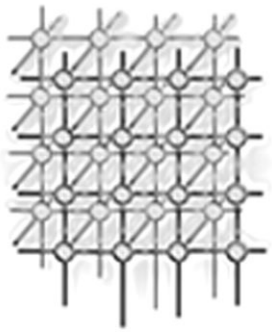
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2004; **16**:1453–1476
Published online 14 September 2004 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.870

Semantic Link Network Builder and Intelligent Semantic Browser



H. Zhuge*,†, R. Jia and J. Liu

China Knowledge Grid Research Group,
Key Lab of Intelligent Information Processing,
Institute of Computing Technology, Graduate School, Chinese Academy of Sciences,
Beijing, People's Republic of China

SUMMARY

Semantic Link Network (SLN) is a semantic Web model using semantic links—the natural extension of the hyperlink-based Web. The SLN-Builder is a software tool that enables definition, modification and verification of, as well as access to the SLN. The SLN-Builder can convert a SLN definition into XML descriptions for cross-platform information exchange. The Intelligent Semantic Browser is used to visualize the SLN and carry out two types of reasoning in browsing time: small granularity reasoning by chaining semantic links and large granularity reasoning by matching semantic views of SLN. With the help of the reasoning mechanism, the browser can recommend the content that is semantically relevant to the current browsing content, and it enables users to foresee the end-side content of a semantic link chain. This paper presents the design and implementation of the SLN-Builder and the intelligent semantic browser as well as key algorithms. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: algorithm; browser; matching; reasoning; semantic link; Semantic Web

1. INTRODUCTION

Since the invention of the World Wide Web, more and more people have become reliant on it to share information by publishing and browsing Web pages, which also leads to exponential growth of Web pages. This phenomenon makes it difficult to accurately and effectively locate, access and maintain Web information, although many efforts have been made to resolve the issue [1–5]. HTML-based Web pages are designed to be read, but the information expressed cannot be understood easily by application software like search engines. It is difficult, therefore, for the current Web to support intelligent services [6]. The Semantic Web is proposed to resolve this issue.

The concept of the Semantic Web was first proposed by Berners-Lee *et al.* in their book *Weaving the Web* [7]. Information is given well-defined meanings, better enabling computers and people to work

*Correspondence to: Hai Zhuge, China Knowledge Grid Research Group, Key Lab of Intelligent Information Processing, Institute of Computing Technology, Graduate School, Chinese Academy of Sciences, Beijing, People's Republic of China.
†E-mail: zhuge@ict.ac.cn

Contract/grant sponsor: National Grand Fundamental Research 973 Program; contract/grant number: 2003CB316901
Contract/grant sponsor: National Science Foundation of China

Received 18 July 2003
Revised 7 November 2003
Accepted 16 November 2003

together [8,9]. Ontology mechanisms play an important role in information sharing across various documents on the Web [6,10,11]. By specifying domain ontology, a document can be understood by applications and applications can also understand one another to a certain extent. Many markup languages are proposed to represent a document's content [12–15]. With the XML schema, the XML (http://www.w3.org/TR-REC-xml) can reflect the structured data, which is useful in raising the accuracy of information retrieval [16]. The Resource Description Framework (RDF) is a foundation for processing metadata using the object–property–value model, which also enables interoperability between Internet applications that work in cooperation. The RDF schema (RDFS) provides a means to define vocabulary, structure and constraints for expressing metadata of Web resources [17], thus expressing more formal semantic information than the RDF. The topic map approach uses such notions as topics, associations and occurrences to solve the issue of large quantities of unorganized information [18]. The XML Topic Maps (XTM) can represent certain structural semantics on the Web, therefore enabling more efficient Web retrieval [18]. Conceptual linking is an ontology-based open hypermedia mechanism that aims to allow documents to be linked via metadata describing their contents and hence to improve the consistency and breadth of linking Web documents at retrieval time and authoring time [19].

However, the machine-understandable semantics may not be suitable for the user to understand. The ideal semantic expression would be achieved by obtaining a balance between the machine-understandable semantics and human-understandable semantics. The most important factor contributing to the success of the current Web is its simplicity and readable style of Web pages.

A semantic link can be one of the following seven types: *cause-effective link*, *implication link*, *subtype link*, *similar-to link*, *instance link*, *sequential link* and *reference link* [20], which provide the basics for describing the external semantics. The Semantic Link Network (SLN) is a Semantic Web model using semantic links to extend the hyperlinks of the current Web. A SLN consists of semantic nodes and semantic links. A semantic node can be an atomic node (a piece of text or image) or a complex node (another SLN). As the semantic links are the natural and smooth extension of the hyperlink in semantics, the SLN can inherit the research results on hyperlink. The SLN can further make use of the characteristics of the semantic link in reasoning and semantic operations.

In addition to the ability to represent the semantic relationship between entity resources, the semantic links can also represent semantic relationships between abstract contents—concepts or the SLN consisting of concepts and semantic links. The abstract SLN reflects a kind of high-level semantics of the corresponding entity resources.

This paper presents the design and implementation of the SLN-Builder, which enables definition, modification and verification of, as well as access to the SLN. Furthermore, the intelligent semantic browser is described, which can visualize the SLN in a human-readable form and carry out reasoning and content recommendation during browsing. The matching algorithms and operations for implementing the intelligent semantic browser are also presented.

2. GENERAL ARCHITECTURE

The *SLN-Builder* and the *Intelligent Semantic Browser* are connected through XML descriptions—a set of semantic-link XML descriptions and document-content XML descriptions. Their relationship is depicted in Figure 1.

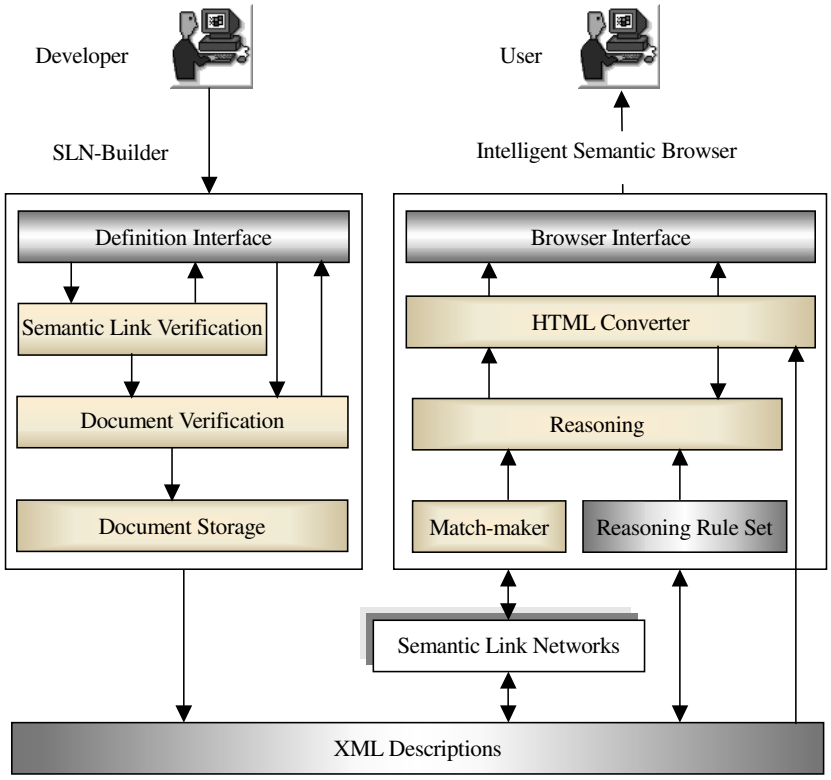


Figure 1. General architecture.

The SLN-Builder comprises four components: definition interface, semantic link verification, document verification and document storage. Using the definition interface, users can add and delete semantic links easily. The semantic link verification component checks whether a semantic link is stored in the correct format. The document verification component checks mismatching tags and cross-nesting tags in the whole document. The document storage component is responsible for saving the semantic link structures and document contents in the form of XML descriptions.

