Chapter 1

The Need For a New Proposal to Evaluate Similarity Between Fuzzy Predicates Defined In a FuzzyPprogram

In chapter **??** we have mentioned about some existing works in the field of similarity concept in Fuzzy Logic, and also the *SBM* method [**?**] has been highlighted as an intriguing work in the field. In this chapter the *SBM* method is observed in detail, and the layout of the section is as follows:

In Structured Based Measurement, a tree is built for each predicate, so comparing two predicates is indeed comparing two trees. In the light of that, section 1.1.1 introduces the approach for building a tree for predicate. In section 1.1.2, the algorithm to obtain similarity between two trees is described in detail. Then the shortcomings of the approach are displayed with their corresponding examples in section 1.2.

1.1 Structured Based Measurement

1.1.1 Predicate tree

Firstly, the question of "how to build a tree for a predicate in *RFuzzy* Program" is the main focus. For convenience, the corresponding tree for the predicate is called as a *predicate tree*. The first step is transforming an *RFuzzy* Program, to obtain the subset of program which is used to create the predicate tree in section 1.1.1.1. The approach of constructing predicate trees from subset of program is presented in section 1.1.1.2. In SBM method, to be able to compare two predicate trees, the structure of the concerning trees should be the same. Moreover they ought to preserve the embedded meanings. To ensure this, the concept “equivalent trees” is introduced in section 1.1.1.3. According to “equivalent trees”, the predicate tree could be constructed with *identities*, which is claimed to maintain the semantic meaning but reform the structure. This is described in section 1.1.1.4.

1.1.1.1 Transforming RFuzzy program

In an *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, there is the need of filtering and reforming some parts of P. This procedure is called *transformation*, in which two steps are taken, one is filtering and the other is reforming. In this section, the details *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,

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

   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. F_c is to combine the credibility of the rule and the truth value of the body. F is to combine the truth values of the subgoals in the body.

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

   $$A \longleftarrow v$$

   where c and F_c are omitted.

   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* is out of consideration of similarity between predicates.

   As a result, 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 is not related to any other predicates, and therefore it is out of consideration in similarity.

All in all, 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 is not 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 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,

$$p(\vec{x}) \xleftarrow{\delta_1,\cdot} \varphi_1^1$$

$$\vdots$$

$$p(\vec{x}) \overset{\delta_1,\cdot}{\longleftarrow} \varphi_1^{k_1}$$

$$\vdots$$

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

$$\vdots$$

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

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, $T_{new} = T'$.

1.1.1.2 Constructing the 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 1.1.1. **(Atomic predicate)**. Predicates appearing only in the bodies of rules in R_{new}, and never appearing in the heads of any rules, are **atomic predicates**, since they are not defined by any other predicates.

Definition 1.1.2. **(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.

Atomic predicate is written as p_a, and *complex predicate* as p_c. *Atomic predicate* p_a is represented 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.

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.

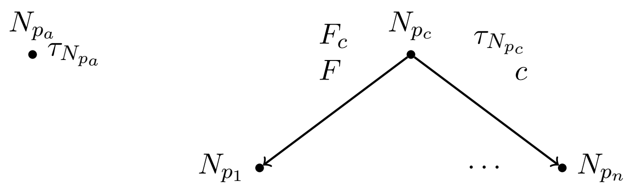


FIGURE 1.1: Predicate Tree

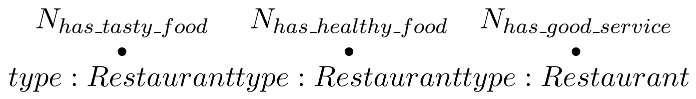
Example 1.1.3. Here is a part of short RFuzzy program,

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

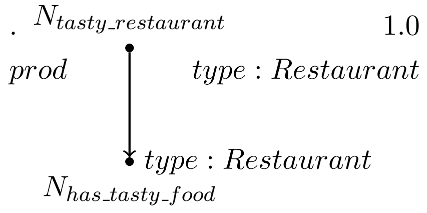
$$\begin{array}{llll} tasty\_restaurant(X) & \xleftarrow{1.0,.} & prod & has\_tasty\_food(X)\\ good\_restaurant(X) & \xleftarrow{0.8,.} & prod & has\_healthy\_food(X), has\_good\_service(X)\end{array}$$

