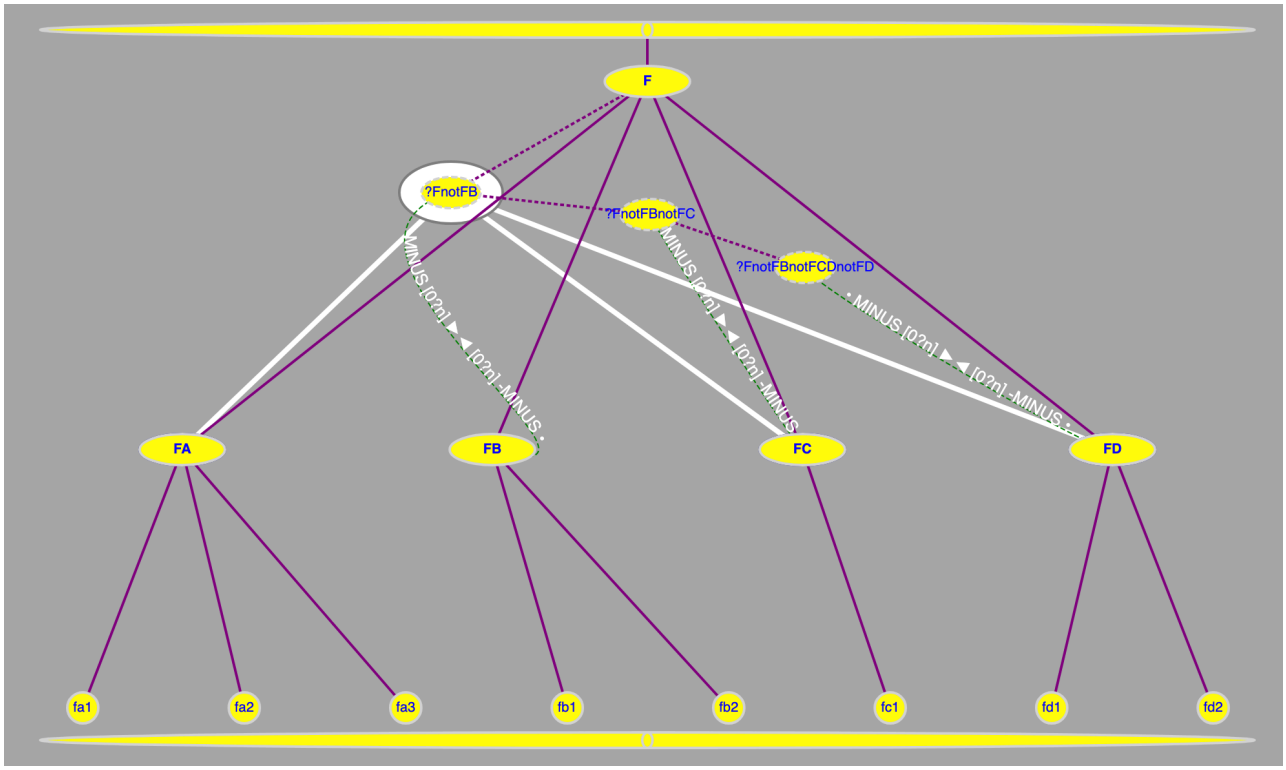


The special relationship MINUS is used to give the complement of units, so the query concept

?FnotFB

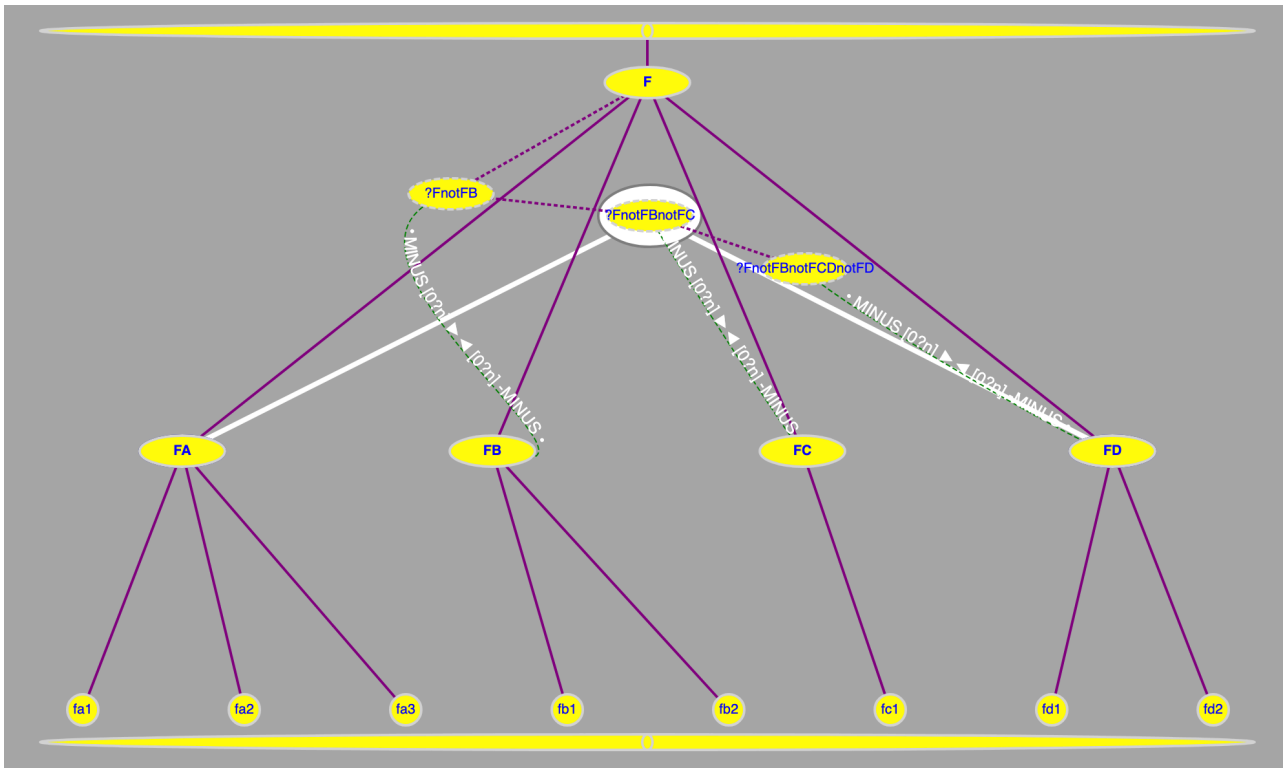
evaluates to the F units, which are not FB units.