The intelligent semantic browser comprises five components: the match-maker, the reasoning rule set, the reasoning mechanism, the HTML Converter, and the browser interface. The match-maker presents a matching algorithm between SLNs or views of SLNs. The reasoning rule set contains the rules that describe characteristics of a semantic link. The match-maker is responsible for discovering the proper SLN that semantically matches the current browsing SLN. The reasoning rule set supports small granularity reasoning—semantic link chaining. Large granularity reasoning is based on the transitivity of the semantic inclusion relationship between views of SLN. The HTML Converter

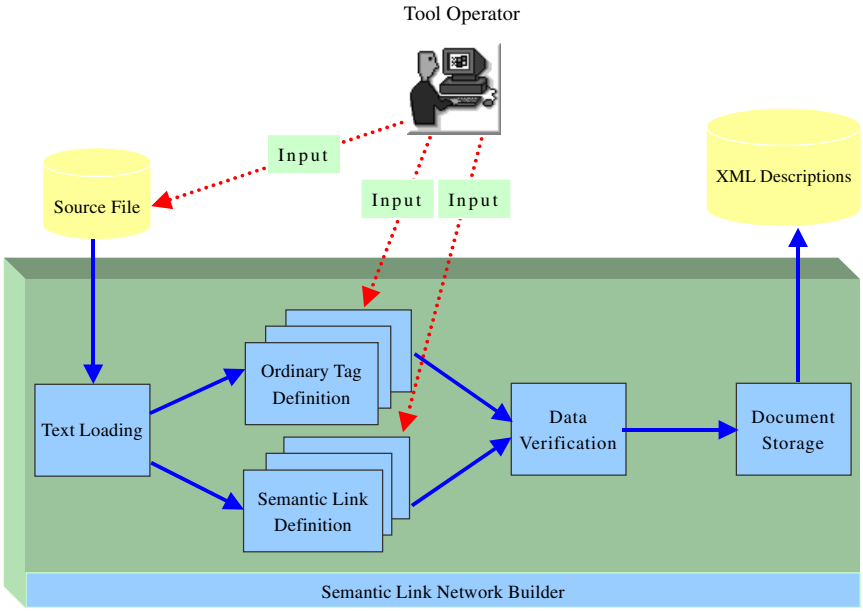


Figure 2. Workflow of the SLN-Builder.

transforms the XML description of the SLN and the reasoning result into a special form of Web page so that the ordinary browser can display it in a uniform way.

3. IMPLEMENTATION OF SLN-BUILDER

The workflow of the SLN-Builder is shown in Figure 2. The source input of the SLN-Builder is a text file, and the final output consists of the XML descriptions of the SLN corresponding to the source file. The user should first upload the text file, and then use the SLN-Builder to define tags and semantic links according to his/her understanding of the text. After that, the data verification component is executed to check the correctness of the semantic link definition and tag definition. Finally, the document storage component is executed to create the XML descriptions.

There are two types of tags: one is the ordinary tag that is frequently used in a document to reflect the structure of a document, e.g. 'ArticleTitle', 'Author', 'Abstract', 'Introduction' and so on, and the other is the self-definition tag. Figure 3 shows the interface of the tag definition. Figure 4 is the definition result of Figure 3. Figure 5 shows the interface for defining the semantic links.

According to whether a semantic link is a cross-document, it can be classified into two types: *intra-document link* and *inter-document link*, which can be created by the functions of the 'InternalLink'

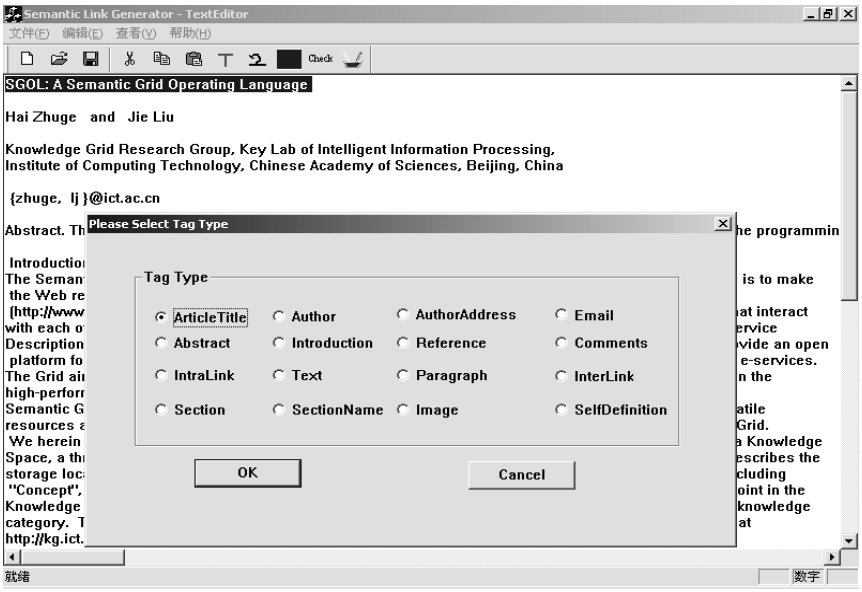


Figure 3. Interface of ordinary tag definition.

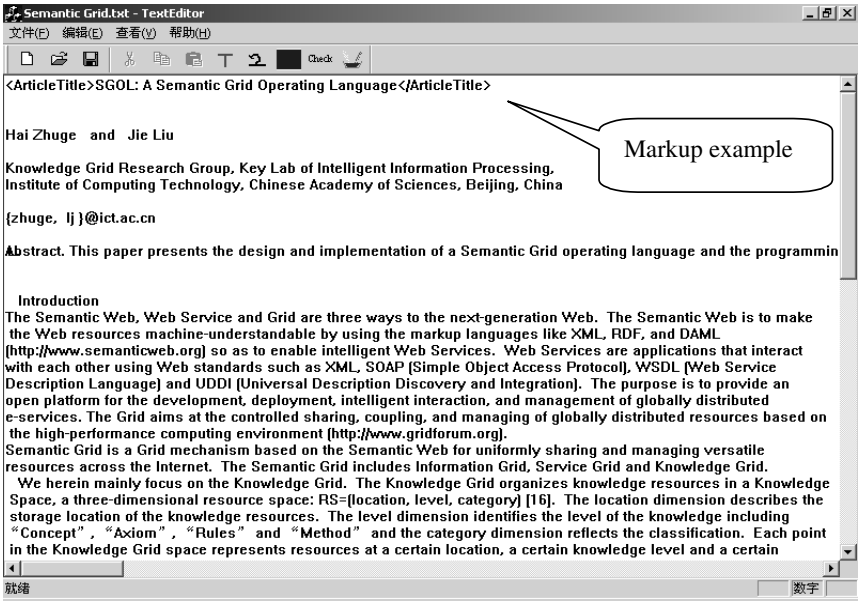


Figure 4. An example of a markup document using the definition tool.

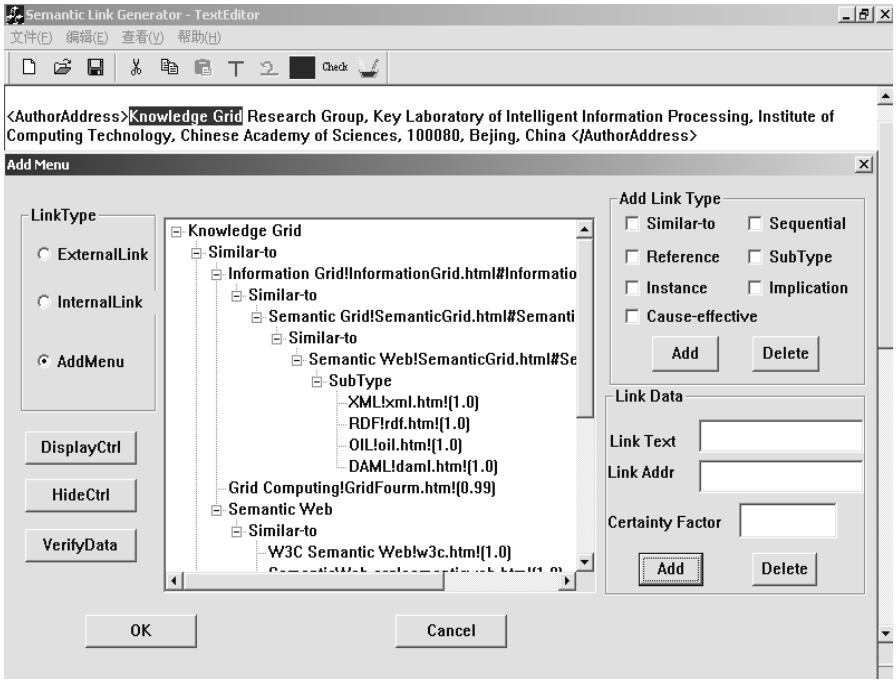


Figure 5. The interface for defining semantic links.

button and the 'ExternalLink' button of the interface shown in Figure 5. The definition result is stored in the following data structure.

```
<Entity PredecessorID = string PredecessorText = string>
    <SemanticRelation>
        <Similar-to>
            Link specification
        </Similar-to>
        <Sequential>
            Link specification
        </Sequential>
        <Reference>
            Link specification
        </Reference>
        <SubType>
            Link specification
        </SubType>
```


<Instance>
Link specification
</Instance>
<Cause-effective>
Link specification
</Cause-effective>
<Implication>
Link specification
</Implication>
</SemanticRelation>
</Entity>,

where the 'Link specification' takes the following form:

<Link>
<SuccessorID> ... </SuccessorID >
<SuccessorText> ... </SuccessorText>
</Link>
...
<Link>
<SuccessorID> ... </SuccessorID >
<SuccessorText> ... </SuccessorText>
</Link>

Each type of semantic link includes a predecessor node and a successor node identified respectively by the tags 'PredecessorID' and 'SuccessorID', which are maintained automatically by the SLN-Builder. The data structure represented by the tag 'Entity' describes all the semantic links that are emitted from the same predecessor node. It can contain one or more types of semantic link, such as 'Similar-to', 'Sequential', 'SubType' and so on. The tags 'PredecessorText' and 'SuccessorText' describe textual descriptions of the predecessor node and successor node, respectively. The connection of semantic links forms a semantic link chain that only ends at a hyperlink. If a semantic link is a hyperlink, its data structure is organized as follows:

<Entity PredecessorID = string PredecessorText = string>
<ExternalAddress> ... </ExternalAddress>
<CertaintyFactor> ... </CertaintyFactor>
</Entity>,

where the tag 'ExternalAddress' describes a hyperlink like #introduction or http://kg.ict.ac.cn/. 'CertaintyFactor' reflects the belief of the user on establishing the semantic link [21,22].

The data verification component comprises the semantic link verification function and the document verification function. Because semantic links are stored in the fixed data structure introduced above, if users make errors when defining semantic links, an invalid data structure may be formed. The semantic link verification function checks whether semantic links are saved in the correct format. The document verification function checks the correctness of the resulting document. It is mainly responsible for checking for the following two errors: (1) *checking tag mismatching*, which is to ensure that the

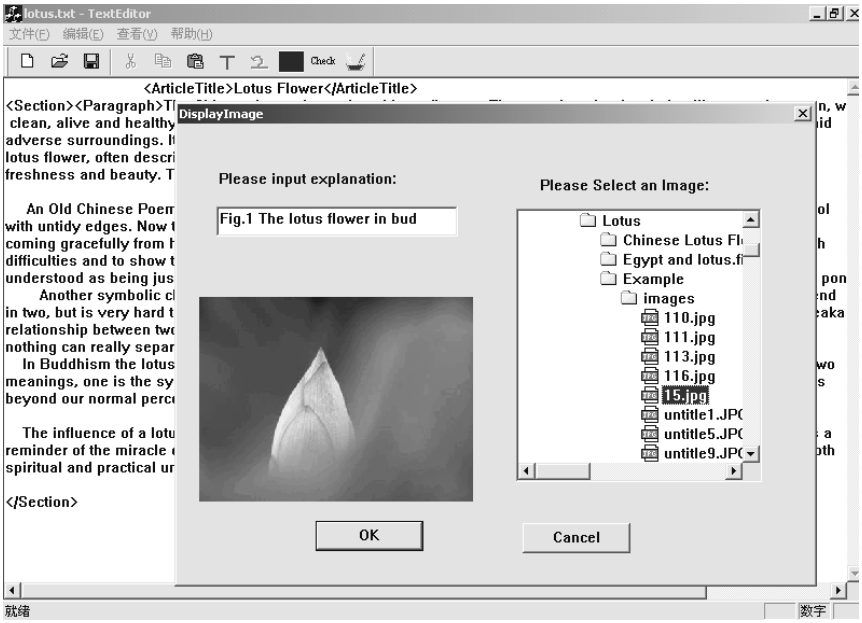


Figure 6. The interface of previewing an image.

resulting document is a well-formed document, i.e. if there is a tag like '<Mark>', then there must be a corresponding ending tag like '</Mark>'; and (2) *checking cross-nesting*, which means that if there is a tag 'T' contained within another tag pair, the ending tag '</T>' is also contained within the tag pair. A tag pair is composed of a tag and its corresponding ending tag.

The data storage component is responsible for converting the final semantic link definition into two types of XML descriptions: document-content XML descriptions and semantic-link XML descriptions. For searching semantic links easily, document-content XML descriptions and semantic-link XML descriptions are stored independently. The semantic links in semantic-link XML descriptions can be accessed through the ID number provided by document-content XML descriptions.

Besides supporting semantic links between document texts, the SLN-Builder also supports semantic links between images. Users can use the SLN-Builder to embed images and establish semantic links between images [23]. Figure 6 shows an interface for previewing an image when defining a semantic link.

4. MATCHING BETWEEN SLNs

The algorithm for matching between SLNs is the basis for determining the inclusion relationship between SLNs. In the following discussion, we assume that there are no repeated semantic links between the same node pair and that both vertex set and edge set of a SLN are not empty.

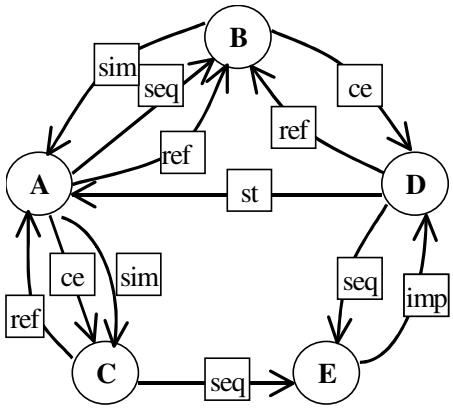


Figure 7. A semantic link network G.

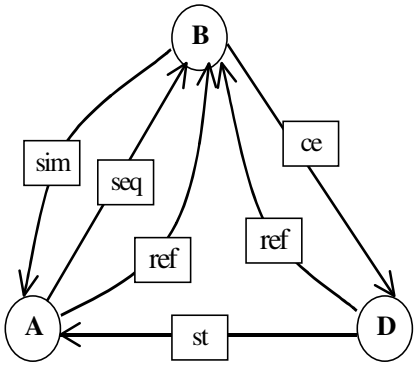


Figure 8. A full induced sub-semantic network of G $G_v(V)$.

Definition 1. Let $G = (V, E)$ be a SLN, $G' = (V', E')$ be a subgraph of $G = (V, E)$. For every edge $e = e(\mathrm{x}, \mathrm{y}) \in E$, if $x \in V'$ and $y \in V'$, then $e \in E'$, we call $G' = (V', E')$ a *full induced sub-semantic-graph* of G with vertex set V', denoted as $G_V(V')$.

Definition 2. A SLN G can be expressed in a SLN-*matrix* denoted as $M(G)$. Every element in $M(G)$ is an empty set or a set of semantic factors in {*ce*, *imp*, *st*, *sim*, *ins*, *seq*, *ref*}. The node sequence of rows and columns of $M(G)$ is the same.

Take the SLN G in Figure 7 as an example, $G_V(V')$ shown in Figure 8 is the full induced sub-semantic graph of G with $V' = \{\mathrm{A}, \mathrm{B}, \mathrm{D}\}$. The *SLN-matrix* of G is $M(G)$ (see Figure 9), which is

$$M(G) = \begin{pmatrix} null & \{seq, ref\} & \{ce, sim\} & null & null \\ \{sim\} & null & null & \{ce\} & null \\ \{ref\} & null & null & null & \{seq\} \\ \{st\} & \{ref\} & null & null & \{seq\} \\ null & null & null & \{imp\} & null \end{pmatrix}$$

Figure 9. The SLN-matrix of the SLN G.

composed of five columns and five rows, i.e. A, B, C, D and E in sequence. The *null* element in $M(G)$ means empty set.

The relationship between an order pair of SLNs $\langle G, G' \rangle$ ($G = (V, E)$ and $G' = (V', E')$) includes five types: *intersection, empty, equal, inclusion* and *inverse inclusion* as follows

(a) *Intersection*: there is at least one element that is contained in both E and E'.
(b) *Empty*: the intersection of E and E' is an empty set.
(c) *Equal*: every element in E is also in E', and every element in E' is also in E.
(d) *Inclusion*: every element in E is also in E'.
(e) *Inverse inclusion*: every element in E' is also in E.

The relationship between G and G' can be determined according to the following steps.

(1) Let $V_{\text{int}} = V \cap V'$, and $G1$ and $G2$ be their full induced sub-semantic networks with vertex set V_{int}. Let $G1 = G1(V_{\text{int}}, E1) = G_V(V_{\text{int}})$ and $G2 = G2(V_{\text{int}}, E2) = G'_{V'}(V_{\text{int}})$. The relationship between $G1$ and $G2$ can be determined by algorithm *Rel_SLN_Vertex*, which is also suitable for determining the relationship between any two networks that have the same vertex set.
(2) According to the relationship between $G1$ and $G2$, the relationship between G and G' can be determined by algorithm *Rel_SLN*.

