

Architecture of a Semantic Networking Language

© Pierre Lévy, CRC, FRSC, University of Ottawa, version 2, 2008 / 03 / 25

0. Introduction

0.0. Definitions

0.0.0. Categories

A category (a universal, a set, a class) can be identified by its extension (list of individuals that are members of the category) or by its intention, or essence (the common property of the individuals that are members of the category).

0.0.1. Semantic Space

Proposition

The *semantic space* is the abstract container of essences, that are categories identified by their intention.

Comments

Other (and more common) names for the intensive identity of a category are : signification, meaning, signified, sense...

The semantic space is unique and infinite (open-ended).

0.0.2. Semantic Positions

A unique category (a unique essence) occupies a unique *position* in the semantic space.

0.0.3. Semantic Steps

A semantic step is a transition from the semantic position of a category to the semantic position of another category.

0.0.4. Semantic Networks

A semantic network is a set of semantic steps, or an oriented graph of categories semantic positions.

0.0.5 Semantic Paths

A semantic path is a connected set of steps in a semantic network.

0.1. Semantic engineering foundations

0.1.0. Initial problem

The initial problem of semantic engineering is to find a coordinate system of the semantic space (that is to say a semantic positioning code for categories) matching two interdependant constraints.

First constraint: there should be *one single* code (or one single semantic address) for a *unique* semantic position.

Second constraint: the coding system should allow *automatable* production of semantic steps that can be *automatically* organized into relevant semantic networks and semantic paths.

0.1.1. Solution principle: differential semantics

The solution of the initial problem implies the invention of a formal and generative system of *qualitative differences* to be used as a coordinate system for the semantic space. The semantic differences coding system must be (a) formal and (b) generative in order to encompass an open-ended space by computational means.

Coded by such a formal generative system, an essence, or a *signification* (the intensive identity of a category) can become an object of automatable manipulation.

More specifically, the notion of automatable manipulation of semantic codes can be decomposed into three properties.

- 1) Semantic codes can be processed as variables (input and output) by a lot of different computable functions.
- 2) These computable functions can be composed (arranged in sequences, in loops and in parallel).
- 3) Transition from inputs to outputs can be used as the basis for the computation of semantic steps, and therefore functions - and compositions of functions - can compute semantic networks and semantic paths.

0.1.2. IEML: the Information Economy MetaLanguage

IEML meets the constraints of the initial problem and the requirements of the principle solution.

It allows the coding of a practically infinite number of significations from a relatively short list of co-dependent qualitative differences :

- role differences (source|destination|translator)¹,
- layer differences (primitives|events|relations, ideas|phrases|semes)

¹ Notation convention: throughout this text, the symbol "|" is used to denote the set operator "union".

- existential differences (empty|full)
- cognitive pattern differences (polar-verb|ternary-noun)
- elementary differences (virtual|actual|sign|being| thing).

The formal system of qualitative differences generated by IEML casts (or projects) a regular coordinate system on the semantic space.

0.1.3. SNL: the Semantic Networking Language

The goal of this paper is to show that semantic steps, networks and paths between IEML coded semantic positions can be generated automatically. In order to suggest this possibility, the architecture of a programming language called SNL (Semantic Networking Language) able to generate determined semantic motion between semantic positions is outlined here.

IEML is the semantic position coding system and SNL is the operating system of this code, allowing the automatic generation of paths in the semantic space. These paths can be used for navigation or for information value channeling.

0.1.4. CIP: the Collective Intelligence Protocol

0.1.4.1. Relationship between SNL and IEML

IEML is made for SNL and SNL for IEML. They are complementary.

IEML generate the set of possible variables and SNL the set of possible operators on these variables.

Again : IEML is a code for positions in the semantic space and SNL a language allowing the programming of transitions between positions.

Again: IEML stands for identifying positions in the semantic space and SNL stands for programming motion between semantic positions.

0.1.4.2. The CIP has been *invented* and the semantic space has been *discovered*

Taken together, IEML and SNL form the Collective Intelligence Protocol (CIP).

The CIP is a meta-linguistic convention. It is the result of an *invention*. This symbolic tool has not been "discovered" but designed and built according to the purpose of mapping and navigating the semantic space.

Such a symbolic tool is the last layer of a thick stack of technical conditions (among which computers, programming languages, the Internet and the Web) that will make possible, first, the *exploration* of the semantic space and, second, the *management* of information economies.

The CIP cannot be anything else than a invented technical convention. But this invented convention paves the way for the scientific *discovery* of the semantic space.

0.1.5. Semantic engines

The CIP boosts the Turing machine with a semantic engine. IEML works like a semantic identification system for the cells of the infinite recording tape of the universal Turing machine. SNL works as a semantic guiding system for the movements of the read-write head of the universal Turing Machine.

From a more practical perspective, a database is augmented by a *semantic engine* when...

- 1) Its recorded items are indexed/identified by IEML codes.
 - 2) The semantic navigation systems and the value circulation channels of the database are generated by SNL programs.
- A semantic engine is made of semantic programs (or SNL-coded semantic circuits) working on a semantic domain (a subset of the IEML-coded semantic space).
A semantic engine supports semantic motion.

0.1.6. Semantic engineering activity

The goal of semantic engineering is to build the most efficient *semantic engines* in relation to the needs of human communities.

Its basic task consists in *matching* a set of natural language descriptors to a set of IEML codes.

The bottom line of good matching is a mini-max *optimization* process.

What is to *minimize* is the number of IEML categories descriptors *created* to map the semantic domain of a community of users.

What is to *maximize* is the number and variety of automatable relevant *semantic steps* interconnecting the semantic domain.

These semantic steps - organized in semantic paths travelling into semantic networks - are supposed to be useful vehicles for the collaborative cognitive activities of the community of users of a database. Semantic engineers design the semantic networking of databases in order to augment the collective intelligence of the communities of users.

Semantic engineering is not a purely deductive science. It will be necessary, for the new discipline, to accumulate systematically a wealth of experience, suitable for individual and collective learning, in order to meet its goal : building useful semantic engines for the digitized memory of human communities.

0.2. SNL overview

0.2.1 Elementary Network Generators

Definition

Elementary Network Generators (ENG) are abstract machines with valid IEML expressions in input and valid IEML expressions in output. They create semantic steps between input categories and output categories.

An ENG can be defined by a couple of operators (KNO, QUA) where KNO is a knot operator and QUA is a qualifier.

I will describe in detail the knots operators and the qualifiers below, but I can already say in this introduction that the knots operators have to do with the nature of the cognitive operations (sort, replace, analyze, synthesize...) on categories and that the qualifiers are related to the "differences" addressed by the knots operators (like: full *versus* empty, polar *versus* ternary, etc.).

Comments

- The ENG is the "formal neuron" of semantic engineering.
- The same categories can be the input and the output of different ENGs. So a category plays the role of a crossroad - or a node - between semantic links woven by ENGs. The meaning of a category cannot be separated from the network of links *crossing* this category.
- ENG can be read "Elementary *Navigation* Generators" or "Elementary *Neuron* Generator" and the Semantic *Networking* language is also a Semantic *Navigation* language or a Semantic *Neurons* Language.

0.2.2. Semantic circuits

Description

A *semantic circuit* is made of ENGs disposed in sequence, in parallel and/or in loop.

Comments

The semantic circuit is the "neural network" of semantic engineering.

A semantic circuit is a "spider" of semantic networks and semantic navigations.

Metaphorically, the IEML grid is the "warp" and the network woven by the circuit is the "weft" of the semantic cloth.

The very fact that ENGs can be *composed* - and that any one of them can take in input the output of any other - shows that IEML maps "one semantic space" across layers and networks.

0.2.3. Semantic networks

Definition

Semantic networks are trans-layers graphs of IEML-coded categories woven by semantic circuits.

Comment

Semantic networks may be used for at least three (interdependant) purposes : search and navigation, semantic differences (rather than semantic distance) measurement, value distribution.

Comparisons and "semantic differences" measurements between categories are necessarily related to identified semantic networks used as measurement instruments.

Concerning value distribution: as semantic networks are also semantic *neural networks*, the semantic neurons can be programmed to process information value...

0.3. Plan of this text

The first chapter will describe *gestalt* and *mereologic classes* of categories, that will be used in the formal definition of ENGs.

The second chapter will explain the notion of *extent* of an ENG, that is basically the "depth" of its operation in regard of the layer stack.

The third and fourth chapters expound the knot and the qualifier that are the two principal parts of an ENG.

The sixth chapter gives the detail of the workings of the seven basic ENGs, with several examples.

Finally, the seventh chapter explains the semantic circuits and develops two examples : the *abilities table* and the *standard order* circuits.

1. Gestalt and mereologic classification

1.1. Gestalt classification

1.1.0. Introduction

The new syntax, version 3.0 (14 feb 2008) makes valid (or well formed) flows and categories that were forbidden in the previous versions (empty source players with full destination players and empty destination players with full translator players). In order to process these new categories, I have created two classification systems addressing the structure of the flows and ultimately based on the existential composition (fullness, emptiness or completeness) of their role players.

1.1.1. Population classification

I distinguish three "population" classes:

- the J (mixed) population class
- the V (vacant) population class,
- the P (populated) population class

The three classes J, V and P are defined in a recursive way.