Example CRLLP--8 A chain of queries with MINUS

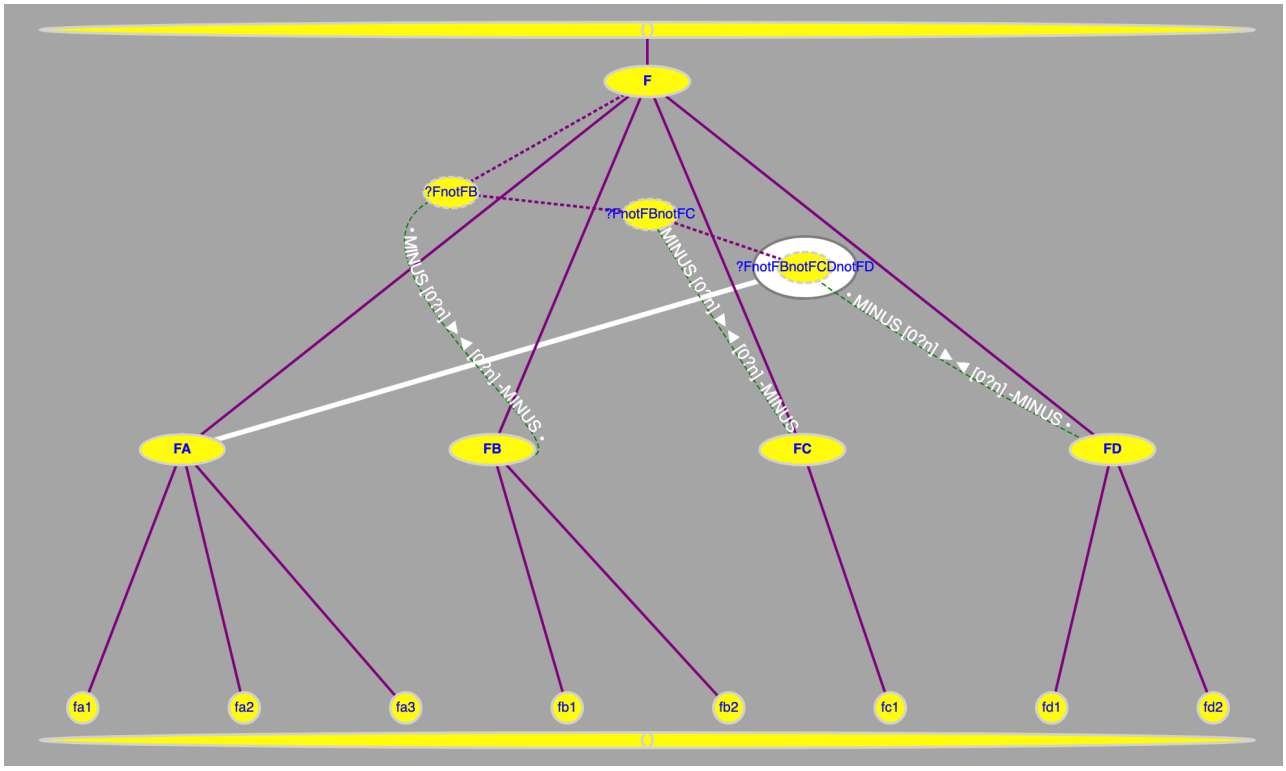


The special relationship MINUS is used to give the extension of the superconcept minus the extension of a subconcept. This model has a chain of query concepts, each restricting F by more MINUS relationships of the subconcepts of F.

?FnotFB restricts F to not FB.

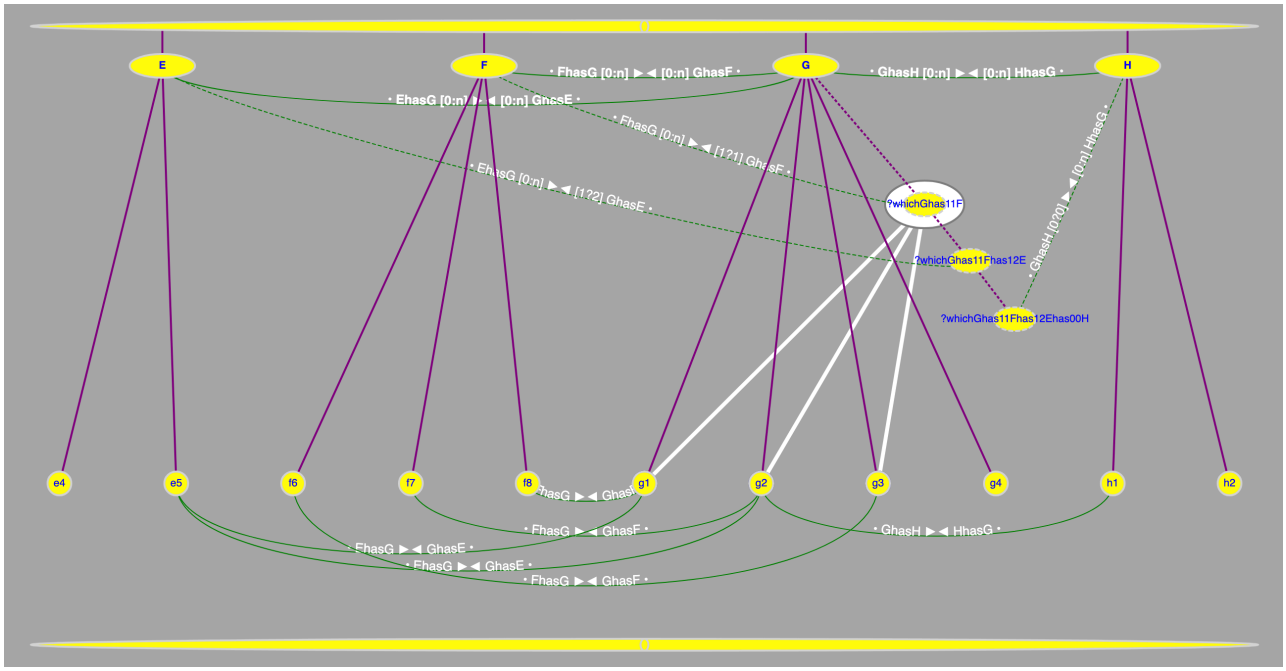


?FnotFBnotFC further restricts F to not FC.



?FnotFBnotFCnotFD further restricts F to not FD.

Example CRLLP--9 A chain of queries with various quantifiers



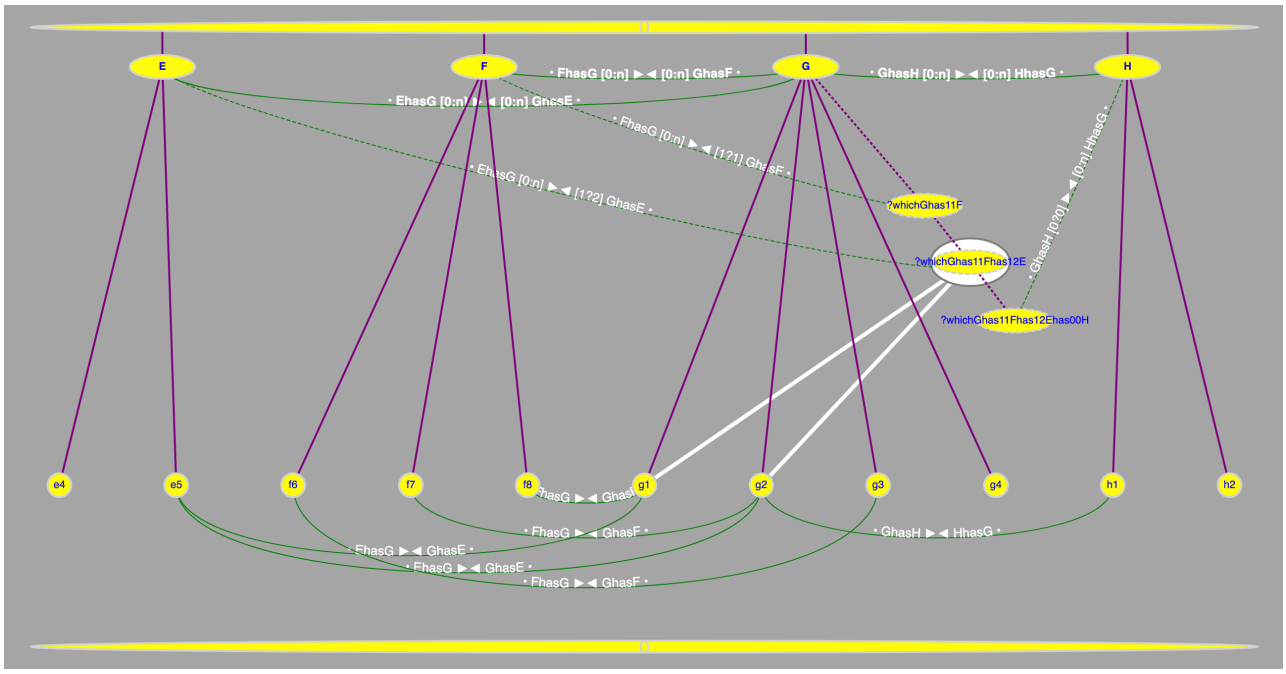
This model has a chain of query concepts, each restricting the G units by more relationships.

