

# Syntactic Folding and its Application to the Information Extraction from Web Pages

Jörg Herrmann

Deutsches Forschungszentrum für  
Künstliche Intelligenz GmbH  
Stuhlsatzenhausweg 3  
66123 Saarbrücken, Germany  
herrmann@dfki.de

## Abstract

The paper deals with investigations concerning potential structures of documents that will be subject to automated information extraction. The focus is on folding principles and their influence on the recognition of certain data in a document undergoing the extraction.

## Introduction

The topic of our work is information extraction from the Internet.

There are a couple of approaches which deal with the problem of recognizing structural data in semi-structured documents for retrieval of user specified information from these and from similar documents (possibly of the same source), in an automatic or semi-automatic way (Freitag 1996), (Soderland 1997), (Kushmerick 1997).

Ideally, structural information shall be learned by presenting only samples of text segments which a user wants to extract from these pages to a learning device, without any need to specify details of how the desired samples can be localized within the document. The learning device should generate a procedure, a *wrapper*, that – reading the same documents – puts out a collection of information, including the samples and, hopefully, extending them in terms of finding similar items.

These approaches led to a variety of wrapper classes, e.g. *LR-wrappers* (Kushmerick 2000), *island wrappers* (Grieser *et al.* 2000), *T-wrappers* (Thomas 1999) and further variants of them with different characteristics, for instance (Hsu & Dung 1998), (Muslea, Minton, & Knoblock 1999). Extracting the fundamentals of these approaches, our goal consists in a specification of the area of application for a wrapper under investigation, and in finding out some rules for their conscious selection and generation.

On the basis of (Grieser *et al.* 2000) and a comparison with Kushmerick's wrapper classes, this publication will focus on the question how tupels, that shall be

extracted from a document in the manner above, can interrelate. It is *not our intention* to set the mentioned wrapper concepts into a relation in terms of investigating the acceptable languages in detail.

To restrict the domain, we make a couple of assumptions: The information, we are looking for, should be encapsulated by syntactic expressions that are comparable in some manner. We suppose there are a lot of web pages which are organized in a similar way and which make use of the same expressions for structuring information. However, we do not require a specific set of such expressions, so the approach is not restricted to HTML-documents. HTML serves more for an application domain.

## Island Wrappers

In (Thomas 1999) the author introduced so-called *Island Wrappers* for the information extraction from web pages.

$$\begin{aligned}
 w(V_1, V_2, \dots, V_n, X_1 L_1 V_1 R_1 \dots X_n L_n V_n R_n X_{n+1}) \leftarrow & \\
 p_{\ell_1}(L_1), nc - p_{r_1}(V_1), p_{r_1}(R_1), & \\
 nc - p_{\ell_2}(X_2), p_{\ell_2}(L_2), nc - p_{r_2}(V_2), p_{r_2}(R_2), & \\
 \vdots & \\
 nc - p_{\ell_n}(X_n), p_{\ell_n}(L_n), nc - p_{r_n}(V_n), p_{r_n}(R_n). & \\
 nc - p_{\ell_1}(X) \leftarrow not\ c - p_{\ell_1}(X). & \\
 c - p_{\ell_1}(X) \leftarrow p_{\ell_1}(X). & \\
 c - p_{\ell_1}(XY) \leftarrow c - p_{\ell_1}(X). & \\
 c - p_{\ell_1}(XY) \leftarrow c - p_{\ell_1}(Y). & \\
 \vdots & \\
 nc - p_{r_n}(X) \leftarrow not\ c - p_{r_n}(X). & \\
 c - p_{r_n}(X) \leftarrow p_{r_n}(X). & \\
 c - p_{r_n}(XY) \leftarrow c - p_{r_n}(X). & \\
 c - p_{r_n}(XY) \leftarrow c - p_{r_n}(Y). &
 \end{aligned}$$

Figure 1: Island Wrapper as AEFS

One might distinguish certain ideas underlying that concept:

- There is a document containing the information of interest in the form of tuples  $t = (t_1, \dots, t_m)$ ,  $t_i \in \Sigma^+$  for any fixed  $m$ ,
- the quantity of such tuples per document  $D$  is greater than 1:  $\|T_D\| > 1$  for  $T_D = \{t | t \text{ in } D\}$ , and
- the tuple embedding in  $D$  follows a predefined pattern, a so-called *wrapper*.

In (Grieser *et al.* 2000) island wrappers have been represented by *Advanced Elementary Formal Systems (AEFS)*, an advancement of *Elementary Formal Systems (EFS)* known from Smullyan (Smullyan 1961). Figure 1, taken from (Grieser *et al.* 2000), shows such an AEFS.

Within this AEFS,  $V_1, \dots, V_n$  denote extraction variables for storage of the information a user wants to extract from a document. The  $L_i$  and  $R_i$  are replacement symbols for delimiters and the  $X_i$  play the role of wild cards. Their combination in the last argument of the predicate  $w$  specifies a pattern resp. the order in which the substitutions of the variables are expected to occur in the document. The subsequent constraints reduce the search space for these substitutions. For instance,  $V_1$  must not contain a segment of the document that can be matched by  $R_1$ .

Alternatively, the language  $T_X(D)$  accepted from a document  $D$  by an island wrapper  $X$  might be represented as follows<sup>1</sup>:

$$T_X(D) = \{t | t \in \widehat{\mathcal{T}}\}$$

with

$$\exists f \in \mathcal{T}: \circ(f * t) = D \quad (1)$$

$$\exists f' l, r \in \mathcal{T}: \circ[f' * (l \otimes t \otimes r)] = \mathcal{D} \quad (2)$$

$$\forall i: L_i^{left} = \{l_i\}, \quad L_i^{right} = \{r_i\} \quad (3)$$

$$\forall i \leq m: t_i \notin (\Sigma^* \circ L_i^{right} \circ \Sigma^*) \quad (4)$$

$$\forall i < m: f'_{i+1} \notin (\Sigma^* \circ L_{i+1}^{left} \circ \Sigma^*) \quad (5)$$

where  $\mathcal{T}$  denotes the set of tuples with nonempty components, and  $\widehat{\mathcal{T}}$  denotes a superset that allows empty components too.

Condition (1) states that the order of tuple components  $t_i$  is fixed. Furthermore, those components are subwords of the underlying document  $D$ , what in turn implies there is no need to investigate tuple semantics. One has to look for syntactic structures only.

The  $l_i$  are called *left anchors*, the  $r_i$  *right anchors* (cf. (Grieser *et al.* 2000)) and represent left resp. right delimiters encapsulating the desired information  $t_i$  (cond. (2)/(3)).

<sup>1</sup>The operator definitions can be found in the appendix.

Island wrappers require that tuple components must not contain elements from the corresponding right anchor language (cond. (4)). The same applies to filling words between the anchors of two consecutive components w.r.t. the left anchor involved (cond. (5)). One might argue to what extent those last two requirements are appropriate, but obviously the initial idea was to separate tuple components from delimiters in a plain way.

## Wrapper Versions

An approach to the design of algorithms for the extraction of tuples by means of specified wrappers, or for learning of those wrappers, consists in analyzing the ideas underlying the island wrappers above.

First of all we need some understanding of the kind of information  $T_X(D)$  we intend to extract by a wrapper  $X$  from a given document  $D$ . For our purposes, condition (1) from the last section will provide a first approximation:

$$T_X(D) \subseteq \{t | t \in \widehat{\mathcal{T}}, \exists f \in \mathcal{T}: \circ(f * t) = D\}$$

Requiring that tuples  $t$  should not contain empty components  $t_i = \epsilon$  is a feature we could drop. For convenience, when comparing the results with other approaches which rely on it, we retain it.

As mentioned, we adopt the view there should exist left and right delimiters that encapsulate the tuple components. But here, we hit the first problem. What are delimiters? May they overlap or not? What does it mean to draw them from a specific *anchor* language? Should, for instance, anchor languages concern a specific tuple position, and what is the result if we presuppose or ignore this assumption?

It turns out, one has to consider the mutual position of tuples, though within the definition of island wrappers this insight is invisible.

Island wrappers are designed to accept any individual tuple regardless of its position in the document as long as it meets at least the requirements outlined in the previous section. These innocuous requirements, however, have implications on the relation between tuples, on their position and on the syntactic structure of the delimiters and the components themselves. We skip this question for a while.

Looking for a most simple relation between two tuples, we come out with a strict sequence

$$\forall t, t': t_1 \triangleright t'_1 \longrightarrow t_m \triangleright t'_m \quad (A)$$

where the relational symbol  $\triangleright$  denotes precedence in the underlying document. Strict sequences as a general presumption have been used in (Chidlovskii, Ragetli, & de Rijke 2000), and in (Kushmerick 2000).

For a wide variety of documents such assumptions may hold, for a table organized as a sequence of columns, it fails (fig. 2).

On the asset side, it does not require much sophistication to treat similar documents too. Thus, relation

```

< table >
  < cities >
    < item > Frankfurt < \item >
    < item > London < \item >
    < item > Moscow < \item >
    < item > NewYork < \item >
    < item > Tokio < \item >
    :
  < \cities >
  < temperature >
    < item > 10C < \item >
    < item > 12C < \item >
    < item > 5C < \item >
    < item > 15C < \item >
    < item > 11C < \item >
    :
  < \temperature >
< \table >

```

Figure 2: Example for a Set of Non-sequential Tuples

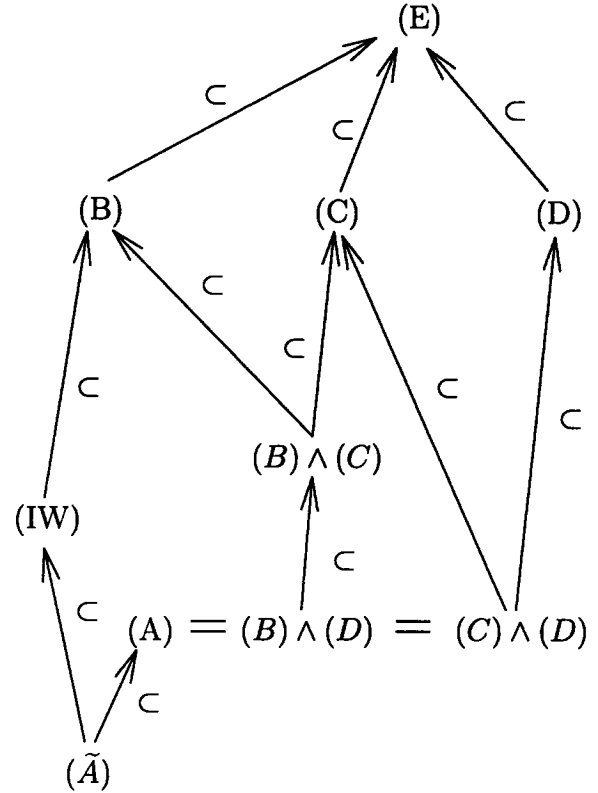


Figure 3: Folding Relations

(B) specifies a plain generalization of (A):

$$\forall t, t', \forall i: t_i \triangleright t'_i \longrightarrow t_i \triangleright t'_i \quad (B)$$

On the debit side, (B) increases the restrictions for the delimiters of  $t$  and  $t'$  respectively for the necessary knowledge where  $t'$  starts in fact. With other word, each version of folding  $t$  and  $t'$  will require a specific treatment.

A third ordering principle can be called *displacement by catching up*:

$$\forall t, t', \forall i: t_i \triangleright t'_i \triangleright t_{i+1} \longrightarrow t'_{i+1} \triangleright t_{i+1} \quad (C)$$

This type of a folding relation is of some special interest since it allows to construct rather complex structures.

Finally, we regard *strict embedding*:

$$\forall t, t', \forall i, \forall j > 1: t_1 \triangleright t'_1 \triangleright t_j \longrightarrow t'_i \triangleright t_j \quad (D)$$

Strict embedding defines context-free structures. Thus, it supports languages like XML or SGML, that – beside general programming languages – definitely represent proper extensions of the regular languages, to a greater extend than any of the other wrapper classes (T-wrappers, island wrappers or OCLR wrapper, etc.) mentioned above can do.

Fig. 3 provides a general survey concerning the relation between these folding principles. Here, type (E) denotes unrestricted folding.

We refrain from discussing the inclusions in detail and direct the reader's attention to island wrappers again. Where do they belong to?

Condition (2) in the previous section says, there is no overlap between the right delimiter  $r_i$  of a tuple component  $t_i$  and the left delimiter  $l_{i+1}$  of the next component:

$$\forall i < m, \exists f_{i+1} \neq \epsilon: r_i \circ f_{i+1} \circ l_{i+1} \sqsubseteq D \quad (6)$$

In contrast, document type (A) allows any delimiter between successive components. As a consequence, there is no proper inclusion. We only could restrict the family  $\mathcal{F}_A$  of documents corresponding type (A)

$$\mathcal{F}_A = \{D \mid \exists X, T: T_X(D) = T \wedge \forall t, t' \in T: t_1 \triangleright t'_1 \longrightarrow t_m \triangleright t'_1\} \quad (7)$$

to a subfamily  $\mathcal{F}_{\tilde{A}}$  by adding condition (6) to the definition of  $\mathcal{F}_A$ . For getting

$$\mathcal{F}_{\tilde{A}} \subset \mathcal{F}_{IW}$$

where  $\mathcal{F}_{IW}$  denotes the family of documents that are processable by island wrappers, we need conditions (3)...(5) too, for arbitrary choices of  $L_i^{left}$  and  $L_i^{right}$ . Conversion of  $\mathcal{F}_B$  by the same suggestions to  $\mathcal{F}_{\tilde{B}}$ , finally leads to

$$\mathcal{F}_{IW} \subset \mathcal{F}_{\tilde{B}} \subset \mathcal{F}_B$$

## The Extraction Task

In this section we discuss the task of extracting some tuples of interest from a document.

Let  $D$  be such a document. We are looking for a set  $T_X(D) \subseteq (\Sigma^*)^m$  of tupels extractable from  $D$  by means of a wrapper  $X$ .

One can distinguish two sorts of knowledge.

Local knowledge contains characteristics that focus on individual tupel components. They tell how components are encapsulated by structuring information. In the current approach we use pre- and suffixes in terms of left and right anchors/delimiters for each tupel component.

For considering exactly one left and one right delimiter per component, two tupels  $\beta, \gamma \in (\Sigma^+)^m$  may serve for the anchor representation.

Then for any  $t \in T_X(D)$  the local knowledge consists of two relations:

$$\exists f \in \widehat{\mathcal{T}}: \circ[f * (\beta \otimes t)] = D \quad (8)$$

and

$$\exists f' \in \widehat{\mathcal{T}}: \circ[f' * (t \otimes \gamma)] = D \quad (9)$$

In comparison, island wrappers would require a special version:

$$\exists f \in \widehat{\mathcal{T}}: \circ[f * (\beta \otimes t \otimes \gamma)] = D$$

Second, global knowledge defines folding of tupels. We did discuss this topic during the previous section. For documents of the mentioned type (B), e.g., we get

$$\forall t \in T_X(D), \forall i, \exists t'_i: (t_i \triangleright t'_i) \wedge \neg(t_i \triangleright t'_i) \quad (10)$$

The extraction becomes more sophisticated if we allow sets of alternative anchors for each component. Again, there is a wide range of freedom for refinements. Whereas, for example, on the local level island wrappers do not treat<sup>2</sup> each element  $\beta$  from a potential set  $A_\beta$  of anchor representations individually, as in

$$\exists \beta \in A_\beta, \exists f \in \widehat{\mathcal{T}}: \circ[f * (\beta \otimes t)] = D$$

and one has to search over

$$\widetilde{A}_\beta = A_{\beta_1} \times A_{\beta_2} \times \dots \times A_{\beta_m}$$

for

$$A_{\beta_i} = \{\beta_i | \beta \in A_\beta\}$$

other wrappers obviously can permit this treatment.

