Structured_Based Measurement

Lu JingWei
FREE UNIVERSITY OF BOLZANO
TECHNICAL UNIVERSITY OF MADRID
phoenixlv0108@gmail.com

December 29, 2010

1 Structured Based Measurement

By Interpretation Based Measurement in (ref), similarity between predicates is obtained, and used to answer queries in a fuzzy way. For example, as known, *Beautiful* and *Popular* have high similarity value, in an incomplete knowledge base, querying *popular(X)*, not only returns results from *popular(X)*, but also from *Beautiful(X)* . In this way, fuzzy query-answering gives more similar choices for users. However, in real life, such as "geek" and "unsociable" share a large amount of people in each group, by Interpretation Based Measurement, they would be connected with higher similarity value, but semantically, they don't refer to similar concepts according to their linguistics definitions. In this case, asking for "geek", the "computer scientist" returns, so does "unsociable", asking for "unsociable", "shy people" returns, so does "geek". Obviously, not only those real similar concepts are gained, but some untruthful results are obtained at the same time, since Interpretation Based Measurement builds strong similarity between "unsociable" and "geek". In order to avoid this disadvantage, we propose Structured Based Measurement, in which similarity between predicates is defined from their embedded meanings. In RFuzzy Framework, the embedded meaning of predicate is presented by its structure.

In Structured Based Measurement, we build a tree for each predicate, to compare two predicates, indeed, is to compare two trees. Section 1.1 demonstrates the approach to build a tree for predicate. In section 1.2, the algorithm to obtain similarity between two trees is described in detail. Its complexity is presented in section 1.3.

1.1 Predicate tree

In this section, we will answer the question that how to build a tree for a predicate in RFuzzy Program. For convenience, we call the corresponding tree for the predicate as *predicate tree*. We start to transform RFuzzy Program, generate the subset of program we use to create predicate tree, which is introduced in

section 1.1.1. The approach of constructing predicate trees from subset of program is presented in section 1.1.2. In order to compare two predicate trees, the structure of them should be the same, and also preserve the embedded meanings. For this, the concept "equivalent trees" is introduced in section 1.1.3. According to "equivalent trees", the predicate tree could be constructed with *identities*, which maintain the semantic meaning but reform the structure. It is described in section 1.1.4.

1.1.1 Transform RFuzzy program

In a *RFuzzy program* $P = (R, D, T)$, R is a set of *fuzzy clauses*, D is a set of *default value declarations* and T is a set of *type declarations*. In order to construct a tree for each predicate in P, we need to filter and reform some parts of P. This procedure is called *transformation*, in which there are two steps are taken, one is filtering and the other is reforming. In this section, the details about *how* and *why* are presented.

The result of first step *filtering* is a tuple $P' = (R', D', T')$, which is generated from P by following constrains,

1. Refined R'

   R as a set of *fuzzy clauses*, includes fuzzy clauses, which are written as,

   $$A \xleftarrow{c,F_c} F(B_1, ..., B_n)$$

   where $A \in TB_{\Pi,\Sigma,V}$ is called the head, $B_1, ..., B_n \in TB_{\Pi,\Sigma,V}$ is called the body, $c \in [0, 1]$ is the credibility value, and $F_c \in \{\&_1, ..., \&_k\} \subset \Omega^{(2)}$ and $F \in \Omega^{(n)}$ are connectives symbols for the credibility and the body, respectively.

   A *fuzzy fact* is a special case where $c = 1$, F_c is the usual multiplication of real numbers " . " , $n = 0$, $F \in \Omega^{(0)}$. It is written as

   $$A \longleftarrow v$$

   where c and F_c are omit.

   The similarity between predicates is a relevant value of comparison between two predicates, rather than the absolute value described in *fuzzy facts*. Therefore, *fuzzy facts* will be out of consideration of similarity between predicates.

   Thus, R' is a set of all *fuzzy clauses*, but not *fuzzy facts*, notated as, $R' = R \backslash R_{facts}$.

2. Refined D'

   A *default value declaration* for a predicate $p \in \Pi^{(n)}$ is written as

   $$default(p/n) = [\delta_1 \text{ if } \varphi_1, ..., \delta_m \text{ if } \varphi_m]$$

where $\delta_i \in [0,1]$ for all i. The φ_i are first-order formulas restricted to terms in p from TU_{Σ,V_p}, the predicates $=$ and $\neq$, the symbol true and the junctors $\wedge$ and $\vee$ in their usual meaning, which are *'and'*, *'or'*.

There is a special case called *unconditional* default value declaration where $m = 1$ and $\varphi_1 = true$ in the default value declarations. In this case, the predicate p/n will not be related to any other predicates, therefore, it is out of consideration of similarity.