?whichGhas11F restricts the G units to those with one F unit related.

Notice that GhasF(1?1) is like EITHER-OR, e.g. for g2

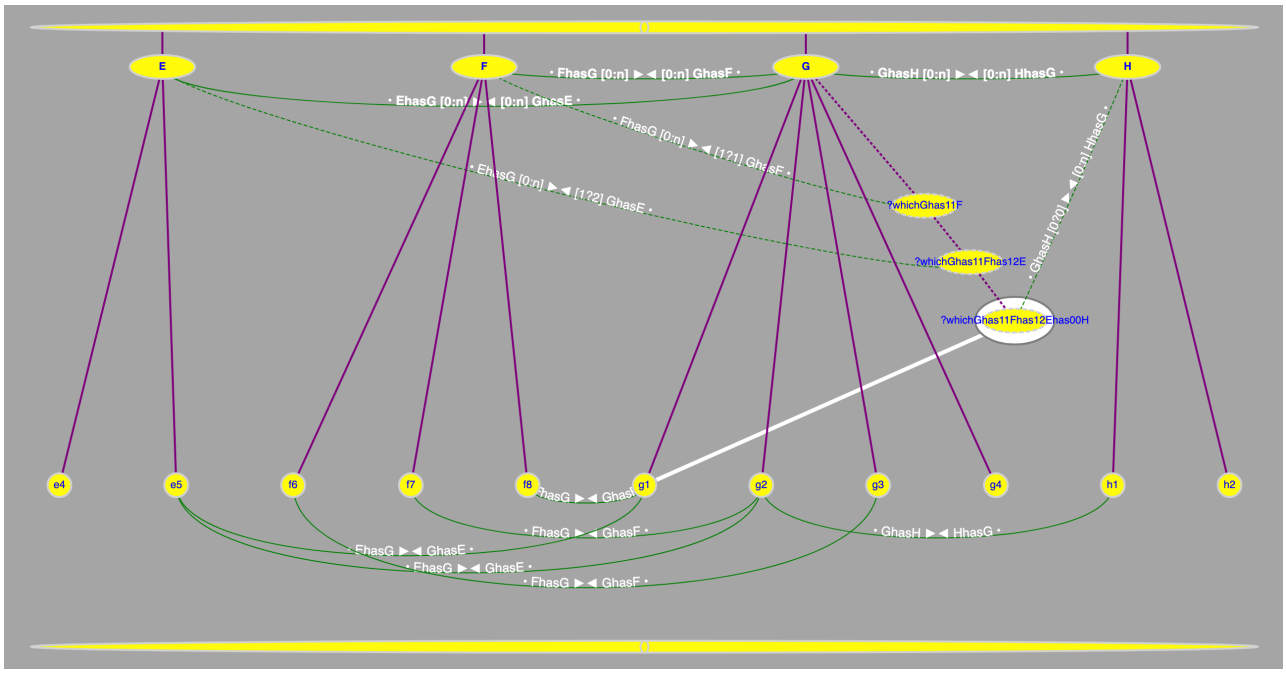
EITHER g2•GhasF▶f6 (NO) OR g2•GhasF▶f7 (YES) OR g2•GhasF▶f8 (NO).

TOP
> G
• GhasE [0:n] ▶ E
• GhasF [0:n] ▶ F
• GhasH [0:n] ▶ H
> ?whichGhas11F
• GhasF [1?1] ▶ F
> g1
• GhasF ▶ f8
> g2
• GhasF ▶ f7
> g3
• GhasF ▶ f6



?whichGhas11Fhas12E further restricts the G units to those with one or two E units related.

TOP
> **G**
• **GhasE [0:n] ▶ E**
• **GhasF [0:n] ▶ F**
• **GhasH [0:n] ▶ H**
> ?whichGhas11F
• GhasF [1?1] ▶ F
> ?whichGhas11Fhas12E
• GhasE [1?2] ▶ E
> g1
• GhasE ▶ e5
• GhasF ▶ f8
> g2
• GhasE ▶ e5
• GhasF ▶ f7

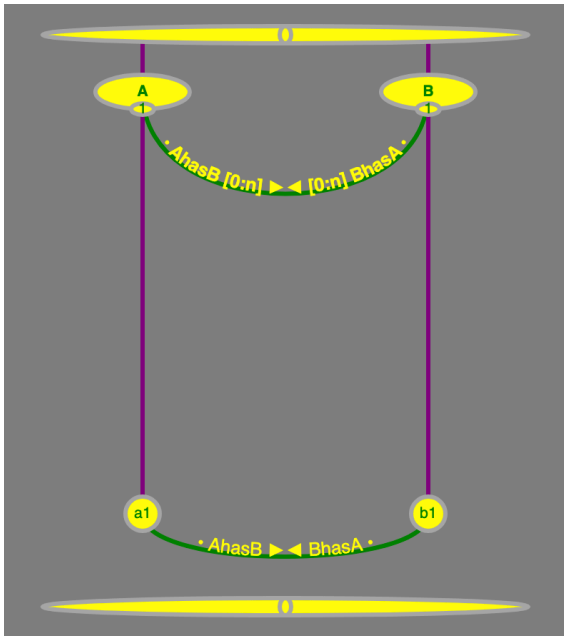


?whichGhas11Fhas12Ehas00H further restricts the G units to those without H units related.

TOP
- > **G**
  - • **GhasE [0:n]** ▶ **E**
  - • **GhasF [0:n]** ▶ **F**
  - • **GhasH [0:n]** ▶ **H**
    - > ?whichGhas11F
      - • GhasF [1?1] ▶ F
        - > ?whichGhas11Fhas12E
          - • GhasE [1?2] ▶ E
            - > ?whichGhas11Fhas12Ehas00H
              - • GhasH [0?0] ▶ H
            - > g1
              - • GhasE ▶ e5
              - • GhasF ▶ f8

Notice that GhasF(1?1), GhasE(1?2), and GhasH(0?0) are ANDed, so the result is the same, whether the sequence is different, or they were attached to one and the same query concept.

Example CRLLP-11 Meta-level CRLL model of CRLL models, example AB



TOP
> A
• AhasB [0:n] ▶ B
> a1
• AhasB ▶ b1
> B
• BhasA [0:n] ▶ A
> b1
• BhasA ▶ a1

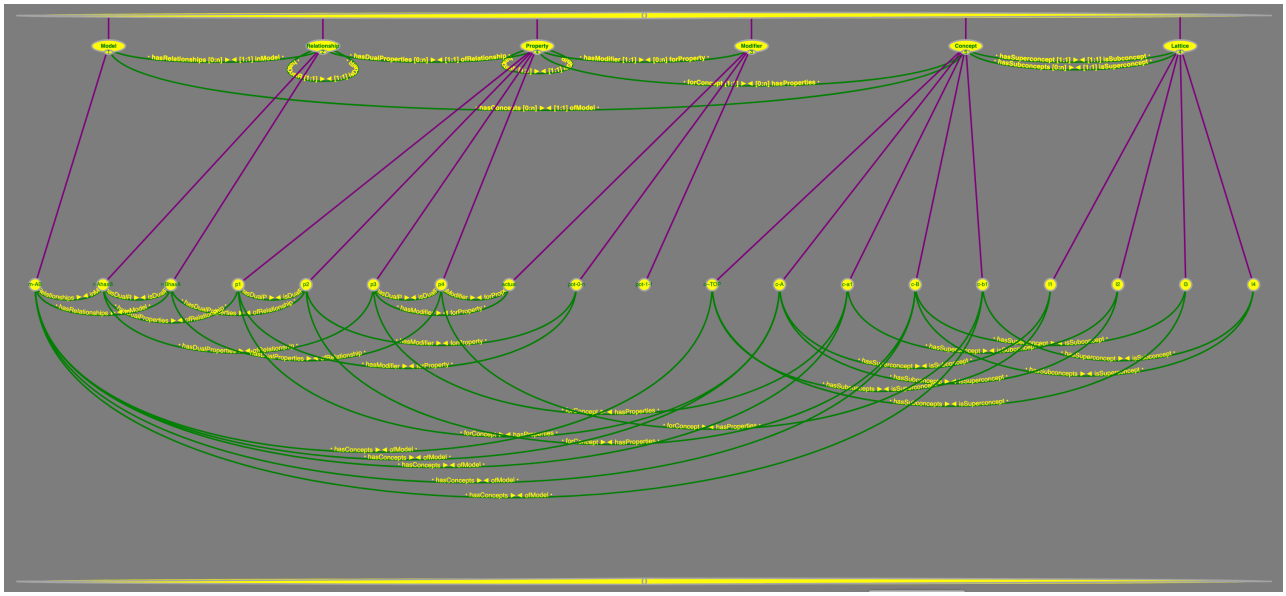
A	• AhasB [0:n] ▶ B
a1	b1
B	• BhasA [0:n] ▶ A
b1	a1

Example AB is the object of the example meta-level CRLL model.