Rule for layer 1

If the category is complete (*I:**)

Then the category is mixed (J)

If the category is empty (*E:**)

Then the category is vacant (V)

If the category is full (*F:**)

Then the category is populated (P)

Rules for layers 2,3,4,5,6 (generated flows)*Mixed flows (J)*Mixed class

If the role players are [all mixed (J)] OR [void (V) AND mixed (J)]

Then the category is mixed (J)

Mixed subclasses

(JJJ) (JJV) (JVJ) (JVV) (VJJ) (VJV) (VVJ)

*Void flows (V)*Void class

If the three role players are vacant (V)

Then the category is vacant (V)

The void class contains only one subclass (VVV)

*Populated flows (P)*Populated class

If at least one of the three role players is populated (P)

Then the category is populated (P)

Populated subclasses

(JJP) (JPJ) (JPV) (JPP) (VJP) (VVP) (VPJ) (VPV) (VPP) (PJJ) (PJV) (PJP) (PVJ)
(PVV) (PVP) (PPJ) (PPV) (PPP)

1.1.2. Rationality classification*Semantic rationality*

The rational order for an information flow is : 1) source, 2) destination, 3) translator, because the source comes first as necessary condition for a flow and

because the destination, or goal, is logically anterior to the translator (that is a mean, or a medium)

From this basic rational order, I define *semantic rationality* as the property of a flow in which these three conditions are met.

1) the source_player is populated
So(P)

2) if the destination_player is vacant, the translator_player is vacant,
IF De(V) THEN Tr(V)

3) if the translator_player is populated, the destination_player is populated.
IF Tr(P) THEN De(P)

Semantic irrationality

A flow is irrational if...

- 1) at least one of its role players is populated
- 2) at least one of the three continuity rule is broken

The four rationality classes

Rational and irrational are defined in terms of population subclasses

From these definitions, four rationality classes can be defined:

- Rational (Z)
- Irrational (Y)
- Void (W) when all the roleplayers are vacant.
- Ambiguous (X)

A flow is amiguous when the presence of mixed role players prevents from deciding between rationality and irrationality. A ambiguous flow is not certainly rational, not certainly disrational and not certainly void.

Rational flows (Z)

PJV PVV PPJ PPV PPP

All the rational flows are populated.

Some (PJV and PPJ) flow structures contain mixed role players, but these do not impede the continuity decision.

Disrational flows (Y)

VPJ VVP VJP VPV VPP PVP

All the disrational flows are populated.

Some (VPJ and VJP) flow structures contain mixed role players, but these do not impede the discontinuity decision.

Ambiguous flows (X)

JJV JVJ JVV VJJ VJV VVJ
 JJP JPJ JPV JPP PJJ PJP PVJ

The first line shows the mixed ambiguous flows and the second line, the populated ambiguous flows.

Void flows (W)

VVV

1.1.3. Summary

Population classification $dPOP = (P|J|V)$
 Topological classification $dRAT = (Z|Y|X|W)$

1.2. Mereologic classification**1.2.1. Rule for all mereologic classifications**

The *class* of a generated category is given (recursively) by the *class of its source player*.

1.2.2. Mereologic classifications

| | |
|------------------------------------|---------------------|
| - Existential classification | $dI = (I E F)$ |
| - Cognitive pattern classification | $dF = (O M)$ |
| - Primitive classification | $dOM = (U A S B T)$ |
| - Polar classification | $dO = (U A)$ |
| - Ternary classification | $dM = (S B T)$ |

2. Extent of Elementary Network Generators**2.0. Introduction**

The goal of this chapter is to introduce the reader to the notion of "extent" of a semantic networking operation. It is difficult to understand the concept without example. So, I will first give an example of elementary network generator (ENG) and then use this example in the explanation of the "extent" concept.

An ENG is made of a knot and a qualifier. Let's take - as example of a knot - a analytical taxonomic operator, noted [ANA=], "ANA" for analysis and "=" for "same layer". As "qualifier" of this knot, I chose an existential difference of the source role_player. The ENG has the (simplified) form: [ANA=]#[So(dI)] The formal expression means : taxonomic analysis in function of the existential class of the source. Basically, the operator [ANA=]#[So(dI)] will analyse into "E"s

(emptiness) and "F"s (fullness) the "I"s (completeness) of the source of the expression in input.

I will now give the rules governing the maximal or *last layer extent* of the networking operation, with an example illustrating each rule. Then I will define *layer n-1* and *same layer* extents.

2.1. Rule for *last layer* extent

If the analysis of the *first* role player of the selected role player (here : So) has not been done,

Then perform the analysis of the first role player of the selected role player, according to the selected classification (here : dI).

Example 1 :

Input : *I~~- I~~- I~~-'** == *I~~~'*

Operator : [ANA=]#[So(dI)]

Output :

→*E:I:I. I:I:I. I:I:I.- I~~- I~~-'** == *E~~~'*

→*F:I:I. I:I:I. I:I:I.- I~~- I~~-'** == *F~~~'*

If the analysis of the *first* role player of the selected role player has been done,

Then perform the analysis on the *second* role player of the selected role player, according to the selected classification.

Example 2

Input : *F~~~** == *F:I:I. I:I:I. I:I:I.- I~~- I~~-'**

Operator : [ANA=]#[So(dI)]

Output :

→*F:I:I. E:I:I. I:I:I.- I~~- I~~-'** == *F~.E~.I~.- I~~- I~~-'**

→*F:I:I. F:I:I. I:I:I.- I~~- I~~-'** == *F~.F~.I~.- I~~- I~~-'**

If the analysis of the *second* role player of the selected role player has been done,

Then perform the analysis on the *third* role player of the selected role player, according to the selected classification.

Example 3

*F:I:I. F:I:I. I:I:I.- I~~- I~~-'** == *F~.F~.I~.- I~~- I~~-'**

Operator : [ANA=]#[So(dI)]

Output :

→*F:I:I. F:I:I. E:I:I.- I~~- I~~-'** == *F~.F~.E~.- I~~- I~~-'**

→*F:I:I. F:I:I. F:I:I.- I~~- I~~-'** == *F~.F~.F~.- I~~- I~~-'**

If the analysis of the *third* role player of the selected role player has been done,

Then perform the analysis on the *second* role player of the first role player of the selected role player, according to the selected classification.

Example 4

Input: *F:I:I. F:I:I. F:I:I.- I~~- I~~-' ** == *F~.F~.F~- I~~- I~~-' **

Operator : [ANA=]#[So(dI)]

Output :

→*F:E:I. F:I:I. F:I:I.- I~~- I~~-' ** == *F:E:I. F~.F~- I~~- I~~-' **

→*F:F:I. F:I:I. F:I:I.- I~~- I~~-' ** == *F:F:I. F~.F~- I~~- I~~-' **

If the analysis of the *second* role player of the first role player of the selected role player has been done,

Then perform the analysis on the *third* role player of the first role player of the operator-selected role player, according to the operator-selected classification.

Example 5

Input: *F:F:I. F:I:I. F:I:I.- I~~- I~~-' ** == *F:F:I. F~.F~- I~~- I~~-' **

Operator : [ANA=]#[So(dI)]

Output :

→*F:F:E. F:I:I. F:I:I.- I~~- I~~-' ** == *F:F:E. F~.F~- I~~- I~~-' **

→*F:F:F. F:I:I. F:I:I.- I~~- I~~-' ** == *F:F:F. F~.F~- I~~- I~~-' **

And so on...

If all the role players of every layer for the selected role of the input set expression have been analysed,

Then, the analysis is finished.

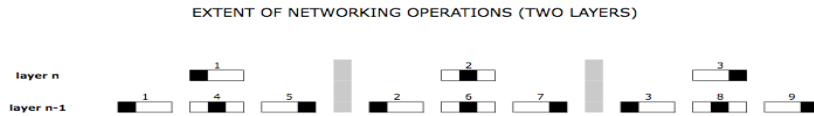
Example 6

Input : *F:F:F. F:E:F. F:E:E.- I~~- I~~-' **

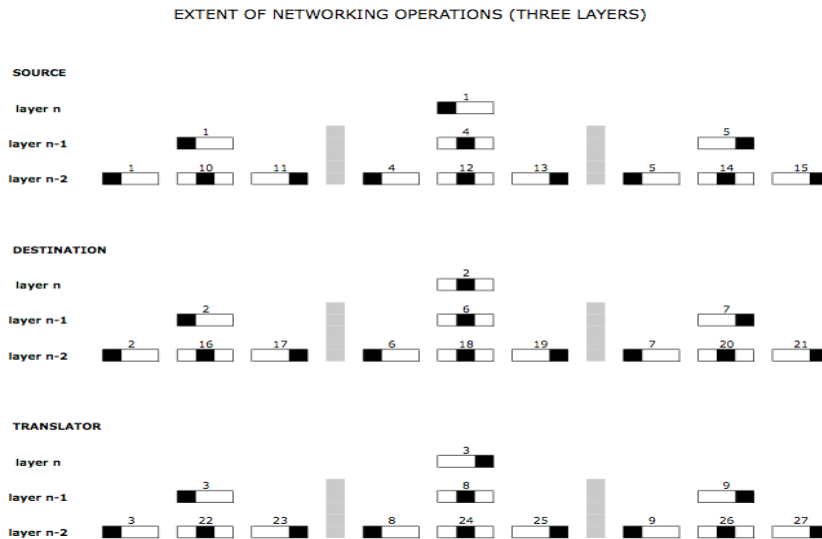
Operator : [ANA=]#[So(dI)]

The analysis cannot go further with this operator, it is terminated. To continue the analysis, the user must chose another operator, by changing the role ex : [De(dI)] or by changing the classification ex : [So(dOM)].

The following diagram shows the principle of recursive operation for any ENG, and not only for the [ANA=] example (the selected set belongs to layer n+1).



Principle of recursive operation for three layers (the selected set belongs to layer n+1)



2.2. Rule for *layer n-1* and *same layer extents*

When the extent of an ENG is "layer n-1" instead of "last layer" the operation stops at layer n-1 instead of continuing until the last layer.

When the extent of an ENG is "layer n" or "same layer", the operation stops at layer n. It transforms the source of the selected role, then its destination, then its translator and then it stops.

