# ~~Thought~~ Synthesis of Semantic Structures

## Particle model for thought synthesis

### Thought and Particle Signatures

Let P be a thought particle. A thought particle can be an *object particle* denoted with V (corresponding to a vertex in the thought DAG) or a *connecting particle* denoted with A (corresponding to an arc in the thought DAG). The object particle is also known as *V-particle* and the connecting particle is known as *A-particle*.

When parsing a new thought we construct the thought path which is also known as the *main thought path*:

Fig 1

When a new thought is introduced to the environment various processes take place spontaneously.

In the process *Association* the thought path is augmented to a DAG which may look like:

Fig 2

|\_\_\_\_\_\_\_|| |

| |\_\_

|\_\_

Compound object particle is a subset of a thought path starting with object particle followed by connecting particle , etc, and ending with object particle . It is denoted by square brackets in which the sub-particles are enclosed:

[ Vstart A1 V1 … Vend ]

Thought radical is a piece of a thought which starts and/or ends with connecting particle. It is denoted by braces in which the sub-particles are enclosed:

{ Astart V1 … Aend }

The radical can be considered a generalized connecting particle analogous to the compound object particle.

Signature of a thought particle is a matrix S[i,j] where 0 < i < m, 0 < j < n

Rule for calculation of the thought signature of a compound thought particle

Vcomp = [ Vstart A1 V2 … Vend ]

Sig(Vstart) = Sstart[i,j]

Sig(A1) = S1[i,j]

Sig(V2) = S2[I,j]

Sig([Vstart A1 V2]) = (Sig(Vstart)) + (Sig(V2)) which is also mxn matrix

The last line can be written with the following syntax using radicals:

Sig([Vstart A1 V2]) = Sig({Vstart A1}) + Sig({A1 V2})

Obviously the signature of every compound thought particle is mxn matrix.

Every connecting particle signature encodes the operation LA which will be applied to the object particle on the left and the operation RA which will be applied to the object article on the right. Those operations LA and RA will preserve the original information contained in the signature of object particle which is being operated on and will add additional information pertaining to the link particle to those.

Arcs which are not on the main thought path and which connect V-particles on the main thought path are modeled via A-particles. The latter have a special property turned on - `*remote`* of type *`link`* for arcs connecting V-particle on the main thought path and a V-particle in a different thought. The property *secondary* of type *`link`* is turned on for A-particle which connects two V-particles on the main thought path. We write this statement as:

Prop(Ak*,`link`*,*`secondary`*)=true; Prop(Ak+1,*`link`*,*`remote`*)=true

When computing the signature of a compound particle P with secondary path given for example with:

P = [ V1 A1 V2 A3 V3 ]

|\_\_\_\_Ak\_\_\_|

We serialize the two paths

Sig(P) = Sig([V1 A1 V2 A3 V3]) + Sig([V1 Ak V3])

The *secondary* and *remote* link properties will affect the LA and the RA operators in a very specific way. For the case of the secondary link LAk(V1) and RAk(V3) will shift the signatures of V1 and V3 such that the final matrix Sig(V1) + LAk(V1) + Sig(V3) + RAk(V3) will contain information for this secondary link and the particles it connects. Similar expansion holds for the case of remote link. For instance for the subparticle P = [V1 A1 V2 A3 V3] shown on Fig 2 we can write

Sig(P) = Sig([V1 A1 V2 A3 V3]) + Sig([V1 Ak V3]) + Sig([V3 Ak+1 Vother])

Here Sig([V3 Ak+1 Vother]) = LAk+1(V3)+RAk+1(Vother)

For the case of the remote link LAk+1(V3) and RAk+1(Vother) will shift the signatures of V3 and Vother such that the final matrix Sig(V3) + LAk+1(V3) + Sig(Vother) + RAk+1(Vother) will contain information for this remote link as well as the particles it connects including the remote object particle Vother.

Particle properties and encoding them within particle signature

Properties are encoded in specific regions in the particle signature. Each property has a type, a name and value. Property can be thought of as magnifying lens which exposes particular region from the particle signature. Not all properties expose bits from the particle signature which are adjacent and some properties may expose overlapping areas from the signature. In other words

Let us have p defined properties on particle P. Then it is possible that

Prop(P, prop-typei, prop-namei) set-intersect Prop(P, prop-typej, prop-namej) <> empty set for some 0 < i, j < p

Prop(P, prop-type, prop-name) set-intersect B

Where B is a continuous block of bits from the particle signature

The tendency of a particle to establish attraction/repellence toward another particle is achieved through the mediation of *match-seeking* particle (aka M-particle) or *repelling particle* (aka R-particle).

*Match-seeker particle*

Match-seeker particle is denoted with M aka M-particle. We do not usually depict this kind of particles in our graph representations.

