# Thought Synthesis

## 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 know as the *main thought path*:

Fig 1

< V1 A1 V2 A3 V3 A4 … Ak-1 Vk >

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

< V1 A1 V2 A3 V3 A4 … Ak-1 Vk >

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

| |\_Ak+2\_Vanother

|\_Ak+1\_Vother

Compound object particle is a subset of a thought path starting with object particle Vstart followed by connecting particle A1, etc, and ending with object particle Vend. 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 lense which exposes particular region from the particle signature.

The tendency 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. We do not usually depict those kind of particles in our graph representations.

Attaches to a specific property with type *`key-match`*, with specific name and the 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.

Prop(V,*`key-match`*)=key-value

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

|

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

Repelling particle

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

Attaches to a specific property with type *`key-match`*, name and the value is a vector which allows 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 `*key-match*` property of object particle V can attach either an M-particle or an R-particle.

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)

The 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 have affinity for each other