A Definition of the Pattern Language

Inductive definition:[1]

QuotedString ::=′ "′String′"′
XML-Tag ::= String
Operator ::= String
SynSet ::= QuotedString; POS ::= QuotedString
Constraint ::= Operator′(′ QuotedString* ′)′ | ′!′Operator′(′ QuotedString* ′)′
Pattern ::= QuotedString
Pattern ::= "SynSet :″ SynSet
Pattern ::=′ \′XML-Tag(′[′Pattern′]′)?
Pattern ::=′ (′Pattern′)′
Pattern ::= Pattern′ ′* Pattern
Pattern ::=′ (′Pattern(′|′Pattern)+′)′
Pattern ::=′ ∗′
Pattern ::=′ (′Pattern′)?′
Pattern ::=′ (′Pattern Pattern+′)%′
Pattern ::=′ (′Pattern′)*′
Pattern ::=′!(′Pattern′)′
Pattern ::= Pattern′ =:′ String
Pattern ::=′ $′String′$′
Pattern ::= Pattern′{′ Constraint* ′}′

The following productions make the specification of patterns more explicit, compact and readable although they are redundant and can be expressed by accessing attributes of corresponding XML elements

Pattern ::= "PF :″ String
Pattern ::= "POS :″ POS("[PF :″ String"]″)?
Pattern ::=′ [′Syntactic_Category(′:′ Pattern)?′]′

Semantics:

QuotedString: matches if the textual content of an XML element corresponds with the QuotedString

[1] Definition is written in EBNF standard used by W3C for XML specification (see http://www.w3.org/TR/)

′\′XML-Tag(′[′Pattern′]′)?: matches if an XML element specified by the tag is found and the element has a structure specified by the pattern in square brackets.

Pattern′ ′ * Pattern: denotes sequence pattern. Sequences of more than two elements can be obtained applying this production recursively.

′(′Pattern(′|′Pattern)+′)′ : union, matches either pattern

′∗′ matches any XML fragments satisfying the ENSS requirements II-IV

′(′Pattern′)?′: indicates that a pattern occurs either once or does not occur at all (option)

′(′Pattern Pattern+′)%′: patterns in parentheses can occur in any order (permutation).

′(′Pattern′)*′: Pattern inside the parentheses can occur any number of times (Kleene closure)

′!(′Pattern′)′: Negation. ′!′expresses that the inner pattern does not occur

Pattern′ =:′ String: allows to reference parts of a pattern by a variable for later reuse. Besides, after a successful match the variable is bound to the XML fragment matching the assigned Pattern

′$′String′$′: After a variable has been bound to an XML fragment, it can be accessed using the reference pattern (by enclosing the variable in dollar signs). Furthermore, a pattern assigned to a variable in the assignment pattern can be reused using the reference pattern.

Constraint ::= Operator′(′ QuotedString* ′)′: constraints are boolean expressions. Operator is a boolean function that can be applied to the XML fragment matching the pattern. Constraints play the similar role as the filter functions in XPath. An example of a constraint is a comparison of attribute values of an XML element.

Pattern′{′ Constraint* ′}′: constraints can be specified for any pattern. They are defined on the XML fragments matching the pattern and not on single XML nodes.

“PF :′′ QuotedString addresses every inflection form of the word with the principal form specified by String. Principal form for verbs is infinitive, for nouns it is nominative and singular etc.

′[′Syntactic_Category′]′ matches a text segment if it has the specified syntactic category

′[′Syntactic_Category′ :′ Pattern′]′ restricts the scope of a pattern to the specified syntactic constituent (e.g. paragraph, sentence, verb phrase, etc.). If the outer scope of the whole pattern is not specified, default restriction to a sentence is assumed.