Thus, D' is all the *conditional* default value declarations, notated as, $D' = D \backslash D_{unconditional}$.

3. Refined T'

$$T = T_{term} \cup T_{predicate}$$

T_{term} is a set of all *term type declarations*, which assign a type $\tau \in \mathcal{T}$ to a term $t \in \mathbb{HU}$ and is written as $t : \tau$. $T_{predicate}$ is a set of all predicate type declarations. A *predicate type declaration* assigns a type $(\tau_1, ..., \tau_n) \in \mathcal{T}^n$ to predicate $p \in \Pi^n$ and is written as $p : (\tau_1, ..., \tau_n)$, where τ_i is the type of p's i-th argument.

The T_{term} is not related to predicates at all, so it will not be taken into account of the similarity between predicates.

Therefore, $T' = T_{predicate}$.

After filtering the RFuzzy program $P = (R, D, T)$ into $P' = (R', D', T')$, the second step is taken to reform D' into D_{new}. A *default value declarations* in D' is

$$default(p/n) = [\delta_1 \text{ if } \varphi_1, ..., \delta_m \text{ if } \varphi_m]$$

Since φ_i are FOL formulas, it will always could be in a DNF(Disjunctive Normal Form), $\varphi_i^1 \vee ... \vee \varphi_i^{k_i}$, then $default(p/n) = \delta_i$ if φ_i is written in the *fuzzy clauses* format $A \xleftarrow{c,F_c} F(B_1, ..., B_n)$.

$$p(\vec{x}) \xleftarrow{\delta_{i,\cdot}} \varphi_i^1$$

$$\vdots$$

$$p(\vec{x}) \xleftarrow{\delta_{i,\cdot}} \varphi_i^{k_i}$$

Then $default(p/n) = [\delta_1 \text{ if } \varphi_1, ..., \delta_m \text{ if } \varphi_m]$ could be reformed as,

$$p(\vec{x}) \xleftarrow{\delta_{1,\cdot}} \varphi_1^1$$

$$\vdots$$

$$p(\vec{x}) \xleftarrow{\delta_{1,\cdot}} \varphi_1^{k_1}$$

$$\vdots$$

$$p(\vec{x}) \overset{\delta_{m,\cdot}}{\longleftarrow} \varphi_m^1$$

$$\vdots$$

$$p(\vec{x}) \overset{\delta_{m,\cdot}}{\longleftarrow} \varphi_m^{k_m}$$

Apparently, the reforming procedure from D' into D_{new} preserves the semantic equivalence. The result of *reforming* functioning over P' is $P_{new} = (R_{new}, T_{new})$, where $R_{new} = R' \cup D_{new}$ a union of the *fuzzy clauses* and *default value declarations* in the same form of *fuzzy clauses*'s, $T_{new} = T'$.

1.1.2 Construct predicate tree

Every predicate in the *fuzzy program* P_{new} could be represented as a tree. A formal description of the approach of generating a corresponding tree for a certain predicate begins with the definition of *atomic predicate* and *complex predicate*.

Definition 1.1. ***(Atomic predicate)***. *Predicates only appearing in the bodies of rules in R_{new}, and never appearing in the heads of any rules, are* ***atomic predicates****, since they can not be defined by any other predicates.*

Definition 1.2. ***(Complex predicate)***. *Predicates appearing in the heads of rules in R_{new} are* ***complex predicates****, in the sense that they could be represented by the bodies of the rules with them as heads.*

We write *atomic predicate* as p_a, and *complex predicate* as p_c. We represent *atomic predicate* p_a in a tree-form, which is a node with information τ_{p_a}, and without any children nodes, where τ_{p_a} is the *type* of p_a. *Complex predicate* p_c by definition must appear in the head of some rules in P_{new}, whose form is,

$$p_c(\vec{t}) \overset{c,F_c}{\longleftarrow} F(p_1(\vec{t_1}), ..., p_n(\vec{t_n}))$$

Then the tree of p_c has a root N_{p_c}, with its node information, c, F_c, F, and τ_{p_c}, which is the *type* of p_c. The branches from N_{p_c} are N_{p_1}, ..., N_{p_n}, which are expanded as corresponding trees recursively. Thus, the leaves in the tree are *atomic predicates*, which never appear in the heads of any rules in R_{new}, and can not be expanded any more.

