# Semantic tree operations

Let us introduce the *tree tuple* where represents certain node - semantic particle or a subtree.

Here is a tree tuple factor which encodes uniquely the position of the node in the tree.

The following operations are defined for tree tuple factors:

We have a base of tree tuple factors which are defined as the digits greater than 0 of -nary number system such that . We define an operation `` denoting digit concatenation . Obviously,

for any pair

Note that the latter implies that

for any tuple where

Encoding a complete -ary tree of height with the algebraic notation above:

. Further we will assume that .

In general we have:

where

Obviously, we have at most distinct terms which represent nodes i.e. semantic values.

The expression for the tree also can be written as:

where and are the *node factors* given with . The node values are the values ordered in increasing order of . This order corresponds to *level order traversal* of the -ary tree. Note that with appropriately defined comparison operation `<` we can model different ways of traversing the -ary tree. For instance, if we define `<` as the comparison for the values of we will have ordering which corresponds to the *preorder traversal* of the tree.

*Example*

Peter is Dimitar’s son.

Dimitar’s son has a friend in the neighborhood and his friend’s name is James.

* James is Peter’s friend

Peter is the son of Dimitar.

The son of Dimitar has a friend in the neighborhood and the name of his friend is James.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ \ / \

\_\_\_\_\_\_\_\_

/ | \ / \ \ / \

/ | \ | / \ / \

/ \

T

V

/ \

/ | \

V V

Expressing with the algebraic notation discussed earlier:

which is expanded to:

Expressing with the algebraic notation yields:

+

which is expanded to:

Semantic Aggregation:

\_\_\_\_\_\_\_\_\_

/ \ / \

\_\_\_\_\_ V

/ \ \ / \

V V

Properties and Dependent Properties:

--- o ---

\ | / .. / | \

--- o --- \_

.. / | \ .. / \ | \

|

A particle is defined as a tree-like structure of attached properties as depicted on the figures above.

Let us construct the V particle of the verb “*is*”:

: *key*: ‘particle\_type’, *value*: ‘verb’

: *key*: ‘is\_transitive’, *value*: ‘true’

: *key*: ‘plurality’, *value:* ‘singular’

: *key*: ‘grammatical\_person’, *value:* ‘third\_person’

: *key*: ‘tense’, *value:* ‘present’

: *key*: ‘text’, *value:* ‘is’

o ---

|

---

/ | \

Each property (-particle) is constructed from a key (-particle) and a value (-particle).

Each key and value particles are registered in a particle registry . The purpose of the particle registry is to facilitate learning of new keys and values and use them for parsing new semantic constructs.

Between any two keys and , such that it is obeyed exactly one of the following 3 relations:

Here the operator indicates that there is no parent-child relationship between and , the operator indicates that is a subkey (or a child) of and indicates that is a superkey (or a parent) of .

For instance, in the example above

Semantic properties of a V particle

The semantic properties of a particle are under the key ‘semantic\_properties\_available’. The semantic properties of a particle can be frequently updated depending on the context and upon refining concepts as a result of parsing and analysis of new semantic structures.

In our example we can deduce the following semantic properties:

: *key*: ‘semantic\_properties\_available’, *value:* ‘true’

: *key*: ‘represents\_an\_entity’, *value:* ‘true’

SN.key\_relation(k1, k2): KeyRelation

Enum KeyRelation: {‘keys\_unrelated’, ‘key\_left\_is\_parent\_of\_key\_right’, ‘key\_left\_is\_ancestor\_of\_key\_right’, ‘key\_left\_is\_child\_of\_key\_right’, ‘key\_left\_is\_descendant\_of\_key\_right’}

SN.key\_children(k): set()

SN.key\_descendents(k): set()

How do we add properties to a particle? Either via semantic programming or via inference.

Simple scenario to add property via inference: