LAIR: A Language for Automated Semantics-Aware Text Sanitization based on Frame Semantics

Steffen Hedegaard, Søren Houen, Jakob Grue Simonsen
Department of Computer Science, University of Copenhagen (DIKU)
Universitetsparken 1, DK-2100 Copenhagen Ø
Denmark
shedegaard@stud.ku.dk, diku@houen.net, simonsen@diku.dk

***Abstract*—We present LAIR: A domain-specific language that enables users to specify actions to be taken upon meeting specific semantic frames in a text, in particular to rephrase and redact the textual content. While LAIR presupposes superficial knowledge of frames and frame semantics, it requires only limited prior programming experience. We have implemented a LAIR compiler and integrated it in a pipeline for automated redaction of web pages. We detail our experience with automated redaction of web pages for subjectively undesirable content; initial experiments suggest that using a small language based on semantic recognition of undesirable terms can be highly useful as a supplement to traditional methods of text sanitization.**

***Keywords*-Redaction; sanitization; frame semantics; domain-specific languages**

I. Sanitization: redaction and expurgation of natural language

Sanitization (aka. redaction) is the removal of sensitive information from a text so that it may be distributed to a certain audience. Traditional sanitization is done by blacking out parts of the texts, typically when governments release declassified documents or a hospital releases medical records. However, sanitization may be broadly construed as removing unwanted words, information or meaning from text, also commonly called *expurgation* or *bowdlerization*. Until recently, manual sanitization was the norm (see Fig. 1), but recently academic [5] and commercial tools have appeared to automate parts of the process.

Sanitization traditionally focuses on removing data such as names, social security and bank account numbers, and occasionally single offensive terms as seen in online fora and games.

To supplement the above approaches we propose a less intrusive and less work-intensive form of sanitization than mere blacking-our or removal of offensive terms. We want to remove *or* rephrase potentially sensitive *sentences* in natural language, the purpose being to render a text appropriate for a certain type of readers (e.g. people with low-security clearance or small children) while retaining as much of the semantic meaning of the original text as possible.

As an example, consider the following sentence from the

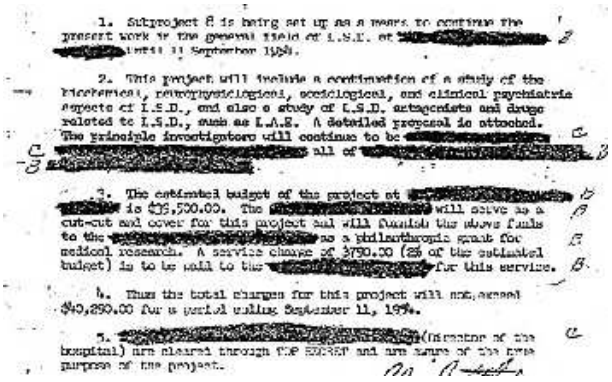


Figure 1. Manually redacted document [1]

May 1,2009 Wikipedia entry on statesman Gabriel Slaughter:

> "Slaughter was appointed justice of the peace in Mercer County."

A simple word-based tool redacting a text for mention of violent behaviour would recognize the word "Slaughter" and redact either the word or a larger part of the text. However, the semantic content of the sentence is distinctly non-violent, bar the named entity "Slaughter". Conversely, were we to write the sentence

> "Slaughter killed foes by the dozen!"

violence is almost certainly involved in the intended semantics and should be appropriately censored, e.g. by blacking out mention of the person named Slaughter or by rephrasing the sentence, depending on reader context. We propose that such *sanitization by semantic analysis* is useful for automated expurgation of undesirable content.

For this purpose we present a small domain-specific language: *Language for Automatically Inferred Redaction* (**LAIR**). In **LAIR**, program snippets may be written that paraphrase or redact natural language texts in English based on Fillmore's notion of *semantic frames*. The language allows manipulation of the text based on actions to be performed on frames or elements therein.

As an example, the tiny **LAIR** program snippet below

searches through all semantic frames of a document and replaces the name of any “Killer” in all “Killing” frames by “J. Doe”:

nf := **namedframes**(**frames**,’Killing’);
fesubst(nf,’Killer’,’J. Doe’);

Thus, when executing the program snippet on the text “Slaughter was appointed justice of the peace in Mercer County. Slaughter killed foes by the dozen!”, we obtain “Slaughter was appointed justice of the peace in Mercer County. J. Doe killed foes by the dozen!”.

LAIR programs are simple and require only limited prior familiarity with general-purpose programming languages; however, superficial knowledge of frames and frame semantics and a list of existing frames names and frame element names are required to fully utilize the capabilities of **LAIR**. We imagine a trained non-IT professional using **LAIR** programs in combination with other tools for automated redaction to write a small suite of **LAIR** snippets that are then deployed and run automatically on large corpora of texts, for example censoring web content. Though expressly designed for sanitization, **LAIR** could in principle be used for any type of frame-based automated paraphrasing of text.