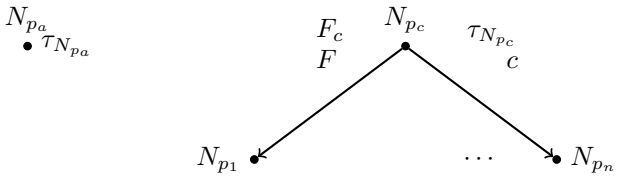


Figure 1: Predicate Tree

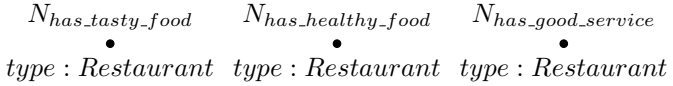
Example 1.3. *Here is a part of short RFuzzy program,*

$$\begin{array}{ll} has\_tasty\_food: & (Restaurant) \\ has\_healthy\_food: & (Restaurant) \\ has\_good\_service: & (Restaurant) \\ tasty\_restaurant: & (Restaurant) \\ good\_restaurant: & (Restaurant) \end{array}$$

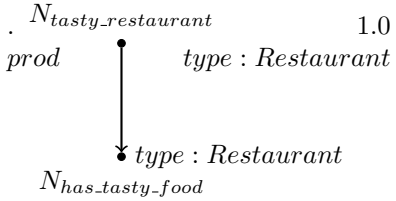
$$\begin{array}{llll} tasty\_restaurant(X) & \overset{1.0,.}{\longleftarrow} prod & has\_tasty\_food(X) \\ good\_restaurant(X) & \overset{0.8,.}{\longleftarrow} prod & has\_healthy\_food(X), has\_good\_service(X) \end{array}$$

$$Sim(has\_healthy\_food, has\_tasty\_food) = 0.6$$

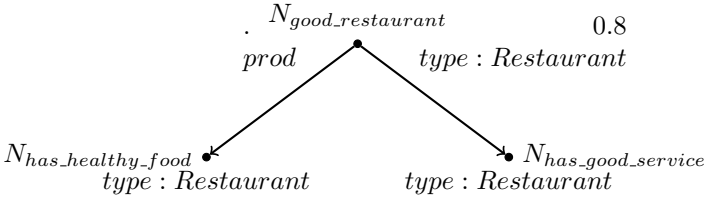
The corresponding trees for atomic predicates has_tasty_food, $has\_healthy\_food$, $has\_good\_service$ are,



The corresponding tree for complex predicate $tasty\_restaurant$ is,



The corresponding tree for complex predicate $good\_restaurant$ is,



If there are several rules defining a certain predicate p, then there are different corresponding trees for p, since the expanding procedure is recursive, which means, the children of p node could also be defined in several other rules, then more possible trees are generated.

Proposition 1.4. ***Number of corresponding tree for predicate*** p *The number of the corresponding trees for each predicate is exponential over the number of rules* n*.*

Let P be a RFuzzy program, it has rules in the following form,

$$
\begin{aligned}
p() &: -p_1(\vec{X}) \\
p_1(\vec{X}) &: -p_{11}(\vec{X}) \\
p_1(\vec{X}) &: -p_{12}(\vec{X}) \\
p_{11}(\vec{X}) &: -p_{111}(\vec{X}) \\
p_{11}(\vec{X}) &: -p_{112}(\vec{X}) \\
p_{12}(\vec{X}) &: -p_{121}(\vec{X}) \\
p_{12}(\vec{X}) &: -p_{122}(\vec{X}) \\
&\vdots
\end{aligned}
$$

continue, for each predicate p_i, there are two rules defining it, in each rule, there is only one atom in the body.

Suppose that there is n rules in P, and the number of trees could be generated for predicate p is $2^{n/2}$, which is exponential of n. However, practically, the number of corresponding trees for certain predicate will not achieve the exponential, since the rules will be defined more reasonable, rather than the given example.

In order to retrieve the similarity between two predicates, creating all their corresponding trees is not optimal approach, considering the possible exponential explosion. Therefore, there is a more efficient algorithm is proposed in the following sections.

1.1.3 Equivalence between predicate trees

In order to compare two predicate trees with different number of children, we reconstruct them with the same number of children, preserving the original semantic meaning. For this, the concept "equivalent tree" is introduced in this section.

For each rule in P_{new} that $A \xleftarrow{c,F_c} F(B_1, ..., B_n)$, under its interpretations, it is viewed as a function in the following form,

$$A^{\mathcal{I}} = OP(c, B_1^{\mathcal{I}}, ..., B_n^{\mathcal{I}})$$

where $OP = (\hat{F}_c, \hat{F})$ is a pair, representing the operations over *credit value* c, and other interpretation over atoms B_i in the body of the rule.