3. Knots

3.0. Introduction

A knot can be defined as a triple (THR, EXT, OBJ), where THR is a thread, EXT is the extent of the knot operator and OBJ is the "layer object" of the knot operator.

THR can have at least four values : SOR (sort), REP (replace), ANA (analyze) and SYN (synthetize)

EXT can have the values of layers differences and OBJ can have the value of layer differences and the values of layer places.

3.1. Knots vocabulary

3.1.1. Threads

| | |
|-----|------------------------------|
| THR | Threads |
| ANA | Analysis (toward contained) |
| SYN | Synthesis (toward container) |
| REP | Replace (substitution) |
| SOR | Sort (ordering) |

Others kinds of threads can be defined.

3.1.2. Layers

| | |
|--------------------------------|--------------|
| LAY | layers |
| PLA (: . - ' , _) | layer places |

| | |
|-----------------------------|-------------------|
| dPLA (= + - ≠ ≈) | layer differences |
| = | Same layer |
| + | layer n+1 |
| - | layer n-1 |
| ≠ | different layers |
| ≈ | whatever layer |

3.2. Categories of Knots

A knot - between brackets - is symbolized by a thread preceded and followed by a layer selection. [KNO] == [dPLA THR LAY]

The place difference before the thread symbolize the extent of the knot.

The layer symbol after the thread symbolize the object of the knot.

| | |
|---------|--------------------------------------------------|
| [ANA] | Analysis |
| [ANA=] | Taxonomic analysis |
| [=ANA=] | Taxonomic analysis to the extent of same layer |
| [-ANA=] | Taxonomic analysis to the extent of layer n-1 |
| [≈ANA=] | Taxonomic analysis to the extent of last layer |
| [ANA-] | Composition analysis |
| [=ANA-] | Composition analysis to the extent of same layer |
| [-ANA-] | Composition analysis to the extent of layer n-1 |
| [≈ANA-] | Composition analysis to the extent of last layer |
| [SYN] | Synthesis |
| [SYN=] | Taxonomic synthesis |
| [=SYN=] | Taxonomic synthesis to the extent of same layer |
| [-SYN=] | Taxonomic synthesis to the extent of layer n-1 |
| [≈SYN=] | Taxonomic synthesis to the extent of last layer |

| | |
|------------|---------------------------------------------------|
| [SYN+] | Composition synthesis |
| [=SYN+] | Composition synthesis to the extent of same layer |
| [-SYN+] | Composition synthesis to the extent of layer n-1 |
| [≈SYN+] | Composition synthesis to the extent of last layer |
| [REP] | Replace |
| [REP PLA] | Replace at a specified layer |
| [REP:] | Replace at primitive layer |
| [REP.] | Replace at event layer |
| [REP-] | Replace at relation layer |
| [REP'] | Replace at idea layer |
| [REP,] | Replace at phrase layer |
| [REP dPLA] | Replace at differential layer |
| [REP=] | Replace at same layer |
| [REP-] | Replace at layer n-1 |
| [SOR] | Sort |
| [SOR≠] | Sort by layers |
| [SOR≈] | Sort by content |
| [=SOR≈] | Sort by content to the extent of same layer |
| [-SOR≈] | Sort by content to the extent of layer n-1 |
| [≈SOR≈] | Sort by content to the extent of last layer |

4. Qualifiers

4.0. Introduction

A qualifier is made by one or three roles, each role being specified by a determinator.

4.1. Difference vocabulary

4.1.1. Roles

dROL
(So|De|Tr|Fl) role
role choices ("Fl" denotes the whole flow)

4.1.2. Determinators

dDET
(dGES|dMER) role determination
categories of determinators

dGES
(dRAT|dPOP) gestalt determination : defined by flow (Fl) *or* by
(So|De|Tr)
gestalt classes

dRAT
(W|X|Y|Z) rationality range
rationality class choices

dPOP
(V|J|P) population range
population class choices

dMER
(dI|dF|dOM|dO|dM) mereologic determination : defined only by
(So|De|Tr)
mereologic classes

dI
(I|E|F) existential range
existential class choices

dF
(O|M) cognitive pattern range
cognitive pattern choices

dOM
(U|A|S|B|T) elementary range
elementary class choices

dO
(U|A) polar range
polar class choices

dM
(S|B|T) ternary range
ternary class choices

! Used when no determination for a role (the role player stays the same or is determined by a diagonal operator)

Standard order for choices

| | |
|-------------------|---------------------|
| roles | (So / De / Tr) |
| population | (P / X / V) |
| rationality | (Z / Y / X / W) |
| existence | (I / E / F) |
| cognitive pattern | (O / M) |
| primitive | (U / A / S / B / T) |

4.2. Qualifiers structures

4.2.1. General Form of Elementary Network Generators

An ENG is composed of a knot and a qualifier. The qualifier can be "unplayed" like in the case of "sort by layers" [SOR≠] that has no qualifier.

The knots [KNO] and qualifiers [QUA] are represented between square brackets.

The qualifiers and the knots are connected by a pound "#" to build an ENG.

ENG == [KNO]#[QUA]

4.2.2. General Form of Qualifiers

A qualifier is basically made of ...

1) roles, belonging to dROL

2) roles determinations, belonging to dDET (see below).

Each role is followed by its determination between parenthesis.

The determined roles [So(...) / De(...) / Tr(...)] are separated and ordered by slashes "/"

Two roles can also be separated by an inverse slash "\" in case of inversion

[REP]#[So\De]

Inside the parenthesis, the choices of the determinations may be ordered by slashes "/"

So(dF: O/M)

If the choice order is omitted, the standard order (see above) is applied.

Roles can always be nested in roles, like in : [So(...) / De(De(...)) / Tr(...)]

"De(De(...))" means "the destination of the destination".

4.2.3. Qualifiers templates for various knots

There are seven templates numbered from (1) to (7)

[ANA=] | [ANA-] | [SYN=]

For ([ANA=] | [ANA-] | [SYN=]) the qualifiers operators can have only two forms:

- (1) [KNO]#[Fl(dGES)]
- (2) [KNO]#[So(dDET) / De(dDET) / Tr(dDET)]

[SYN+]

For [SYN+], the two preceding forms are correct, plus the following one (3)

- (3) [SYN+]#[So(...) / De(...) / Tr(...)]

In (3), the roles (So|De|Tr) may be determined by one choice (I|E|F|O|M|U|A|S|B|T) instead of one range (dDET).

In this case, the class corresponding to the choice (at layer n + 1) is given to the selected role in output.

[SOR≈]

There are only two templates of qualifiers for sorting by content [SOR≈]

- (1) [SOR≈]#[Fl(dGES)]

- (4) [SOR≈]#[So|De|Tr (dDET)]

In the template (4), only *one* of the three flow roles can be selected.

[SOR≠]

- (5) There is *no qualifier* in case of sorting by layers [SOR≠]

[REP]

There are two forms for the "replace" knot [REP]

- (6) [REP]#[So\De] or [REP]#[So\Tr] or [REP]#[Tr\De]

This is the "role inversion" form.

- (7) [REP]#[So(...) / De (...) / Tr (...)]

In (7), the roles can be determined by :

- one dMER range
- one dMER choice
- a list of choices taken in the dMER ranges
- a "W"
- a "!"

5. Detail of Elementary Network Generators

5.1. [ANA=]#[QUA] taxonomic analysis

The taxonomic analysis ENG produces subsets (at layer n) of the input set (at layer n).

The output sets get automatically the same layer as the input set.

[ANA=] rule1: For dI, the result of the analysis can be only (E|F) i.e. "I" is excluded.

[ANA=] rule2: For dGES, *I:I:I.** , *I~~~**, *I~~~~', *I~~~~, ** and *I~~~~_** are excluded from X and J

Without these rules (1 and 2), the analysis would be endless in case of looped operation.

Example 7

*I:I:S:.** → [≈ANA=]#[Fl(dRAT)]}
 → "Z" = *F:F:S:.** /
 → "Y" = *E:I:S:.** / *E:F:S:.** / *F:E:S:.** / *E:E:S:.** / *E:F:S:.*
 → "X" = *I:F:S:.** / *F:I:S:.** / *I:E:S:.**

Example 8

*I:F:S:.** → [≈ANA=]#[So(dI) / De(dF) / Tr(!)]
 → "So(E)" = *E:F:S:.**
 → "So(F)" = *F:F:S:.**
 → "De(O)" = *I:O:S:.**
 → "De(M)" = *I:M:S:.**

Example 9

*I:F:S:.. A:O:M:.. -** → [≈ANA=]#[So(dI) / De(dI) / Tr(!)]
 → "So(E)" = *E:F:S:.. A:O:M:.. -**
 → "So(F)" = *F:F:S:.. A:O:M:.. -**

Remark : " De(dI) " means that the taxonomic existential analysis must be done for the destination role, but the operation cannot be performed because it is already done.

Example 10

*I:F:S:. A:O:M:. -** → [≈ANA=]#[So(dI) / De(dOM) / Tr(!)]
 → "So(E)" = *E:F:S:. A:O:M:. -**
 → "So(F)" = *F:F:S:. A:O:M:. -**
 → "De(U)" = *F:F:S:. A:U:M:. -**
 → "De(A)" = *F:F:S:. A:A:M:. -**

5.2. [ANA-]#[QUA] composition analysis

The composition analysis ENG produces embedded components (of layer n-1) of the input set (at layer n).