Example AB has a diagram, a tree display, and a table display.

The meta-level CRLL model of example AB also has a diagram, a tree display, and a table display.

The meta-level concepts Model, Relationship, Property, Modifier, Concept, and Lattice may be queried by query concepts, that is, they may be quantified.



TOP
> Model
· hasConcepts [0:n] ▶ Concept
· hasRelationships [0:n] ▶ Relationship
> m-AB
· hasConcepts ▶ c--TOP
· hasConcepts ▶ c-A
· hasConcepts ▶ c-a1
· hasConcepts ▶ c-b1
· hasConcepts ▶ c-B
· hasRelationships ▶ r-AhasB
· hasRelationships ▶ r-BhasA

Model	· hasConcepts [0:n] ▶ Concept	· hasRelationships [0:n] ▶ Relationship
m-AB	c--TOP	
〃	c-A	
〃	c-a1	
〃	c-b1	
〃	c-B	
〃		r-AhasB
〃		r-BhasA

> Relationship
· hasDualProperties [0:n] ▶ Property
· hasDualR [1:1] ▶ Relationship
· inModel [1:1] ▶ Model
· isDualR [1:1] ▶ Relationship
> r-AhasB
· hasDualProperties ▶ p1
· hasDualProperties ▶ p3
· hasDualR ▶ r-BhasA
· inModel ▶ m-AB
> r-BhasA
· hasDualProperties ▶ p2
· hasDualProperties ▶ p4
· inModel ▶ m-AB
· isDualR ▶ r-AhasB

Relationship	· hasDualProperties [0:n] ▶ Property	· hasDualR [1:1] ▶ Relationship	· inModel [1:1] ▶ Model	· isDualR [1:1] ▶ Relationship
r-AhasB	p1			
〃	p3			
〃		r-BhasA		
〃			m-AB	
r-BhasA	p2			
〃	p4			
〃			m-AB	
〃				r-AhasB

> Property
· forConcept [1:1] ▶ Concept
· hasDualP [1:1] ▶ Property
· hasModifier [1:1] ▶ Modifier
· isDualP [1:1] ▶ Property
· ofRelationship [1:1] ▶ Relationship
> p1
· forConcept ▶ c-A
· hasDualP ▶ p2
· hasModifier ▶ pot-0-n
· ofRelationship ▶ r-AhasB
> p2
· forConcept ▶ c-B
· hasModifier ▶ pot-0-n
· isDualP ▶ p1
· ofRelationship ▶ r-BhasA
> p3
· forConcept ▶ c-a1
· hasDualP ▶ p4
· hasModifier ▶ actual
· ofRelationship ▶ r-AhasB
> p4
· forConcept ▶ c-b1
· hasModifier ▶ actual
· isDualP ▶ p3
· ofRelationship ▶ r-BhasA

Property	· forConcept [1:1] ▶ Concept	· hasDualP [1:1] ▶ Property	· hasModifier [1:1] ▶ Modifier	· isDualP [1:1] ▶ Property	· ofRelationship [1:1] ▶ Relationship
p1	c-A				
〃		p2			
〃			pot-0-n		
〃					r-AhasB
p2	c-B				
〃			pot-0-n		
〃				p1	
〃					r-BhasA
p3	c-a1				
〃		p4			
〃			actual		
〃					r-AhasB
p4	c-b1				
〃			actual		
〃				p3	
〃					r-BhasA

> Modifier
· forProperty [0:n] ▶ Property
> actual
· forProperty ▶ p3
· forProperty ▶ p4
> pot-0-n
· forProperty ▶ p1
· forProperty ▶ p2
> pot-1-1

Modifier	· forProperty [0:n] ▶ Property
actual	p3
〃	p4
pot-0-n	p1
〃	p2

> **Concept**
- **hasProperties [0:n] ▶ Property**
- **hasSubconcepts [0:n] ▶ Lattice**
- **hasSuperconcept [1:1] ▶ Lattice**
- **ofModel [1:1] ▶ Model**

> c--TOP
- hasSubconcepts ▶ l1
- hasSubconcepts ▶ l3
- ofModel ▶ m-AB

> c-A
- hasProperties ▶ p1
- hasSubconcepts ▶ l2
- hasSuperconcept ▶ l1
- ofModel ▶ m-AB

> c-a1
- hasProperties ▶ p3
- hasSuperconcept ▶ l2
- ofModel ▶ m-AB

> c-B
- hasProperties ▶ p2
- hasSubconcepts ▶ l4
- hasSuperconcept ▶ l3
- ofModel ▶ m-AB

> c-b1
- hasProperties ▶ p4
- hasSuperconcept ▶ l4
- ofModel ▶ m-AB

Concept	· hasProperties [0:n] ▶ Property	· hasSubconcepts [0:n] ▶ Lattice	· hasSuperconcept [1:1] ▶ Lattice	· ofModel [1:1] ▶ Model
c--TOP		l1		
〃		l3		
〃				m-AB
c-A	p1			
〃		l2		
〃			l1	
〃				m-AB
c-a1	p3			
〃			l2	
〃				m-AB
c-B	p2			
〃		l4		
〃			l3	
〃				m-AB
c-b1	p4			
〃			l4	
〃				m-AB

> **Lattice**
- **isSubconcept [1:1] ▶ Concept**
- **isSuperconcept [1:1] ▶ Concept**