Definition 3 (subtraction operation of SLN-matrix). For two SLNs $G = (V, E)$ and $G' = (V', E')$, if $V = V'$, let $R(G)$ be the result matrix of subtraction operation, $R(G) = M(G) - M(G')$.

$$R(G_{i,j}) = M(G_{i,j}) - M(G'_{i,j}) = \bigcup_{i=1}^{3} W_i^k, k = 0, 1$$

where W_i^k can be obtained from Table I.

$R_{i,j}(G) = \{\text{“0”}\}$ means that there is at least one element that is contained in both $M_{i,j}(G)$ and $M_{i,j}(G')$. $R_{i,j}(G) = \{\text{“+”}\}$ means that there is at least one element that is contained in $M_{i,j}(G)$ but not in $M_{i,j}(G')$. $R_{i,j}(G) = \{\text{“−”}\}$ means that there is at least one element that is contained in $M_{i,j}(G')$ but not in $M_{i,j}(G)$.

Based on the above definitions, Algorithm 1 can be introduced to determine the relationship between two SLNs.

Table I. The value of W_i^k.

W_1^0	W_1^1	W_2^0	W_2^1	W_3^0	W_3^1
null	{"0"}	null	{"+"}	null	{"−"}

```
Rel_SLN_Vertex (SLN G1, SLN G2){
    // G1 = G1(V, E1) and G2 = G2(V, E2) are semantic link networks, M(G1) and M(G2) are
    n × n SLN matrices.
    Pre_Process(M(G1), M(G2)); // establish corresponding relationship between SLN nodes
    and the rows or columns of SLN-matrix and also establish the corresponding relationship
    between rows and columns of two SLN-matrices.
    Subtract(M(G1), M(G2), R(G)); // R(G) = M(G1) − M(G2)
    RtnStr = Result(R (G));
    return RtnStr; // the returned value belongs to {"intersection", "empty",
                   "equal", "inclusion", "inverse inclusion"}
}
```

Algorithm 1. The algorithm of *Rel_SLN_Vertex*.

```
Subtract(MATRIX M(G1), MATRIX M(G2), RESULT_MATRIX R(G)){
    //Precondition: the vertex set of G1 is the same as G2.
    //Input: M(G1) and M(G2) are two SLN-matrices
    //Output: R(G): R(G) = M(G1) − M(G2)
    Initialize(R(G)); // let every element of R(G) be null.
    if (V1 != V2) return;
    for every R_{i,j}(G){
        if (there exists an element that is contained in both M_{i,j}(G1) and M_{i,j}(G2))
            Add "0" to R_{i,j}(G);
        if (there exists an element that is only contained in M_{i,j}(G1) but not contained in M_{i,j}(G2))
            Add mark "+" to R_{i,j}(G);
        if (there exists an element that is only contained in M_{i,j}(G2) but not contained in M_{i,j}(G1))
            Add mark "−" to R_{i,j}(G);
    }
}
```

Algorithm 2. The algorithm of *Subtract*.

```
Result(RESULT_MATRIX R(G)){
    Zero = Plus = Sub = 0;
    for every R_{i,j}(G){
        if (Zero == 0 && "0" is contained in R_{i,j}(G)) Zero = 1;
        if (Plus == 0 && "+" is contained in R_{i,j}(G)) Plus = 1;
        if (Sub == 0 && "−" is contained in R_{i,j}(G)) Sub = 1;
    }
    if (Zero == 0) return "empty";
    if (Zero == 1 && Plus == 1 && Sub == 0) return "inverse inclusion";
    if (Zero == 1 && Plus == 1 && Sub == 1) return "intersection";
    if (Zero == 1 && Plus == 0 && Sub == 0) return "equal";
    if (Zero == 1 && Plus == 0 && Sub == 1) return "inclusion";
}
```

Algorithm 3. The algorithm of *Result*.

```
Rel_SLN(SLN G1, SLN G2){
    if (V1 == V2)    return Rel_SLN_Vertex(G1, G2);
    if (V1 ⊂ V2) {
        Let G3 = G3(V1, E3) = G2_{V2}(V1);
        Rtn = Rel_SLN_Vertex(G1, G3);
        if (Rtn == "empty" || Rtn = "inclusion") return Rtn;
        if(Rtn == "equal") return "inclusion";
        if((Rtn == "intersection") || (Rtn == "inverse inclusion")) return "intersection";
    }
    if (V1 ⊃ V2) {
        Let G3 = G3(V2, E3) = G1_{V1}(V2);
        Rtn = Rel_SLN_Vertex(G2, G3);
        if (Rtn == "empty") return Rtn;
        if((Rtn == "inclusion") || (Rtn == "equal")) return "inverse inclusion";
        if((Rtn == "intersection") || (Rtn == "inverse inclusion")) return "intersection";
    }
    Let V_int = V1 ∩ V2;    // let V_int be the intersection of set V1 and V2
    if V_int == Φ return "empty";
    if V_int ≠ Φ{
        Let G3 = G3(V_int, E3) = G1_{V1}(V_int) and G4 = G4(V_int, E4) = G2_{V2}(V_int);
        Rtn = Rel_SLN_Vertex(G3, G4);
        if Rtn == "empty" return "empty";
        if (Rtn != "empty") return "intersection";
    }
}
```

Algorithm 4. The algorithm of *Rel_SLN*.

Based on Algorithm 1, Algorithm 4 determines the relationship between any two SLNs $G1 = G1(V1, E1)$ and $G2 = G2(V2, E2)$ and returns a value belonging to {"intersection", "empty", "equal", "inclusion", "inverse inclusion"}.

According to the relationship returned by Algorithm 4, we can find the SLN that contains richer semantics. For example, if the returned value is 'inclusion', then $G2$ has richer semantics; if the returned value is 'inverse inclusion', then $G1$ contains richer semantics. The proof of the correctness of the basic algorithm *Rel_SLN_Vertex* will be given in Appendix B.

5. MATCHING BETWEEN HYPER-SLNs

A Hyper-SLN includes the complex node—a node itself is a SLN. Algorithm 4 only applies to the SLN consisting of atomic nodes. Algorithm 5 is used to determine the relationship between Hyper-SLNs. *Rel_HyperSLN* first uses *Rel_SLN* to determine the relationship between SLNs by viewing all the complex nodes as atomic nodes, and then determines the relationship between corresponding complex nodes by calling *Rel_HyperSLN* recursively.

Supposing the names of nodes in a SLN are different from each other in the sense of ontology, Algorithm 6 establishes the correspondence relationship between the nodes in $G1$ and $G2$.

6. SLN REASONING

The SLN reasoning can be carried out in either large granularity or small granularity modes. Large granularity reasoning aims to discover the SLN with the richest semantics among a set of SLNs based on the matching algorithm *Rel_SLN*. For a given SLN set $S = \{G1, G2, \ldots, Gn\}$, Algorithm 4 can obtain a set of SLNs that have the relationship 'inclusion' or 'inverse inclusion'. Suppose that $\{G_{s1}, G_{s2}, \ldots, G_{sm}\}$ $(1 \leqslant si \leqslant n, 1 \leqslant i \leqslant m)$ is the matching result that satisfies $G_{s1} \subseteq G_{s2} \subseteq \cdots \subseteq G_{sm}$. So when users want to get semantic information from G_{si}, G_{sm} can be used to replace $G_{si} (i = 1, 2, \ldots, m-1)$ in applications. However, Algorithm 5 should be used to replace *Rel_SLN* for reasoning in case there are complex nodes in SLNs.

Small granularity reasoning means that a semantic link in a SLN can deduce a semantic link chain according to the chaining rules of the semantic links. A semantic link can be the following primitive types: cause-effective link (denoted as *ce*), implication link (denoted as *imp*), subtype link (denoted as *st*), similar-to link (denoted as *sim*), instance link (denoted as *ins*), sequential link (denoted as *seq*), and reference link (denoted as *ref*).

Reasoning rules of small granularity reasoning are constructed by the transitive characteristic and implication characteristic. The transitive characteristic applies to the same type reasoning. For example, if we have two semantic links d—$ce \rightarrow d'$ and d'—$ce \rightarrow d''$, we can get the reasoning result d—$ce \rightarrow d''$ according to the transitive characteristic of the cause-effective link. So the above reasoning process can be represented as the following reasoning rule: d–$ce \rightarrow d'$, d'—$ce \rightarrow d'' \Rightarrow d$—$ce \rightarrow d''$.

If the types of the two semantic links are different, the reasoning process is carried out according to the implication characteristic, for example, if we have two semantic links d—$ce \rightarrow d'$, and d'—$imp \rightarrow d''$, the following reasoning result holds: d—$ce \rightarrow d''$ according to the implication characteristic. This reasoning process can be represented as the following reasoning rule d—$ce \rightarrow d'$, d'—$imp \rightarrow d'' \Rightarrow d$—$ce \rightarrow d''$. However, several types of semantic links have the implication characteristic as introduced in [20].

```
Rel_HyperSLN(SLN G1, SLN G2){
    Pre_Process(G1, G2);
    String Rtn = Rel_SLN(G1, G2);// view complex nodes as atomic nodes.
    if ( (Rtn == "empty") || (Rtn == "intersection")) return Rtn;
    Let V1join = V2join = Vjoin = V1 ∩ V2;
    Suppose V1join = {V1j1, V1j2, ..., V1jm} and V2join = {V2j1, V2j2, ..., V2jm}; //V1ji = V2ji, i ∈ [1, m].
    if ( Rtn == "inclusion"){
      for every V1ji (i = 1, 2, ..., m){
        if ((V1ji is a complex node and V2ji is an atomic node) || (V1ji is an atomic node and
          V2ji is a complex node)) return "intersection";
        if (V1ji is a complex node and V2ji is a complex node){
          Let G3 = G3(V3, E3) be the SLN expanded by the complex node V1ji;
          Let G4 = G4(V4, E4) be the SLN expanded by the complex node V2ji;
          String subRtn = Rel_HyperSLN(G3, G4);
          if ((subRtn != "equal") && (subRtn != "inclusion")) return "intersection";
        }
      }// end for
      return "inclusion";
    }
    if ( Rtn == "inverse inclusion"){
      for every V1ji (i = 1, 2, ..., m){
        if ((V1ji is a complex node and V2ji is an atomic node) || (V1ji is an atomic node and
          V2ji is a complex node))    return "intersection";
        if (V1ji is a complex node and V2ji is a complex node){
          Let G3 = G3(V3, E3) be the SLN expanded by the complex node V1ji;
          Let G4 = G4(V4, E4) be the SLN expanded by the complex node V2ji;
          String subRtn = Rel_HyperSLN(G3, G4);
          if((subRtn!="equal")&&(subRtn !="inverse inclusion")) return "intersection";
        }
      }// end for
      return "inverse inclusion";
    }
    BOOL bEqual = bInc = bUnInc = FALSE;
    if ( Rtn == "equal"){
      for every V1ji, (i = 1, 2, ..., m){
        if ((V1ji is a complex node and V2ji is an atomic node) || (V1ji is an atomic node
          and V2ji is a complex node)) return "intersection";
        if (V1ji is a complex node and V2ji is a complex node){
          Let G3 = G3(V3, E3) be the SLN expanded by the complex node V1ji;
          Let G4 = G4(V4, E4) be the SLN expanded by the complex node V2ji;
          String subRtn = Rel_HyperSLN(G3, G4);
          if ((subRtn == "empty")||(subRtn == "intersection")) return "intersection";
          if ((!bEqual) && (subRtn == "equal")) bEqual = TRUE;
          if ((!bInc) && (subRtn == "inclusion")) bInc = TRUE;
          if ((!bUnImp) && (subRtn == "inverse inclusion")) bUnInc = TRUE;
        }
      }// end for
      if ((bEqual ) && (!bInc) && (!UnInc)) return "equal";
      if ((bInc) && (!bUnInc)) return "inclusion";
      if ((bUnInc) && (!Inc)) return "inverse inclusion";
    }
}
```

Algorithm 5. Algorithm *Rel_HyperSLN* inputs Hyper-SLNs $G1 = G1(V1, E1)$ and $G2 = G2(V2, E2)$, and then returns a string belonging to {"empty", "intersection", "equal", "inclusion", "inverse inclusion"}.


```
Pre_Process(SLN G1, SLN G2){
    For every node N in G1{
        if (there exists a corresponding node N′ in G2 whose name has the same meaning as
        N in the sense of ontology)
            Establish the corresponding relationship between N and N′;
    }
}
```

Algorithm 6. The algorithm of *Pre_Process*.

7. INTELLIGENT BROWSER

7.1. Implementation of reasoning mechanism

We use the following data structure to support reasoning:

<Entity PredecessorID = string PredecessorText = string>
<ExternalAddress> ... </ExternalAddress>
<CertaintyFactor> ... </CertaintyFactor>
SR
</Entity>,

where SR denotes the 'SemanticRelation' data structure described in Section 3, Entity = (ExternalAddress & CertaintyFactor) | SR, and 'ExternalAddress' is used to determine whether the semantic link is an inter-document link that supports inter-document reasoning.

The reasoning process starts from a semantic link whose successor node points to a hyperlink. For a specific semantic link l_1: d—*sim* $\rightarrow$ d' in an XML file, the reasoning process aims to find all the semantic links from the local XML descriptions. All these semantic links have the 'similar-to' relationship and they are deduced from d'. Using the tag 'SuccessorID' of d', the reasoning mechanism finds the data structure DT_1 in which all the semantic links take d' as their predecessor node. If there is no tag 'ExternalAddress' in DT_1, all the semantic links in DT_1 that have a 'similar-to' relationship are put into the stack *InDocStack*. These semantic links are marked as $l_2, \ldots, l_n$. If there exists the tag 'ExternalAddress' in DT_1, the destination address represented by 'ExternalAddress' is analyzed as follows: if the destination address does not end up with 'MENU' plus ID such as 'MENU1001', the semantic link cannot be used for reasoning and it should be added to the reasoning result set *RS*. If the destination address ends up with 'MENU' plus ID, such as 'knowledgegrid.html#knowledgeMENU1001', the following reasoning process is carried out. The reasoning mechanism extracts the information about the destination file name like 'knowledgegrid.html' and the ID like '1001' that identifies the predecessor node of a semantic link in the destination XML file according to the destination address of the hyperlink, then it puts the above information into stack *OutDocStack*. At the same time, l_1 is added to the reasoning result set *RS*. If *InDocStack* is not empty, it pops up the top element l_2. The reasoning process of $l_2$is the same as l_1. The reasoning process stops until *InDocStack* becomes empty. Then, *OutDocStack* pops up the top element for reasoning when it is not empty. The reasoning process of the top element is the same as l_1.

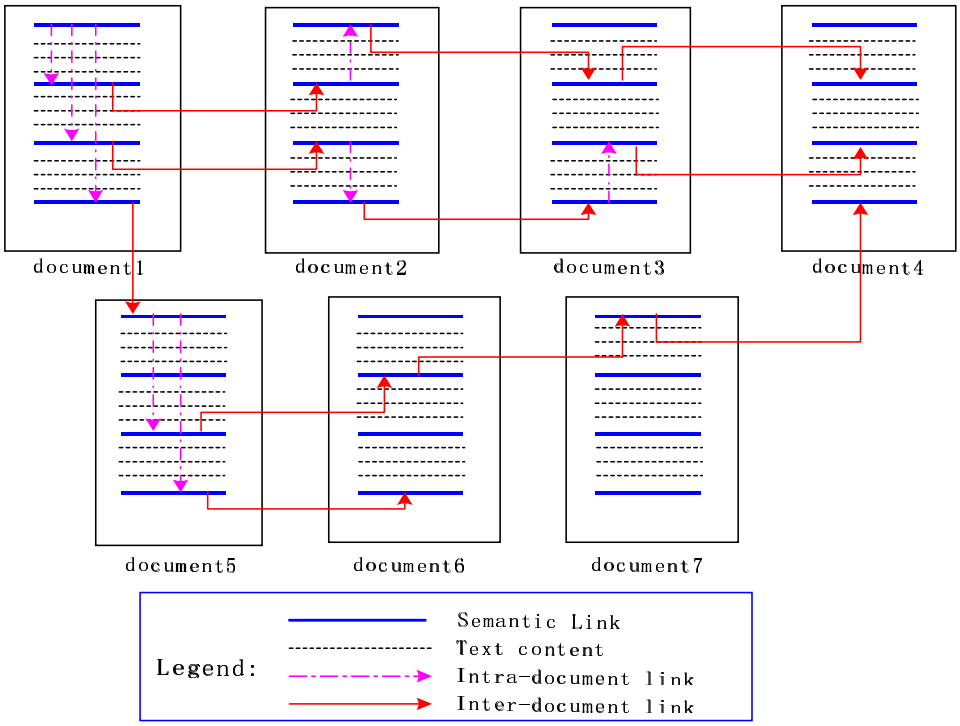


Figure 10. Sketch map of the inter-document and intra-document reasoning process.

The reasoning process stops until both *InDocStack* and *OutDocStack* become empty. All the semantic links deduced from d' of l_1 are included in the reasoning result *RS*. Figure 10 shows a sketch map of the reasoning process.

7.2. HTML Converter and browser interface

The HTML Converter is a component that converts XML descriptions into HTML files. It uses a template file that is responsible for parsing all tags in the XML files and converting document contents and semantic links into Web pages so that the ordinary Web browser can browse them. While browsing, the user can obtain rich semantics displayed by the converted Web pages shown in Figures 12 and 13.

The intelligent semantic browser provides users with three types of interface: the image-link-based interface, the non-reasoning-based interface and the reasoning-based interface. In case the XML descriptions contain some semantic links between images, the HTML Converter can convert them into Web pages containing image information. The intelligent semantic browser provides users with an image link interface as shown in Figure 11. In case a document defined by the SLN-Builder does not

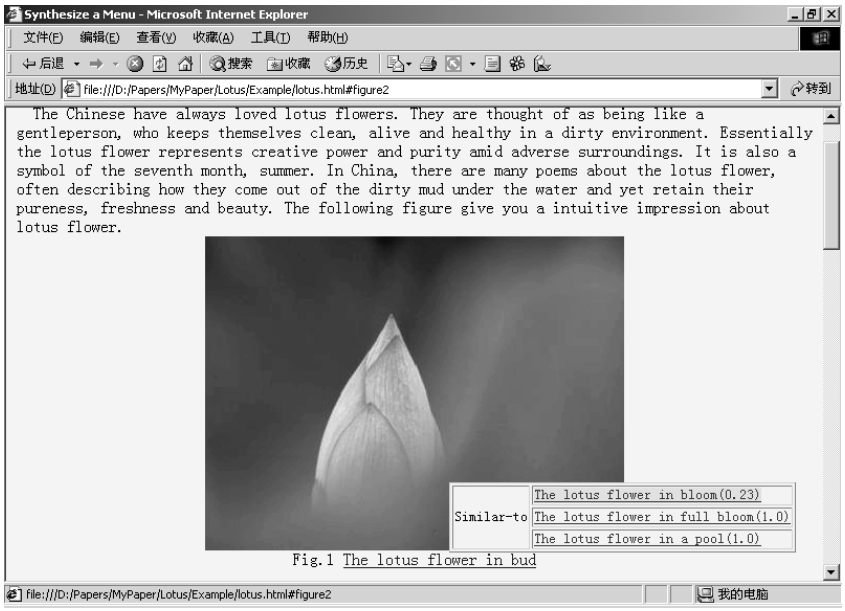


Figure 11. Interface that displays semantic links on image.

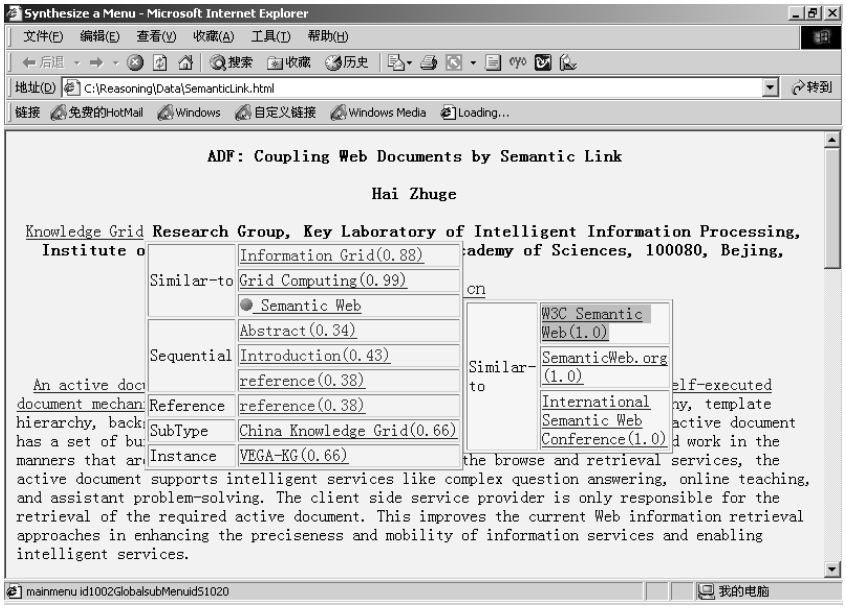


Figure 12. Non-reasoning interface.

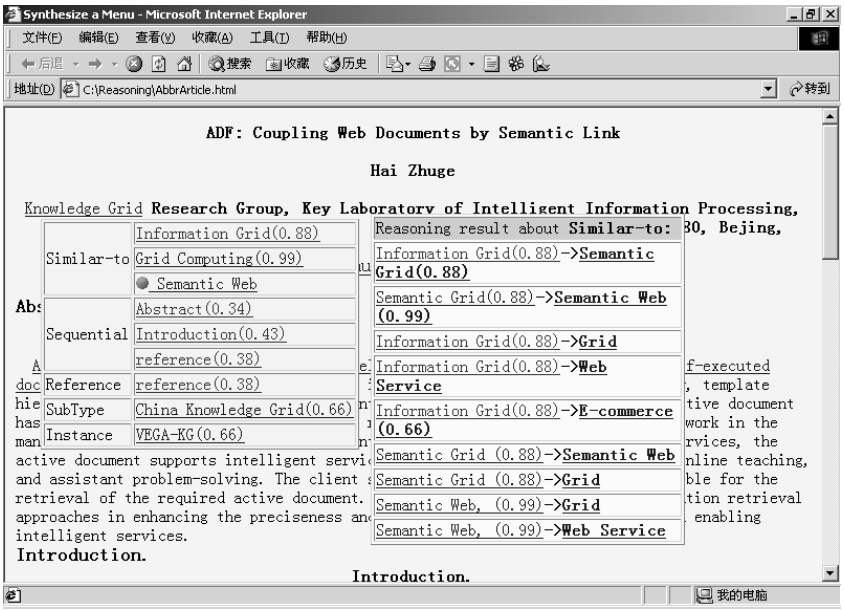


Figure 13. Reasoning interface.

contain any reasoning information, the Web pages converted by the HTML Converter will be displayed by a non-reasoning interface as shown in Figure 12. In case a document contains reasoning information, the Web pages converted by the HTML Converter will be displayed by a reasoning interface as shown in Figure 13.

8. CONCLUSION AND ONGOING WORK

The SLN is the natural and smooth extension of the Web's hyperlink network. This paper presents the implementation of the SLN-Builder and the Intelligent Semantic Browser. The SLN-Builder enables users to conveniently develop the semantic-link-based 'Web', which includes richer semantics than the hyperlink-based Web. The semantic links support intelligent, efficient and precise information service. The intelligent semantic browser can find the semantic rich SLN in a set of SLNs and provide the users with a corresponding semantic link chain when they browse a SLN. The large granularity reasoning is based on the SLN matching algorithms. The small granularity reasoning is carried out according to the transitive and implication characteristics of semantic links. In Appendix A, we present the approach to realize the union of two SLNs, which enables semantic integration and a single semantic image when browsing a large-scale SLN [24]. The proposed approach is a new attempt towards the semantic Web by means of the smooth extension of the Web and establishing the computing model. The research results on the hyperlink Web such as the link analysis and page rank algorithm have paved the road ahead for SLN due to such a smooth extension.

We have noticed that the representation of a large-scale SLN requires a large matrix. Although the SLN-matrix records the information on semantic links and not the entities of Web pages, a lighter representation mechanism is worth considering. One approach to solving this issue is to transform the sparse SLN-matrix into a special vector that contains no empty elements, and the other approach is to transform the sparse SLN-matrix into several small full SLN-matrices.

Ongoing work includes realizing more types of reasoning like analogical reasoning and abstraction-based SLN operations investigating the physical level implementation and the transformation between RDF and SLN, and applying the proposed approach to online education and e-science based on the peer-to-peer networking mode.

APPENDIX A. UNION OPERATION OF SLN

Union operation of SLNs is a kind of semantic integration, which can integrate small-granularity semantic components into a large granularity semantic component. The union operation is useful in forming a complete semantic image (i.e. '*single semantic image*' [24]) during browsing and reasoning. The algorithm *Union_SLN* that realizes the union operation is presented as follows.

Definition 4. For two semantic link networks $G1 = G1(V1, E1)$ and $G2 = G2(V2, E2)$, the union of $G1$ and $G2$ (denoted as $G1 \cup G2$) is also a SLN. Let $G3 = G3(V3, E3)$ be $G1 \cup G2$, $G3$ can be constructed by the following steps.

(1) View all the nodes in $G1$ and $G2$ as atomic nodes, $V3 = V1 \cup V2$ and $E3 = E1 \cup E2$.
(2) If a node $V1_c \in V1$ is a complex node, and $V1_c \notin V1 \cap V2$, then the SLN expanded by $V1_c$ constructs the SLN expanded by the complex node $V3_c$ (corresponding to $V1_c$) of $G3$.
(3) If a node $V2_c \in V2$ is a complex node, and $V2_c \notin V1 \cap V2$, then the SLN expanded by $V2_c$ constructs the SLN expanded by the node $V3_c$ (corresponding to $V2_c$) of $G3$.
(4) If a node $V_c \in V1 \cap V2$ is a complex node, let $G1_{Vc}$, $G2_{Vc}$ be the SLNs expanded by V_c in $V1$ and $V2$, respectively. Let $V3_c \in V3$ be the complex node corresponding to V_c and $G3_{V3c}$ be the SLN expanded by $V3_c$. Then we have $G3_{V3c} = G1_{Vc} \cup G2_{Vc}$.

Take the SLNs shown in Figures A1–A3 for example. For two SLNs $G1$ and $G2$, the union of $G1$ and $G2$ is shown in Figure A3. The SLN matrices of $G1$, $G2$ and $G1 \cup G2$ are $M(G1)$, $M(G2)$ and $M(G1 \cup G2)$, as shown below:

$$M(G1) = \begin{array}{c|cccc} & A & C & D & B \\ A & null & \{sim, ce\} & \{ce\} & \{ins, ce\} \\ C & seq & null & \{seq, imp\} & null \\ D & null & null & null & \{ref, st\} \\ B & null & null & null & null \end{array}$$

$$M(G2) = \begin{array}{c|ccccc} & A & E & C & F & B \\ A & null & \{sim\} & \{seq\} & null & \{ce, ref, st\} \\ E & null & null & null & null & \{seq\} \\ C & null & \{imp\} & null & null & null \\ F & null & \{sim, seq\} & null & null & null \\ B & null & null & null & \{ce, ins\} & null \end{array}$$

$M(G1 \cup G2) =$

	A	C	D	B	E	F
A	*null*	$\{sim, ce, seq\}$	$\{ce\}$	$\{ins, ce, ref, st\}$	$\{sim\}$	*null*
C	$\{seq\}$	*null*	$\{seq, imp\}$	*null*	$\{imp\}$	*null*
D	*null*	*null*	*null*	$\{ref, st\}$	*null*	*null*
B	*null*	*null*	*null*	*null*	*null*	$\{ce, ins\}$
E	*null*	*null*	*null*	$\{seq\}$	*null*	*null*
F	*null*	*null*	*null*	*null*	$\{sim, seq\}$	*null*

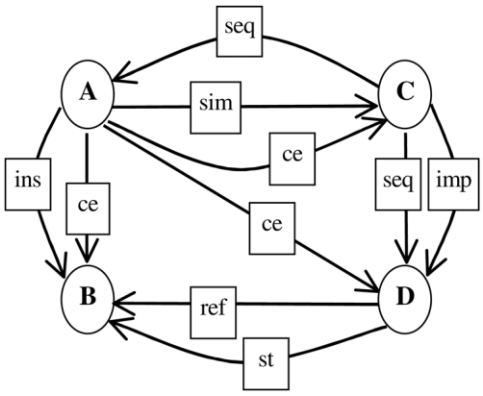


Figure A1. A semantic link network $G1$.

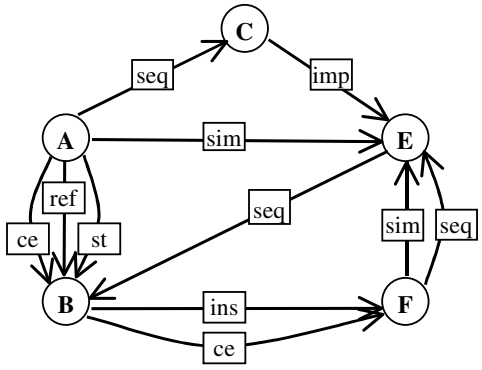


Figure A2. A semantic link network $G2$.

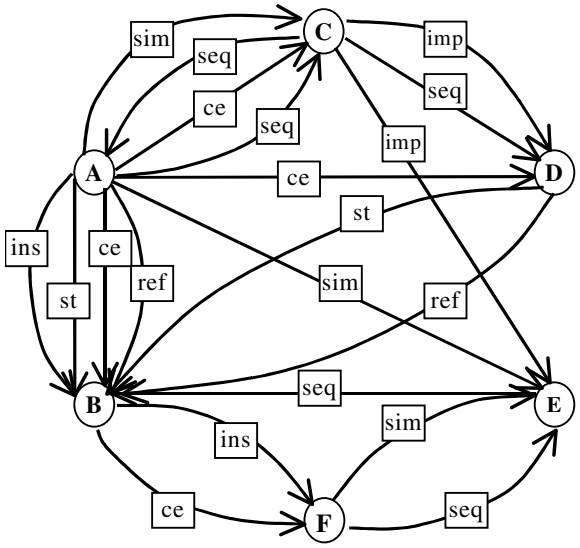


Figure A3. A semantic link network $G1 \cup G2$.

Union_SLN(SLN $G1$, SLN $G2$, SLN $G3$){
$V_{\text{int}} = V1 \cap V2$;
$d = |V1| + |V2| - |V_{\text{int}}|$;
$V3 = V1 \cup V2$;
Let $L1$, $L2$ and $L3$ be arrays with one dimension;
Set all the nodes in $V1$, $V2$, and $V3$ to array $L1$, $L2$, and $L3$ respectively and ensure each location of Li only contains one node;
Initialize($M(G3)$); // $M(G3)$ is a $d \times d$ SLN-matrix, every $M_{i,j}(G3)$ is set to *null*
For every $M_{i,j}(G3)(1 \leqslant i \leqslant d, 1 \leqslant j \leqslant d)$ {
Let Node_i $= L3[i]$ and Node_j $= L3[j]$;
if (both Node_i and Node_j belong to V_{int}) {
Let $V1\_i$, $V1\_j$, $V2\_i$, $V2\_j$ satisfy the following equations:
Node_i $= L1[V1\_i]$, Node_j $= L1[V1\_j]$, Node_i $= L2[V2\_i]$, Node_j $= L2[V2\_j]$;
$M_{i,j}(G3) = M_{v1\_i,v1\_j}(G1) \cup M_{v2\_i,v2\_j}(G2)$;// union two sets.
}
if (only one node belongs to V_{int}){
Suppose that another node belongs to Vk ; //k belongs to {1, 2}
Node_i $= Lk[Vk\_i]$, Node_j $= Lk[Vk\_j]$;
$M_{i,j}(G) = M_{vk\_i,vk\_j}(Gk)$;
}
if (neither Node_i or Node_j belongs to V_{int}){
if (both Node_i and Node_j belong to Vk){//k belongs to {1, 2}
Let Vk_i and Vk_j satisfy the equations: Node_i $= Lk[Vk\_i]$, Node_j $= Lk[Vk\_j]$;
$M_{i,j}(G) = M_{vk\_i,vk\_j}(Gk)$;
}
if(Node_i and Node_j are not in the same SLN) $M_{i,j}(G3) = null$;
}
}//end for
}

Algorithm A1. The algorithm Union_SLN for the union of two SLNs.

```
Union_HyperSLN(SLN G1, SLN G2, SLN G3){ //G3 = G1 ∪ G2;
    Pre_Process(G1, G2); //refer to the definition of Pre_Process in Section 5.
    Let V_int = V1 ∩ V2;
    Union_SLN(G1,G2,G3); //view complex nodes in G1 and G2 as atomic nodes.
    For every complex node cNode in G3 {// union operation between complex nodes.
        If( cNode belongs to V_int){
            Let G4 = G4(V4, E4) be the SLN expanded by the complex node cNode in V1;
            Let G5 = G5(V5, E5) be the SLN expanded by the complex node cNode in V2;
            Let G6 = G6(V6, E6) be the SLN expanded by the complex node cNode in V3;
            String Rtn = Rel_HyperSLN(G4, G5);
            If (Rtn == "equal") Let G6 = G4;
            If (Rtn != "equal")
                Rename cNode in V1 and V2 as different node names;
                Union_HyperSLN(G1,G2,G3,); //since the SLNs are changed, the union
                operation must be carried out from the beginning.
        }
        if( cNode only belongs to Vk){ //k belongs to {1,2}
            Let G_Vk,cNode = G_Vk,cNode(V_Vk,cNode, E_Vk,cNode) be the SLN expanded by the complex
            node cNode in Vk;
            Let G_V3,cNode = G_V3,cNode(V_V3,cNode, E_V3,cNode) be the SLN expanded by the complex
            node cNode in V3;
            Let G_V3,cNode = G_Vk,cNode;// G_V3,cNode is set to G_Vk,cNode.
        }
    }//end for
}
```

Algorithm A2. The algorithm *Rel_HyperSLN* for determining the relationship between two hyper-SLNs. The algorithm *Union_SLN* only applies to the SLNs consisting of atomic nodes. For semantic link networks including complex nodes, the following algorithm *Union_HyperSLN* is used to the union of SLNs. The algorithm *Union_HyperSLN* includes two steps: (1) it first views the complex nodes in two networks as atomic nodes, and then uses *Union_SLN* to union them, and (2) it uses *Rel_HyperSLN* to decide how to union complex nodes.

