Component Matching Using Fuzzy Logic

Zhang Ting
Prof. Giovanni De Micheli
tingz, nanni@cs.stanford.edu
Computer System Lab
Stanford University

March 4, 2000

– Typeset by FoilTeX – Informal Presentation

abstract

We present an automatic component matching system in JavaCAD framework. The system transforms design specification from XML representation to document object model (DOM) and apply fuzzy logic scoring and aggregation algorithm on the tree-structured model to compute component ranking. Also this system provides an approach to address functionality matching problem using result-checking techniques.

Introduction

- IP Component Matching
- Component Matching Architecture (In JavaCAD Framework)
- XML Representation of IP Component Specification.
- Fuzzy Logic Matching Algorithm.
- Result-Checking Functionality.
- Experiment
- Conclusion

IP Component Matching

1. IP Reuse is needed
   (a) to reduce circuit design complexity
   (b) to speed up design process
2. IP Component Matching
   (a) to facilitate IP reuse
3. Our Contribution
   (a) XML representation for IP specification as interface between IP provider and IP user
   (b) Application of fuzzy logic to find out match component
   (c) Provide result-checking for functionality identification

Architecture

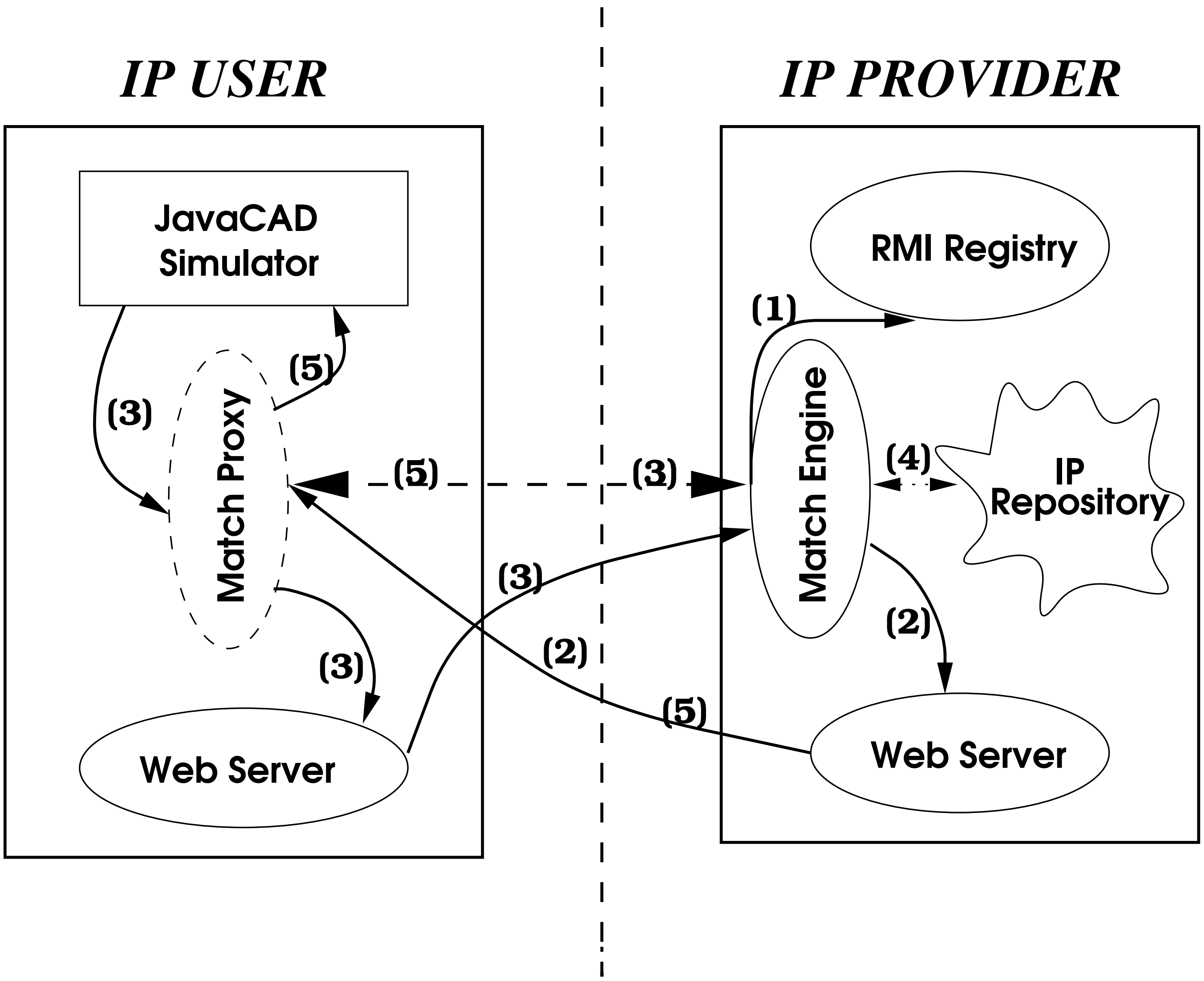


Figure 1: IP Component Matching Architecture

Work Flow

1. *Match Engine* Registering RMI service.

2. Client (IP user) ships the virtual proxy of *Match Engine* to its location using HTTP through webserver.

3. Constructing final query objects by applying specific fuzzy scoring methods and aggregation algorithms.

4. Retrieving IP components from IP repository and rank them according to proximity.

5. Behavioral level codes are wrapped for simulation.

XML Representation

1. SGML expressiveness on structured documents without complexity.

2. Structured data are easy to process, avoiding ad hoc code.

3. Next generation of web language with stylistic customization.

4. Semi-structured data organization model (vs. the traditional relational one).

Example: Data Type Definition(DTD)

```
<!-- DTD for ip_repository version 1.0 -->
<!-- parameter entities: -->
<!ENTITY % global_id "ID ID #IMPLIED">
<!-- matching scorer declaration:-->
<!ELEMENT ip_scorer (parameter)*>
<!ATTLIST ip_scorer
  name  CDATA  #REQUIRED>
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
  type  CDATA  #REQUIRED
  value  CDATA  #REQUIRED>
<!-- value declaration:-->
<!ELEMENT value (#PCDATA)>
<!-- ip specification declaration:-->
<!ELEMENT ip_specification
  (ip_scorer?, area, power,
   throughput, portmap, description)>
<!ATTLIST ip_specification
  %global_id;>
<!ELEMENT area (ip_scorer?, value)>
```

```
<!ELEMENT power (ip_scorer?, value)>
<!ELEMENT throughput (ip_scorer?, value)>
<!ELEMENT portmap (ip_scorer?, port+)>
<!ELEMENT port (ip_scorer?)>
<!ATTLIST port
  %global_id;
  name  CDATA  #REQUIRED
  mode  (in | out | inout | clock)  #REQUIRED
  size  CDATA  #REQUIRED>
<!ELEMENT description (ip_scorer?, value)>
```

Table 1. DTD Sample for IP Repository

Example: XML Representation of IP Repository

```
<?xml version='1.0' encoding='osi-8' standalone='no'?>
<!DOCTYPE ip_repository
          SYSTEM "http://.../ip_repository.dtd">
<!--  A SAMPLE of IP Component Repository  -->
<ip_repository>
  <!-- logistic information: eg creator, time etc. -->
  <repository_header
    repository_creator="Zhang Ting" ...
  </repository_header>
  <repository_body>
    <ip_component ID="ip1">
      <!-- logistic/taxonomy information: -->
      <ip_header
        name="Huffman Encoder"
        category="Image Processor"
        ...>
      </ip_header>
      <ip_specification ID="s1">
        <area>
```

```
      <ip_scorer name="NumericFuzzyScorer">
        <parameter type="double" value="1">
        </parameter>
        ...
      </ip_scorer>
      <value>9.5</value>
    </area>
    <power>6.2</power>
    <throughput>4.0</throughput>
    <portmap>
      <port ID="p1" name="difference"
            mode="in" size="8">
      </port>
      ...
      <port ID="p6" name="bitstream_produced"
            mode="out" size="1">
      </port>
    </portmap>
    <description>Sample of Huffman Encoder
    </description>
  </ip_specification>
  <ip_implementation
    code_level="Behavior"
```

```
        code_type = "Java"
        code_base = "javacad...HuffmanEncoder">
      </ip_implementation>
    </ip_component>
    ...
  </repository_body>
</ip_repository>
```

Table 2. XML Representation of Sample IP Repository

Example: Verilog vs XML Interface Representation

```
<!-- Verilog Code -->
...
module fibNumberGen(startingValue, flag, numConsumed,fibNum);
        input    [15:0]   startingValue;
        input             flag;
        output            numConsumed;
        output   [15:0]   fibNum;
...

<!-- XML Code -->
...
        <portmap>
          <port ID="in1" name="startingValue"
                mode="in" size="16">
          </port>
          <port ID="in2" name="flag"
                mode="in" size="1">
          </port>
          <port ID="out1" name="numConsumed"
```

```
                  mode="out" size="1">
            </port>
            <port ID="out2" name="fibNum"
                  mode="out" size="16">
            </port>
        </portmap>
...
```

Table 3. Example: Verilog vs XML Interface Representation

Matching Using Fuzzy Logic

1. Why Fuzzy Logic

2. Fuzzy Object

3. Atomic Query

4. Conjunction Aggregation

5. Weighted Aggregation

6. Fagin Aggregation

7. Matching Algorithm

Why Fuzzy Logic

1. Neural Net: Fixed pattern matching, very high complexity
2. Baysian Network: Causal reasoning, high complexity
3. Fuzzy Logic: approximate matches, low complexity

Fuzzy Object

Following is a geometric representation of a fuzzy number.

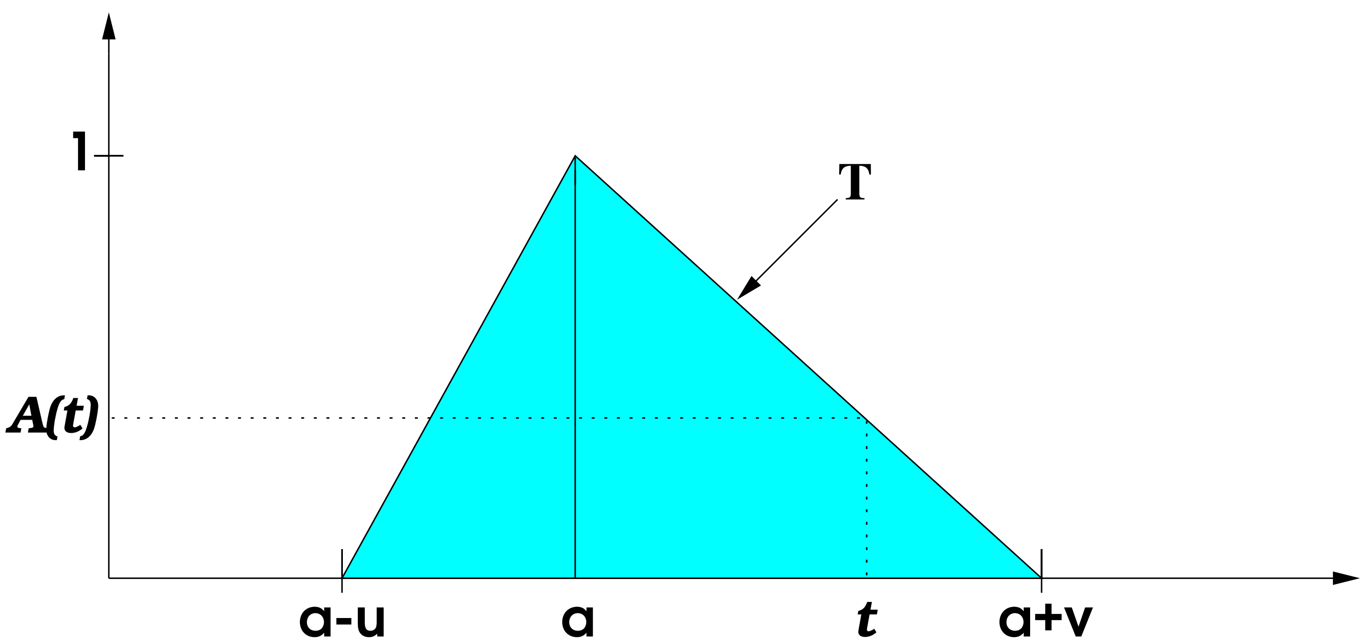


Figure 2: Triangular Fuzzy Number

$$\mathcal{A}(t) = \begin{cases} 1 - (a-t)/u & \text{if } a - u \le t \le a \\ 1 - (t-a)/v & \text{if } a \le t \le a + v \\ 0 & otherwise \end{cases}$$

Similarly, we can have fuzzy vector, region and functions.

Atomic Query

It is to score atomic attribute. Component attributes are scored according to specification which is represented as a fuzzy object.

Example: Specification: Fuzzy Number 20, u=v=2

$$
\begin{array}{ll}
\text{Value } 21.0 & \text{Score } = 0.5 \\
\text{Value } 20.5 & \text{Score } = 0.75 \\
\text{Value } 20.2 & \text{Score } = 0.9
\end{array}
$$

Conjunction Aggregation

Given a aggregation function A, we like to see the following properties:

$$A(x, y) = A(y, x) \qquad (1)$$

$$A(A(x, y), z) = A(x, A(y, z)) \qquad (2)$$

$$A(x, y) \leq A(u, v) \text{ given } x \leq u \text{ and } y \leq v \qquad (3)$$

$$A(x, 1) = x \text{ and } A(x, 0) = 0 \qquad (4)$$

The four attributes are in turn called symmetry, associtivity, monotonicity and existence of unit. We call it $\mathcal{T}$-norm.

Example:

1. Minimum

$$\mathcal{MIN}(x, y) = min\{x, y\}$$

2. Lukasiewicz

$$A(x, y) = max\{x + y - 1, 0\}$$

3. Probabilistic

$$\mathcal{PAND}(x, y) = xy$$

4. Weak

$$\mathcal{WEAK}(x, y) = \begin{cases} min\{x, y\} & \text{if } max\{x, y\} = 1 \\ 0 & otherwise \end{cases}$$

Weighted Aggregation

Given an aggregation function A, a weight vector $\vec{W}$, we like to see the following properties of weighted aggregation function $A_{\vec{W}}$

$$A_{\vec{W}}(x_1, \cdots, x_n) = A(x_1, \cdots, x_n) \quad \text{given } w_1 = w_2 = \cdots = w_n \quad (5)$$

$$A_{\vec{W}}(x_1, \cdots, x_n) = A_{\vec{W} \setminus w_i}(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n) \quad \text{given } w_i = 0 \quad (6)$$

$$A_{\vec{W}}(x_1, \cdots, x_n) \text{ is continuous with respect to } \vec{W} \quad (7)$$

Fagin Aggregation Function

Given an order weight vector $\vec{W} = (w_1, w_2, \cdots, w_n)$ where $w_n \leq w_{n-1} \leq \cdots \leq w_1$ and unweighted aggregation function $A(x_1, \cdots, x_n)$, the Fagin aggregation $A_{\vec{W}}$ is

$$
\begin{aligned}
A_{\vec{W}}(x_1, \cdots, x_n) = & \\
& (w_1 - w_2) \times A(x_1) + \\
& 2 \times (w_2 - w_3) \times A(x_1, x_2) + \\
& 3 \times (w_3 - w_4) \times A(x_1, x_2, x_3) + \\
& \cdots + \\
& n \times w_n \times A(x_1, x_2, \cdots, x_n) \quad (8)
\end{aligned}
$$

Let

$$w_{n+1} = 0$$

$$\vec{X}_i \stackrel{\text{def}}{=} (x_1, x_2, \cdots, x_i)$$

we have a concise format:

$$A_{\vec{W}}(\vec{X}_n) = \sum_{i=1}^{n}(i \times (w_i - w_{i+1}) \times A(\vec{X}_i)) \qquad (9)$$

Tree Structure Matching

Matching is essentially to find out the similarity between two DOM trees.

In general, a matching is processed at three steps beginning at leaf nodes and backing up to the root.

1. atomic scoring at leaf nodes.
2. conjunction aggregation at intermediate nodes.
3. weighted aggregation at intermediate nodes.

In implementation, the last two steps execute together for efficiency.

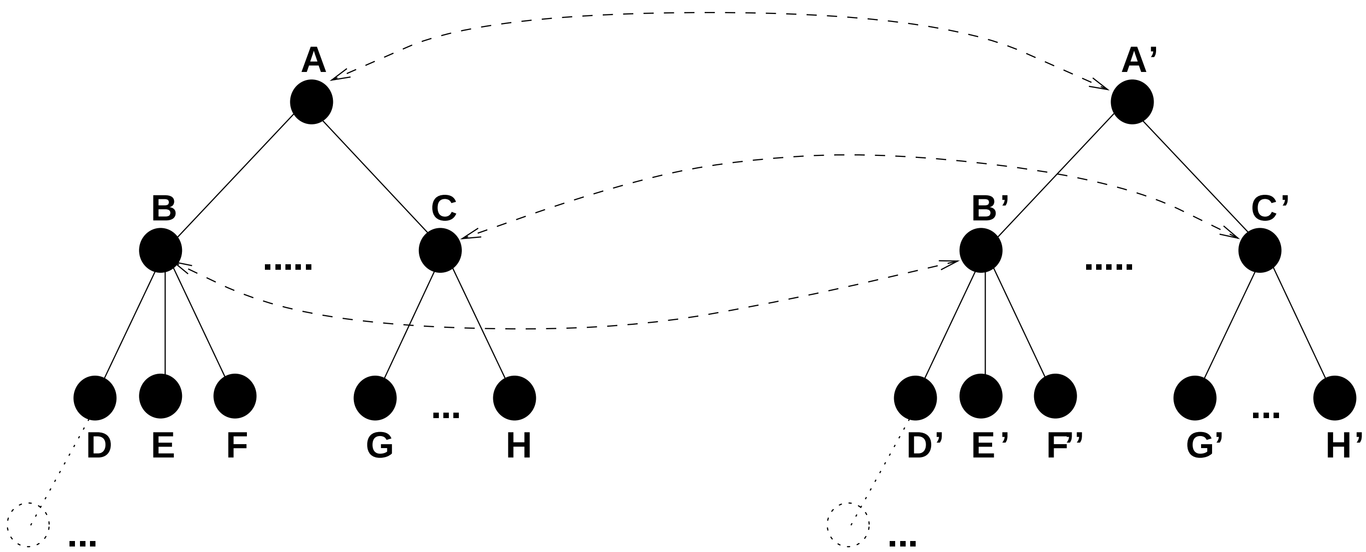


Figure 3: Fuzzy Tree Matching

Example: Unweighted Aggregation

Assumptions:

P: Power, T: Timing, A: Area.

Use triangular fuzzy number member function (in figure 2) as scoring function.

Use $u = v = 2$ for power scoring and $u = v = 1$ for the other two.

Use $\mathcal{MIN}$ as aggregation function.

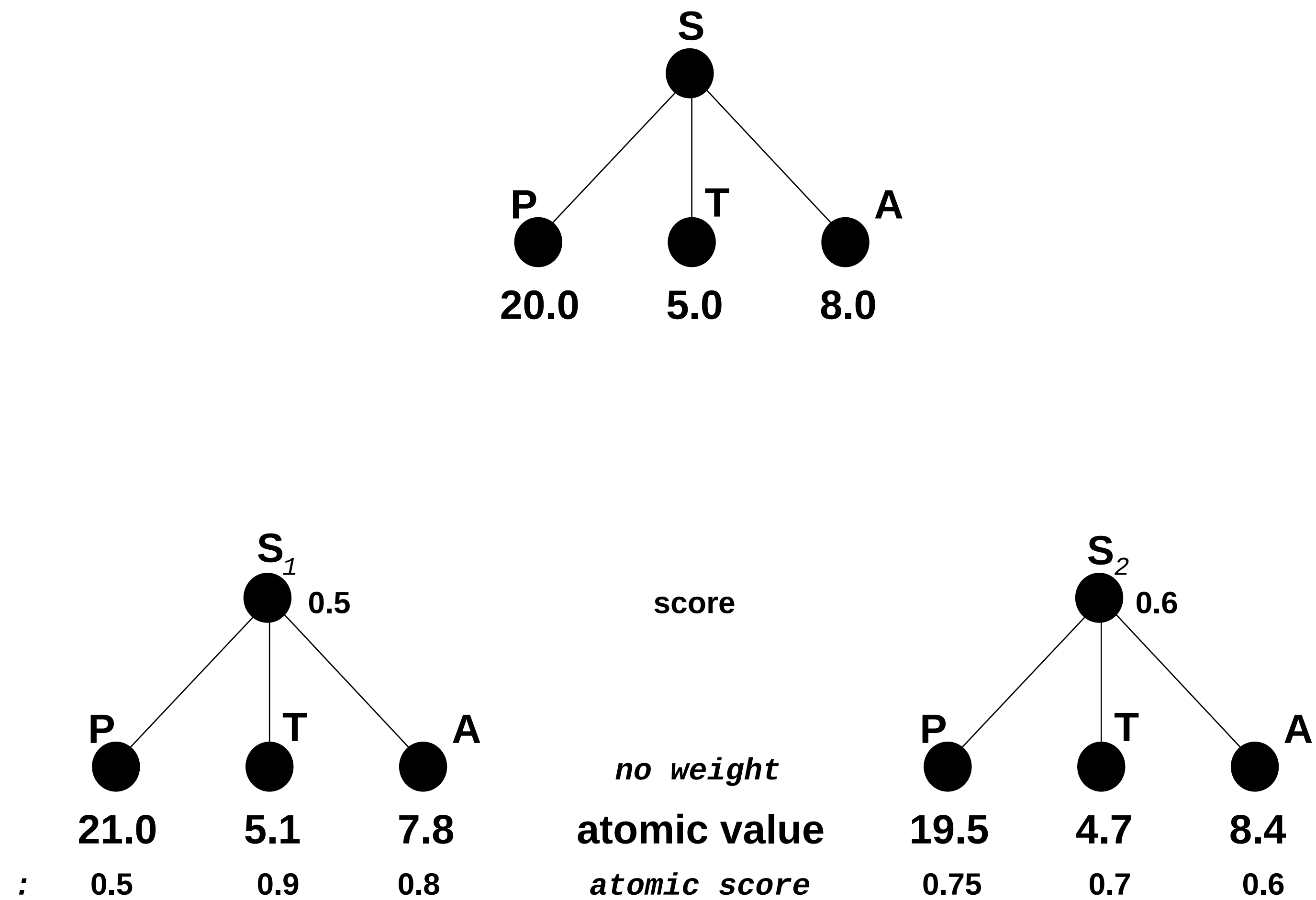
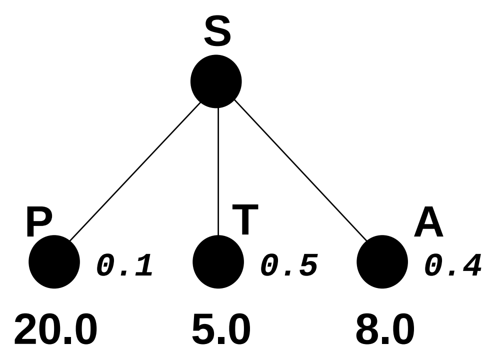


Figure 4: Example: Unweighted Aggregation

Example: Weighted Fagin Aggregation

Continued example: Due to big weight on timing, S_1 now has higher ranking than S_2.



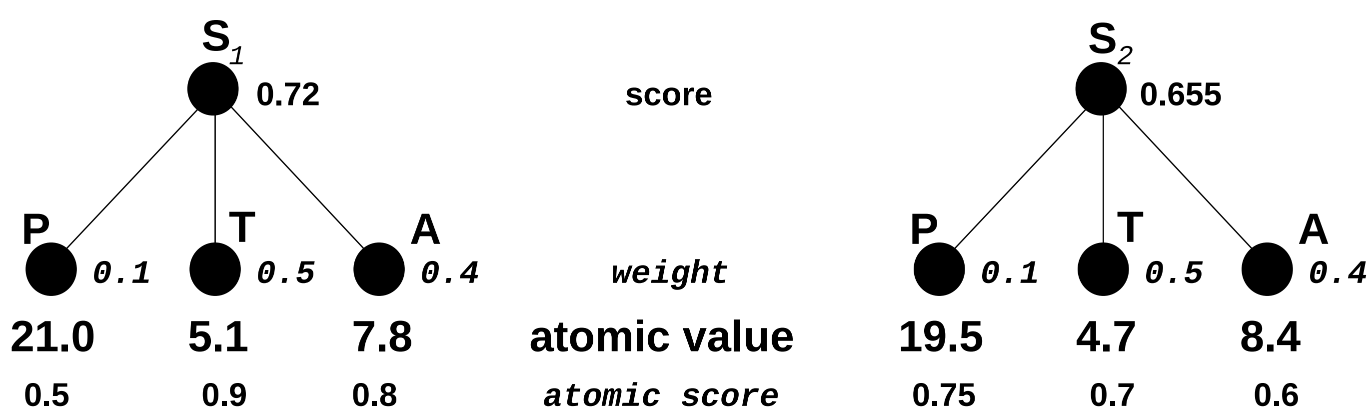


Figure 5: Example: Weighted Fagin Aggregation

Checking Functionality

1. Functionality checking is crucial in distinguishing different components with the same exterior parameters.

2. Information retrieval(Text based)
   Assumptions: Keywords are assumed to appear in the description text independently. Individual keyword matching score s is a function of the occurrence probability p and occurrence time n,

$$s = f(p, n) = 1 - p^n \tag{10}$$

   Naive approach, useful as auxiliary method, but generally text description can hardly reflect functionality precisely.

3. Formal Verification

(a) deductive approach.
Problem: PSPACE complexity on specification, generally undecidable

(b) model checking.
Problem: State explosion

Both are unsuitable for matching domain and suffer high complexity.

4. Testing
Problem: ad hoc approach, no solid theory foundation.

5. Result-checking
Reason program correctness using probability, no complexity overhead.

Result-Checking Functionality

A program $\mathcal{P}$ (implementing a functionality) and a checker $\mathcal{V}$ can be viewed as a interactive proof system.

$\mathcal{P}$ claims it can fulfill some functionality $\mathcal{F}$ and tries to convince $\mathcal{V}$ using the following challenge and response protocol:

1. $\mathcal{P}$ computes the result on input instance $\mathcal{I}$ and gives the result to $\mathcal{V}$.

2. $\mathcal{V}$ repeat this step in k iterations. In $i-th$ iteration, $\mathcal{V}$ generates new input instance $\mathcal{I}_i$ according to randomly generated bits and send $\mathcal{I}_i$ to $\mathcal{P}$.

3. $\mathcal{P}$ "figure out" what random bits generated by $\mathcal{V}$ in the k iterations. $\mathcal{P}$ does this by computing each $\mathcal{I}_i$ and examine the result.

4. $\mathcal{V}$ checks whether $\mathcal{P}$ made a good "guess". If it is, $\mathcal{V}$ acknowledges that $\mathcal{P}$'s claim is right.

Result-checking Graph Non-isomorphism

Here we gives an example on result-checking graph non-isomorphism.

Suppose $\mathcal{P}$ is the program which can determine graph non-isomorphism given any two graphs G_1 and G_2. Suppose $\mathcal{P}$ now claims that G_1 and G_2 are non-isomorphism. $\mathcal{P}$ needs to convince $\mathcal{V}$ that fact by passing the following challenges posted by $\mathcal{V}$:

1. Repeat k iterations:
   In i-th iteration, $\mathcal{V}$ generates two new input instances $\mathcal{I}_i^1 = (G_1, G')$ and $\mathcal{I}_i^2 = (G_2, G')$ according to the following rule:
   $\mathcal{V}$ throws a coin, if it is head, $G' = Pr(G_1)$. If it is tail, $G' = Pr(G_2)$. Here Pr is a random permutation keeping isomorphism.
   $\mathcal{V}$ sends $(\mathcal{I}_i^1, \mathcal{I}_i^2)$ to $\mathcal{P}$.

2. $\mathcal{P}$ "figure out" the results of coin throwing in the k iterations. An obvious way to do so is to compute

$\mathcal{I}_i^1$ and $\mathcal{I}_i^2$. If $\mathcal{I}_i^2$ is true, (i.e., G' is non-isomorphic to G_2), then $\mathcal{P}$ knows $\mathcal{V}$ got head round i. Similarly, if $\mathcal{I}_i^1$ is true, (i.e., G' is non-isomorphic to G_1), then $\mathcal{P}$ knows $\mathcal{V}$ got tail round i.

3. $\mathcal{V}$ checks whether $\mathcal{P}$ "guesses" correctly for all k results of coin throwing. If it is, $\mathcal{P}$ passes the test. Otherwise, $\mathcal{V}$ claims $\mathcal{P}$ is cheating.

It is obvious that if G_1 and G_2 are indeed isomorphic, $\mathcal{I}_i^1$ and $\mathcal{I}_i^2$ are identical instances up to random permutation. In other words, $\mathcal{P}$ gains no knowledge after seeing these two instances. Thus $\mathcal{P}$ can only figure out the coin throwing result by pure guess. It follows that $\mathcal{P}$ can only fool $\mathcal{V}$ with probability 2^{-k} after k iteration of challenges.

Result-checking Advantages/Disadvantages

1. Advantages
   (a) Computation Efficiency (Polynomial-time)
   (b) Transparent to implementation details.
   (c) Constructible by Reduction. (Beigel Theorem)

2. Disadvantages
   (a) Suitable in mathematical domain.
   (b) Non-trivial checker design

Experiment(Unfinished)

Proposal:

1. Choose a typical component, e.g., Huffman Encoder/Decoder.

2. Generate a large set of similar specifications by randomly tuning attributes of the original one.

3. Measure match preciseness in terms of tolerance of query specification variations.

It is based on the assumption that from the viewpoint of a particular specification, its homogeneous variety can simulate huge database of heterogeneous components.