> l1
- isSubconcept ▶ c-A
- isSuperconcept ▶ c--TOP

> l2
- isSubconcept ▶ c-a1
- isSuperconcept ▶ c-A

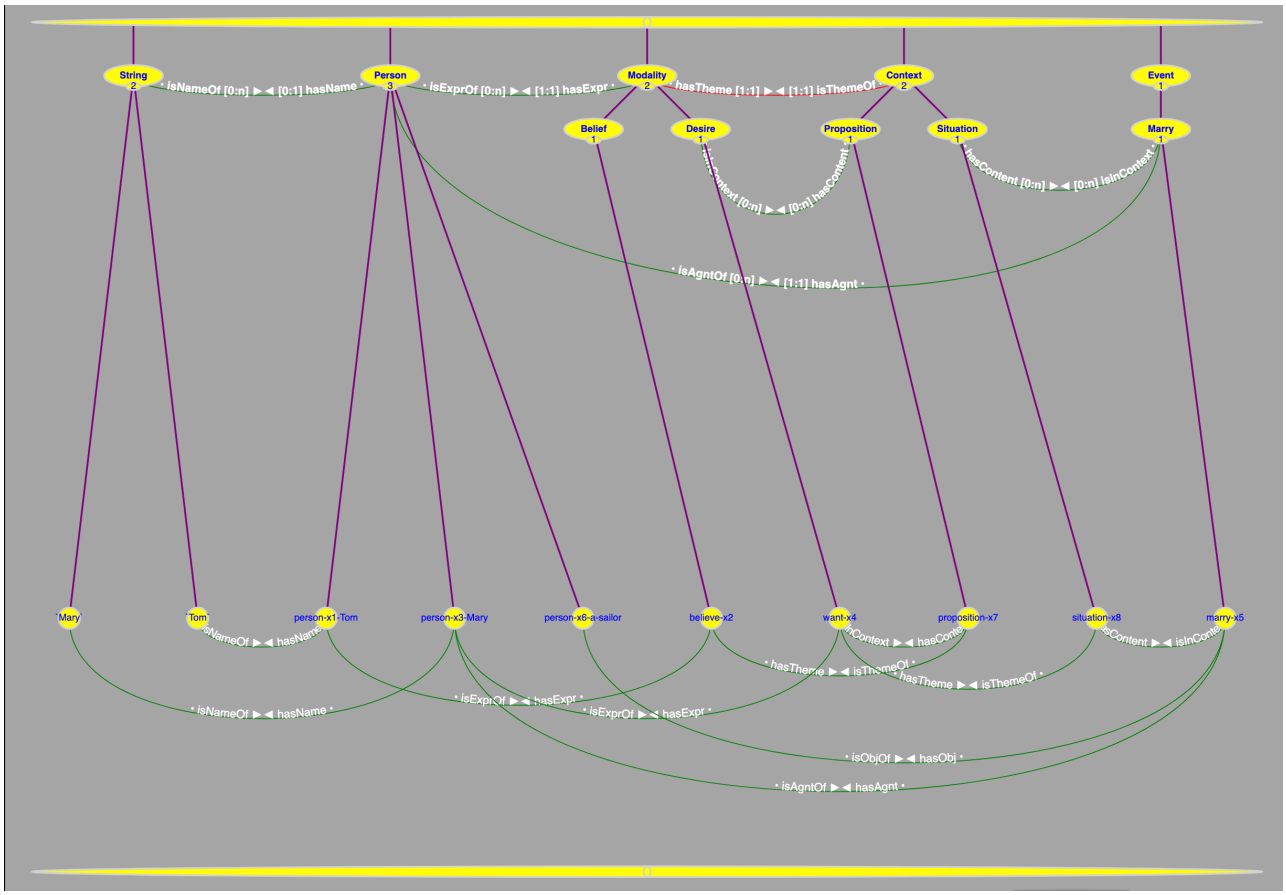
> l3
- isSubconcept ▶ c-B
- isSuperconcept ▶ c--TOP

> l4
- isSubconcept ▶ c-b1
- isSuperconcept ▶ c-B

Lattice	· isSubconcept [1:1] ▶ Concept	· isSuperconcept [1:1] ▶ Concept
l1	c-A	
〃		c--TOP
l2	c-a1	
〃		c-A
l3	c-B	
〃		c--TOP
l4	c-b1	
〃		c-B

Examples from references

Example CRLLP-21 "Tom believes Mary wants to marry a sailor.”



John F. Sowa models the sentence

"Tom believes Mary wants to marry a sailor."

with conceptual graphs and compares to CGIF and KIF, see [2002].

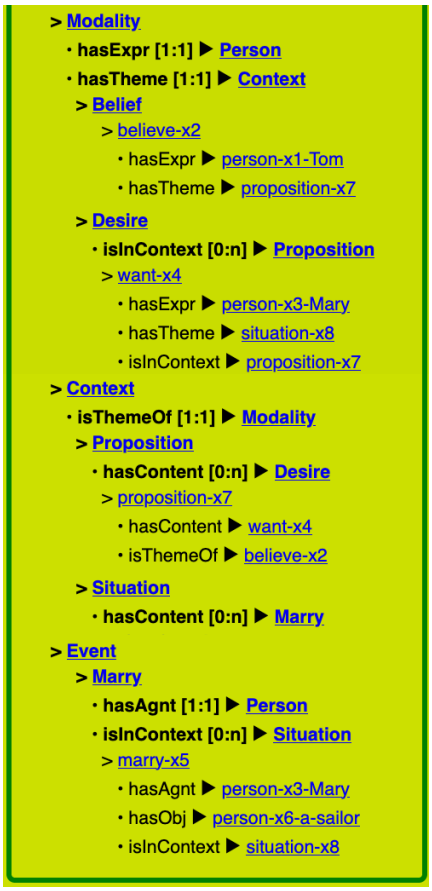
The x-numbers in the concept labels are chosen to match Sowa's CGIF and KIF statements.

The CRLL model here models the sentence with this sequence of relationships:

person-x1-Tom
• isExprOf ▶◀ hasExpr •
believe-x2
• hasTheme ▶◀ isThemeOf •
proposition-x7

TOP
> **String**
• **isNameOf [0:n] ▶ Person**
> `Mary`
• isNameOf ▶ person-x3-Mary
> `Tom`
• isNameOf ▶ person-x1-Tom
> **Person**
• **hasName [0:1] ▶ String**
• **isAgntOf [0:n] ▶ Marry**
• **isExprOf [0:n] ▶ Modality**
> person-x1-Tom
• hasName ▶ `Tom`
• isExprOf ▶ believe-x2
> person-x3-Mary
• hasName ▶ `Mary`
• isAgntOf ▶ marry-x5
• isExprOf ▶ want-x4
> person-x6-a-sailor
• isObjOf ▶ marry-x5

• hasContent ►◄ isInContext •
want-x4
• hasExpr ►◄ isExprOf •
person-x3-Mary
• hasTheme ►◄ isThemeOf •
situation-x8
• hasContent ►◄ isInContext •
marry-x5
• hasAgnt ►◄ isAgntOf •
person-x3-Mary
• hasObj ►◄ isObjOf •
person-x6-a-sailor



A modality (believe-x2, want-x4) indicates that an event (marry-x5) is not or may not be a fact.

A context (proposition-x7, situation-x8) contains the theme of a modality.

Modal logics aim to interpret modalities as intended.

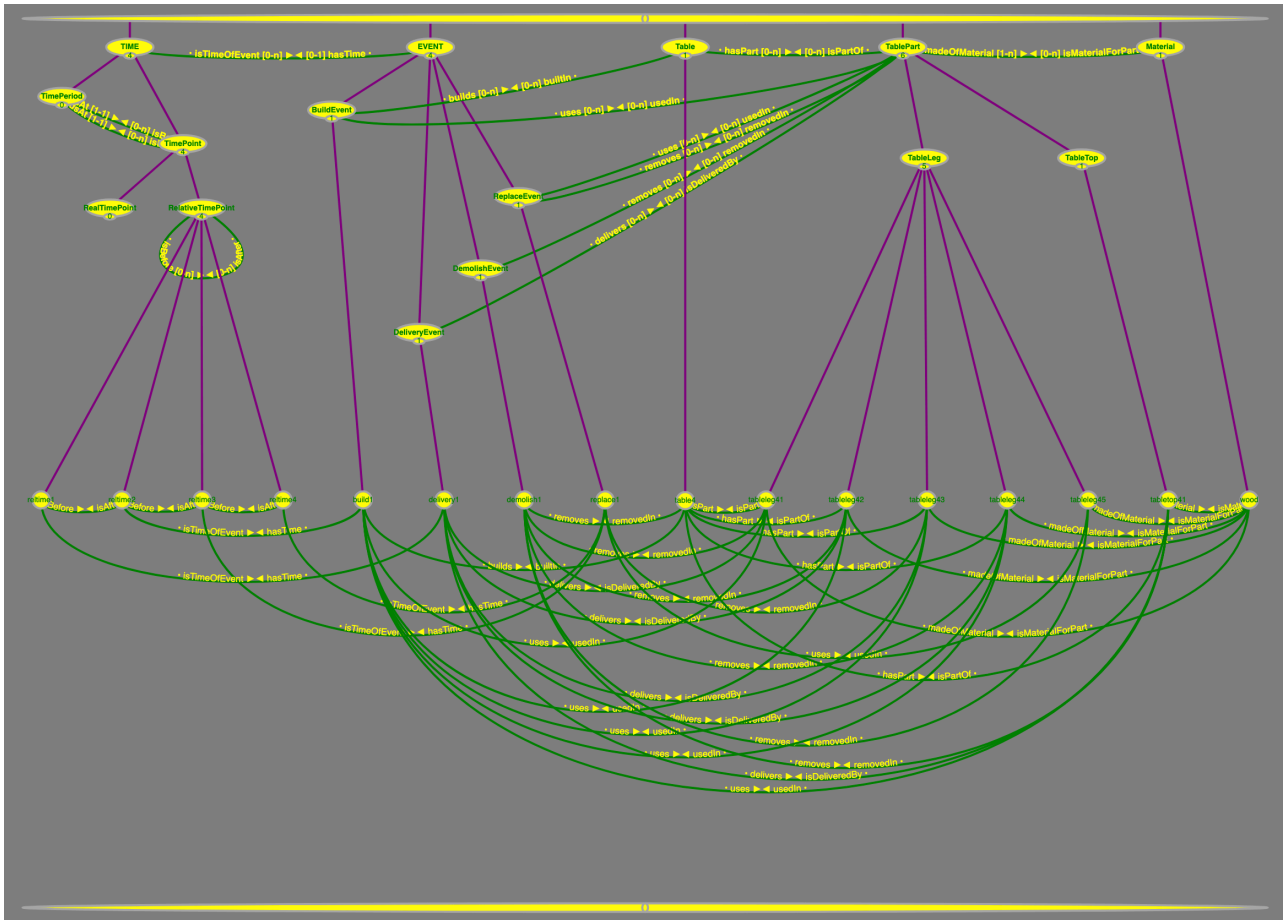
Sowa has two versions of the sentence:

"There is a sailor that Tom believes Mary wants to marry."

"Tom believes there is a sailor that Mary wants to marry."

They are modelled by moving the sailor out of the contexts or into another context.

Example CRLLP-31 FOUST Case 1



Borgo et al (2022a): Foundational ontologies in action. Understanding foundational ontology through examples

Case 1: (Section 3.1. Composition/constitution)“There is a four-legged table made of wood. Some time later, a leg of the table is replaced. Even later, the table is demolished so it ceases to exist although the wood is still there after the demolition.”

GOAL: The example aims to show if and how the ontology models materials, objects, and components and the relationships among them.

FOCUS: The relationship between the wood and the table and the table’s parts over time. (Artefacts and functions are not the focus.)

- **TOP**
  - **> TIME**
    - **• isTimeOfEvent [0-n] ▶ EVENT**
      - **> TimePeriod**
        - **• beginsAt [1-1] ▶ TimePoint**
        - **• endsAt [1-1] ▶ TimePoint**
      - **> TimePoint**
        - **• isBeginOf [0-n] ▶ TimePeriod**
        - **• isEndOf [0-n] ▶ TimePeriod**
          - **> RealTimePoint**
          - **> RelativeTimePoint**
            - **• isAfter [0-n] ▶ RelativeTimePoint**
            - **• isBefore [0-n] ▶ RelativeTimePoint**
              - > reltime1
                - • isBefore ▶ reltime2
                - • isTimeOfEvent ▶ delivery1
              - > reltime2
                - • isAfter ▶ reltime1
                - • isBefore ▶ reltime3
                - • isTimeOfEvent ▶ build1
              - > reltime3
                - • isAfter ▶ reltime2
                - • isBefore ▶ reltime4
                - • isTimeOfEvent ▶ replace1
              - > reltime4
                - • isAfter ▶ reltime3
                - • isTimeOfEvent ▶ replace1
  - **> EVENT**
    - **• hasTime [0-1] ▶ TIME**
      - **> BuildEvent**
        - **• builds [0-n] ▶ Table**
        - **• uses [0-n] ▶ TablePart**
          - > build1
            - • builds ▶ table4
            - • hasTime ▶ reltime2
            - • uses ▶ tableleg41
            - • uses ▶ tableleg42
            - • uses ▶ tableleg43
            - • uses ▶ tableleg44
            - • uses ▶ tabletop41
      - **> DeliveryEvent**
        - **• delivers [0-n] ▶ TablePart**
          - > delivery1
            - • delivers ▶ tableleg44
            - • delivers ▶ tableleg41
            - • delivers ▶ tabletop41
            - • delivers ▶ tableleg43
            - • delivers ▶ tableleg42
            - • hasTime ▶ reltime1
      - **> DemolishEvent**
        - **• removes [0-n] ▶ TablePart**
          - > demolish1
            - • removes ▶ tabletop41
            - • removes ▶ table4
            - • removes ▶ tableleg42
            - • removes ▶ tableleg41
            - • removes ▶ tableleg45
            - • removes ▶ tableleg44
      - **> ReplaceEvent**
        - **• removes [0-n] ▶ TablePart**
        - **• uses [0-n] ▶ TablePart**
          - > replace1
            - • hasTime ▶ reltime3
            - • hasTime ▶ reltime4
            - • removes ▶ tableleg43
            - • uses ▶ tableleg45
  - **> Table**

> **Table**
- **builtIn [0-n] ▶ BuildEvent**
- **hasPart [0-n] ▶ TablePart**

> table4
- builtIn ▶ build1
- hasPart ▶ tableleg44
- hasPart ▶ tableleg42
- hasPart ▶ tabletop41
- hasPart ▶ tableleg41
- hasPart ▶ tableleg43
- removedIn ▶ demolish1

> **TablePart**
- **isDeliveredBy [0-n] ▶ DeliveryEvent**
- **isPartOf [0-n] ▶ Table**
- **madeOfMaterial [1-n] ▶ Material**
- **removedIn [0-n] ▶ DemolishEvent**
- **removedIn [0-n] ▶ ReplaceEvent**
- **usedIn [0-n] ▶ BuildEvent**
- **usedIn [0-n] ▶ ReplaceEvent**

> **TableLeg**

> tableleg41
- isDeliveredBy ▶ delivery1
- isPartOf ▶ table4
- madeOfMaterial ▶ wood
- removedIn ▶ demolish1
- usedIn ▶ build1

> tableleg42
- isDeliveredBy ▶ delivery1
- isPartOf ▶ table4
- madeOfMaterial ▶ wood
- removedIn ▶ demolish1
- usedIn ▶ build1

> tableleg43
- isDeliveredBy ▶ delivery1
- isPartOf ▶ table4
- madeOfMaterial ▶ wood
- removedIn ▶ replace1
- usedIn ▶ build1

> tableleg44
- isDeliveredBy ▶ delivery1
- isPartOf ▶ table4
- madeOfMaterial ▶ wood
- removedIn ▶ demolish1
- usedIn ▶ build1

