# Anti-Unification Based Learning of T-Wrappers for Information Extraction

**Bernd Thomas**
Institut für Informatik, Universität Koblenz
Rheinau 1, 56075 Koblenz, Germany
Email: bthomas@uni-koblenz.de

### Abstract

We present a method for learning *wrappers* for multi-slot extraction from semi-structured documents. The presented method learns how to construct automatically wrappers from positive examples, consisting of text tuples occurring in the document. These wrappers (*T-wrappers*) are based on a feature structure unification based pattern language for information extraction. The presented technique is an inductive machine learning method based on a modified version of *least general generalization* (*TD-Anti-Unification*) for a subset of feature structures (*tokens*).

## Introduction

The programming of hand-crafted wrappers (Wiederhold 1992) for information extraction (*IE*) from semi-structured documents, e.g. web pages, is a tedious and time consuming process. Furthermore, hand-crafted wrappers are not usable for intelligent web information agents whose task is to collect information from documents autonomously, that have not been analyzed before. Thus it is obvious that they need the capability to learn how to construct wrappers automatically.

Our approach to *automatic wrapper construction* is based on inductive machine learning techniques. The major differences to related approaches in this field (Asish & Knoblock 1997; Califf & Mooney 1997; Kushmerick 1997) are: Our learned wrappers (*T-Wrappers*) are based on a *features structure* (Smolka & Treinen 1994) and *unification*-based (Knight 1989) pattern language (Thomas 1999b) for multi slot *IE*. This approach offers various possibilities to constrain the matching and extraction process by unification techniques on special feature structures (tokens). The main intention for its development was to use *logic programs* (Lloyd 1987) in combination with wrapper techniques to build intelligent web *mediators* (Wiederhold 1992) and information agents.

Instead of using examples that need hand-crafted tagging (Soderland 1997) or labeling (Kushmerick 1997) our approach works with examples consisting of text tuples occurring in the document. Therefore simple *copy* and *paste* actions can provide examples to our *learner*, no knowledge about internal processes or data structures is required. Furthermore we use an un-supervised learning method, which makes it suitable for use with intelligent information agents. The presented approach is able to learn *T-Wrappers* given only a handful of *positive* examples (1 to 10) and not restricted to special structural representations (e.g. tables).

Let us depict the general idea of learning *T-Wrappers* by a small example: If a user reads a web page containing a list of publications, he might think of a relation like: $publication(Author, Title)$. The instances of this relation are the information that can be found in the list of publications shown on the particular web page (e.g. $publication(Knuth, Latex)$). Now we ask the user to determine some instances of this particular relation, such that they are contained in the list of publications. These instances will be our positive set of examples $E^+$. Then the learning task is to find a definition for the relation *publication* such that all examples from $E^+$ and all other listed publications are instances of the relation *publication*. This means we have to learn an extraction procedure that is able to extract the instances in $E^+$ and remaining instances presented on the page. We show that such an extraction procedure directly corresponds to a certain wrapper class, the *T-Wrappers*. Next we will give a brief introduction into *T-Wrappers*. Then our basic learning method is described. Thereafter the generalization of automatically generated extraction patterns from example instances and techniques for the automatic construction of *T-Wrappers* from generalized extraction patterns is defined. Finally a brief conclusion is presented.

## T-Wrappers

The basic intention of *T-Wrappers* is to extract information from semi-structured documents and to represent it as relational instances. This relational representation allows the concepts of *T-Wrappers* and *logic programs* (Lloyd 1987) to be combined more easily and to guide the extraction process with interfering inference processes. Therefore we interpret *T-Wrappers* as special theories and handle *Logic Programs* extended with *T-Wrappers* by means of *theory reasoning* (Stickel 1985). These methods are discussed in (Thomas 1999b).