B Pattern Matching Algorithm for Selected Patterns

We write the main body of the algorithm as the function *match* since it is easier to denote numerous iterations of the algorithm as recursive function calls. The algorithm decides whether pattern P matches the context node n (for simplicity multiple matchings are disregarded).

```
INITIALIZE PATTERN STACK WITH P, INITIALIZE ELEMENT STACK WITH n

match (Stack patternstack, Stack elementstack){

if (patternstack.isEmpty() && elementstack.isEmpty())  return (true, false)
if (patternstack.isEmpty()) return (false, false)
REMOVE THE TOP PATTERN FROM THE PATTERN STACK

switch(category of removed pattern)
case SEQUENCE:
  Put all sequence elements on the stack so that the first one is on top

case NEGATION:
    BACK UP THE STACKS
    patternstack.put(p.ipatterns.elementAt(0));
    if (!match(patternstack, elementstack).success) //mr denotes matching result
         mr=(true, true); //hint for the upper backtr. procedure to continue
         RESET THE STACKS
    else mr=(false, false); // no backtracking in the upper calls

case WILDCARD:
    if (patternstack.isEmpty())
        REMOVE ALL ELEMENTS FROM THE ELEMENT STACK
        return (true, _)
    BACK UP THE STACKS
    while ((mr=match (patternstack, elementstack)).backtracking)
        if (mr.success && mr_backup==null)
           BACKUP MATCHING RESULT AND STACKS
        if (elementstack.isEmpty() && mr_backup==null)
           return  (false, _)
        if (elementstack.isEmpty()) break
        RESET PATTERNSTACK
        ADD ALL CHILDREN OF THE CURRENTLY PROCESSED ELEMENT ON THE ELEMENT STACK
```

```
case Pattern.UNION:
    i=0
    BACK UP THE STACKS
    do  {if (mr.success && mr_backup==null)
            BACK UP MATCHING RESULT AND STACKS
         RESET THE STACKS
         PUT THE I-th INNER PATTERN OF UNION PATTERN ON THE PATTERN STACK
        } while ((ur=match (patternstack, elementstack)).backtracking && ++i <
          NUMBER OF INNER PATTERNS)

if (mr_backup!=null && mr!=null && (mr.success || mr.backtracking))
    mr=mr_backup
    RESET THE STACKS TO THE BACKED UP STATE
if (!mr.success)
    if (backtracking pattern ||elementstack.isEmpty())
        return mr
    PUT ALL CHILDREN OF THE CURRENTLY PROCESSED ELEMENT ON THE ELEMENT STACK
    PUT THE CURRENTLY PROCESSED PATTERN ON THE PATTERN STACK
if (!patternstack.isEmpty() && !elementstack.isEmpty())
    return match(patternstack, elementstack)
if (patternstack.isEmpty() && elementstack.isEmpty())
    return mr

return match(patternstack, elementstack)
}
```

C Algebraic Transformations for Recursive Backtracking Patterns

Idempotence:

$(*)? = *$
$(*)^* = *$
$(A|*) = *$
$(A^*)\% = A^*$
$(A^*)? = A^*$

Semi-Idempotence:

$(A^*\ A)\% = A^*$
$(A^*\ A?)\% = A^*$
$A? \mid A = A?$
$A^* \mid A = A^*$
$A^* \mid A? = A^*$
$(A\ B)\% \mid (A \mid B)^* = (A \mid B)^*$

Sequence transformations:

$* * = *$
$A^*\ A = A^*$
$A^*\ A? = A^*$

Subsumption:

$(A?)^* = A^*$

Commutativity:

$A \mid B = B \mid A$

Associativity:

$(A \mid (B \mid C)) = ((A \mid B) \mid C)$

Distributive Law:

$A\ (B \mid C) = (A\ B) \mid (A\ C)$

DeMorgan Rule is not valid:

$!(A \mid B) \neq !A \mid !B$

D Anhang gemäß Promotionsordnung

Zusammenfassung

Internet, Presse, wissenschaftliche Veröffentlichungen beherbergen gigantische Informationsmengen, die als natürlichsprachliche Texte vorliegen. Die Informationen in natürlichsprachlichen Quellen sind für Computer jedoch kaum zugänglich, und auch für den Menschen ist der Zugang oft mit langer Suche verbunden. Das Ziel dieser Dissertation ist die Entwicklung eines Verfahrens, mit dessen Hilfe im Voraus spezifizierte Informationen im Text automatisch gesucht und in eine formale, abfragbare Struktur überführt werden. Durch Extraktion und Speicherung der Informationen in einer strukturierten Form (z.B. als Relationen in der Datenbank) werden ein effizienter Anfragemechanismus und bessere Administration der extrahierten Inhalte ermöglicht. Strukturierte Informationen können ohne menschliche Interaktion vom Computer weiterverarbeitet werden, indem sie z.B. zur Erstellung von Ontologien, Wissensbasen und Datenanalyse dienen.

Die ersten IE Systeme basierten auf handgeschriebenen, auf einen Anwendungsbereich zugeschnittenen Extraktionsregeln, die einen hohen manuellen Aufwand erforderten und auf andere Anwendungsbereiche nicht übertragbar waren. Der in der Dissertation vorgestellte Ansatz kompensiert diese Unzulänglichkeiten durch den Einsatz des maschinellen Lernens anstelle eines menschlichen Experten. Extraktionsregeln werden automatisch aus einer Menge von Beispielextraktionen, die von einem Menschen in einem Trainingskorpus annotiert sind, durch induktives Lernen abgeleitet. Da außer dem Trainingkorpus keine Ressourcen erforderlich sind, kann das Verfahren in unterschiedlichen Anwendungsbereichen und sogar Sprachen angewendet werden.