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

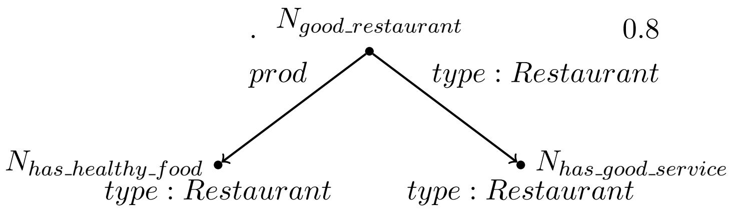
The corresponding trees for atomic predicates *has_tasty_food*, *has_healthy_food*, *has_good_service* are,



The corresponding tree for complex predicate *tasty_restaurant* is,



The corresponding tree for complex predicate *good_restaurant* is,



1.1.1.3 Equivalence between predicate trees

In order to compare two predicate trees with different number of children, they are reconstructed 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 1.1.4. **(Identity for complex predicate).** There exists a formula α under the operations of RFuzzy Program, which makes

$$OP(c, B_1^{\mathcal{I}}, ..., B_n^{\mathcal{I}}, \alpha^{\mathcal{I}}) = OP(c, B_1^{\mathcal{I}}, ..., B_n^{\mathcal{I}}) \tag{1.1}$$

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

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

$$A \xleftarrow{c,F_c} F(B_1, ..., B_n) \tag{1.2}$$

could be rewritten as

$$A \xleftarrow{c,F_c} F(B_1, ..., B_n, \alpha, ..., \alpha) \tag{1.3}$$

Since the procedure of rewriting preserves the semantic equivalence, the corresponding trees of rules 1.2 and 1.3 represent the same semantic meaning.

Definition 1.1.5. **(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.

1.1.1.4 Reconstructing predicate trees with equivalence

In this section, the approach of constructing predicates tree with equivalence is displayed, which is used for expanding two comparing nodes with the same structure in the algorithm introduced in section 1.1.2.

- *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 1.2. 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.

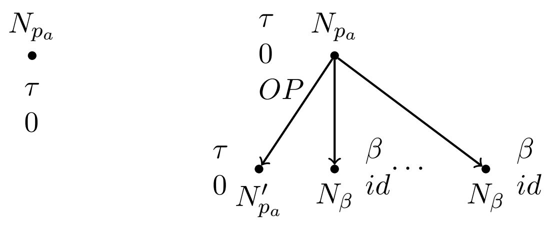


FIGURE 1.2: Atomic predicate tree with equivalence

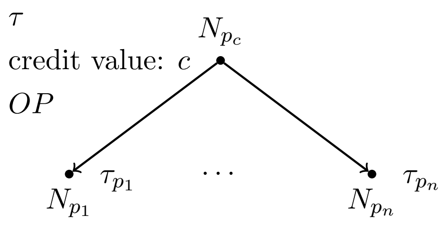
- *Complex predicate*

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

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

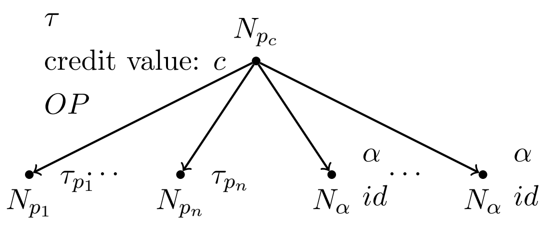
  and for $OP = (\hat{F_c}, \hat{F})$, there exists an identity α. τ, and τ_i are represented types of p_c and p_i respectively.

  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 1.1.6. Continuation of the example 1.1.3, reconstruct predicate trees with equivalence.

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

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

The predicate tree for *has_tasty_food* is represented in figure 1.3.

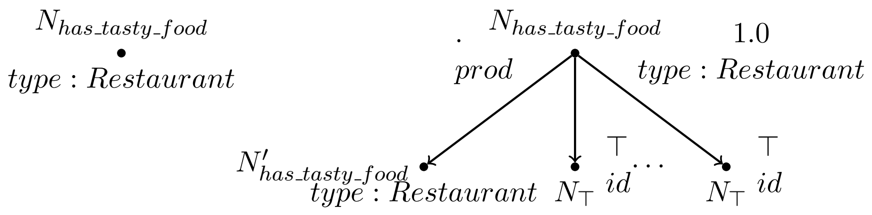


FIGURE 1.3: two equivalent predicate trees for has_tasty_food

The predicate tree for *tasty_restaurant* is represented in figure 1.4, and 1.5 which is extended with equivalence.