In general a *T-Wrapper* defines a relation, where its

instances are obtained by extracting information from a document with an associated pattern (*T-Pattern*). A *T-Wrapper* is said to be successful if its set of instances is not empty. The general definition of a *T-Wrapper* is as follows *wrapper_name(D,$\vec{V}$,P) :=* $\{\vec{V}_\sigma \mid$ *with* $\sigma$ *a substitution obtained from matching pattern P with variables V on document D*$\}$ The following example *T-Wrapper* definition is capable of extracting links from HTML documents:
*template* $link(D, Link, Desc) := \#any$ *and* $html(tag = a, href = Link)$ *and* $Desc = +not(html\_end(tag = a))$.

Two basic assumptions must hold to apply our IE techniques as described in (Thomas 1999b): (1) *Only semi-structured documents are considered.* Here semi-structured means: documents containing special annotations, like tags as in *HTML* or command sequences as in *LaTeX* documents. Such text sequences are used as anchors, that mark the beginning and end of the relevant information to be extracted. (2) *Documents are pre-processed (tokenized).* That means, any document is transformed into a list of special *feature structures* (Smolka & Treinen 1994).

**Tokens and T-Patterns** We call a *simple* and acyclic feature structure a *token* if and only if it has a feature named `type` and all feature values consist either of variables or constants. That is, no feature value consists of another feature. Variables are written in capital letters and constants are quoted if they start with a capital letter. Furthermore, we choose a term notation for feature structures that does not code the features to a fixed argument position. Instead the arguments of the annotated term are extended, by the notation `Feature = Value`. For example the text `<b>Pentium 90</b>` may be written as the list of tokens: *[token(type=html, tag=b), token(type=word, txt='Pentium'), token(type=int, val=90), token(type=html_end, tag=b)].* Tokens are not used only to classify the symbol sequences appearing in a document, they also can provide additional information depending on the tokenizer used. This may be of great interest for additional linguistical processing of documents. To recognize a certain token or token sequence in a tokenized document we need techniques to match a given token description with a corresponding token. For feature structures a special unification was defined in (Shieber 1986). For our purposes we need a modified version of this unification, the *token-unification*.

**Definition 1 (Token-Unification)** *Let* $T_1 = token(f_1 = v_1, f_2 = v_2, \ldots, f_j = v_j)$ $j \geq 1$ *and* $T_2 = token(f'_1 = v'_1, f'_2 = v'_2, \ldots, f'_k = v'_k)$ $k \geq 1$ *be tokens. Let* $A_1 = \bigcup_{i=1}^{j}\{(f_i, v_i)\}$ *and* $A_2 = \bigcup_{i=1}^{k}\{(f'_i, v'_i)\}$ *be the feature-value sets of the tokens* $T_1$ *and* $T_2$.
*It must hold that* $\{f_1, \ldots, f_j\} \subseteq \{f'_1, \ldots, f'_k\}$. *If this is not the case, we call* $T_1$ *and* $T_2$ *not token-unifiable, written* $T_1 \not\sqcup T_2$.
*Let* $F = \{f \mid (f, v) \in A_1\}$ *be the set of features* $T_1$ *and* $T_2$ *have in common. The terms* $T'_1$ *and* $T'_2$ *are defined as follows:* $T'_1 := token(f_1 = v_1, \ldots, f_n = v_n)$ *and* $T'_2 := token(f_1 = v'_1, \ldots, f_n = v'_n)$ *with* $f_i \in F \wedge (f_i, v_i) \in A_1 \wedge (f_i, v'_i) \in A_2 \wedge 1 \leq i \leq n \wedge n = |F|$.
$T_1$ *and* $T_2$ *are token-unifiable* iff $T'_1$ *is unifiable with* $T'_2$. *The most general unifier (mgU)* $\sigma$ *of* $T_1$ *and* $T_2$ *is the mgU of* $T'_1$ *and* $T'_2$ *wrt. the usual definition (Lloyd 1987). We write* $T_1 \sqcup_\sigma T_2$ iff $T_1$ *is token-unifiable with* $T_2$ *and* $\sigma$ *is the mgU of the unification from* $T'_1$ *with* $T'_2$. □

*Token-Unification* is motivated by the idea of interpreting the *left* token to be a pattern to match the *right* token, it allows us to set up feature constraints in an easy way. By simply adding a feature to the *left* token we restrict the number of possible matches or by decreasing its features we extend it. To ease the construction of *T-Patterns*, tokens are written as *token_type($m_1$=$w_1$,...,$m_n$=$w_n$)*, for example *token(type=word, txt=money)* ≡ *word(txt=money)*. The needed retransformation, to be able to test for *token-unification* is then straightforward. Therefore we do not differentiate between these two notations in the following sections.

**Example 1 (Token-Unification)**
*html(href=Y)* $\sqcup_\sigma$ *token(type=html, tag=a, href='pubs.html') with* $\sigma$*= [Y/'pubs.html]*
*word(len=4,txt=X)* $\not\sqcup$ *token(type=word, txt=telefon, len=7)*

*T-Patterns* consist of the following basic elements: *Iteration operators* like # or +; operators for *Pattern conjunctions*: *and*, *or*; *Extraction variable assignments*: $Var = pattern$; *Negation or exclusion of tokens*: $not(token_1, \ldots, token_n)$. The whole language of *T-Wrappers* and the semantics of *T-Patterns* is described in detail in (Thomas 1999b).

**Definition 2 (T-Pattern Match)** *Let p be a T-Pattern then* $\mathcal{L}(p)$ *defines a set of token sequences described by p. A T-Pattern p matches on a tokenized document D, iff* $\exists\, s \in \mathcal{L}(p)$ *and* $\exists\, d$ *subsequence of D such that* $|s| = |d|$ *(s and d have the same number of tokens) and d is the sequence of the first* $|s|$ *token in D and* $\exists\, \Sigma$ *such that* $\forall i = 1 \ldots |s|$ : $t_i^s \sqcup_\Sigma t_i^d$, *where* $t_i^s$ *denotes the i-th token of s. d is called the matched sequence of p on D. The substitution* $\sigma_i$ *of an extraction variable* $X_i = pattern_i$ *occurring in p is the token sub-sequence* $ex_i$ *from d matched by the sub-pattern* $pattern_i$ *of p. The substitution* $\Sigma$ *is called* extracted information *from D by p.* □

**Example 2 (Token-Matching)** *Given a document like "Order 10 apples and 60 oranges." and its tokenization as D=[word(txt='Order'), int(val=10), word(txt=apples), word(txt=and), int(val=60), word(txt=oranges)]. Let p (# any and int(val=X) and word(txt=Y) be a T-Pattern. Matching p on D computes the substitution* $\Sigma$*={ (X / 10, Y / apples), (X / 60, Y / oranges)}. Matching a T-Wrapper* $price(D, X, Y) := p$ *on D results in the set tuples* $\{(100, apples), (60, oranges)\}$.

## Learning T-Wrappers

The general task of learning *T-Wrappers* can be stated as follows: Let an arbitrary web page $D$ be given that contains information representable as a set of instances $E$ of a relation $r$, where each argument of an instance is occurring in $D$. Let $E^+ \subset E$ be a set of examples. The learning task is to find an *T-Pattern p* and corresponding *T-Wrapper t* with the help of $E^+$ such that $t(D, \vec{V}_p, p) \subseteq E$. This leads to the definition

of three different types of wrappers that are important for machine learning based IE tasks:

**Definition 3 (sound, correct, perfect)** *Let $E^+$ be a set of example relation instances of a relation r for a document D. Further let $E \supseteq E^+$ be the set of all relation instances of r from D. A learned* T-Wrapper *T from $E^+$ for r wrt. D is*

**sound** *iff $E^+ \subseteq T$.*

**correct** *iff $T \subseteq E$ and T is* sound.

**perfect** *iff $E = T$.* □

The method presented in this paper is separated into three major steps: (1) automatic example pattern generation, (2) generalization of example patterns, (3) automatic construction of *T-Wrappers*.

**Example Pattern Generation** For semi-structured documents it showed that the surrounding text parts (*anchors*) of the information to be extracted are of major interest. Thus the key idea of our approach is to *collect* these *anchors* and to learn a general pattern for the recognition of them. This involves the construction and matching of T-Patterns for each given example in $E^+$. For example, let $E^+$ = {*tv(18.30,News), tv(20.15,Voyager*} be an example set wrt. a web page containing the daily TV-program. The pattern generation, according to a certain wrapper class, and matching process results in a set of instantiated (*grounded*) patterns with following general structure as shown in Example 3. Whereas an $Anchor_i$ is a list of matched token sequences, e.g. *[html(tag=b), html(tag=img, src=star.gif)]*, and $Arg^j$ denotes a pattern for the $j-th$ argument of the example instances.

**Example 3 (General Structure of T-patterns)**
*$Begin\_Anchor$ and $Arg_1$ and $Middle\_Anchor_2$ and $Arg_2$ and ... $Middle\_Anchor_n$ and $Arg_m$ and $End\_Anchor$*

Because we generate one example pattern for each given relation instance, we already know the instantiations of the argument extraction patterns. All we have to do is to instantiate the extraction variables of the *Arg* patterns with the appropriate tokenized arguments of the example instances. This is the basic idea to find the surrounding anchors. These sub-patterns have to match because we know that the arguments of an example relation instance is contained in the document. To recognize the surrounding text parts of these arguments special extraction patterns are defined. They will extract the surrounding token sequences of the example arguments which then will be used for the generalization process.

**Relation Instances and Example Patterns** If we talk about example instances and the generation of example patterns some basic assumptions are to be taken: *Examples:* Our methods work on positive examples exclusively, which means the relation instances provided have to be valid instances due to the actual web page; the arguments have to appear as text on the page. Furthermore the examples should all share a basic structural syntactic type. Otherwise the attempt to learn a common syntactical representation makes no sense. *Left to right text appearance:* The argument order of the example instances represent the linear appearance of the arguments in the document, e.g. $arg_i$ must appear before $arg_{i+1}$ in text. *Shortest argument distance:* It is assumed that the text sequence between two relation arguments $arg_i$ and $arg_i + 1$ occurring in a document does not contain either $arg_i$ or $arg_i + 1$. *Minimal match:* Because of the *shortest argument distance* it follows, that we always have to find the minimal matched sequence *s* of an example pattern *p*.

**Definition 4 (Minimal Match)**
*Let $r(arg_1, \ldots, arg_n)$ be an example relation instance wrt. a document D. Let p be a T-Pattern matching $arg_1, \ldots, arg_n$ in D. Then $s \in \mathcal{L}(p)$ is a* minimal match *of p iff s matches on D and there is no $m \in \mathcal{L}(p)$ such that m matches on D with $|m| < |s|$.* □

**Generating Example T-Patterns** Assume we want to learn a *T-Wrapper* that extracts all instances of a relation *r* with arity *n*, which set of instances *I* is presented on a web page *D*. Let $e := r(arg_1, \ldots, arg_n) \in I$ be an example instance of *r* shown on *D*. We proceed as follows:

1. Tokenize each argument $arg_i$ of the example instance $e \in I$.
2. Create an example pattern with *n* extraction variables $Ex_i = \#any$ with $i = 1 \ldots n$, where *n* is the number of arguments of *e*.
3. Instantiate the extraction variables $Ex_i$ of the example pattern with the according tokenized arguments $arg_i$.
4. According to the wrapper class to use, insert extraction variables $A_m$ and special anchor patterns for later detection of the *anchors*. The number and location of these $A_m = pattern$ units depend on the chosen wrapper class.
5. Match the created pattern *p* on *D*.
6. Calculate the minimal match of *p*.

After matching *p* on *D* the example pattern is fully instantiated (*grounded*), that means, for every extraction variable occurring in *p* exists a substitution $\sigma$ calculated by the matching procedure (Definition 2). Our main goal is to find *anchors* for the further generalization process of the example patterns. The use of extraction variables for *anchors* allows us easily to obtain the relevant *anchors* for each example by means of unification processes (Definition 1). How these anchor token sequences in fact will look like depends on the anchor patterns of the chosen wrapper class. We leave out the discussion of various wrapper classes (Thomas 1999a) and give a brief remark remark on wrapper classes for the generation of example patterns: (1) Restrictions on the anchor patterns is highly important to keep the number of match alternatives low. (2) Restrictions on anchor patterns can be too restrictive and thus causes the whole example pattern to fail (*no sound wrapper*). (3) Considering the text sequences between two arguments as *one* anchor leads often to huge anchors. Especially when the distance between the arguments increases. (4) Too Short anchors will lead to over-generalizations. In general anchors vary, because the web page author is not that disciplined to use always the same syntactical structures to represent semantically related information in the same way or additional informations are presented. Therefore the anchors are *shifted* within the text.

Normal term generalization on short anchors (e.g. least general generalization (lgg) (Plotkin 1971)) will lead to over-generalization. (5) What is needed is a wrapper class that enumerates as few matches as possible for a given example instance (performance), but still ensures that all the given relation instances are matched (*soundness*) and that performs well on finding anchors (*correctness*).

**Island-Wrapper** We mainly focus on the *Island Wrapper* class (Thomas 1999a) that is suited for a variety of extraction tasks, especially multi-slot extraction where the text distance between extraction arguments vary a lot. This class uses an iterative method to estimate the maximal length (*island radius*) of the relevant surrounding text parts (anchors) such that the generated pattern still returns the minimal match.

**Example 4 (Island Wrapper for generation)**

| | |
|---|---|
| *Begin* | *any* |
| $Arg_i$ | *tokenized example argument i* |
| *Middle* $Anchor_i$ | $Right_i$ = *times(N,any) and any_but(*$Arg_i$*) and* $Left_i$ = *times(N,any)* |
| *End* | *any* |

times(N,any) *matches N arbitrary tokens;* any_but(A) *matches every sequence except the token sequence A.*

Thus we think of information *islands* by separating the *middle anchors*, e.g. $Anchor_2^1$, into a left pattern, building the right island part (right anchor) of the preceeding extraction argument and a corresponding right pattern, building the left island part (left anchor) of the succeeding argument. A middle pattern is then allowed to match any sequence but the one matched by the succeeding right anchor. The length of the sequences matched by the left and right anchors is determined by the estimated island radius. We use *Island Wrappers* for the generation of example patterns (Example 4) and the construction of learned *T-Wrappers* (Example 8).

## Generalization of T-Patterns

After the calculation of the *minimal grounded pattern* for each example, we finally have a set of example patterns we can start to generalize. But instead of generalizing the whole set of patterns we only generalize on the matched anchors of these patterns. Thus we learn a more general T-Pattern for the recognition of *anchors*.

**Generalization of Feature Structures** Our main intention to learn a *T-Wrapper* from a set of grounded example patterns is based on the assumption to learn by generalizing matched text-sequences (token sequences). So far we figured out that the important sub-sequences of a match sequence are the anchor sequences. These token sequences are a list of special feature structures. Therefore to generalize anchors means to provide a method to generalize feature structures.

Next we give a definition for the generalization of acyclic and simple feature structures (tokens). This generalization is closely related to the definition of *anti-unification* (Knight 1989) or the calculation of the *least general generalization* of terms (Plotkin 1971). The reason why the standard generalization methods are not applicable is the variable position and number of feature-value pairs tokens can have. Furthermore, we specify a special *least general generalization* for tokens, the *TD-Anti-Unification*.

**Definition 5 (TD-Anti-Unification)** *Let $T_1$ and $T_2$ be tokens with $T_1 := token(f_1 = v_1, \ldots, f_n = v_n)$ and $T_2 := token(f'_1 = v'_1, \ldots, f'_m = v'_m)$ tokens. Let $t_1$ and $t_2$ be the token types of $T_1$ and $T_2$ that is given by $(type = t_1) \in \{f_1 = v_1, \ldots, f_n = v_n\}$ and $(type = t_2) \in \{f'_1 = v'_1, \ldots, f'_n = v'_n\}$.*
*Let $A_1 = \bigcup_{i=1}^{n}\{(f_i, v_i)\}$ and $A_2 = \bigcup_{i=1}^{m}\{(f'_i, v'_i)\}$ be the feature-value sets of the tokens $T_1$ and $T_2$.*
*Further let $F = \{(f, v_1, v_2) | (f, v_1) \in A_1 \wedge (f, v_2) \in A_2\}$ be the set of triples, where $f$ denotes the features $T_1$ and $T_2$ have in common.*
*The* T-Anti-Unification *of tokens $T_1$ and $T_2$, written as $T_1 \sqcap T_2$, is defined as follows:*

$$T_1 \sqcap T_2 := token(f_1 = u_1, \ldots, f_k = u_k)$$
$$with \bigcup_{i=1}^{k}(f_i, u_i) = \{(f, u) | (f, v_1, v_2) \in F \wedge u = anti\text{-}unify(v_1, v_2)\}$$

*The function $anti-unify$ calculates the anti-unification (generalization) of two terms according to the usual definition as defined in (Knight 1989).*
*The* TD-Anti-Unification *of tokens $T_1$ and $T_2$, written as $T_1 \sqcap^D T_2$, is defined as follows:*

$$\underline{If}\ t_1 \neq t_2\ \underline{then}\ T_1 \sqcap^D T_2 := \{T_1, T_2\}$$
$$\underline{else}\ T_1 \sqcap^D T_2 := \{T_1 \sqcap T_2\} \qquad \square$$

The definition of *TD-Anti-Unification* is strongly motivated by the practical usage of *T-Anti-Unification* for the generalization of anchor sequences. The *T-Anti-Unification* as defined in Definition 5 of two tokens with different types results in a token with variable token type. This generalized token is too general for the subsequent matching process of relevant anchor sequences. For example, the anti-unification of $word(txt =' hallo') \sqcap html(tag = b)$ results in a variable what corresponds to the token *any*. Keeping both tokens as generalization and interpreting the result as a disjunction of both, gives a *least general generalization* which suits the process of constructing anchor patterns from generalized anchor sequences much better and does not lead to over-generalization.

**Example 5 (T/TD-Anti-Unification)**
*html(tag=a, href="a.html") ⊓ html(tag=img, src="a.gif") = html(tag=X)*
*html(tag=img, src="dot.gif", width=100) ⊓ html(tag=img, src="circle.gif") = html(tag=img,src=X)*
*word(txt="Hallo") ⊓ int(val=999) = token(type=X)*
*word(txt="Hallo") $\sqcap^D$ int(val=999) = {word(txt="Hallo"), int(val=999)}*

**Anchor Matrices** Assume the case that *n* grounded example patterns are given. Each pattern contains the same number of anchors. If these example patterns belong to the class of *island wrappers* they have exactly $(2 * r)$ anchors where $r$ is the arity of the example relation instances. Example 6 shows a general example island-pattern set. Now we enumerate the anchors of each example pattern from 1 to $2 * r$ and build anchor sets $AS_i$ (multi sets) labeled $i = 1$ to $2 * r$, such that $AS_i$ contains all anchors from the example set with anchor number $A_i^k$ with $k = 1\ to\ n$. Thus an anchor set $AS_i$

consists of all token sequences that have been matched by each anchor *i* of the example patterns 1 to *n*.

An anchor matrix $M^{AS_i}$ according to a given anchor set $AS_i$ is built by constructing every row of $M^{AS_i}$ from $a \in AS_i$. Thus the tokens of a sequence in $AS_i$ can be indexed as $M^{AS_i}_{r,c}$, where *r* denotes the row and *c* the column of the matrix $M^{AS_i}$. In Example 6, an example anchor set and the corresponding anchor matrix is shown.