Definition 1.5. ***(Identity for complex predicate).*** *There exists a formula α under the operations of RFuzzy Program, which makes*

$$OP(c, B_1^{\mathcal{I}}, ..., B_n^{\mathcal{I}}, \alpha^{\mathcal{I}}) = OP(c, B_1^{\mathcal{I}}, ..., B_n^{\mathcal{I}})$$

for each interpretation $\mathcal{I}$. Therefore, the formula is called identity for OP. The existence of α depends on the operation OP.

If *identity* α exists for some OP, then the rule

$$A \xleftarrow{c,F_c} F(B_1, ..., B_n) \quad (1)$$

could be rewritten as

$$A \xleftarrow{c,F_c} F(B_1, ..., B_n, \alpha, ..., \alpha) \quad (2)$$

Since the procedure of rewriting preserves the semantic equivalence, the corresponding trees of rules (1) and (2) represent the same semantic meaning.

Definition 1.6. ***(Identity for atomic predicate).*** *For some certain $OP = (\hat{F}_c, \hat{F})$, there exists a formula β under the operations of RFuzzy Program, which makes,*

$$A^{\mathcal{I}} = OP(c, A^{\mathcal{I}}, \beta^{\mathcal{I}})$$

for any interpretation $\mathcal{I}$, where c is a real number in the range of $[0, 1]$. Therefore, β is called identity for OP.

1.1.4 Reconstruct predicate tree with equivalence

In this section, we present the approach of constructing predicate tree with equivalence, which is used to expand two comparing nodes with same structure in the algorithm introduced in section 1.2

- *Atomic predicate*

  Suppose that p_a is a *atomic predicate*. Its type is τ. The identity for some OP associated with p_a is β.

  The corresponding tree of it is a node N_{p_a} with information, which are τ of p_a and a number 0, indicating that p_a is atomic predicate. There are no branches for the node N_{p_a}.

  With the equivalence extension, the corresponding tree is presented in figure 2. The root is a node N_{p_a} with information, which are the *type* of p_a τ, the operation OP and atomic mark 0. The branches are a node N'_{p_a} and several nodes N_β. N'_{p_a} carries information τ, which is p_as type and atomic mark 0. N_β carries information of identity β and identity mark id, the number of such nodes depends on two comparing predicates.

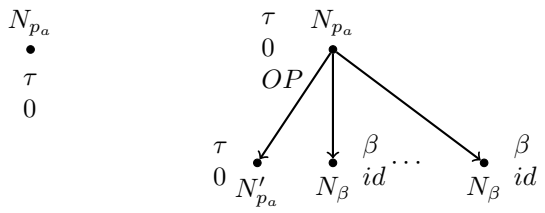


Figure 2: Atomic predicate tree with equilvalence

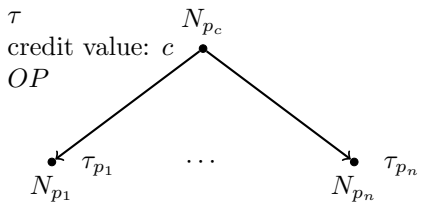
- *Complex predicate*

  For *complex predicate*, there would be at least one rule defining it as,

$$p_c(\vec{t}) \xleftarrow{c,F_c} F(p_1(\vec{t_1}), ..., p_n(\vec{t_n}))$$

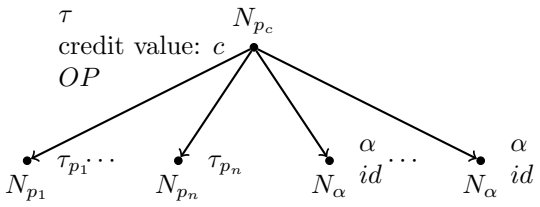
  and for $OP = (\hat{F}_c, \hat{F})$, there exists an identity α. τ, and τ_i are represented types of p_c and p_i respectively.

  The corresponding tree for p_c is



  where N_{p_c} is the root with information which are τ, OP and credit value c. N_{p_i}s are the branches carrying their *types* as information.

  With the equivalence extension, the tree is



  N_{p_c} is the root with information which are τ, OP and credit value c. N_{p_i}s are the branches with their *types* as information, and several nodes N_α carrying information of identities α with identity mark id are added as extra branches depending on two comparing predicates.

Example 1.7. *Continue example 1.3, reconstruct predicate trees with equivalence.*

$$
\begin{array}{ll}
has\_tasty\_food: & (Restaurant) \\
has\_healthy\_food: & (Restaurant) \\
has\_good\_service: & (Restaurant) \\
tasty\_restaurant: & (Restaurant) \\
good\_restaurant: & (Restaurant)
\end{array}
$$

$$
\begin{array}{lll}
tasty\_restaurant(X) & \overset{1.0,.}{\longleftarrow} prod & has\_tasty\_food(X) \\
good\_restaurant(X) & \overset{0.8,.}{\longleftarrow} prod & has\_healthy\_food(X), has\_good\_service(X)
\end{array}
$$

The predicate tree for has_tasty_food is represented in figure 3.