A **LAIR** compiler, full language definition, manual and accompanying documents, and a web interface for sanitization of web pages with **LAIR** programs is available at `http://www.diku.dk/~simonsen/lair/lair.html`

A. Related work

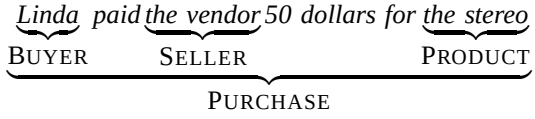
In the ERASE framework [5], an algorithm is devised that queries a domain-specific database of relational entities (e.g. persons, products, diseases) where some entities are flagged as protected. ERASE then redacts text so protected entities cannot be disclosed while maintaining a hopefully readable document. The redaction performed by ERASE is highly efficient, but is done on a per-term basis, rather than per-phrase or per-sentence and currently does not offer any elaborate means to paraphrase redacted material rather than simply block it.

There are a number of commercial software products for document redaction, for example e-Redact [3], RapidRedact[4], and AiREDACT [2]. All of these are closed-source, but their online documentation and ad-hoc experiments reveal that they are based on manual selection of text to be redacted, matching of a number of fixed patterns (for example social security or credit card numbers), or (undisclosed) machine learning algorithms on lexical tokens. In all cases, redaction by the commercial products we surveyed replaces potentially dangerous strings by black boxes or other semantically meaningless structures.

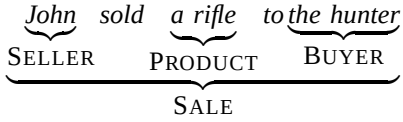
In the area of formal languages, the SCISSOR project aims at parsing and compiling natural language text to several existing domain-specific formal languages, e.g. for expressing playing instruction for soccer-playing robots [8]. The purpose of SCISSOR is to map sentences in natural-language to sentences in a formal language; in contrast, we develop a formal language for manipulating natural languages by semantic parsing.

II. Frames and frame semantics

A *frame* [11] is, loosely, a cognitive structure representing an underlying conceptual structure. For example, the sentence



Represents the conceptual structure of a purchase. The sentence is a frame with *frame name* PURCHASE and includes the frame elements “Linda” (with *frame element name* BUYER), “the vendor” (frame element name SELLER) and “the stereo” (frame element name PRODUCT). A similar cognitive frame containing the same frame elements, but a very different sentence is:



Which in this case is a SALE frame. Both examples could also be classified as a more general frame COMMERCE_SCENARIO which includes a BUYER, SELLER and a PRODUCT.

A. FrameNet

The Berkeley FrameNet project [6], [12] is a database of annotated frames and frame elements that may be used as a resource by a semantic parser for tagging elements of texts in natural language.

FrameNet operates with:

- Semantic Frames: A Frame indicates the semantic meaning of the text, for example COMMERCE_SCENARIO which can be invoked by words such as *Buy* and *Sell*. Frames are universal across languages.
- Frame Elements (FE): Words contributing to the frame-such as the BUYER, SELLER and PRODUCT element.
- Lexical Units (LU): A pairing of word and meaning. Lexical units are *frame evoking words* such as *Buy* and *Sell*. FrameNet provides multiple annotated examples for each lexical unit.

The relationship of these are summed up in the following sentence from [12]:

> *“For example, the* APPLY_HEAT *frame describes a common situation involving a* COOK, *some* FOOD, *and a* HEATING_INSTRUMENT, *and is evoked by words such as bake, blanch, boil, broil, brown, simmer, steam, etc. We call these roles frame*

element(FE)'s and the frame-evoking words are LUs in the APPLY_HEAT *frame.*"

In the FrameNet database, each frame and frame element has been manually annotated with one or more examples of the context in which the lexical unit carries the meaning of the frame implied, as a frame invoking word can invoke more than one frame depending on the context, as shown by the lexical unit *"turned"* in the following two examples:

Anakin turned to the dark side.
BECOMING

Obi-Wan turned to face the Sith lord.
MOVING_IN_PLACE

Both frames are invoked by the LU *"turned"* but have radically different meaning in the contexts. FrameNet gives examples of use for each frame the LU can invoke, but it is left to a Semantic parser to distinguish between the frames and invoke the correct one.

There are several (shallow) semantic parsers that utilize FrameNet to parse and tag words and sentence fragments with frame, FE and LU annotations, e.g. Shalmaneser [7] and the LTH parser [10], and parsers [13], [9] combining input from FrameNet with other databases.