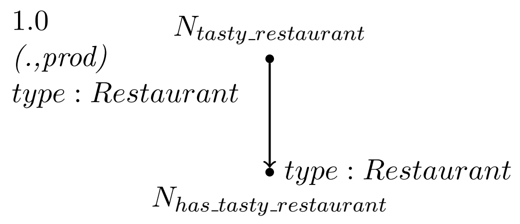


Figure 1.4: predicate tree for tasty_restaurant

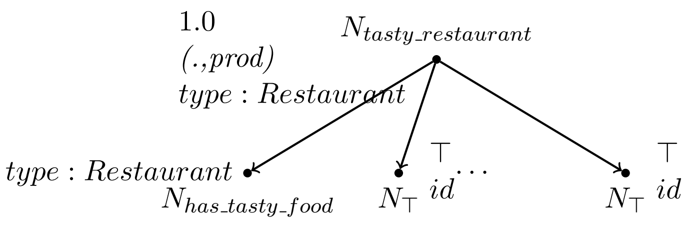


Figure 1.5: predicate tree for tasty_restaurant with equivalence extension

1.1.2 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 until infinite level.

FIGURE 1.6: 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
     $$p_b(\vec{t}) \xleftarrow{c,F_c} F(p_1(\vec{t_1}), ..., p_n(\vec{t_n}))$$

     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* $\tau_{p_1}, \ldots, \tau_{p_n}$ of $p_1, \ldots, p_n$ 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}.

- *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

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

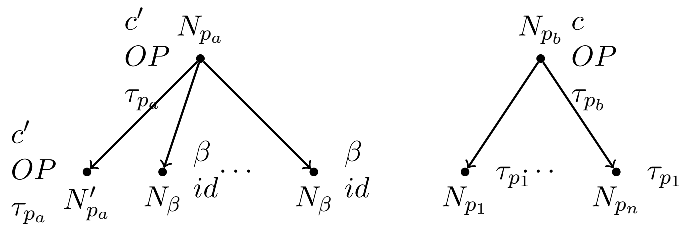


Figure 1.7: Comparison between atomic and complex predicates

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

$$p_b(\vec{t}) \overset{c^b, F^b_c}{\longleftarrow} F^b(p^b_1(\vec{t^b_1}), ..., p^b_m(\vec{t^b_m}))$$

with $OP_b = (F^b_c, F^b)$. 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, $F^a_c \neq F^b_c$ 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 $F^a_c = F^b_c$ 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.

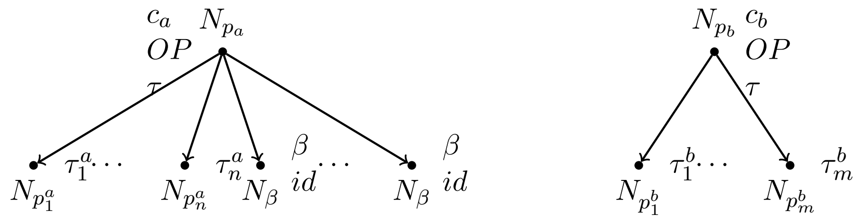


Figure 1.8: Comparison between two complex predicates

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, called *decision middle set*, defined as,

$$M_{ds} = \{(level, degree) | level \neq -\infty, degree \neq u\} \tag{1.4}$$

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

$$M_v = \frac{\sum_{i=1}^{m} v_i + 1 - |c^a - c^b|}{n+1} \tag{1.5}$$

where m is the cardinality of M_{ds}, v_i is an element in M_{ds}.

The meaning of $1 - |c^a - c^b|$ is as follows:

Credit value c^a and c^b are the trust degrees of fuzzy rules within head p_a and p_b, respectively. They are values in unit interval $[0,1]$. We define the distance between c^a and c^b by $1-norm$, which is $|c^a - c^b|$. The similarity between them is $1 - |c^a - c^b|$. c^a and c^b are considered as a special pair of combinations, whose similarity is counted into the middle result of similarity between p_a and p_b.

Furthermore *BMS* requires an extension regarding the identity predicate. Firstly, the purpose of introducing identity α for an operation $OP = (\hat{F_c}, \hat{F})$ is to build two predicate tree in the same structure, where two predicate trees have the identical OP and the same number of children. The procedure of reconstructing predicate tree with equivalence preserves the original semantics. When comparing identity α and an arbitrary predicate p, the similarity between them is defined as $Sim(\alpha, p) = (l, v)$, where l is the level the identity is on, and v is an arbitrary value in $[0,1]$. The value v will not affect the decision of choosing the optimal combination. The reason is shown in next point.