Extraktionsregeln beinhalten Sprachmuster, die typische Ausdrucksformen der extrahierten Information in einem Textcorpus widerspiegeln. Wir erweitern verbreitete Modelle der natürlichen Sprache, indem wir eine reichhaltige Musterspezifikationssprache definieren, die reguläre Ausdrücke, Permutation, Negation und hierarchische XML Strukturen unterstützt. Sprachliche Muster sind nicht auf ein festes Kontextintervall beschränkt, sie bilden komplette Sätze als primäre semantische Einheiten der natürlichen Sprache ab. Die Ausdrucksfähigkeit der entwickelten Mustersprache erlaubt, nicht triviale Phrasen und Sätze mit relevanter Information abzubilden.

Sprachliche Muster werden auf linguistisch vorverarbeitete Texte angepasst, die mit XML annotiert sind, um gewünschte Informationen zu finden. Indem wir die sprachlichen Muster der Extraktionsregeln als XML Anfragen betrachten, wird das Problem der IE auf die Auswertung der XML Anfragen zurückgeführt. Wir haben eine formale Semantik und einen effizienten Algorithmus zur Anfrageauswertung für die Musterspezifikationssprache entwickelt. Damit wurde eine neue XML Anfragesprache geschaffen, die für Anfragen auf den mit XML annotierten Texten besonders geeignet ist.

Im Rahmen der semantischen Vorverarbeitung der Texte schlagen wir eine neue Methode für die Bestimmung der Synonyme vor. Wir bauen einen lexikalischen Graphen durch Verbindung der lexikalischen Terme entsprechend der syntaktischen Struktur der Sätze auf und erhalten so eine implizite Kontextdarstellung. Wir zeigen, dass die Synonymiemetrik, die die Längen der Pfade zwischen zwei Termen, die Anzahl und die Besonderheit ihrer gemeinsamen Nachbarn berücksichtigt, zufrieden stellende Ergebnisse auf einem Testcorpus mit 200 Synonymmengen erreicht. Identifizierte Synonyme dienen zur Abstraktion der lexikalischen Elemente während der Induktion von Extraktionsregeln.

Die am Anfang aus den vom Menschen annotierten Beispielextraktionen generierten Extraktionsregeln werden verallgemeinert, um mögliche ähnliche Ausdrucksmöglichkeiten der gesuchten Information im Text zu erfassen. Die Verallgemeinerung ist formal spezifiziert und umfasst neben der Regelverschmelzung Abstraktion der einzelnen Regeln und Substitution der extrahierten Abschnitte im Kontext der unterschiedlichen Regeln. Für die Definition eines Ähnlichkeitsabstands zwischen zwei Regeln und für die Regelverschmelzung wurden ein Algorithmus entworfen, der optimales Alignment zwischen zwei Sequenzen in minimaler Laufzeit findet (was eine Erweiterung des bekannten LCS Problems darstellt), und seine Korrektheit bewiesen. Um eine allmähliche Verallgemeinerung der Regeln zu erreichen, flossen Validierung und Korrektur der Regeln in den Lernalgorithmus ein.

Die Leistungsfähigkeit unseres Ansatzes wird mit Hilfe eines Vergleichs mit den Ergebnissen aktueller IE-Systeme demonstriert, bei dem GROPUS vergleichbare oder sogar beste Resultate in Abhängigkeit von der Art der Texte erreicht. Das Potential des Systems wird in einer neuartigen Untersuchung durch den Vergleich mit menschlichen Ergebnissen ermittelt. Aufgrund des unterschiedlichen Verhaltens von untersuchten Ansätzen auf unterschiedlichen Korpora werden Schlussfolgerungen über die Tauglichkeit der statistischen und regel-basierten Ansätze für unterschiedliche Textarten gemacht. Quantitative Untersuchung wird ergänzt durch die Analyse, welche Faktoren die Extraktionsqualität entscheidend beeinflussen, was die Fehlerquellen sind etc. Abschließend gehen wir auf die Fragen ein, unter welchen Bedingungen die Anwendung von IE-Techniken heute nützlich ist, welche Arten von Text verarbeitet werden können, und charakterisieren die Umgebungen, in denen der vorgestellte Einsatz sinnvoll eingesetzt werden kann.

Erklärung

Ich versichere, die vorliegende Dissertation auf Grundlage der in der Arbeit genannten Hilfen und Hilfsmittel selbständig verfasst zu haben.

Berlin, 28.11.2007

(Peter Siniakov)