

Semantic tree operations

D. Gueorguiev 1/6/2021

Let us introduce the *tree tuple* (k, v) where v represents certain node - semantic particle or a subtree. Here k is a tree tuple factor which encodes uniquely the position of the node v in the tree.

The following operations are defined for tree tuple factors:

We have a base of m tree tuple factors k_1, k_2, \dots, k_m which are defined as the digits greater than 0 of $(m + 1)$ -nary number system such that $0 = k_0 < k_1 < k_2 < \dots < k_m$. We define an operation `*` denoting digit concatenation $k_i * k_j = (m + 1)k_i + k_j$. Obviously,

$$k_i * k_j > k_i \text{ for any pair } i, j = 1..m$$

Note that the latter implies that

$$k_{i_1} * k_{i_2} * \dots * k_{i_n} > k_{j_1} * k_{j_2} * \dots * k_{j_{n-1}} \text{ for any tuple where } i_p, j_q = 1..m, p = 1..n, q = 1..n - 1$$

$$(k_i, (k_j, v_j)) = (k_i * k_j, v_j)$$

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

$$T = (k_0, v_0^0). \text{ Further we will assume that } k_0 = 0.$$

$$v_0^0 = (k_0, v_0^1) + (k_1, v_1^1) + (k_2, v_2^1) + \dots + (k_m, v_m^1)$$

In general we have:

$$v_q^p = (k_0, v_q^{p+1}) + (k_1, v_{(q-1)m+1}^{p+1}) + (k_2, v_{(q-1)m+2}^{p+1}) + \dots + (k_m, v_{qm}^{p+1})$$

$$\text{where } q = 1..m^h, p = 1..h$$

Obviously, we have at most $\frac{(m^{h+1}-1)}{m-1}$ distinct terms v_q^p which represent nodes i.e. semantic values.

The expression for the tree also can be written as:

$$T = \sum_{i=0}^N (k_i^*, v_i) \text{ where } N \leq \frac{(m^{h+1}-1)}{m-1} \text{ and } k_i^* \text{ are the node factors given with } k_i^* = k_{i_1} * k_{i_2} * \dots *$$

$k_{i_n}; n \leq h$. The node values v_i are the values v_q^p ordered in increasing order of k_i^* . This order corresponds to *level order traversal* of the m -ary tree. Note that with appropriately defined comparison operation ` $<$ ` we can model different ways of traversing the m -ary tree. For instance, if we define ` $<$ ` as the comparison for the values of $(m + 1 - k_{i_1})m^{n-1} + (m + 1 - k_{i_2})m^{n-2} + \dots + (m + 1 - k_{i_n})$ 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.

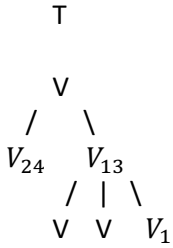
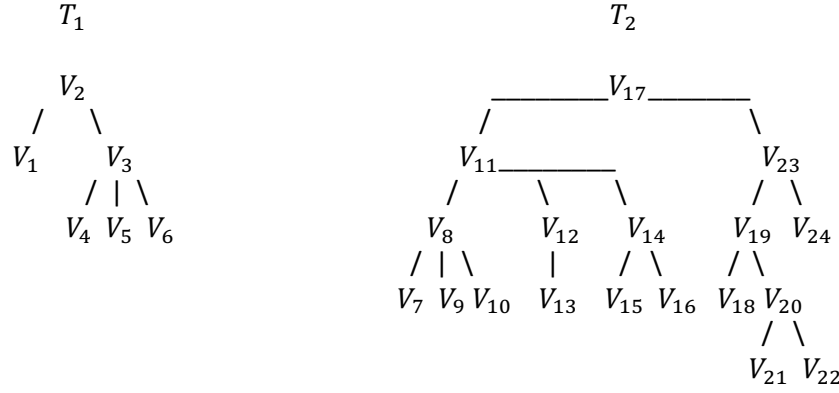
\Rightarrow James is Peter's friend

Peter is the son of Dimitar.

$$V_1 \ V_2 \ V_3 \ V_4 \ V_5 \ V_6$$

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

$$V_7 \ V_8 \ V_9 \ V_{10} \ V_{11} \ V_{12} \ V_{13} \ V_{14} \ V_{15} \ V_{16} \ V_{17} \ V_{18} \ V_{19} \ V_{20} \ V_{21} \ V_{22} \ V_{23} \ V_{24}$$