Lastly, the algorithm is concluded with the termination step. 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 until both comparing predicates are atomic, which makes the algorithm terminate.

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

Definition 1.1.7. **Completely Built**. The two comparing predicates are represented as trees, and expanded and valued in synchronization with each other until 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 1.1.8. Here is a short RFuzzy program from the example 1.1.3,

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

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

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

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'. For this pair of predicates, their *types* are the same, that

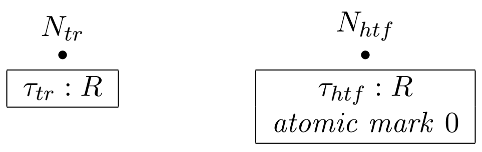


FIGURE 1.9: Similarity between tr and htf

  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

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

     tr is expanded on the left of the figure 1.14, 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).

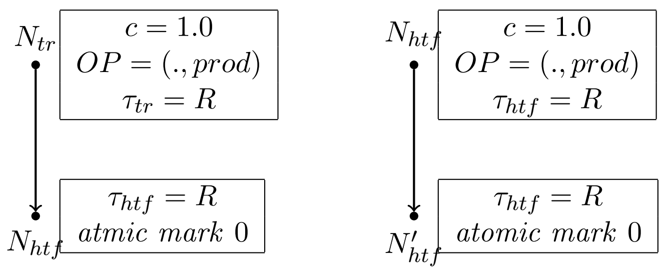


FIGURE 1.10: Expansion of tr and htf

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,

N_{tr}
$\tau_{tr} : R$
N_{gr}
$\tau_{gr} : R$

FIGURE 1.11: Similarity between tr and gr

  $\tau_{tr} = \tau_{gr} = R$. And, the similarity value between them is not directly defined in RFuzzy program. Then, two steps are taken afterwards,

  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, which

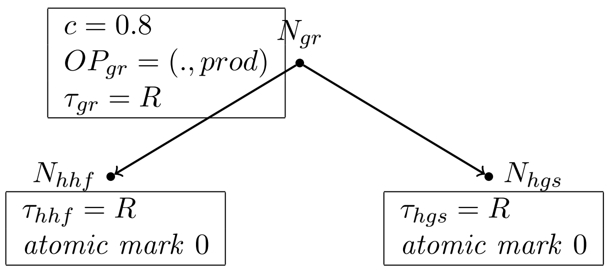


FIGURE 1.12: Expansion of gr

  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

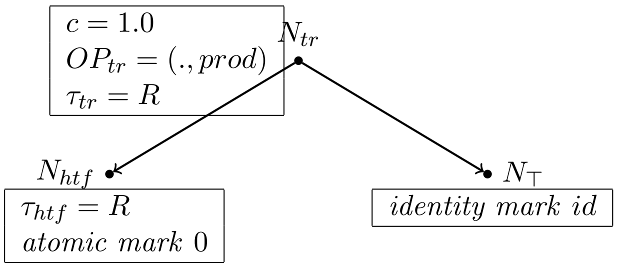


Figure 1.13: Expansion of tr

combinations of predicates in level -1 can be achieved, 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 $M_{r_1} = (-1, 0.6\dot{3})$ and from M_{s_1} is $M_{r_2} = (-1, 0.4\dot{3})$. 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 0.63 between *good_restaurant* and *tasty_restaurant* achieved in the level -1.

1.2 Shortcomings of *SBM*

Even though the *SBM* methodology is a promising approach, it does not come without some serious shortcomings. These problems lie in three main characteristics of the method, namely:

- Construction of the predicate three
- Search algorithm for similar predicates
- The similarity evaluation function

In this section we demonstrate all these deficiencies with solid examples.

1.2.1 Inclusion of Credibilities in Similarity Evaluation Function

The first defect of the *SBM* approach comes with the way it utilizes the credibility values of the fuzzy rules in the similarity evaluation function for the predicates. The way that *SBM* adopts proves to be problematic when the branching factor is small. Since it's directly summed with the values of similarity pairs in the numerator of the equation, when the cardinality n is small, the values of credibilities simply overshadow of similarity pairs.