To sum up, given a set of delimiter pairs  $(\beta, \gamma)$ , a document  $D$  and a folding relation  $F$ , the extraction task consists in finding instantiations of  $t$  such that conditions (8), (9), (10) hold.

### The Learning Task

Let  $D$  be a document,  $X$  any initial wrapper, and  $T_{+/-} = T_+ \cup T_-$  be a set of marked tupels, where  $T_+ \subseteq \mathcal{T} \times \{1\}$  and  $T_- \subseteq \mathcal{T} \times \{0\}$ . For both,  $T_+$  and

<sup>2</sup>We refer to condition (3) in the definition of island wrappers.

$T_-$ , we require that the tupels have been drawn from  $D$ :

$$T_{+/-} \subseteq \{t | \exists f \in \mathcal{T}: \circ(f * t) = D\} \times \{0, 1\}$$

$T_+$  denotes examples that the user wants to extract from  $D$ , whereas  $T_-$  denotes tupels that the user did reject. The order in which these tupels have been collected does not matter, so there are no additional requirements to them.

Learning takes place by adjusting  $X$  such that

$$T_X(D) \cap T_- = \emptyset$$

and

$$T_+ \subseteq T_X(D)$$

For the kind of documents we discussed before, this incorporates to find a folding relation  $\mathcal{F} \in \{A, \dots, D\}$  and a minimal set  $P$  of pairs  $(\beta, \gamma)$  such that the extraction task can be solved locally and global. By utilization of the inclusions from fig. 3 one can navigate in a controlled way in terms of testing simple folding relations first.

It should be mentioned, that the cardinality  $\|t\|$  of the tupels in  $T_{+/-}$  does not induce the necessity of considering an additional parameter in the general case. This is more a peculiarity of island wrappers or their representation in fig. 1. When dealing with different cardinalities during the extraction of one document, it is sufficient to extend  $P$  accordingly.

### Conclusions

We did discuss a collection of principles for combining and folding information in a document.

Starting from island wrappers, certain refinements of the underlying assumptions concerning the structure of a potential document have been investigated. A characterization of the extraction as well as of the intended learning process was given. It became clear, that the concept of island wrappers is rather poor what concerns the treatable documents, but there needs not much sophistication for usefull refinements. The resulting document types, laying behind the folding relations under observation, provide a basis for the automatic construction of algorithms which are intended for extracting tupels of subwords from a wide range of documents.

The complexity of the learning process is still an open question and will be subject of an upcoming publication.

### Appendix

This appendix provides definitions for non-standard operators utilized in the paper.

In terms of standard symbols, the following ones have been used:

- $\emptyset$  ..... for the empty set,
- $\epsilon$  ..... the empty word,
- $\mathbb{N}_+$  .. the set of positive natural numbers,
- $\Sigma$  .... an arbitray alphabet,
- $\circ$  ..... the concatenation operator for strings.

**Definition 1 (document)**

A document  $D$  is a non-empty word over  $\Sigma$ :

$$D \in \Sigma^+$$

**Definition 2 (tupel)**

Tupels are elements from  $\mathcal{T}$  or  $\hat{\mathcal{T}}$ , where

$$\mathcal{T} = \bigcup_{m \in \mathbb{N}_+} (\Sigma^+)^m$$

and

$$\hat{\mathcal{T}} = \bigcup_{m \in \mathbb{N}_+} (\Sigma^*)^m$$

For any  $t \in \hat{\mathcal{T}}$  the cardinality of  $t$  is

$$\|t\| = \max\{i \mid t_i \neq \epsilon\}$$

We overload the operational symbol  $\circ$  for the concatenation of tupel components.

**Definition 3 (tupel concatenation  $\circ$ )**

Let

$$\circ: \hat{\mathcal{T}} \rightarrow \Sigma^*$$

with

$$\forall t \in \hat{\mathcal{T}}: \circ t = t_1 \circ t_2 \circ \dots \circ t_{\|t\|}$$

**Definition 4 (merging of tupels  $*$ )**

Let  $\tau$  be any tupel and  ${}^l_k \tau$  be the subtupel

$${}^l_k \tau = (\tau_k, \dots, \tau_l)$$

for

$$1 \leq k \leq l \leq \|\tau\|$$

where  $l$  and  $k$  may disappear for  $k = 1$  or  $l = \|\tau\|$ .