APPENDIX B. THE PROOF OF THE CORRECTNESS OF *Rel_SLN_Vertex*

The basic algorithm *Rel_SLN_Vertex* is correct.

Proof. We can say the algorithm is correct if we can prove the relationship that is deduced from the value of *Zero* in the function *Result* is just the relationship that the function *Result* returns. Since the relationship between two SLNs only includes five types, we can prove the correctness of the algorithm according to the following five cases.

(1) Empty—Zero = 0. According to the function *Result*, Zero = 0 indicates that element '0' is not contained in any $R_{i,j}(G)$, which implies that there is no element that is contained in both $M_{i,j}(G)$ and $M_{i,j}(G')$ according to the function *Subtract* and the definition of $R_{i,j}(G)$. According to the definition of the 'empty' relationship, we can say that Zero = 0 really means that the intersection of the edge sets of G and G' is 'empty'.

(2) Inverse inclusion—Zero = 1, Plus = 1 and Sub = 0. According to the function *Result*, Sub = 0 indicates that element '−' is not contained in any $R_{i,j}(G)$, which implies that there is no element that is contained in $M_{i,j}(G')$ but not in $M_{i,j}(G)$ according to the function *Subtract* and the definition of $R_{i,j}(G)$. So all the edges of G' must be contained in G.

According to the function *Result*, Plus = 1 indicates that there is at least one element $R_{i,j}(G)$ that contains element '+', which implies that there is an edge that is contained in $M_{i,j}(G)$ but not in $M_{i,j}(G')$ according to the function *Subtract* and the definition of $R_{i,j}(G)$. So there is at least one edge that is contained in G but not in G'.

According to the conclusion of (1), Zero = 1 means that the intersection of the edge set of G and G' is not empty. According to the above conclusions and the definition of the 'inverse inclusion' relationship, we can conclude that 'Zero = 1&&Plus = 1&&Sub = 0' really means by 'inverse inclusion'.

(3) Inclusion—Zero = 1, Plus = 0, and Sub = 1. According to the function *Result*, Sub = 1 indicates that there is at least one element $R_{i,j}(G)$ that contains element '−', which implies that there is an element that is contained in $M_{i,j}(G')$ but not in $M_{i,j}(G)$ according to the function *Subtract* and the definition of $R_{i,j}(G)$. Hence, there is at least one edge that is contained in G' but not in G.

According to the function *Result*, Plus = 0 indicates that element '+' is not contained in any $R_{i,j}(G)$, which implies that there does not exist any element that is contained in $M_{i,j}(G)$ but not in $M_{i,j}(G')$ according to the function *Subtract* and the definition of $R_{i,j}(G)$. Hence, all the edges of G must be contained in G'.

According to (1), Zero = 1 means that the intersection of the edge set of G and G' is not empty. According to the conclusions introduced above and the definition of the 'inclusion' relationship, we can conclude that 'Zero = 1&&Plus = 0&&Sub = 1' really means by 'inclusion'.

(4) Intersection—Zero = 1, Plus = 1, and Sub = 1. According to (2) and (3), Sub = 1 means that there is at least one edge that is contained in G' but not in $G$and Plus = 1 means that there is at least one edge that is contained in G but not in G'. According to the conclusion of (1), Zero = 1 means that the intersection of the edge set of G and G' is not empty. According to the conclusions introduced above and the definition of the 'intersection' relationship, we can conclude that 'Zero = 1&&Plus = 1&&Sub = 1' really means by 'intersection'.

(5) Equal—Zero = 1, Plus = 0, and Sub = 0. According to (2) and (3), Sub = 0 means all the edges of G' are contained in G and Plus = 0 means all the edges of G are contained in G'. According to the conclusion of (1), Zero = 1 means that the intersection of the edge set of G and G' is not empty. According to the above conclusions and the definition of the 'equal' relationship, we can conclude that 'Zero = 1&&Plus = 0&&Sub = 0' really means by 'equal'.

Since the relationship derived from the above five cases is the same as the returned corresponding relationship, we can conclude that the algorithm *Rel_SLN_Vertex* is correct.

ACKNOWLEDGEMENTS

The authors thank all team members of China Knowledge Grid Research Group (http://kg.ict.ac.cn) for their diligent work and cooperation.

REFERENCES

1. Kleinberg J, Lawrence S. The structure of the Web. *Science* 2001; **294**(30):1849–1850.
2. Adamic LA, Huberman BA. Power-law distribution of the World Wide Web. *Science* 2000; **287**(24):2115.

3. Thistlewaite P. Automatic construction and management of large open Webs. *Information Processing and Management* 1997; **33**(2):145–159.
4. Tudhope D, Taylor C. Navigation via similarity: Automatic linking based on semantic closeness. *Information Processing and Management* 1997; **33**(2):233–242.
5. Heflin J, Hendler J. A portrait of the Semantic Web in action. *IEEE Intelligent Systems* 2001; **16**(2):54–59.
6. Hendler J. Agents and the semantic Web. *IEEE Intelligent Systems* 2001; **16**(2):30–37.
7. Berners-Lee T, Fischetti M, Dertouzos TM. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper: San Francisco, 1999.
8. Semantic Web. http://www.w3.org/2001/sw/ [May 2003].
9. McHraith SA, Son TC, Zeng, H. Semantic Web services. *IEEE Intelligent Systems* 2001; **16**(2):46–53.
10. Maedche A, Staab S. Ontology learning for the semantic Web. *IEEE Intelligent Systems* 2001; **16**(2):72–79.
11. Staab S, Schnurr H-P, Studer R, Sure Y. Knowledge processes and ontologies. *IEEE Intelligent Systems* 2001; **16**(1):26–34.
12. Decker S, Melnik S, van Harmelen F, Fensel D, Klein M, Broekstra J, Erdmann M, Horrocks I. The Semantic Web: The roles of XML and RDF. *IEEE Internet Computing* 2000; **15**(3):63–74.
13. Fensel D, van Harmelen F, Horrocks I, McGuiness D, Patel-Schneider P. OIL: An ontology infrastructure for the Semantic Web. *IEEE Intelligent Systems* 2001; **16**(2):38–45.
14. Hendler J, McGuinness D. The DARPA agent markup language. *IEEE Intelligent Systems* 2000; **15**(6):72–73.
15. Klein M. XML, RDF, and relatives. *IEEE Internet Computing* 2001; **16**(2):26–28.
16. Martin P, Eklund PW. Knowledge retrieval and the World Wide Web. *IEEE Intelligent Systems* 2000; **15**(3):18–25.
17. Broekstra J, Klein M, Decker S, Fensel D, van Harmelen F, Horrocks I. Enabling knowledge representation on the Web by extending RDF schema. *Computer Networks* 2002; **39**(5):609–634.
18. XML Topic Maps. http://www.topicmaps.org/xtm/1.0/ [November 2003].
19. Carr L, Hall W, Bechhofer S, Goble C. Conceptual linking: Ontology-based open hypermedia. *Proceedings of the 10th International World Wide Web Conference*, May 2001. ACM Press: New York, 2001; 334–342.
20. Zhuge H. Active e-Document Framework ADF: Model and tool. *Information and Management* 2003; **41**:87–97.
21. Green SJ. Building hypertext links by computing semantic similarity. *IEEE Transactions on Knowledge and Data Engineering* 1999; **11**(5):713–730.
22. Henzinger MR. Hyperlink analysis for the Web. *IEEE Internet Computing* 2001; **5**(1):45–50.
23. Srihari RK, Zhang Z, Rao A. Intelligent indexing and semantic retrieval of multimodel documents. *Information Retrieval* 2000; **2**:245–275.
24. Zhuge H. China's e-science Knowledge Grid environment. *IEEE Intelligent Systems* 2004; **19**(1):13–17.