Example 1.2.1. The following simple example displays one such scenario:

$$\begin{array}{ll} good\_basketball\_player: & (Player) \\ bad\_basketball\_player: & (Player) \\ good\_technique: & (Player) \\ egoism: & (Player) \end{array}$$

$$\begin{array}{lll} good\_basketball\_player(X) & \overset{0.95,.}{\longleftarrow} prod & good\_technique(X). \\ bad\_basketball\_player(X) & \overset{0.9,.}{\longleftarrow} prod & egoism(X)\ . \end{array}$$

$$Sim(good\_technique, egoism) = 0.1$$

There is no need for expansion in this case for the predicate trees as they're structurally equivalent. The built trees are depicted in figure 1.14.

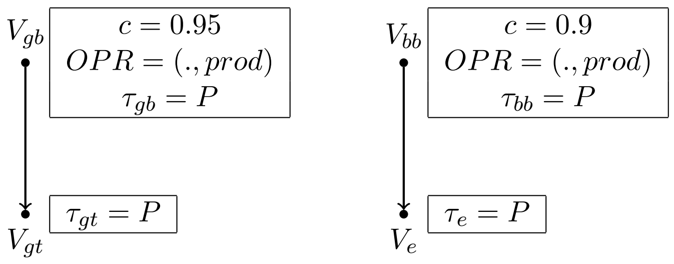


FIGURE 1.14: Predicate trees for *good_basketball_player* and *bad_basketball_player*

By inspecting the example, one should **not** expect the predicates *good_basketball_player* and *bad_basketball_player* to be similar at all. We shall pursue via performing steps of *SBM* in order to see if the expectation is consistent with the methodology's calculation.

$M_s = \{$ (good_technique, egoism)$\}$, $c^a = 0.95$, $c^b = 0.9$, $n = 1$.

$$\begin{aligned} M_v &= \frac{\sum_{i=1}^{1} v_i + 1 - |c^a - c^b|}{n+1} \\ &= \frac{0.1 + 1 - |0.95 - 0.9|}{1+1} \\ &= \frac{1.05}{2} \\ &\cong \mathbf{0.53} \end{aligned} \tag{1.6}$$

So on contrary to what was expected, *SBM* calculates *good_basketball_player* and *bad_basketball_player* to be somewhat similar. As mentioned earlier, this problem is caused by the fact that the equation does not integrate the credibility values in a proper way, thus in some cases (such as the branching factor is low) the emphasize is so high that it dominates the similarity values between the subpredicates.

1.2.2 Convergence to Identity Predicate

Earlier it was stated that in order to compare two predicates, *SBM* needs them to have the same tree structure. And in order to accomplish this when they are not equal, the missing branches and leaves of the smaller tree are filled with identity predicates. Moreover a default similarity value is defined between an arbitrary predicate and the identity predicate.

Unfortunately this introduces a couple of problems concerning the precision of the algorithm's final result. Similar to the effect that credibility values had in the previous section, this time we may have such an overwhelming impact from the default similarity values that is defined between the identity predicate and an arbitrary predicate. One will especially encounter this kind of scenarios when one tree has high branching factor compared to the other one.

Example 1.2.2. We see an example of such a case in the following program:

$classy\_restaurant$:	$(Restaurant)$
$good\_restaurant$:	$(Restaurant)$
$well\_trained\_waiters$:	$(Restaurant)$
$expensive\_inventory$:	$(Restaurant)$
$has\_good\_service$:	$(Restaurant)$
$has\_healthy\_food$:	$(Restaurant)$
has_tasty_food :	$(Restaurant)$
$has\_nice\_surroundings$:	$(Restaurant)$
$has\_high\_reputation$:	$(Restaurant)$

$$\begin{array}{lll} classy\_restaurant(X) & \overset{1.0,.}{\longleftarrow} prod & well\_trained\_waiters(X), expensive\_inventory(X). \\ good\_restaurant(X) & \overset{0.95,.}{\longleftarrow} prod & has\_healthy\_food(X), has\_good\_service(X), \\ & & has\_nice\_surroundings(X), has\_high\_reputation(X), \\ & & has\_tasty\_food(X). \end{array}$$

$$Sim(well\_trained\_waiters, has\_good\_service) = 0.9$$

$$Sim(expensive\_inventory, has\_nice\_surroundings) = 0.8$$

Regarding to the program we should expect the predicates *classy_restaurant* and *good_restaurant* to have a high similarity degree.