Attaches to a specific property with specific prop-name and prop-value. The property value is a vector which is a key allowing the match-seeker particle to be attached to this property. The match-seeker particle exposes a pattern serving as an attraction of an association link particle which would recognize the pattern and attach to the match-seeker. Each match-seeker particle has a property *`charge`* (type *`default`*) with a value indicating the strength of the charge.

M(key-pattern, prop-value)-V(prop-value)

|

A(key-pattern,key-pattern2)-----M(key-pattern2,prop-value2)-Vother(prop-value2)

*Repelling particle*

Repelling particle is denoted with R aka R-particle. We do not usually depict this kind of particles in our graph representations.

Attaches to a specific property with prop-value, a vector allowing a repelling particle to be attached to this property. The repelling particle exposes a pattern serving to repel an *association-link* particle which has a property matching the pattern. Each repelling particle has a property *`charge`* (type *`float`*) with a value indicating the strength of the charge.

To each property of object particle V can be attached multiple M-particles and R-particles.

Synthesizing new properties and removing existing properties

Property removal and addition of new properties can happen at *t-execution*.

The syntax for creating a new property and removing/unregistering an existing one is:

Prop\_new = *register-property*(particle, property\_type, property\_name)

*unregister-property*(particle, property\_type, property\_name)

A property is represented by property matrix P[i,j]=Prop(V, prop\_type, prop\_name), 0 < i < mp; 0 < j < np such that mp < m and np < n. Also for each P[i,j] there exist S[i1,j1] such that P[i,j] = S[j1,j1]. Here S=Sig(V) and 0 < i1 < m, 0 < j1 < n.

*Property affinity*

If two properties of two different particles have affinity for each other there will be either M-particle or R-particle which will attach to those properties.

Adding properties at *t-recombination*

Properties can be cloned from another existing particle when a new particle is created.

At the time of the *initial t-processing,* naked particles are created which are wired-up in *t-path*.

Naked particle – particle which have only couple of properties - a property of type *`text`* and property of type *`association`*. For a naked particle the *t-association* phase has not commenced yet.

In the *t-analysis* phase through a set of sequential optimization steps the naked particles are determined their type (V-particles, A-particles) and are “dressed-up” with properties.

*Thought Association* (*t-association*)

Identify particles with similar signatures and link them through M-particles and R-particles using *similarity-association*.

Example:

**Context:**

*Apostrophe plus s denotes possession.*

*I know John who is my friend. John has children.*

**New thought:**

*Ivan is John’s son.*

<[V101 A101 V102 A102 V103] A103 V104 A104 V105>

<V1 A1 V2 A2 V3 A3 V4 A5 V5 A6 [V6 A7 V8]>

<V3 A8 V9 A9 V10>

Let Vap =[V101 A101 V102 A102 V103]

Prop(Vap, `text`, `all text values`)=list(“Apostrophe plus s”,”’s”)

The new thought contains the following naked particles (N-particles):

<N11 N12 N13 N14>

text(N11) = “Ivan”

Here we utilized the shortcut text(N11) = Prop(N11, `text`, `text`). Similarly:

text(N12)=”is”

text(N13)=”John’s”

text(N14)=”son”

t-association process starts by sending *default-association* particles to attach to each naked particle on the left and on the right as follows:

<N11 DA1 N12 DA2 N13 DA3 N14>

Then a set of M-particles are created attaching to the *key-match:name* property of each naked particle.

Each M-particle which binds to the same *key-match:name* property of the naked particle creates slightly different binding pattern based on the textual representation of the naked particle. Each of those M-particles will attempt to attract *similarity-association* particle (SA-particle) bound to already processed V-particles. For the case of N13 one of those M-particles will attract SA-particle associated with V3 and another one of those M-particles will attract SA-particle associated with Vap.

Details on the algorithm performing similarity association and dressing of naked particles are elaborated and discussed in *Supplement-9-5-29 (page1-page10)*.

*Partially dressed particle*

Naked particle which has been identified by name and its position in the thought path with respect to the other partially dressed particles has been identified

*Property inference in partially dressed particles*

Let us assume we have a partially dressed particle N together with other partially dressed particles N1, N2, … , Nk which are part of the same thought path. We would like to infer as many properties of those as possible.

S – the set of all allowed property-value pairs for particle N

S1, S2, … , Sk are subsets of S

Existing chain C1: N has the pairs (pi,vj) where (i,j) belongs to S1

Existing chain C2: N has the pairs (pi,vj) where (i,j) belongs to S2

. . .

Existing chain Ck: N has the pairs (pi,vj) where (i,j) belongs to Sk

New chain Cnew: N has only (p0,v0) where p0 is the *name* property and v0 is the *name* value.

Goal: find out the set Snew containing the indices (I,j) of all pairs (pi,vi) which belong to N in the new chain Cnew.

One can obtain Snew by constructing Bayesian network from C1, C2, … , Ck and the sets S1, S2, … , Sk.