Example 11

*F:F:S:.M:O:M:.-U:O:A:.-'*** → [≈ANA-]#[So(!)/De(!)/Tr(!)]
 → "So(!)" = *F:F:S:.M:O:M:.-**
 → "De(!)" = *U:O:A:.-**
 → "Tr(!)" = *E:.-**

Example 12

*F:F:S:.M:O:M:.-U:O:A:.-'*** → [≈ANA-]#[So(dF)/De(!)/Tr(!)]
 → "So(O)" = *O:F:S:.M:O:M:.-**
 → "So(M)" = *M:F:S:. M:O:M:.-**
 → "De(!)" = *U:O:A:.-**
 → "Tr(!)" = *E:.-**

5.3. [SYN=]#[QUA] taxonomic synthesis

The taxonomic synthesis ENG produces "including sets" (at layer n) of the input set of layer n.

Example 13

*U:S:.E:E:S:.-** → [≈SYN=]#[So(dRAT) / De(dRAT) / Tr (!)]
 → "So(Z)" = *(F:|F:F:|F:F:F:|F:I:E:|F:F:I:)(E:E:S:)-**
 → "De(Y)" = *(U:S:)(E:E:F:|E:F:E:|E:F:F:|F:E:F:|E:F:I:|E:I:F:)-**

Example 14

U:S:. → [≈SYN=]#[So(dF)/De(dF)/Tr(dI)]
 →"So(dF)" = *O:S:.*
 →"De(dF)" = *U:M:.*
 →"Tr(dI)" = *U:S:I:.*

There is no result when the operation is not possible or already done, like in the following examples:

Example 15

U:S:. → [≈SYN=]#[So(dF)/De(dF)/Tr(dF)]
 →"So(dF)" = *O:S:.*
 →"De(dF)" = *U:M:.*

Example 16

U:S:I:. → [≈SYN=]#[So(dF)/De(dF)/Tr(dF)]
 →"So#dF" = *O:S:I:.*
 →"De#dF" = *U:M:I:.*

5.4. [SYN+]#[QUA] composition synthesis

The composition synthesis ENG produces sets that are embedding (at layer n+1) the input set (at layer n).

| | |
|-------------------------------------|---------------------------------------------------------------------|
| [SYN+]#[So(!)...] | the input set is reproduced for the role |
| [SYN+]#[So(dGES)...] | the class variation is produced for the role |
| [SYN+]#[So(dMER)...] | the class variation is produced for the role |
| [SYN+]#[So(I E F O M U A S B T)...] | the corresponding class choice (at layer n +1) is given to the role |

Example 17

U:F:S:. → [≈SYN+]#[So(!) / De (!) / Tr (dF)]
 →"Tr(O)" = * U:F:S:. U:F:S:. O~.- **
 →"Tr(M)" = * U:F:S:. U:F:S:. M~.- **

Example 18

U:F:S:. → [≈SYN+]#[So(I) / De (!) / Tr (dI)]
 →"Tr(E)" = ** I~. U:F:S:. E~- **
 →"Tr(F)" = ** I~. U:F:S:. F~- **

Example 19

U:F:S:. → [≈SYN+]#[So(I) / De (!) / Tr (dRAT)]
 →"Tr(Z)" = ** I~. U:F:S:. (F:I:E: | F:E:E: | F:F:I: | F:F:E: | F:F:F:)- **
 →"Tr(Y)" = ** I~. U:F:S:. (E:I:F: | E:E:F: | E:F:I: | E:F:E: | E:F:F: | F:E:F:)- **
 →"Tr(X)" = ** I~. U:F:S:. ((I:I:E:) | (I:E:I:) | (I:E:E:) | (E:I:I:) | (E:I:E:) | (E:E:I:)|
 | (I:I:F:) | (I:F:I:) | (I:F:E:) | (I:F:F:) | (F:I:I:) | (F:I:F:) | (F:E:I:))- **
 →"Tr(W)" = ** I~. U:F:S:. E:- **

5.5. [SOR≠] sort by layers

This is the simplest of all the ENG. It has no qualifier.

Example 20

*M:** / *M:O:** / *O:M:** / *y.** / * s.e.-b.u.-wa.e.-'** / *s.e.-** / *b.u.-** →
 [SOR≠]
 → "PLA : " = *M:**
 → "PLA . " = *O:M:** / *y.** / *M:O:**
 → "PLA - " = *s.e.-** / *b.u.-**
 → "PLA ' " = *s.e.-b.u.-wa.e.-'**

5.6. [SOR≈]#[QUA] sort by content

The ENG distributes the input sets into "folders" corresponding to the range of differences specified in the QUA operator.

Example 21

*M:** / *S:O:B:** / *I:E:S:** / *E:E:U:** / * U:F:S:. U:F:S:. O~- **
 →[SOR≈]#[So(dI:I/E/F)]
 →"So(I) Class" = *I:E:S:**
 →"So(E) Class" = *E:E:U:**
 →"So(F) Class" = *M:** / * U:F:S:. U:F:S:. O~- ** / *S:O:B:**

5.7. [REP]#[QUA] Replace

Example 22

*s.e.-b.u.-wa.e.-'** → [REP=]#[So\De] → *b.u.-s.e.-wa.e.-'**

Example 23

*s.e.-b.u.-wa.e.-'** → [REP-]#[So\De] → *e.s.-u.b.-e.wa.-'**

Example 24

*s.e.-b.u.-wa.e.-'** → [REP:]#[So(dM) / De(!) / Tr(!)]
 → "So(B)" *first* source player primitive falling under "M" = *k.e.-b.u.-wa.e.-'**
 → "So(T)" *first* source player primitive falling under "M" = *d.e.-b.u.-wa.e.-'**
 → "So(B)" *second* source player primitive falling under "M" = *s.e.-m.u.-wa.e.-'**
 → "So(T)" *second* source player primitive falling under "M" = *s.e.-f.u.-wa.e.-'**

Example 25

*s.e.-b.u.-wa.e.-'** → [REP=]#[So(dM) / De(F) / Tr(dO)]
 → "So(B)" = *k.e.-b.u.-wa.e.-'**
 → "So(T)" = *d.e.-b.u.-wa.e.-'**
 → "De(E)" = *s.e.-E:B:.u.-wa.e.-'**
 → "Tr(A)" = *s.e.-b.u.-we.e.-'**

5.8. Provisional notes on semantic steps

After the workshop discussion of the 22 march 2008...

5.8.1. Steps

I propose to name *step* the link between one input and one output of ANA, SYN and REP ENGs.

The input category is the *departure* position (in the semantic space) of the step.

The output category is the *arrival* position of the step.

The ENG operation between a departure and an arrival is a *move*.

So a step is defined by 1) a departure category, 2) an arrival category and 3) an ENG-defined move between the departure and the arrival.

5.8.2. Steps generation

As can be seen from the examples, given one input, ENGs with ANA, SYN and REP threads produce *several* outputs and, therefore, several steps with the same departure and the same ENG, but not the same arrival.

These several outputs from the same input result in a *generation* of steps. In other words, a generation of steps is the set of steps produced by an ENG from one single input category.

The steps from the same generation can be ordered (one could give them numbers). In terms of trees, the unique departure is a *root* and the multiple arrivals are the *leaves*. So the ENG produces a generation of leaves from a root.

5.8.3. Conceptual distinction between input / output transformation and network generation

The ENGs with ANA, SYN and REP threads first create an output from an input, then draw a step between the input and the output categories.

One step at a time, they draw generations of steps that will be part of larger networks.

5.8.4. Special features of the SORTING thread

5.8.4.1. Difference between the SOR and (ANA | SYN | REP) threads

The SOR thread is different from the others.

The main difference is that the category in input is not "transformed" by the sorting operation. It stays exactly the same.

Rather, the input categories are distributed or classified into several (numbered) folders.

If several sorting-thread-ENGs work in sequence, the result is a distribution of the input categories into a hierarchy (a tree) of folders.

Notwithstanding its difference, the sorting ENGs, still build networks, paths and steps (and therefore, ultimately, motion) between IEMML-coded categories.

5.8.4.2. Tree-exploration paths

As the categories are arranged into trees (hierarchies of folders) by sorting ENGs, the path between two categories can be reduced to a series of steps between the nodes of the sorting-thread-tree containing these categories.

5.8.4.3. Linear paths

In the sorting thread trees, the generations *and* the leaves of the same generations can all be numbered. Therefore, considering that there is anyway a standard order between the semantic qualities, the categories arranged into sorting-thread-trees can be automatically ranked - or linearly ordered. In other words, there is always a

linear path crossing one by one all the categories arranged into a sorting-thread-tree.

6. Circuits

6.1. Networking Sequences

Sequence operator ">>" takes the output of the preceding ENG as the input of the following.

$$\text{ENG}_1 \gg \text{ENG}_2 \gg \text{ENG}_3 == \\ [\text{KNO}_1] \# [\text{QUA}_1] \gg [\text{KNO}_2] \# [\text{QUA}_2] \gg [\text{KNO}_3] \# [\text{QUA}_3]$$

When the same knot is used for several ENG in sequence, it is possible to use the following notation "[[]]"

$$[\text{KNO}_1] \# [[\text{QUA}_1] \gg [\text{QUA}_2] \gg [\text{QUA}_3]]$$

6.2. Networking Loops

The loop is noted into braces "{ }"

$$\{\text{ENG}_1 \gg \text{ENG}_2 \gg \text{ENG}_3\} == \\ \{[\text{KNO}_1] \# [\text{QUA}_1] \gg [\text{KNO}_2] \# [\text{QUA}_2] \gg [\text{KNO}_3] \# [\text{QUA}_3]\}$$

$$\{[\text{KNO}_1] \# [[\text{QUA}_1] \gg [\text{QUA}_2] \gg [\text{QUA}_3]]\}$$

The "by default" test for the loop is :

IF operation done
THEN stop

IF operation still possible
THEN continue

The "extent" of the knot determines the duration of the loop.

6.3. Parallel Networking

The output of an ENG can be processed by two (or more) different ENG. In this case, the networking process is "parallel".