> tableleg45
- madeOfMaterial ▶ wood
- removedIn ▶ demolish1
- usedIn ▶ replace1

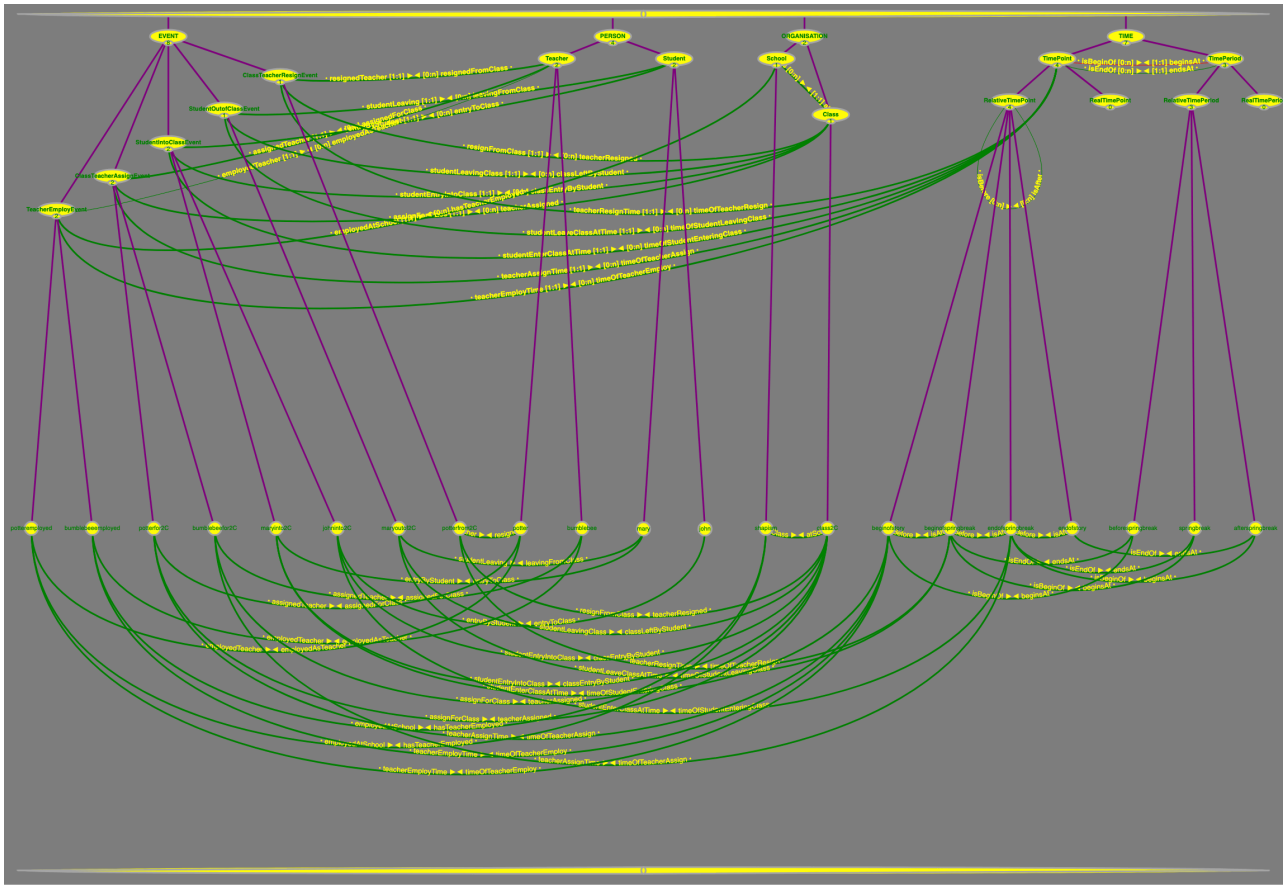
> **TableTop**

> tabletop41
- isDeliveredBy ▶ delivery1
- isPartOf ▶ table4
- madeOfMaterial ▶ wood
- removedIn ▶ demolish1
- usedIn ▶ build1

> **Material**
- **isMaterialForPart [0-n] ▶ TablePart**

> wood
- isMaterialForPart ▶ tabletop41
- isMaterialForPart ▶ tableleg41
- isMaterialForPart ▶ tableleg42
- isMaterialForPart ▶ tableleg43
- isMaterialForPart ▶ tableleg44
- isMaterialForPart ▶ tableleg45

Example CRLLP-32 FOUST Case 2



Borgo et al (2022a): Foundational ontologies in action. Understanding foundational ontology through examples

Case 2: (Section 3.2. Roles)

Mr. Potter is the teacher of class 2C at Shapism School and resigns at the beginning of the spring break. After the spring break, Mrs. Bumblebee replaces Mr. Potter as the teacher of 2C. Also, student Mary left the class at the beginning of the break and a new student, John, joins in when the break ends.

GOAL: The example aims to show if and how the ontology models the relationships between roles, players and organisations.

FOCUS: The change of roles/players; the vacancy of the teaching position; persistence of the class while students come and go.


> EVENT
> TeacherEmployEvent
• employedAtSchool [1:1] ▶ School
• employedTeacher [1:1] ▶ Teacher
• teacherEmployTime [1:1] ▶ TimePoint
> potteremployed
• employedAtSchool ▶ shapism
• employedTeacher ▶ potter
• teacherEmployTime ▶ beginofstory
> bumblebeeemployed
• employedAtSchool ▶ shapism
• employedTeacher ▶ bumblebee
• teacherEmployTime ▶ beginofstory
> ClassTeacherAssignEvent
• assignedTeacher [1:1] ▶ Teacher
• assignForClass [1:1] ▶ Class
• teacherAssignTime [1:1] ▶ TimePoint
> potterfor2C
• assignedTeacher ▶ potter
• assignForClass ▶ class2C
• teacherAssignTime ▶ beginofstory
> bumblebeefor2C
• assignedTeacher ▶ bumblebee
• assignForClass ▶ class2C
• teacherAssignTime ▶ endofspringbreak
> StudentIntoClassEvent
• entryByStudent [1:1] ▶ Student
• studentEnterClassAtTime [1:1] ▶ TimePoint
• studentEntryIntoClass [1:1] ▶ Class
> maryinto2C
• entryByStudent ▶ mary
• studentEnterClassAtTime ▶ beginofstory
• studentEntryIntoClass ▶ class2C
> johninto2C
• entryByStudent ▶ john
• studentEnterClassAtTime ▶ endofspringbreak
• studentEntryIntoClass ▶ class2C
> StudentOutofClassEvent
• studentLeaveClassAtTime [1:1] ▶ TimePoint
• studentLeaving [1:1] ▶ Student
• studentLeavingClass [1:1] ▶ Class
> maryoutof2C
• studentLeaveClassAtTime ▶ beginofspringbreak
• studentLeaving ▶ mary
• studentLeavingClass ▶ class2C
> ClassTeacherResignEvent
• resignedTeacher [1:1] ▶ Teacher
• resignFromClass [1:1] ▶ Class
• teacherResignTime [1:1] ▶ TimePoint
> potterfrom2C
• resignedTeacher ▶ potter
> ClassTeacherResignEvent
• resignedTeacher [1:1] ▶ Teacher
• resignFromClass [1:1] ▶ Class
• teacherResignTime [1:1] ▶ TimePoint
> potterfrom2C
• resignedTeacher ▶ potter
• resignFromClass ▶ class2C
• teacherResignTime ▶ beginofspringbreak
> PERSON
> Teacher
• assignedForClass [0:n] ▶ ClassTeacherAssignEvent
• employedAsTeacher [0:n] ▶ TeacherEmployEvent
• resignedFromClass [0:n] ▶ ClassTeacherResignEvent
> potter
• assignedForClass ▶ potterfor2C
• employedAsTeacher ▶ potteremployed
• resignedFromClass ▶ potterfrom2C
> bumblebee
• assignedForClass ▶ bumblebeefor2C
• employedAsTeacher ▶ bumblebeeemployed
> Student
• entryToClass [0:n] ▶ StudentIntoClassEvent
• leavingFromClass [0:n] ▶ StudentOutofClassEvent
> mary
• entryToClass ▶ maryinto2C
• leavingFromClass ▶ maryoutof2C
> john
• entryToClass ▶ johninto2C
> ORGANISATION
> School
• hasClass [0:n] ▶ Class
• hasTeacherEmployed [0:n] ▶ TeacherEmployEvent
> shapism
• hasClass ▶ class2C
• hasTeacherEmployed ▶ potteremployed
• hasTeacherEmployed ▶ bumblebeeemployed
> Class
• atSchool [1:1] ▶ School
• classEntryByStudent [0:n] ▶ StudentIntoClassEvent
• classLeftByStudent [0:n] ▶ StudentOutofClassEvent
• teacherAssigned [0:n] ▶ ClassTeacherAssignEvent
• teacherResigned [0:n] ▶ ClassTeacherResignEvent
> class2C
• atSchool ▶ shapism
• classEntryByStudent ▶ johninto2C
• classEntryByStudent ▶ maryinto2C
• classLeftByStudent ▶ maryoutof2C
• teacherAssigned ▶ bumblebeefor2C
• teacherAssigned ▶ potterfor2C
• teacherResigned ▶ potterfrom2C
> TIME
> TIME
> TimePoint
• isBeginOf [0:n] ▶ TimePeriod
• isEndOf [0:n] ▶ TimePeriod
• timeOfStudentEnteringClass [0:n] ▶ StudentIntoClassEvent
• timeOfStudentLeavingClass [0:n] ▶ StudentOutofClassEvent
• timeOfTeacherAssign [0:n] ▶ ClassTeacherAssignEvent
• timeOfTeacherEmploy [0:n] ▶ TeacherEmployEvent
• timeOfTeacherResign [0:n] ▶ ClassTeacherResignEvent
> RelativeTimePoint
• isAfter [0:n] ▶ RelativeTimePoint
• isBefore [0:n] ▶ RelativeTimePoint
> beginofstory
• isBefore ▶ beginofspringbreak
• isBeginOf ▶ beforespringbreak
• timeOfStudentEnteringClass ▶ maryinto2C
• timeOfTeacherAssign ▶ potterfor2C
• timeOfTeacherEmploy ▶ bumblebeeemployed
• timeOfTeacherEmploy ▶ potteremployed
> beginofspringbreak
• isAfter ▶ beginofstory
• isBefore ▶ endofspringbreak
• isBeginOf ▶ springbreak
• isEndOf ▶ beforespringbreak
• timeOfStudentLeavingClass ▶ maryoutof2C
• timeOfTeacherResign ▶ potterfrom2C
> endofspringbreak
• isAfter ▶ beginofspringbreak
• isBefore ▶ endofstory
• isBeginOf ▶ afterspringbreak
• isEndOf ▶ springbreak
• timeOfStudentEnteringClass ▶ johninto2C
• timeOfTeacherAssign ▶ bumblebeefor2C
> endofstory
• isAfter ▶ endofspringbreak
• isEndOf ▶ afterspringbreak
> RealTimePoint
> TimePeriod
• beginsAt [1:1] ▶ TimePoint
• endsAt [1:1] ▶ TimePoint
> RelativeTimePeriod
> beforespringbreak
• beginsAt ▶ beginofstory
• endsAt ▶ beginofspringbreak
> springbreak
• beginsAt ▶ beginofspringbreak
• endsAt ▶ endofspringbreak
> afterspringbreak
• beginsAt ▶ endofspringbreak
• endsAt ▶ endofstory
> RealTimePeriod