Furthermore, for any  $\tau, \nu \in \hat{\mathcal{T}}$  let

$$\tau + \nu = (\tau_1, \dots, \tau_{\|\tau\|}, \nu_1) + {}_2 \nu$$

Merging tupels consists of an operation

$$*: \hat{\mathcal{T}} \times \hat{\mathcal{T}} \rightarrow \hat{\mathcal{T}}$$

with

$$\forall \tau, \nu \in \hat{\mathcal{T}}: \tau * \nu = (\tau_1, \nu_1) + ({}_2 \tau * {}_2 \nu)$$

There is yet another joining operation for tupels where exclusively components from the same position are pairwise subject to concatenation.

**Definition 5 (position-based concatenation  $\otimes$ )**

Let

$$\otimes: \hat{\mathcal{T}} \times \hat{\mathcal{T}} \rightarrow \hat{\mathcal{T}}$$

with

$$\forall \tau, \nu \in \hat{\mathcal{T}}: \tau \otimes \nu = (\tau_1 \circ \nu_1) + ({}_2 \tau \otimes {}_2 \nu)$$

**Definition 6 (subword relation  $\sqsubseteq$ )**

For any two words  $u, v \in \Sigma^*$  let

$$u \sqsubseteq v \text{ iff } \exists r, s \in \Sigma^*: r \circ u \circ s = v$$

**Definition 7 (precedence of subwords  $\triangleright$ )**

For any words  $u, v, r \in \Sigma^+$ ,  $u \sqsubseteq r, v \sqsubseteq r$  let

$$u \triangleright v \text{ iff } \exists f: \circ(f * v) = r \wedge u \sqsubseteq f_1$$

Note, in this paper, we refrain from acknowledging overlapping of subwords w.r.t. the precedence relation.

**References**

- Chidlovskii, R.; Ragetli, J.; and de Rijke, M. 2000. Wrapper generation via grammar induction. In *Proc. European Conference on Machine Learning*.
- Freitag, D. 1996. Machine learning for information extraction from online documents: A preliminary experiment. Technical report, School of Computer Science at Carnegie Mellon University.
- Grieser, G.; Jantke, K. P.; Lange, S.; and Thomas, B. 2000. A unifying approach to html wrapper representation and learning. In *Discovery Science, Kyoto, Japan, December 04-06, 2000*.
- Hsu, C., and Dung, M. 1998. Generating finite-state transducers for semistructured data extraction from the web. *Information Systems* 23(8).
- Kushmerick, N. 1997. *Wrapper Induction for Information Extraction*. Ph.D. Dissertation, Dept. of Computer Science & Engineering, University of Washington, Technical Report UW-CSE-97-11-04.
- Kushmerick, N. 2000. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence* 118(1-2):15-68.
- Muslea, I.; Minton, S.; and Knoblock, C. 1999. A hierarchical approach to wrapper induction. In *Proc. 3rd Int. Conf. on Autonomous Agents, Seattle, WA*.
- Smullyan, R. 1961. Theory of formal systems. *Annals of Mathematical Studies* 47.
- Soderland, S. 1997. Learning to extract text-based information from the world wide web. In Heckerman, D.; Mannila, H.; Pregibon, D.; and Uthurusamy, R., eds., *Proc. 3rd International Conference on Knowledge Discovery and Data Mining*, 251. AAAI Press.
- Thomas, B. 1999. Learning t-wrappers for information extraction. In *Workshop on Machine Learning in Human Language Technology, Advanced Course on Artificial Intelligence (ACAI'99), Crete Chania, Greece*.
- Zeng, C., and Arikawa, S. 1999. Applying inverse resolution to efs language learning. In *Proc. Int. Conference for Young Computer Scientists*, 480-487. Int. Academic Publishers.