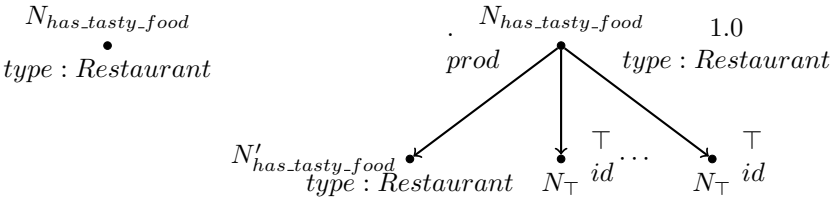


Figure 3: two equivalent predicate trees for has_tasty_food

The predicate tree for tasty_restaurant is represented in figure 4, and 5 which is extended with equivalence.

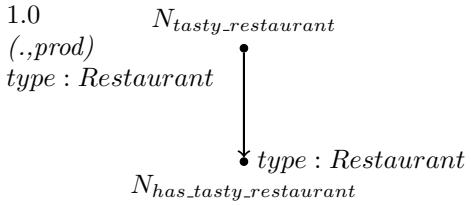


Figure 4: predicate tree for tasty_restaurant

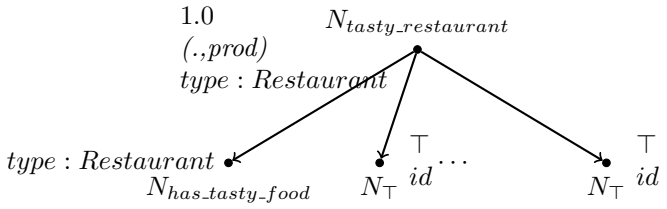


Figure 5: predicate tree for tasty_restaurant with equivalence extension

1.2 Algorithm for similarity

Let p_a and p_b be arbitrary predicates in RFuzzy Program P. The similarity value is represented as a pair $(level, sim\_degree) \in \mathbb{Z}/\mathbb{Z}^+ \times \{[0,1], u\}$, where *level* is a nonpositive integer representing the level of expansion of the corresponding tree, and *sim_degree* is a real number ranging from 0 to 1 representing the similarity degree between two predicates or a mark u to represent that the similarity degree has not been defined.

$(\mathbb{Z}/\mathbb{Z}^+ \times [0,1], \leq)$ is a partial set. Suppose that r_1, r_2 are two real numbers in $[0,1]$, where $r_1 \leq r_2$. We have a partial order such as,

$$\cdots \leq (-3,0) \leq (-3,r_1) \leq (-3,r_2) \leq (-3,1) \leq$$

$$\cdots \leq (0,0) \leq (0,r_1) \leq (0,r_2) \leq (0,1)$$

Therefore, arbitrary two pairs (l_1, v_1), (l_2, v_2) are in $\mathbb{Z}/\mathbb{Z}^+ \times [0,1]$, $(l_1, v_1) \leq (l_2, v_2)$ **iff** $l_1 < l_2$ *or* $l_1 = l_2$ and $v_1 \leq v_2$.

Let p_a and p_b be predicates with types τ_{p_a} and τ_{p_b}, respectively. The similarity between them is defined in the following approach.

- *atomic predicate VS atomic predicate*

  p_a and p_b are *atomic predicates*, which are represented to be isolated nodes with information τ_{p_a} and τ_{p_b} as *type* of p_a and p_b, respectively. If $\tau_{p_a} = \tau_{p_b}$, in the sense that p_a and p_b have the same *type*, then similarity degree between p_a and p_b should be defined directly as a real number in $[0,1]$ in RFuzzy program P as *sim_degree*. Then the similarity value is $Sim(p_a, p_b) = (l, sim\_degree)$, where l is the level of expansion. Once no such information found in RFuzzy program P, return $(-\infty, 0)$ as default similarity value, which means the similarity can not be found even though the predicates could be expanded till infinite level.