Parallel networking is noted this way:

ENG₁

>1> ENG₂

>2> ENG₃

In this scheme, ENG₂ and ENG₃ take in parallel the outputs of ENG₁

6.4. First circuit example : the $\{M:\}M:.e.-M!.A:\{M:\}.-wa.e.-'$ ** table

Example 26

$\{M:\}M:.e.-M!.A:\{M:\}.-wa.e.-'$ ** "paradigm" or table of abilities, skills.

Input : $\{M:\}M:.e.-M!.A:\{M:\}.-wa.e.-'$ ** →

$[\approx ANA=]\#[[So(dM) / De(!) / Tr(!)]>>[So(dM) / De(!) / Tr(!)]>>[So(!) / De(dM) / Tr(!)]>>[So(!) / De(So(De(dM))) / Tr(!)]$

>1> $[REP:]\#[So(dM) / De(!) / Tr(1)]$

>2> $[REP:]\#[[So(dM) / De(!) / Tr(1)]>> [So(So(De(dM))) / De(!) / Tr(1)]]$

>3> $[REP:]\#[[So(!) / De(dM) / Tr(1)]>> [So(!) / De((So(De(dM))) / Tr(1)]]$

>4> $[REP]\#[So(!) / De(!) / Tr(W)]$

>5> $[\approx ANA-]\#[So(!)/De(!)/Tr(!)]$

Output : the whole "abilities" table with a semantic network for navigation links or value circulation (same column, same row, corresponding *achievements* in the $\{M:\}M:.e.-M!.A:\{M:\}.-'$ ** table, etc.)

Here is a decomposition of the result of the first ENG

$\{M:\}M:.e.-M!.A:\{M:\}.-wa.e.-'$ ** → $[\approx ANA=]\#[So(dM) / De(!) / Tr(!)]$

→ S:M:.e.-M!.A:S:.-wa.e.-' (sign related skills)

→ B:M:.e.-M!.A:B:.-wa.e.-' (being related skills)

→ T:M:.e.-M!.A:T:.-wa.e.-' (thing related skills)

Note that the diagonal operator is respected (the first primitive of the source relation and the last primitive of the destination relation stay the same).

At the end of the sequence $[\approx ANA=]\#[[So(dM) / De(!) / Tr(!)]>>[So(dM) / De(!) / Tr(!)]>>[So(!) / De(dM) / Tr(!)]>>[So(!) / De(So(De(dM))) / Tr(!)]$ the whole table is generated. Note that the "analysis" links can be navigated from the analysed categories to their containers. Note also the nested role in the last qualifier.

| | | {M;}M;.e.-M!.A:{M;}.-wa.e.-' | | | | | | | | | |
|--------------------------------------------------|---------------------------------------|--------------------------------------------------------|--------------------------------------------------------------|-----------------------------------------------------|-----------------------------------------------------|--------------------------------------------------------|--------------------------------------------------------|------------------------------------------------------|-----------------------------------------------------|---------------------------------------------------|--|
| | | abilities | | | | | | | | | |
| | | {M;}M;.e.-S!M;.A:{M;}.-wa.e.-' | | | {M;}M;.e.-B!M;.A:{M;}.-wa.e.-' | | | | {M;}M;.e.-T!M;.A:{M;}.-wa.e.-' | | |
| | | {M;}M;.e.-s.A:{M;}.-wa.e.-' | {M;}M;.e.-b.A:{M;}.-wa.e.-' | {M;}M;.e.-t.A:{M;}.-wa.e.-' | {M;}M;.e.-k.A:{M;}.-wa.e.-' | {M;}M;.e.-m.A:{M;}.-wa.e.-' | {M;}M;.e.-n.A:{M;}.-wa.e.-' | {M;}M;.e.-d.A:{M;}.-wa.e.-' | {M;}M;.e.-f.A:{M;}.-wa.e.-' | {M;}M;.e.-l.A:{M;}.-wa.e.-' | |
| | | thought related skills | language related skills | memory related skills | society related skills | affect related skills | world related skills | truth related skills | body related skills | physical space related skills | |
| S!M;.e.-M!.A:S!.-wa.e.-' sign-related skills | S:S;.e.-M!.A:S!.-wa.e.-' grammar | s.e.-s.u.-wa.e.-' to draw / to paint | s.e.-b.u.-wa.e.-' speak a second language | s.e.-t.u.-wa.e.-' to count / to calculate | s.e.-k.u.-wa.e.-' to read / to write | s.e.-m.u.-wa.e.-' to play | s.e.-n.u.-wa.e.-' vocabulary | s.e.-d.u.-wa.e.-' information search | s.e.-f.u.-wa.e.-' playing music / singing | s.e.-l.u.-wa.e.-' reading natural signs | |
| | S:B;.e.-M!.A:S!.-wa.e.-' dialectic | b.e.-s.u.-wa.e.-' visual refinement | b.e.-b.u.-wa.e.-' interpreter / guide | b.e.-t.u.-wa.e.-' logic / programming | b.e.-k.u.-wa.e.-' personal interpretation | b.e.-m.u.-wa.e.-' culture in performing arts | b.e.-n.u.-wa.e.-' literary erudition | b.e.-d.u.-wa.e.-' documentation | b.e.-f.u.-wa.e.-' musical culture | b.e.-l.u.-wa.e.-' scientific culture | |
| | S:T;.e.-M!.A:S!.-wa.e.-' rhetoric | t.e.-s.u.-wa.e.-' image creation / design | t.e.-b.u.-wa.e.-' eloquence | t.e.-t.u.-wa.e.-' software design | t.e.-k.u.-wa.e.-' symbol creation | t.e.-m.u.-wa.e.-' stage / artistic direction | t.e.-n.u.-wa.e.-' literary writing | t.e.-d.u.-wa.e.-' information architecture | t.e.-f.u.-wa.e.-' composing music | t.e.-l.u.-wa.e.-' scientific research | |
| B!M;.e.-M!.A:B!.-wa.e.-' being-related skills | B:S;.e.-M!.A:B!.-wa.e.-' grammar | k.e.-s.a.-wa.e.-' understanding oneself | k.e.-b.a.-wa.e.-' speaking sincerely | k.e.-t.a.-wa.e.-' memory training | k.e.-k.a.-wa.e.-' following a discipline | k.e.-m.a.-wa.e.-' being friend with oneself | k.e.-n.a.-wa.e.-' self-evaluation | k.e.-d.a.-wa.e.-' acknowledging the facts | k.e.-f.a.-wa.e.-' taking care of oneself | k.e.-l.a.-wa.e.-' self-protection | |
| | B:B;.e.-M!.A:B!.-wa.e.-' dialectic | m.e.-s.a.-wa.e.-' understanding others | m.e.-b.a.-wa.e.-' wise words | m.e.-t.a.-wa.e.-' fidelity | m.e.-k.a.-wa.e.-' collaborative attitude | m.e.-m.a.-wa.e.-' solicitude | m.e.-n.a.-wa.e.-' discernment | m.e.-d.a.-wa.e.-' truth seeking | m.e.-f.a.-wa.e.-' to provide welfare | m.e.-l.a.-wa.e.-' law abiding | |
| | B:T;.e.-M!.A:B!.-wa.e.-' rhetoric | n.e.-s.a.-wa.e.-' to give meaning to a group | n.e.-b.a.-wa.e.-' representing a group | n.e.-t.a.-wa.e.-' initiating a tradition | n.e.-k.a.-wa.e.-' being an example | n.e.-m.a.-wa.e.-' inspiring harmony | n.e.-n.a.-wa.e.-' forging values | n.e.-d.a.-wa.e.-' releasing the truth | n.e.-f.a.-wa.e.-' to heal | n.e.-l.a.-wa.e.-' protecting a group | |
| T!M;.e.-M!.A:T!.-wa.e.-' thing-related skills | T:S;.e.-M!.A:T!.-wa.e.-' grammar | d.e.-s.i.-wa.e.-' body / mind disciplines | d.e.-b.i.-wa.e.-' dancing | d.e.-t.i.-wa.e.-' driving | d.e.-k.i.-wa.e.-' team sports | d.e.-m.i.-wa.e.-' subtlety of the senses | d.e.-n.i.-wa.e.-' extreme sports | d.e.-d.i.-wa.e.-' shooting / aiming sports | d.e.-f.i.-wa.e.-' athletics | d.e.-l.i.-wa.e.-' martial arts | |
| | T:B;.e.-M!.A:T!.-wa.e.-' dialectic | f.e.-s.i.-wa.e.-' practical ingenuity | f.e.-b.i.-wa.e.-' communication techniques mastery | f.e.-t.i.-wa.e.-' maintenance | f.e.-k.i.-wa.e.-' financial autonomy | f.e.-m.i.-wa.e.-' esthetics / fashion | f.e.-n.i.-wa.e.-' production control mastery | f.e.-d.i.-wa.e.-' control / measure | f.e.-f.i.-wa.e.-' surgery / body care | f.e.-l.i.-wa.e.-' construction mastery | |
| | T:T;.e.-M!.A:T!.-wa.e.-' rhetoric | l.e.-s.i.-wa.e.-' interconnecting systems | l.e.-b.i.-wa.e.-' computer engineering | l.e.-t.i.-wa.e.-' vehicle design | l.e.-k.i.-wa.e.-' commercial strategy | l.e.-m.i.-wa.e.-' artist | l.e.-n.i.-wa.e.-' tools / machine design | l.e.-d.i.-wa.e.-' metrology | l.e.-f.i.-wa.e.-' biotechnology | l.e.-l.i.-wa.e.-' architect | |

The four "parallel" sequences following the first sequence are about transversal navigation in the table with corresponding achievements in another table. The fifth one produces the analytical decomposition into role players.