**Example 6 (Anchor Matrices)**
*Example Pattern Set:*

$$\begin{array}{ccccccc} A_1^1 & EX_1^1 & A_2^1 & \dots & A_{2*r-1}^1 & EX_r^1 & A_{2*r}^1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ A_1^n & EX_1^n & A_2^n & \dots & A_{2*r-1}^n & EX_r^n & A_{2*r}^n \end{array}$$

*Anchor Columns:*

$$\left|\begin{array}{c} A_1^1 \\ \dots \\ A_1^n \end{array}\right| \quad \dots \quad \left|\begin{array}{c} A_{2*r}^1 \\ \dots \\ A_{2*r}^n \end{array}\right|$$

*Anchor set $AS_i$:*
*{[html(tag=b)], [word(txt=no), html(tag=b), html(tag=h2)], [word(txt=yes), html(tag=i)]}*

*Anchor matrix $M^{AS_i}$:*

$$\left|\begin{array}{l} html(tag=b) \\ word(txt=no)\ \ html(tag=b)\ html(tag=h2) \\ word(txt=yes)\ html(tag=i) \end{array}\right|$$

The generalization of anchor matrices is defined by generalizing each column of a matrix.Therefore generalizing an anchor matrix with *n* columns results in a set with *n* generalized token sets. Example 7 shows the generalized anchor matrix of matrix $M^{AS_i}$ from Example 6.

**Example 7 (Generalized Matrix)**

$$\left|\begin{array}{l} html(tag=b) \\ \\ word(txt=X)\ html(tag=Y)\ html(tag=h2) \end{array}\right|$$

**Generating Generalized T-Patterns** The basic interpretation of a generalized anchor matrix is to interpret each column to form a disjunctive token-pattern and the conjunction of these disjunctive token-patterns form the complete anchor pattern.

An important point to mention is the order of the column tokens. A token $t_1$ of row *i* in column *k* and a token $t_2$ of row *i* in column $k+1$ represents the fact that the corresponding anchor pattern for row *i* matched $t_1$ before $t_2$. Thus the column indices state the order of the matched tokens that has to be preserved and construction of generalized T-Patterns.

Assume we generate the following pattern from the example matrix given in Example 7: *(html(tag=b) or word(txt=X)) and html(tag=Y) and html(tag=h2)*, according to the initial interpretation. Furthermore reconsider the anchors of matrix $M^{AS_i}$ shown in Example 6. Obviously the generated pattern does not match the token sequence in row 1 of matrix $M^{AS_i}$ of Example 6. That means the chosen matrix interpretation and following pattern construction processes clearly violate the condition of a sound wrapper defined in Definition 3. If a sub-pattern of a conjunction of patterns fails to match it follows that the whole pattern does not match. The reason is that matched anchor sequences can differ in its length. Therefore it is simply wrong to generate a pattern that matches only the longest given sequence. More clearly the matched anchor sequences given in Example 6 have a length of $1, 3$ and 2. But the generated anchor pattern matches only token sequences with a length of 3; three conjunctive combined patterns each matching one token.

A solution to this problem is to remember the length of the example anchors and to extend the pattern with an additional operator. A special operator for an alternative match possibility is introduced, whenever the end of a example match sequence is reached. For example: Starting with the disjunction of column 1, that is *(html(tag=b) or word(txt=X))*. Because we know that the first example anchor (row 1) has a length of one, we know that any pattern that would match only more than one token would not suffice this example anymore. Therefore the conjunction *and* and the alternative match operator *?* (matching once or never the following pattern) followed by the remaining patterns for each column is added. Applying this procedure recursively results in the generalized patten *(html(tag=b) or word(txt=X)) and ? (html(tag=Y) and ? html(tag=h2))*.

## Automatic Construction of T-Wrappers

The fact that the learned anchor patterns match the text sequences given as examples does not ensure a overall *correct* wrapper. Because one important point is still left untouched: the patterns used for the extraction of the relation arguments

Therefore using only learned *anchor*-patterns clearly is not enough to construct *T-Wrappers* for *IE*. It is also necessary to restrict the patterns for the extraction of the relation arguments. For example, a sub-pattern like $anchor_1$ *and* $(X = \#any)$ *and* $anchor_2$ is obviously to general. Because it is possible that *#any*[1] matches a sequence described by the anchor pattern $anchor_2$. The pattern *#any* would "ignore" the border of the *anchor*. What is needed is a more restrictive pattern, like $not\_in(anchor_2)$, that matches any sequence not in $\mathcal{L}(anchor_2)$. It does not "ignore" the borders setup by the *anchor* patterns. This leads to the definition of various wrapper classes as discussed in (Thomas 1999a). In this paper we state only the basic idea using the *Island-Wrapper* class for automatic wrapper construction from learned anchor patterns.