Once again we should start via constructing the predicate trees for *BMS*. As the two trees are not structurally equal, the smaller tree, the predicate tree of *classy_restaurant* must be expanded.

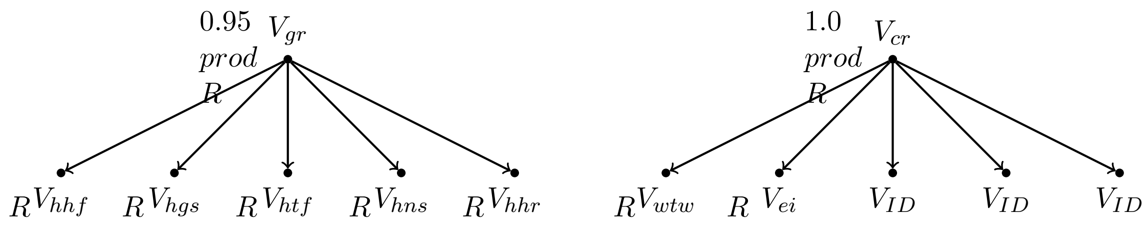


FIGURE 1.15: Predicate trees in expanded form for *SBM*

There is only one level of expansion. Since at every level the M_s that leads to the highest degree of similarity is chosen, the corresponding decision set will be: M_{ds} = { *(well_trained_waiters, has_good_service), (expensive_inventory, has_nice_surroundings), (ID, has_healthy_food), (ID, has_tasty_food), (ID, has_high_reputation)*}

One thing that we should pay attention to is that, since in this example, *BMS* needed the introduction of the *Identity Predicate*, a default similarity value between an arbitrary predicate and the identity predicate should also be defined. Assume *Sim(ID, p) = 0.3* for any arbitrary predicate.

The rest of the variable values are as follows:

$c^a = 0.95$, $c^b = 1$, $n = 5$.

$$\begin{aligned} M_v &= \frac{\sum_{i=1}^{5} v_i + 1 - |c^a - c^b|}{n+1} \\ &= \frac{2.9 + 1 - |0.95 - 1|}{5+1} \\ &\cong \mathbf{0.55} \end{aligned} \tag{1.7}$$

In this case the algorithm was not able to successfully evaluate and conclude that the predicates are actually closely related. This distortion was caused because of the relatively big branching factor difference and the identity predicates introduced for the missing one. As one can see, in *SBM* as ***the missing number of nodes in one tree increases, the similarity degree calculated by the algorithm converges to the default values*** that is defined between an arbitrary predicate and the identity predicate.

One should suspect that an algorithm which does not introduce any such external knowledge, and just use the original information of the knowledge base, might prevent having such shortcomings.

1.2.3 Wrong Filtering

In her work Lu makes the following assumption [**?**]:

...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.

This premise proves to be dangerous as making assumptions on the input setting may cause algorithms to contain flaws.

There is one other faulty behavior of *SBM* which is caused from this relaxed assumption of the the knowledge bases. After every level of expansion, *SBM* checks every predicate pair between the two trees with respect to their resulting similarity degree. Before the next level of expansion is pursued, a filtering is done on the tree by selecting the best pair combination on the level. The problem with this approach is that the information on a prior level is incomplete, and thus wrong steps can be taken when filtering that will cause the loss of crucial information for the main focus, *i.e.* comparing the similarity values of the main predicates.

Since the previous thesis work does not include any examples with predicate trees deeper then just one level, we try to demonstrate the problem with the following example:

Example 1.2.3. The program consists of following type declarations and rules:

$modern\_city:$	$(City)$
$livable\_city:$	$(City)$
$life\_expectancy:$	$(Society)$
$birth\_rate:$	$(Society)$
$social\_welfare:$	$(Society)$
$\#of\_schools$	$(Society)$
$quality\_of\_academic\_staff$	$(Society)$
$\#of\_teachers$	$(Society)$
$compulsory\_schooling\_length$	$(Society)$
$educated\_society:$	$(Society)$
$\#of\_healthy\_individuals$	$(Society)$
$literacy\_rate:$	$(Society)$
$high\_population:$	$(Society)$