Figure 6: Comparison between two atomic predicates

- *atomic predicate VS complex predicate*

  p_a is an atomic predicate, and p_b is a complex predicate. Then, p_a is represented as an isolated node with information which are τ_{p_a} and 0 as atomic mark. While, p_b is represented as an isolated node with information τ_{p_b}. The similarity between them is achieved by following,

  1. If $\tau_{p_a} \neq \tau_{p_b}$ then $Sim(p_a, p_b) = (-\infty, 0)$
  2. If $\tau_{p_a} = \tau_{p_b}$, then search for defined *sim_degree* in RFuzzy program, which is a real number in $[0, 1]$. If there exists, then return the similarity value as (l, sim_degree), where l is the level of expansion which p_a and p_b are on.
  3. If $\tau_{p_a} = \tau_{p_b}$ and there doesn't exist the defined similarity in RFuzzy program, then two steps are taken afterwards,
     - Return a middle result $Sim(p_a, p_b) = (l, u)$, where l is the level of expansion which p_a and p_b are on and u represents the similarity degree is undefined.
     - Expand p_a and p_b in the following way. p_b is expanded according to its rule
     $$p_b(\vec{t}) \overset{c,F_c}{\longleftarrow} F(p_1(\vec{t_1}), ..., p_n(\vec{t_n}))$$
     N_{p_b} is the root with information $INFO_{p_b} = \{c, OP = (\hat{F}_c, \hat{F}), \tau_{p_b}\}$. Its branches are nodes N_{p_1}, ..., N_{p_n} with *type* τ_{p_1}, ..., τ_{p_n} of p_1, ..., p_n respectively. p_a is expanded according to OP in $INFO_{p_b}$, then N_{p_a} is the root with $INFO_{p_a} = \{c', OP = (\hat{F}_c, \hat{F}), \tau_{p_a}\}$. Its branches are nodes N'_{p_a}, N_β, ...,N_β. The number of children of N_{p_a} is n, which is the same as N_{p_b}.

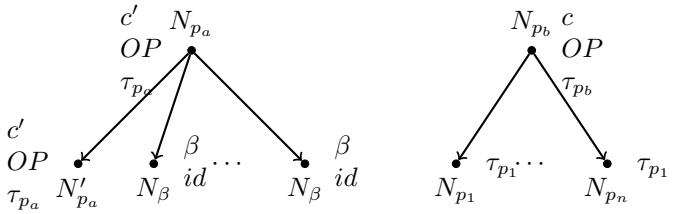


Figure 7: Comparison between atomic and complex predicates

- *complex predicate VS complex predicate*

  p_a and p_b are complex predicates with τ_{p_a} and τ_{p_b} as *type*, respectively. If p_a and p_b are of the different *types*, that is, $\tau_{p_a} \neq \tau_{p_b}$, then $Sim(p_a, p_b) = (-\infty, 0)$ will be returned, otherwise, the expanding procedure will be carried on according to the fuzzy rules. p_a is defined by rule

$$p_a(\vec{t}) \overset{c^a,F^a_c}{\longleftarrow} F^a(p^a_1(\vec{t^a_1}), ..., p^a_n(\vec{t^a_n}))$$

with $OP_a = (F_c^a, F^a)$ and p_b is defined by rule

$$p_b(\vec{t}) \overset{c^b, F_c^b}{\longleftarrow} F^b(p_1^b(\vec{t_1^b}), ..., p_m^b(\vec{t_m^b}))$$

with $OP_b = (F_c^b, F^b)$. There are two cases of p_a and p_b expansion.

1. p_a and p_b are defined by the different OP
   If $OP_a \neq OP_b$, which is, $F_c^a \neq F_c^b$ or $F^a \neq F^b$, then return 0 as similarity degree, and $Sim(p_a, p_b) = (-\infty, 0)$.
2. p_a and p_b are defined by the same OP
   If $OP_a = OP_b$, which is $F_c^a = F_c^b$ and $F^a = F^b$, then same two steps are taken here,
   - Return middle result $Sim(p_a, p_b) = (l, u)$
   - Continue the identity formalization, after which, p_a and p_b have corresponding trees with the same construction, where the roots are of the same type, the same OP, and have the same number of children. The comparing procedure of their children is recursively defined.

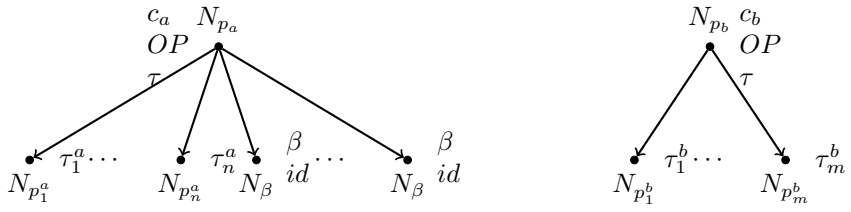


Figure 8: Comparison between two complex predicates

For each expansion, the middle result which is the similarity value before next expansion and comparison, must be returned to filter some pairs of corresponding trees to reduce unnecessary expansion or comparison. The middle result of one level is formalized as follow.