**Example 8 (Island Wrapper for IE)**

| | |
|---|---|
| *Begin* | $Learned\_Left_1$ |
| $Arg_i$ | $Arg_i$ = *longest_not_in(Learned_Left$_i$ or Learned_Right$_i$)* |
| *Middle* $Anchor_i$ | $Learned\_Right_i$ *and shortest_not_in(Learned_Left$_{i+1}$) and* $Learned\_Left_{i+1}$ |
| *End* | $Learned\_Right_n$ |

shortest_not_in(P) *and* longest_not_in(P) *matches the shortest (longest) possible token sequence s iff* $s \notin \mathcal{L}(P)$.

The construction of the final *Island Wrapper* is straightforward now. We just have to use the scheme of a general *island*

[1]Matches arbitrarily often any token; beginning with the shortest possible match

*wrapper* class for information extraction presented in Example 8 and insert the learned anchor patterns and extraction variable patterns.

## Conclusion

We presented an inductive machine learning method for the automatic construction of wrappers for multi-slot extraction from semi-structured documents. *T-Wrappers* are learned from relation instances only, whereas no further interaction is needed. Current tests showed that for some domains (e.g. computer hardware) even the examples can be automatically derived from knowledge databases. To generalize constructed example patterns, the *TD-Anti-Unification* of tokens and a matrix based algorithm for pattern generation is defined. We tested our method on web pages with various structured information: HTML environments like tables, lists; non rigid structures like announcements. Especially the *Island Wrapper* class performed very well in experiments. Currently we are using the presented techniques and learned *T-Wrappers* for developing an intelligent information agent for the web. One advantage of our approach is the small number and the form of examples needed to learn wrappers. Furthermore it has the ability to learn wrappers extracting information tuples separated by large text units and supports the infromation extraction from less rigid formatted documents. Though first tests are very promising, more exhaustive testing is needed. Future work and research on the PAC-Learnability and complexity investigations of our approach must be done.

## References


Asish, N., and Knoblock, C. 1997. Wrapper generation for semi-structured information sources. In *ACM SIGMOD*. Workshop on Management of Semi-structured Data.

Califf, M. E., and Mooney, R. J. 1997. Relational Learning of Pattern-Match Rules for Information Extraction. In *Working Papers of the ACL-97 Workshop in Natural Language Learning*.

Knight, K. 1989. Unification: A multidisciplinary survey. *ACM Computing Surveys* 21(1):93–124.

Kushmerick, N. 1997. *Wrapper Induction for Information Extraction*. Ph.D. Dissertation, University of Washington.

Lloyd, J. 1987. *Foundations of Logic Programming*. Springer-Verlag, 2 edition.

Plotkin, G. 1971. *Automatic Methods of Inductive Inference*. Ph.D. Dissertation, University of Edinburgh.

Shieber, S. M. 1986. *An Introduction to Unification-Based Approaches to Grammar*. Leland Stanford Junior University: CSLI. CSLI Lecture Notes 4.

Smolka, G., and Treinen, R. 1994. Records for Logic Programming. *Journal of Logic Programming* 18:229–258.

Soderland, S. 1997. Learning to extract text-based information from the World-Wide-Web. In *3rd Int. Conference on Knowledge Discovery and Data Mining*.

Stickel, M. 1985. Automated Deduction by Theory Resolution. *Journal of Automated Reasoning* 1:333–355.

Thomas, B. 1999a. Learning T-Wrappers. Technical report, University of Koblenz. to appear.

Thomas, B. 1999b. Logic programs for intelligent web search. In *11th International Symposium on Methodologies for Intelligent Systems - ISMIS*. Springer, Lecture Notes in Computer Sience. to appear.

Wiederhold, G. 1992. Mediators in the architecture of future information systems. *IEEE Computer* 38–49.