$$\begin{array}{lll}
livable\_city(X) & \xleftarrow{0.8,\cdot}prod & literacy\_rate(X), \#of\_healthy\_individuals(X). \\
literacy\_rate(X) & \xleftarrow{1.0,\cdot}prod & compulsory\_schooling\_length(X), \#of\_teachers(X). \\
modern\_city(X) & \xleftarrow{0.7,\cdot}prod & educated\_society(X), high\_population(X), \\
 & & social\_welfare(X). \\
educated\_society(X) & \xleftarrow{1.0,\cdot}prod & \#of\_schools(X), quality\_of\_academic\_staff(X). \\
high\_population(X) & \xleftarrow{1,0,\cdot}prod & birth\_rate(X), life\_expectancy(X).
\end{array}$$

$$Sim(compulsory\_schooling\_length, \#of\_schools) = 0.9$$

$$Sim(\#of\_teachers, quality\_of\_academic\_staff) = 0.85$$

$$Sim(literacy\_rate, social\_welfare) = 0.4$$

$$Sim(\#of\_healthy\_individuals, life\_expectancy) = 0.7$$

Focus of interest is evaluating the similarity degree of *livable_city* and *modern_city*. Let's start calculating this value via our algorithm.

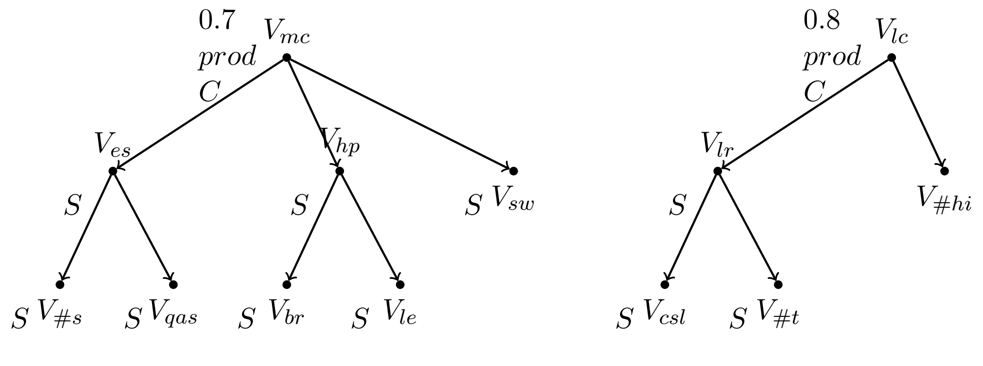


FIGURE 1.16: Predicate trees of *livable_city* and *modern_city* in original form

By know we know that as trees are do not share the same tree strcuture, *SBM* needs reconstructed versions of the tree before the evaluation algorithm can work.

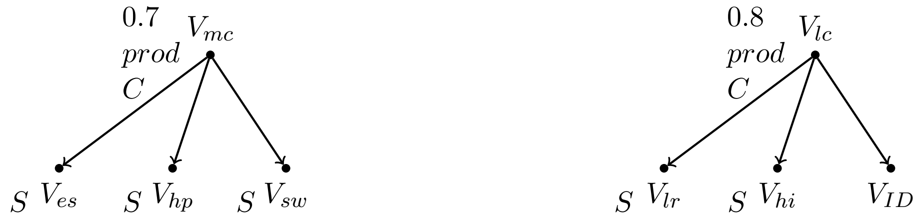


Figure 1.17: Predicate trees after one level of expansion in *SBM*

After the first expansion is done, again an identity predicate is introduced for the missing leaf node in the predicate tree with the root *livable_city*. The problem arise in this step. As we have discussed before, *SBM* makes a filtering of the node-pairs before the next expansion, with respect to the similarity proximities of the pairs in this level. And once again as we have mentioned, this as most of the information hidden in the lower levels is not apparent to the algorithm yet, filtering can cause neglecting some important paths of the tree.

For instance at this point, between the node pairs, only a similarity relation between the predicates *literacy_rate* and *social_welfare* is defined. The algorithm continues expanding, via selecting the *middle set* with the highest similarity value at the current value as the *decision middle set*. So in this example, at the first level $M_{ds} = \{ (V_{sw}, V_{lr}), (V_{hp}, V_{ID}) \}$ or $M_{ds} = \{ (V_{sw}, V_{lr}), (V_{es}, V_{ID}) \}$ as they prove to be the best of the M_s sets.