Suppose that p_a and p_b are on level l in their own trees, and are satisfied expanding requirements with n children for each. After combination of their children, n optimal $Sim(p_i^a, p_j^b)$ are returned and composed as a *middle set* M_s. The possible values in M_s are $(-\infty, 0)$, $(l-1, sim\_degree)$, $(l-1, u)$, where $sim\_degree \in [0, 1]$. There is one subset of M_s,

$$M_s^{'} = \{(level, degree) | level \neq -\infty, degree \neq u\}$$

The *middle result* is $M_r = (l-1, M_v)$, where M_v is defined as,

$$M_v = \frac{\sum_{i=1}^{m} v_i}{n}$$

where m is the cardinality of $M_s^{'}$, v_i is an element in $M_s^{'}$.

For any comparing pair (p_i^a, p_j^b) from children of p_a and p_b, if $Sim(p_i^a, p_j^b) = (l-1, u)$, it means that p_i^a and p_j^b have the same type but the similarity degree have not been defined directly in the RFuzzy program, and it could be reached by expanding them according to certain rules. The expanding procedure could be continued till both comparing predicates are atomic, which makes the algorithm terminate.

The algorithm terminates **iff** the comparing trees of predicates are *completely built*.

Definition 1.8. ***Completely Built***. *The two comparing predicates are represented as trees, and expanded and valued in synchronization with each other till no comparing value* $(level, u)$ *is returned. Then the comparing trees are completely built. The* ***similarity degree*** *is calculated from leaves to root by the definition of middle value, and the* ***level*** *is the lowest level of the comparing trees.*

Example 1.9. *Here is a short RFuzzy program from the example 1.3,*

$$\begin{array}{ll} has\_tasty\_food: & (Restaurant) \\ has\_healthy\_food: & (Restaurant) \\ has\_good\_service: & (Restaurant) \\ tasty\_restaurant: & (Restaurant) \\ good\_restaurant: & (Restaurant) \end{array}$$

$$\begin{array}{lll} tasty\_restaurant(X) & \overset{1.0,.}{\longleftarrow} prod & has\_tasty\_food(X) \\ good\_restaurant(X) & \overset{0.8,.}{\longleftarrow} prod & has\_healthy\_food(X), has\_good\_service(X) \end{array}$$

$$Sim(has\_healthy\_food, has\_tasty\_food) = 0.6$$

The set of *Atomic predicates* is $AP = \{has\_tasty\_food, has\_healthy\_food, has\_good\_service\}$, and the set of *complex predicates* is $CP = \{tasty\_restaurant, good\_restaurant\}$. They have the same *type* 'Restaurant'. There are two tasks as examples for gaining the similarity, one is between $tasty\_restaurant$ and $has\_tasty\_restaurant$, and the other is between $tasty\_restaurant$ and $good\_restaurant$.

- $Sim(tasty\_restaurant, has\_tasty\_food)$

  Since $tasty\_restaurant \in CP$, and $has\_tasty\_food \in AP$, the procedure of obtaining similarity follows the case "atomic predicate VS complex predicate". In an abbreviation, tr is represented as $tasty\_restaurant$, htf is for has_tasty_food, R means 'Restaurant'.

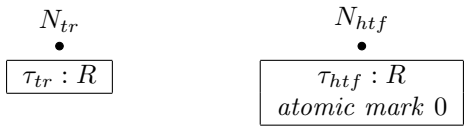


Figure 9: Similarity between tr and htf

For this pair of predicates, their *types* are the same, that is, $\tau_{tr} = \tau_{htf} = R$. And, the similarity value between them is not directly defined in program. Then, two steps are taken afterwards,

1. Return a middle result $Sim_m(tr, htf) = (0, u)$.
2. Expand tr and htf. According to the rule in program, which is

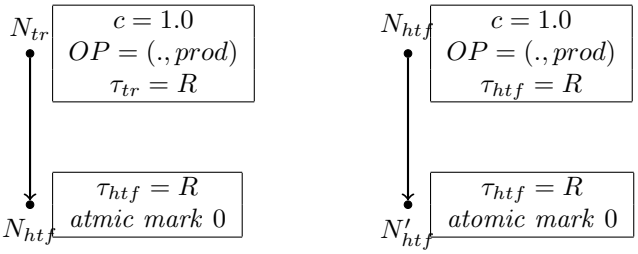


Figure 10: Expansion of tr and htf

$$tasty\_restaurant(X) \xleftarrow{1.0,.} prod\ has\_tasty\_food(X)$$

tr is expanded on the left of the figure 10, and according to $OP = (., prod)$ from tr's rule, htf is expanded on the right side. As seen in the graph, only one combination of predicates in level -1 can be achieved, which is (htf, htf). The similarity value of them is $(-1, 1)$, since they are the same predicate, the similarity between them is 1.