6.5. Second circuit example: "sort all" standard order

Example 27

```
[SOR≠]>>
[=SOR≈]# [Fl(dRAT)]>>
{[≈SOR≈]# [[So(dI)]>>[So(dF)]>>[So(dOM)]
           [De(dI)]>>[De(dF)]>>[De(dOM)]
           [Tr(dI)]>>[Tr(dF)]>>[Tr(dOM)]}}
```

This circuit can sort the whole semantic space. It defines the "standard order".

The path between 2 categories inside the tree produced by the standard order circuit can be used to measure "semantic differences".

Any other *circuit* can be used as a *semantic difference measurement tool*.

7. Conclusion : semantic networking as the essence of semantic engineering

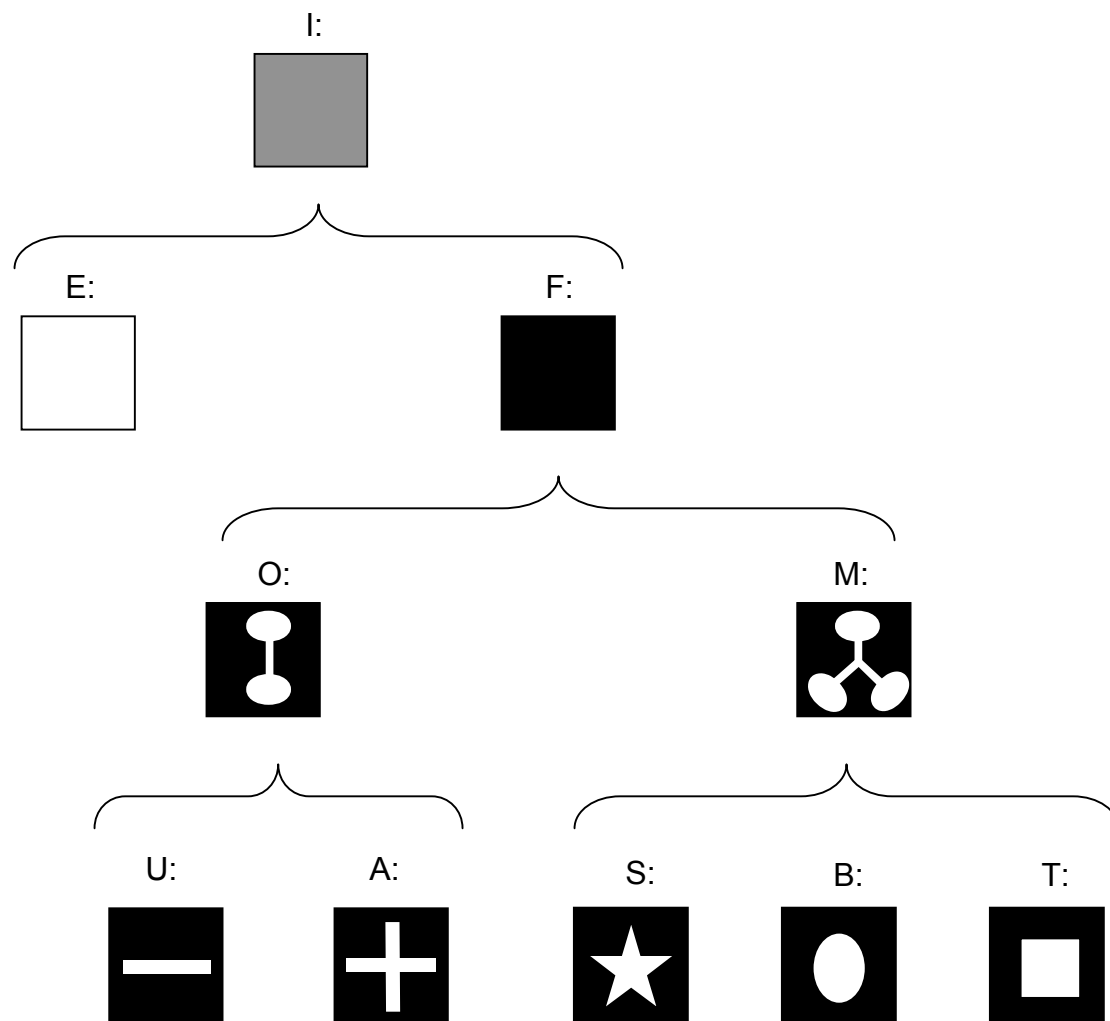
This text describes a semantic networking system for IEMML databases. This semantic networking/navigation system is articulated, for the current state of the formalization, by four combinable functions (sort, replace, analyze and synthetize) processing IEMML syntactic differences.

Attribution of natural language descriptors to IEMML categories *includes* a definition of the semantic networks imposed on these categories.

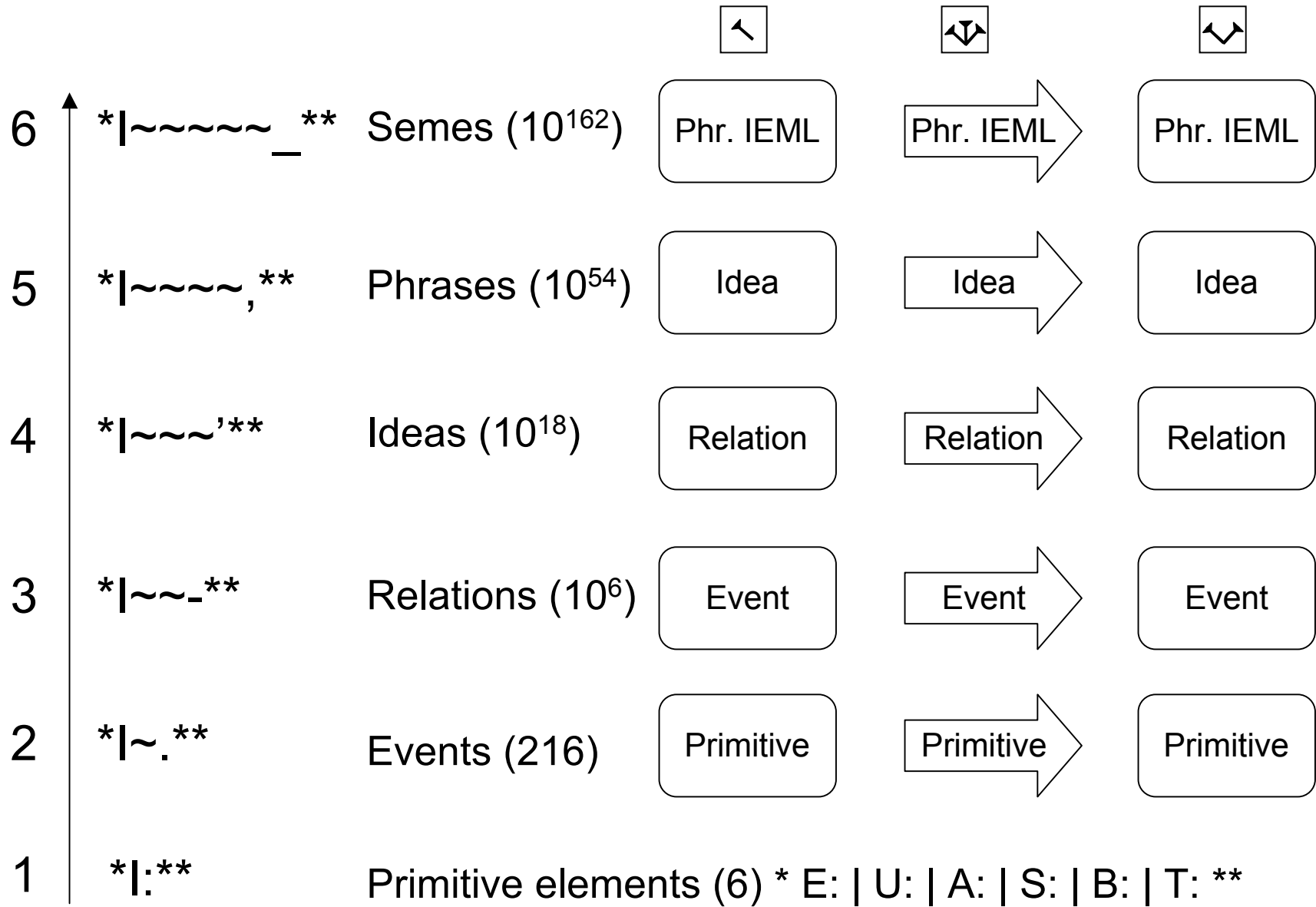
A set of categories should be described (interpreted by natural language descriptors) in order to *maximize* the relevant semantic links that ENG circuits can automatically wave on it.

Building and evolving semantic engines (sets of circuits | sets of categories) finely adapted to the augmentation of human communities collective intelligence is the essence of semantic engineering.