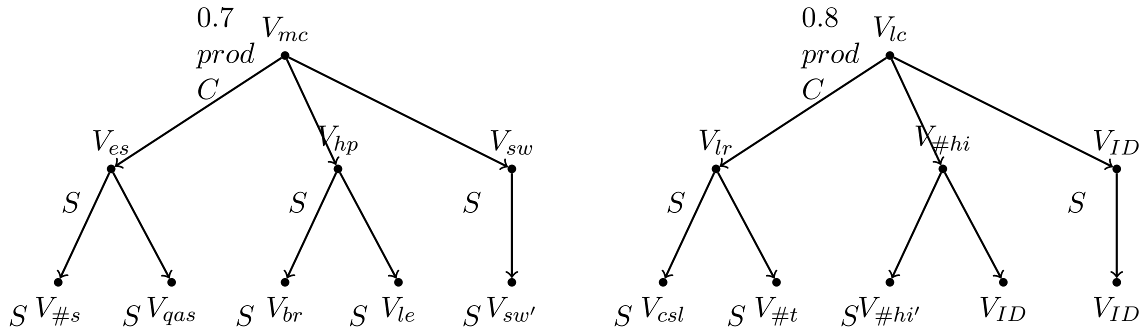


Figure 1.18: Predicate trees after two levels of expansion in *SBM*

However this local optimization approach eliminates other fruitful pair combinations such as *social_welfare* and *educated_society*. In *Figure 6* this pair corresponds to the left subtrees of the main predicate trees. These prove to be the most similar subconcepts of the two as there are two leaf node pairs for which the similarity relation is defined via values *0.9* and *0.85*.

All in all, wrong choice for filtering causes comparing wrong pairs of subpredicates in the end and thus a distorted final similarity degree between the predicates of interest.

1.2.4 Shared Similarity Concepts

Another shortcoming of the *SBM* approach is that since it does a one to one mapping between the subconcepts of the main predicates, it can not utilize the information given in the knowledge base completely where a particular concept is included in more then one similarity relations.

Example 1.2.4. Let us observe the following simple program:

$$\begin{array}{ll}
touristic\_place: & (Land) \\
nice\_destinationt: & (Land) \\
cultural\_venues: & (Sight) \\
natural\_wonders: & (Sight) \\
many\_sights: & (Sight) \\
good\_weather: & (Temperature)
\end{array}$$

$$\begin{array}{llll}
touristic\_place(X) & \overset{1.0,.}{\longleftarrow} & prod & cultural\_venues(X), natural\_wonders(X). \\
nice\_destination(X) & \overset{1.0,.}{\longleftarrow} & prod & good\_weather(X), many\_sights(X).
\end{array}$$

$$Sim(cultural\_venues, many\_sights) = 0.7$$

$$Sim(natural\_wonders, many\_sights) = 0.7$$

In the evaluation step, since both of the similarity degrees are equal the algorithm will conclude that there are two optimal M_{ds}. It will either conclude $M_{ds} = \{\ (V_{cv}\ ,\ V_{ms}\ )\ \}$ and the pair $(V_{nw}\ ,\ V_{gw}\ )$ to be incomparable, or the counter case where $M_{ds} = \{\ (V_{nw}\ ,\ V_{ms}\ )\ \}$ and the pair $(V_{cv}\ ,\ V_{gw}\ )$ to be incomparable. In both cases the algorithm is bound to miss on the similarity relations and thus the evaluation algorithm can not utilize that data.

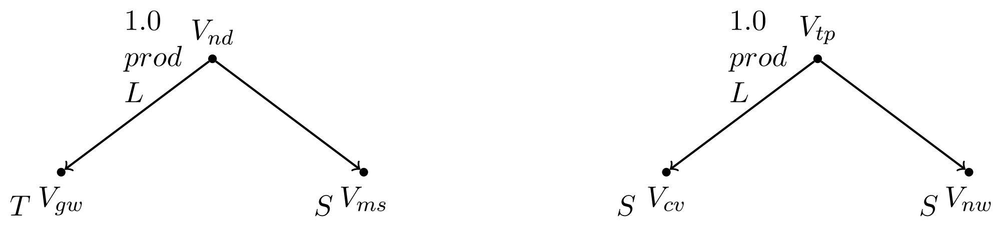


FIGURE 1.19: Predicate trees for *nice_destination* and *touristic_place*

In the case for which the proximity of similarity of the relations were different, that time the algorithm would simply neglect the smaller one. Thus again half of the information in the knowledge base would be lost without chance of recovery.