Thus, the similarity value is 1 between has_tasty_food and $tasty\_restaurant$ achieved in the level -1.

- $Sim(tasty\_restaurant, good\_restaurant)$

  Since $tasty\_restaurant \in CP$, and so is $good\_restaurant$, the procedure of obtaining similarity follows the case "complex predicate VS complex predicate". In an abbreviation, tr is represented as $tasty\_restaurant$, gr is for $good\_restaurant$, R means 'Restaurant'. For this pair of predicates, their *types* are the same, that is, $\tau_{tr} = \tau_{gr} = R$. And, the similarity value between them is not directly defined in RFuzzy program. Then, two steps are taken afterwards,

Figure 11: Similarity between tr and gr

1. Return a middle result $Sim_m(tr, gr) = (0, u)$.
2. Expand tr and gr. gr is expanded acccording to the rule in program,

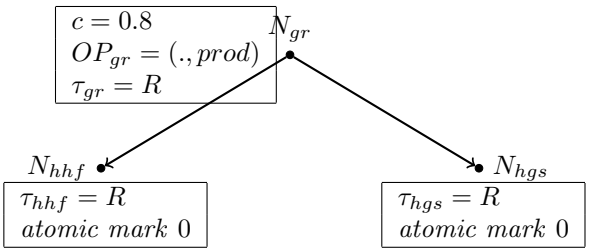


Figure 12: Expansion of gr

which is

$$good\_restaurant(X) \xleftarrow{0.8,.} prod$$
$$has\_healthy\_food(X), has\_good\_service(X)$$

According to the rule

$$tasty\_restaurant(X) \xleftarrow{1.0,.} prod\ has\_tasty\_food(X)$$

and equivalence extension associated with OP_{gr}, the tr is expanded into, Two combinations of predicates in level -1 can be achieved,

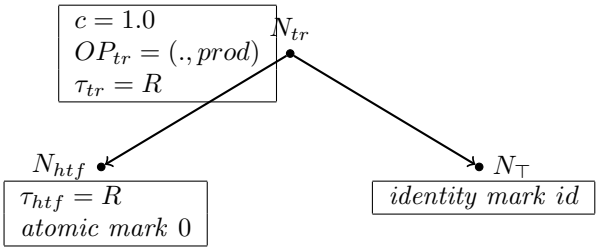


Figure 13: Expansion of tr

which are $M_{s_1} = \{(hhf, htf), (hgs, \top)\}$ and $M_{s_2} = \{(hgs, htf), (hhf, \top)\}$.

In our program, only similarity between has_tasty_food and $has\_healthy\_food$ is defined as

$$Sim(has\_healthy\_food, has\_tasty\_food) = 0.6$$

Suppose that for any predicate p, $Sim(\top, p) = 0.5$, then the similarity achieved from M_{s_1} is $M_{r_1} = (-1, 0.55)$ and from M_{s_1} is $M_{r_2} = (-1, 0.25)$. The former combination M_{s_1} will be chosen. In M_{s_1}, there is no (l, u), which means no undefined similarity in this level l. Thus the comparing trees are *completely built*, the algorithm terminates, and the similarity value is 0.55 between $good\_restaurant$ and $tasty\_restaurant$ achieved in the level -1. .

1.3 Complexity of algorithm

The similarity algorithm is a procedure of comparison and extension alternatively till the predicates are not necessary to or can not be expanded. For each pair of predicates (p_i, p_j), it could be expanded $n_i \times n_j$ different comparing trees, where n_i is the number of the rules with p_i as head, and n_j is the number of the rules with p_j as head. The complexity of calculating the middle result for each level is constant, since the number of children from p_i or p_j is constant. The maximum number of comparison and extension is the level of the comparing trees which are *completely built*, which is less than the number of rules, since one rule associates one expansion, and there are some different rules for one predicate. Suppose that the number of rules in the RFuzzy Program is n, then the complexity of similarity algorithm is approximatively $n_i \cdot n_j \cdot c_1 \cdot c_2 \cdot n$, where the $n_i \cdot n_j$ is the number of possible expansion of comparing p_i and p_j, c_1 is the number of nodes on one level, which is a constant, and c_2 is the number of comparing times for one level to achieve the middle result, and is also a constant. n is represented as the number of levels. The complexity of algorithm is less than $\mathcal{O}(n^3)$.