IEML PRIMITIVE CATEGORIES

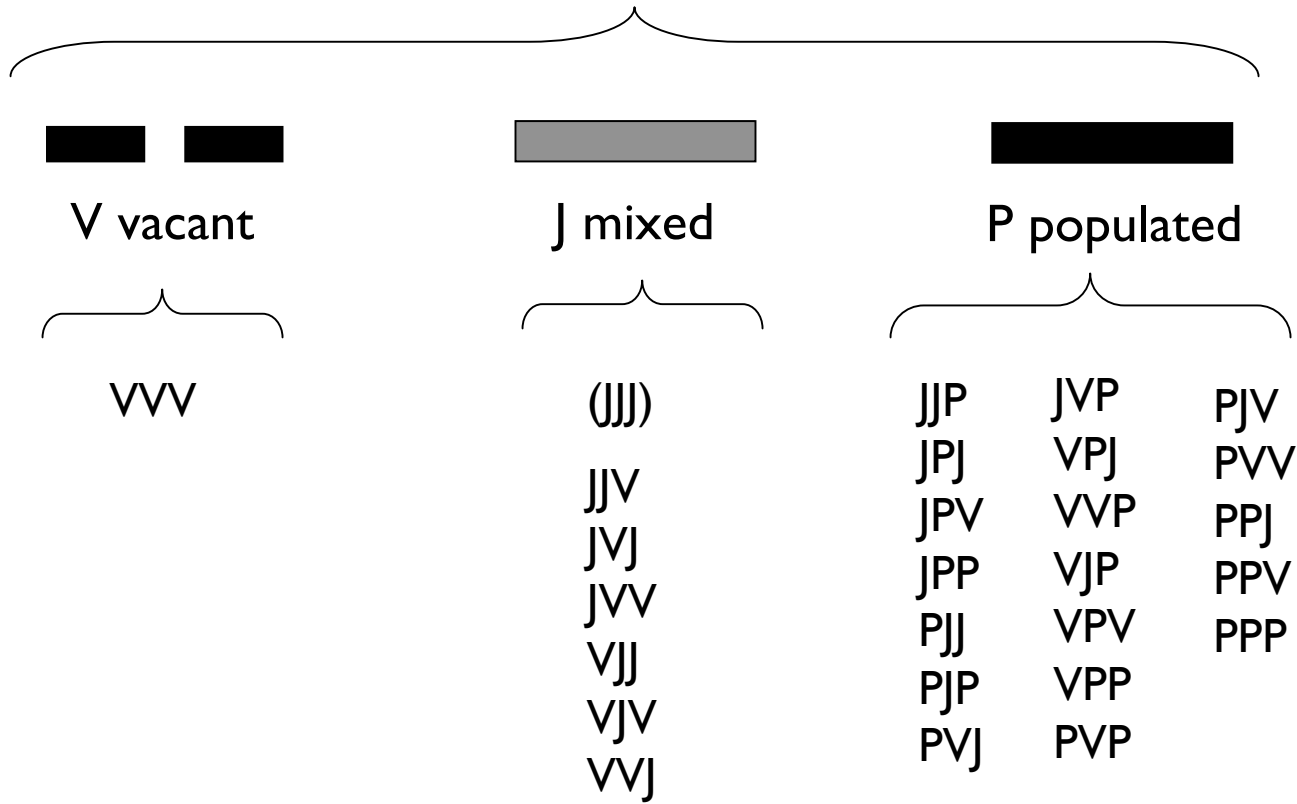


Layers of IEML Semantic Space





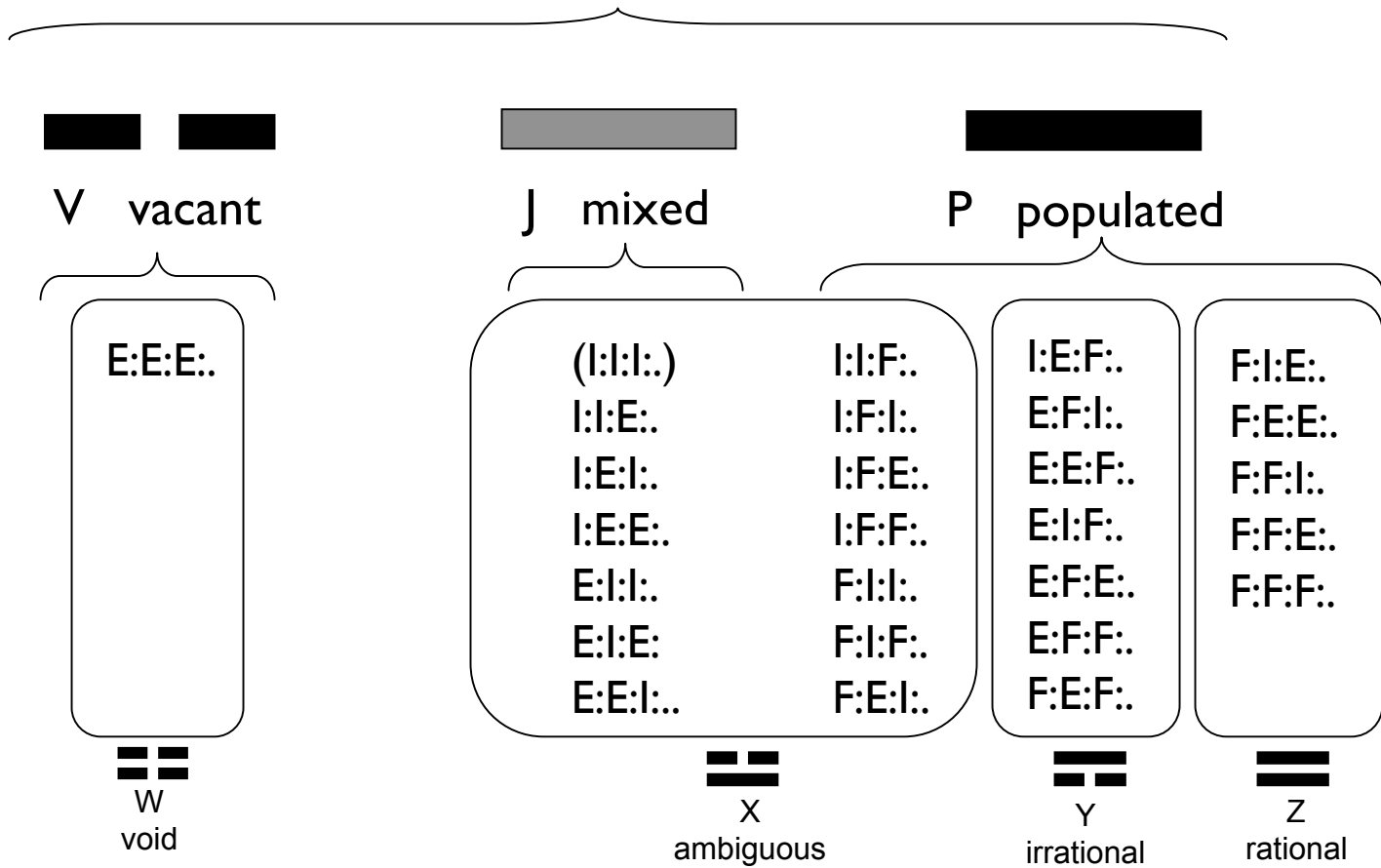
dPOP
population differences



EVENTS CLASSES



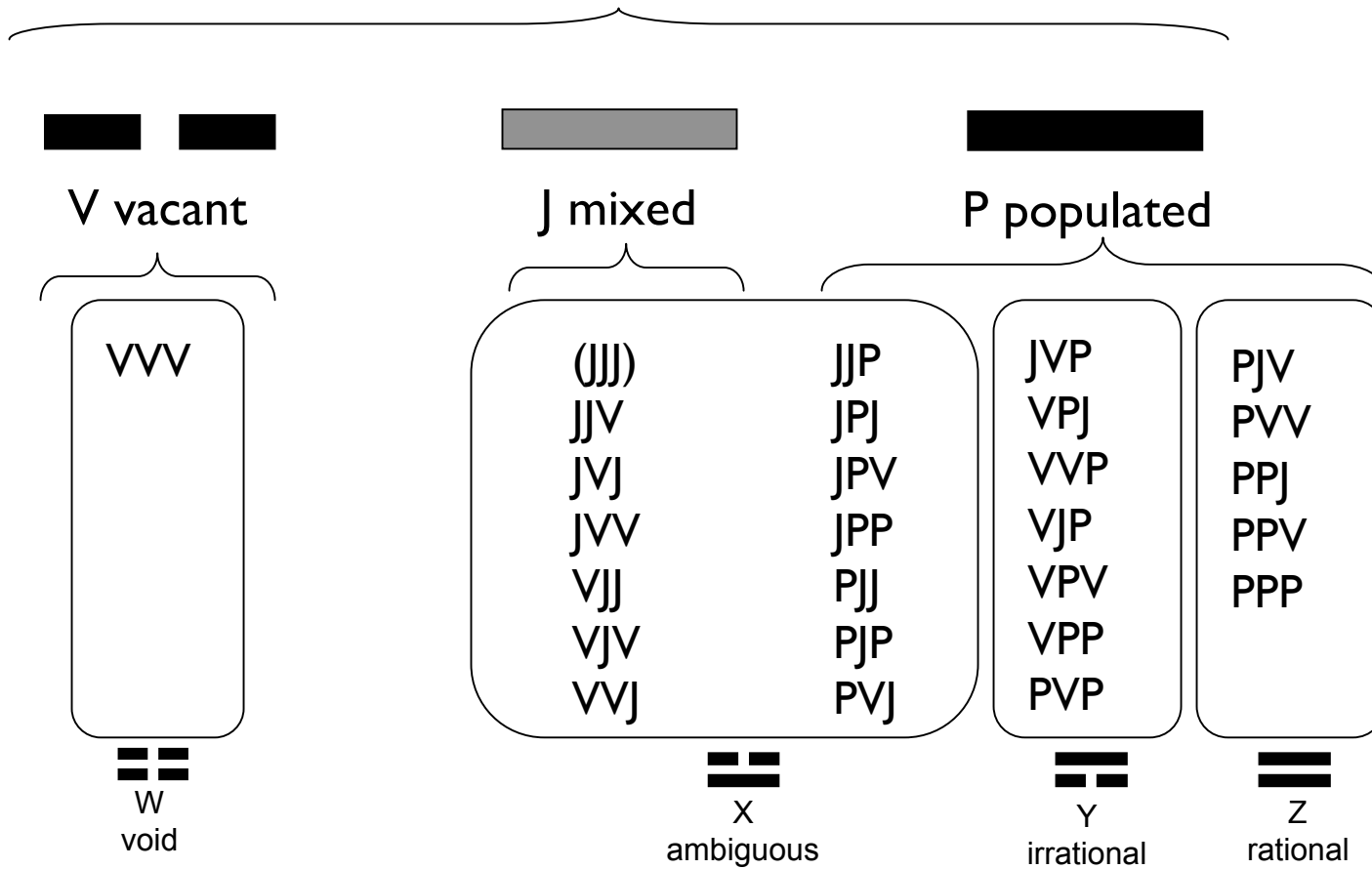
dPOP
population differences



POPULATION / RATIONALITY RELATIONSHIP



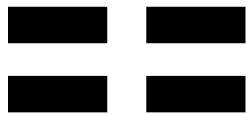
dPOP
population differences



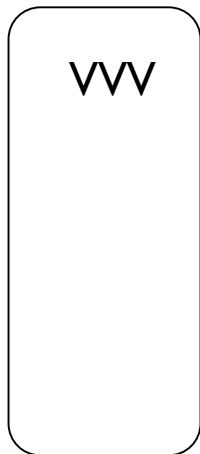


FLOW RATIONALITY

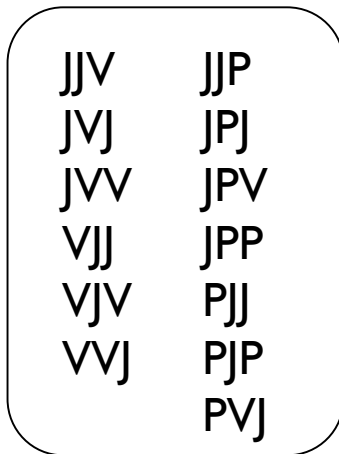
dRAT / rationality differences



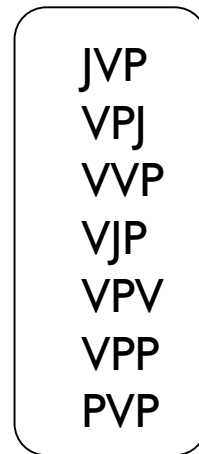
W
void



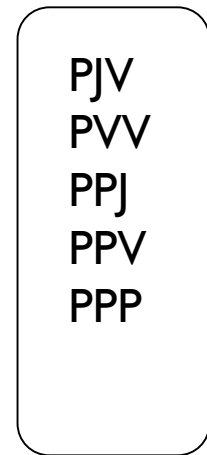