References

[1940] H.S. Leonard, N. Goodman. The Calculus of Individuals and its Uses. The Journal of Symbolic Logic, June 1940, pp. 45-55.

[1970] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, June 1970, pp. 377-387.

[1975] C. J. Date. An Introduction to Database Systems. Pearson.

[1976a] P.P.-S. Chen. The Entity-Relationship Model: Towards a Unified View of Data. ACM Transactions on Database Systems, March 1976.

[1976b] J. F. Sowa. Conceptual Graphs for a Data Base Interface. IBM Journal of Research and Development, July 1976, pp. 336-357.

[1982] D. Bjørner, C. B. Jones. Formal Specification & Software Development. Prentice-Hall.

[1984a] J. F. Sowa. Conceptual Structures, Information Processing in Mind and Machine. Addison-Wesley.

[1984b] M. L. Brodie, J. Mylopoulos, J. W. Schmidt (eds). On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer-Verlag.

[1987] P. Simons. Parts: A Study in Ontology. Oxford University Press.

[1990] B. A. Davey, H. A. Priestley. Introduction to Lattices and Order. Cambridge University Press.

[1991] D. Lewis. Parts of Classes. Basil Blackwell, Oxford.

[1992a] P. Ingwersen. Information Retrieval Interaction. Taylor Graham.

[1992b] J. F. Nilsson. A Concept Object Algebra. In In H. Kangassalo, H. Jaakkola (eds). Information Modelling and Knowledge Bases IV. IOS Press.

[1993a] G. S. Pedersen. Relationship Lattices for Information Modelling. In H. Kangassalo, H. Jaakkola (eds). Information Modelling and Knowledge Bases V. IOS Press.

[1993b] G. S. Pedersen. A Browser for Bibliographic Information Retrieval, Based on an Application of Lattice Theory. In Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, June 1993, Pittsburgh. ACM Press.

[1993c] G. S. Pedersen. Relationship lattices and their role in object-oriented software engineering. Poster session at the Conference on Object-Oriented Programming Systems, Languages, and Applications (00PSLA), 26 September-1 October 1993, Washington, D.C.

[1993d] E. Bertino, L. Martino. Object-Oriented Database Systems, Concepts and Architectures. Addison-Wesley.

[1993e] J. F. Nilsson. An Algebraic Logic for Concept Structures. In H. Kangassalo, H. Jaakkola (eds). Information Modelling and Knowledge Bases V. IOS Press.

[1994a] G. S. Pedersen. Rule Modelling in Relationship Lattices. In H. Kangassalo, H. Jaakkola (eds). Information Modelling and Knowledge Bases VI. IOS Press.

[1994b] G. S. Pedersen. Conceptual Modelling without the Distinction between Individuals and Singleton Sets. Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology, held in conjunction with ECAI'94 (11th European Conference on Artificial Intelligence), Amsterdam, August 1994.

[1994c] G. S. Pedersen. Integrating Object-Orientation and Knowledge Representation through a Formalism based on Lattice Theory. Workshop on Integrating Object-Orientation and Knowledge Representation, held in conjunction with ECAI'94 (11th European Conference on Artificial Intelligence), Amsterdam, August 1994.

[1995] Gert Schmeltz Pedersen. Construction of multimedia software systems for retrieval and presentation of information from heterogeneous sources. Copenhagen Business School. Ph. D. serie 3.95 .

[2000] J. F. Sowa. Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co.

[2002] J. F. Sowa. Architectures for intelligent systems. IBM Systems Journal, vol. 41, no. 3.

[2008a] G. Guizzardi, T. Halpin. Ontological foundations for conceptual modelling. Applied Ontology. IOS Press.

[2008b] Ø. Linnebo, D. Nicolas. Superplurals in English. Analysis.

[2010] L. S. Moss. Extension of example from Natural Logic and Semantics, Lecture Notes in Computer Science, September 2010.

[2016a] A. Oliver, T. Smiley. Plural Logic: Second Edition. Oxford University Press.

[2016b] A. Varzi. Mereology. The Stanford Encyclopedia of Philosophy.

[2018] S. Florio, Ø. Linnebo. Logic and Plurals. In The Routledge Handbook of Collective Intentionality.

[2020] S. Florio, D. Nicolas. Plurals and Mereology. Journal of Philosophical Logic.

[2021a] S. Florio, Ø. Linnebo. The Many and the One - A Philosophical Study of Plural Logic. Oxford University Press.

[2021b] A. C. Varzi, A. J. Cotnoir. Mereology. Oxford University Press.

[2022a] S. Borgo, A. Galton, O. Kutz. Foundational ontologies in action. Understanding foundational ontology through examples. In Applied Ontology. IOS Press.

[2022b] Ø. Linnebo. Plural Quantification. The Stanford Encyclopedia of Philosophy.

[2023] J. Garson. Modal Logic. The Stanford Encyclopedia of Philosophy.

[2024a] D. Nicolas. The Logic of Mass Expressions. The Stanford Encyclopedia of Philosophy.

[2024b] D. Westerståhl. Generalized Quantifiers. The Stanford Encyclopedia of Philosophy.

[2024c] V. Goranko, A. Rumberg. Temporal Logic. The Stanford Encyclopedia of Philosophy.

[2024d] J. Väänänen. Second-order and Higher-order Logic. The Stanford Encyclopedia of Philosophy.

[2026] G.S. Pedersen.Concept Relationship Lattice Logic - where Conceptual Modelling and Plural Logic meet.