Expressing T_1 with the algebraic notation discussed earlier:

$$T_1 = (k_0, v_2) + (k_1, v_1) + (k_2, ((k_0, v_3) + (k_1, v_4) + (k_2, v_5) + (k_3, v_6)))$$

which is expanded to:

$$T_1 = (k_0, v_2) + (k_1, v_1) + (k_2 k_0, v_3) + (k_2 k_1, v_4) + (k_2 k_2, v_5) + (k_2 k_3, v_6)$$

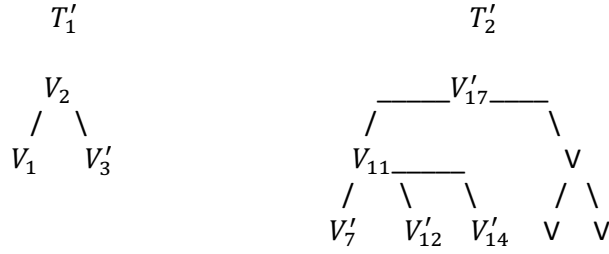
Expressing T_2 with the algebraic notation yields:

$$T_2 = (k_0, v_{17}) + \left(k_1, \left((k_0, v_{11}) + \left(k_1, \left((k_0, v_8) + (k_1, v_7) + (k_2, v_9) + (k_3, v_{10}) \right) \right) + \right. \right. \\ \left. \left(k_2, \left((k_0, v_{12}) + (k_1, v_{13}) \right) \right) + \left(k_3, \left((k_0, v_{14}) + (k_1, v_{15}) + (k_2, v_{16}) \right) \right) \right) + \left(k_2, \left((k_0, v_{23}) + \right. \right. \\ \left. \left. \left(k_1, \left((k_0, v_{19}) + (k_1, v_{18}) + \left(k_2, \left((k_0, v_{20}) + (k_1, v_{21}) + (k_2, v_{22}) \right) \right) \right) \right) + (k_2, v_{24}) \right) \right)$$

which is expanded to:

$$T_2 = (k_0, v_{17}) + (k_1 k_0, v_{11}) + (k_1 k_1 k_0, v_8) + (k_1 k_1 k_1, v_7) + (k_1 k_1 k_2, v_9) + (k_1 k_1 k_3, v_{10}) \\ + (k_1 k_2 k_0, v_{12}) + (k_1 k_2 k_1, v_{13}) + (k_1 k_3 k_0, v_{14}) + (k_1 k_3 k_1, v_{15}) + (k_1 k_3 k_2, v_{16}) \\ + (k_2 k_0, v_{23}) + (k_2 k_1 k_0, v_{19}) + (k_2 k_1 k_1, v_{18}) + (k_2 k_1 k_2 k_0, v_{20}) + (k_2 k_1 k_2 k_1, v_{21}) \\ + (k_2 k_1 k_2 k_2, v_{22}) + (k_2 k_2, v_{24})$$

Semantic Aggregation:



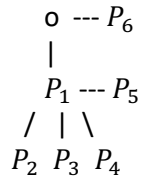
Properties and Dependent Properties:



A V 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”:

P_1 : key: ‘particle_type’, value: ‘verb’
 P_2 : key: ‘is_transitive’, value: ‘true’
 P_3 : key: ‘plurality’, value: ‘singular’
 P_4 : key: ‘grammatical_person’, value: ‘third_person’
 P_5 : key: ‘tense’, value: ‘present’
 P_6 : key: ‘text’, value: ‘is’



Each property (P -particle) is constructed from a key (key -particle) and a value (val -particle). Each key and value particles are registered in a particle registry \mathfrak{R}^P . The purpose of the particle registry \mathfrak{R}^P is to facilitate learning of new keys and values and use them for parsing new semantic constructs. Between any two keys k_1 and k_2 , such that it is obeyed exactly one of the following 3 relations:
 $k_1 \nless k_2 \vee k_1 > k_2 \vee k_1 < k_2$
Here the operator \nless indicates that there is no parent-child relationship between k_1 and k_2 , the operator $>$ indicates that k_2 is a subkey (or a child) of k_1 and $<$ indicates that k_2 is a superkey (or a parent) of k_1 .
For instance, in the example above $key(P_1) \nless key(P_6)$

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:

P_7 : key: `'semantic_properties_available'`, value: `'true'`

P_8 : 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_descendants(k): set()

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

Simple scenario to add property via inference:

Dimitar is a son of John.

V_1 V_2 V_3 V_4 V_5 V_6

The red Ford is a car of Bank of America.

V_7 V_8 V_9 V_{10} V_{11} V_{12} V_{13} V_{14} V_{15} V_{16}

Maria is a daughter of Dimitar.

V_{17} V_{18} V_{19} V_{20} V_{21} V_{22}