X
ambiguous



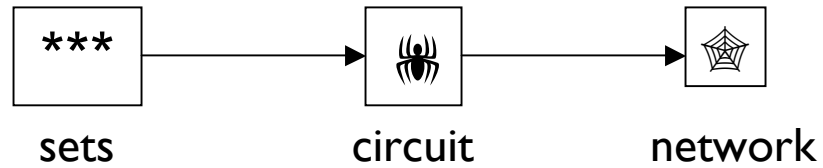
Y
irrational



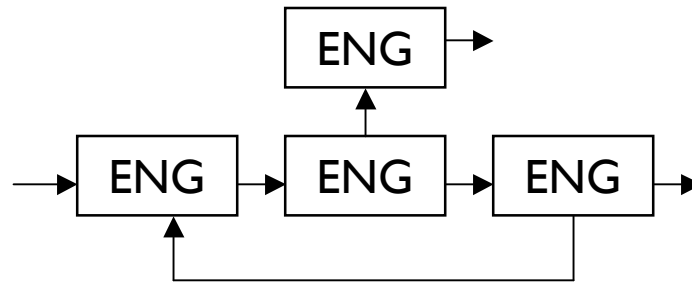
Z
rational



Semantic Networking



circuit =
composition of ENG



elementary network
generator



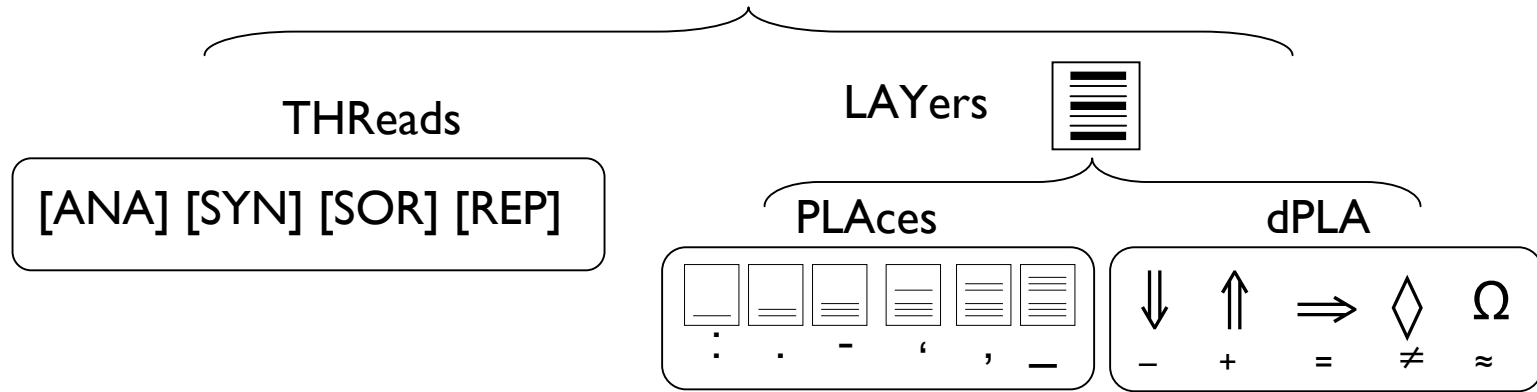
ENG = knot +
qualifier

[KNO]#[QUA]

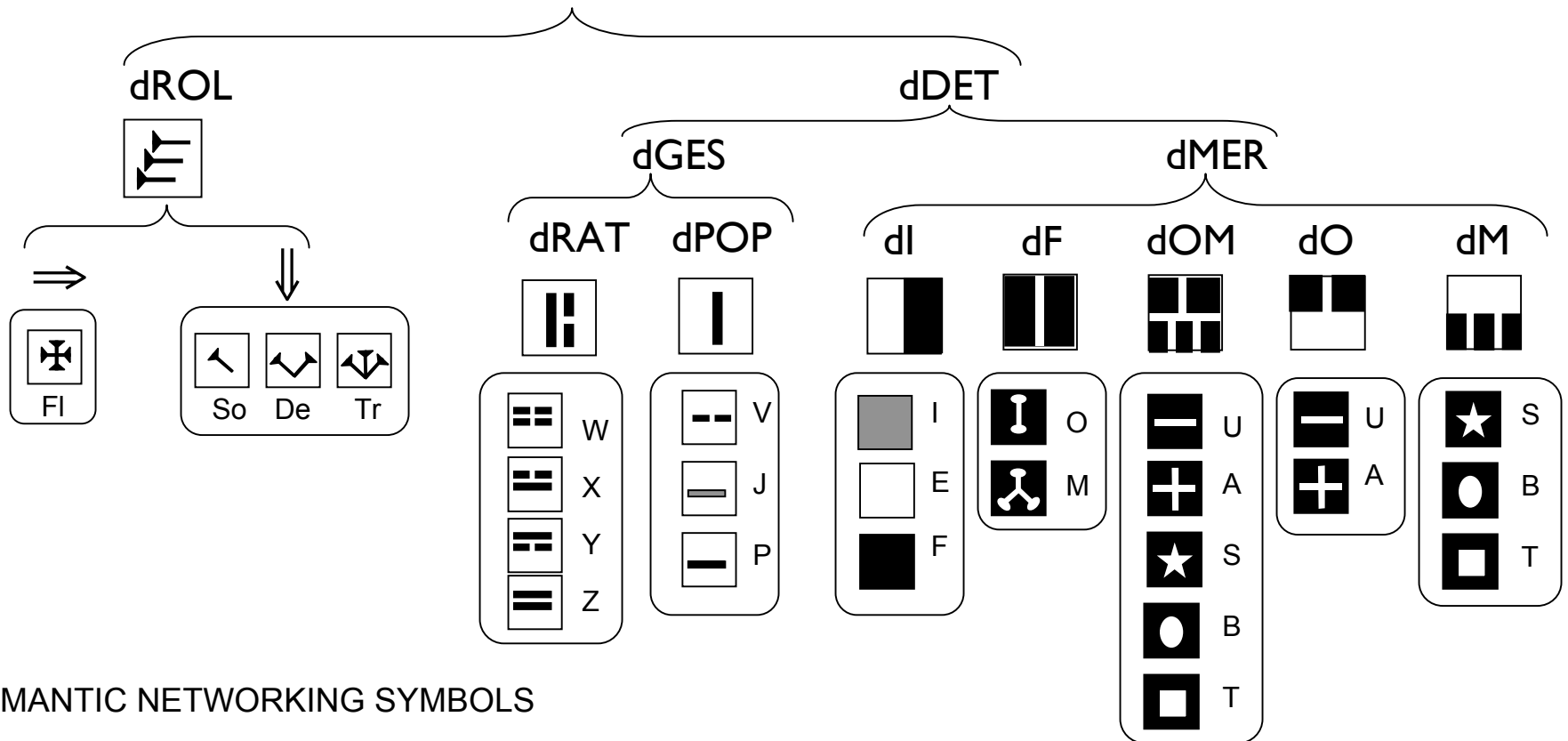
knot and qualifier
composition

[LAY THR LAY]#[dROL(dDET)]

KNOts



QUALifiers



SEMANTIC NETWORKING SYMBOLS

IEML DATABASE

Semantic motion design: Semantic Circuits

Navigation interface: Semantic Networks

IEML dictionary

IEML-tagged data