III. OVERVIEW OF **LAIR**

The *input* to a lair program is text in English natural language where sentences and sentence fragments have been tagged with metadata identifying the sentence as a given *frame* (possibly "don't know") and various fragments of a sentence identified as *frame elements*, and the *output* of a **LAIR** program is a text in English. Thus, in the first example of Section II, the input to a **LAIR**-program would be the text with sentence fragments "Linda", "the vendor", and "the stereo" tagged as frame elements BUYER, SELLER, and PRODUCT, and the entire sentence tagged as a PURCHASE frame.

The tagging of input text is performed by a semantic parser–in all practical uses, a **LAIR** compiler will be pipelined with a semantic parser accepting an input which is an English text in natural language. The exact effect of executing a **LAIR** program in such a pipeline depends highly on the underlying semantic parser. Thus, a **LAIR**-pipeline applied to a certain text may yield different outputs if different semantic parsers are used. This is unavoidable: Unless a text is perfectly tagged with frame information, a feat currently only accomplishable by humans, some variation in the output will happen.

The **LAIR** program itself is a sequence of statements, each designating some action to be taken on collections of frames or frame elements from the input document. Conditional statements and looping over the frames or frame elements of a collection is allowed.

The representation of frames and frame elements in **LAIR** is accomplished by having a notion of *frame objects* and *frame element objects*. Each frame of the input text is represented by a frame object, each of its frame elements by a frame element object. Only frame objects and frame element objects can be manipulated. Thus, the user cannot write a string directly in the program and have it parsed as a frame, but only access frame objects occurring in the input.

IV. EXAMPLES OF **LAIR**-PROGRAMS

This section contains examples of how to do operations on a text using LAIR.

Redact all frames of one kind We replace the text of all frames involving killing with the text "[Removed]".

```
framesubst(namedframes(frames,'Killing'),'[Removed]');
```

Anonymize actors in a frame of one kind In case of a frame involving "killing" with a specific weapon, replace all victims with "B. Bunny" if the name of the killer is "E. Fudd"

```
wkillname := fetextsearch(frames,'Killing','Killer','E. Fudd');
wkillweap := fetextsearch(wkillname,'Killing',
                          'Murder_Weapon','Shotgun');
fesubst(wkillweap,'Victim,'B. Bunny');
```

This first selects all killing frames with 'E. Fudd' as killer and from these selects the frames where the murder weapon is a shotgun. All victims in this collection of frames is then substituted with 'B. Bunny'.

Anonymize only if a specific frame occurs elsewhere In case of a killing occurring anywhere in the text, substitute all cooks and foods in "Apply_heat" frames.

```
murderframes := namedframes(frames,'Killing');
if (not isempty(murderframes)) then
   fesubst(namedframes(frames,'Apply_heat'),'Cook','B. Bunny');
   fesubst(namedframes(frames,'Apply_heat'),'Food','Gerbil');
end
```

If no killing frames occur, no cooks or foods will be anonymized.

Anonymize if name occurs in a specific frame type In case of one or more killings occur in the text, anonymize the occurrences of the names of all killers throughout the entire text (not just in frames involving killing)

```
killerframes := namedframes(frames,'Killing');
foreach murder in killerframes do
        killername := getfe(murder,'Killer');
        textsubst(frames,'killername','E. Fudd');
end
```

Example with actual in- and output As an example of a small program with actual in- and output, consider the removal of frames relating to death or dying:

```
deathframes := namedframes(frames,'Death');
dyingframes := namedframes(frames,'Dying');
framesubst(deathframes,'[REDACTED]');
framesubst(dyingframes,'[REDACTED]');
```

The following text snippet is from the May 1, 2009 Wikipedia entry on Gabriel Slaughter:

"Gabriel Slaughter (December 12, 1767 ? September 19, 1830) was the seventh Governor of Kentucky and was the first person to ascend to that office upon the death of the sitting governor. His family moved to Kentucky from Virginia when he was very young. He became a member of the Kentucky militia, serving throughout his political career. He received a citation from the state legislature in recognition of his service at the Battle of New Orleans. After spending a decade in the state legislature, Slaughter was elected lieutenant governor, serving under Charles Scott. With the War of 1812 looming at the end of his tenure, Slaughter ran for governor against Isaac Shelby, the state's first governor and a noted military leader. Shelby beat Slaughter soundly. Four years later, Slaughter was again elected as lieutenant governor, serving under George Madison. Madison died a short time into his term, whereupon Slaughter became acting governor. He sought to be sworn in as governor, but public sentiment turned against him when he replaced Shelby's son-in-law with John Pope as Secretary of State. Pope was an unpopular figure in Kentucky and, after his appointment, some in the General Assembly began to call for a special election to replace Slaughter. The measure did not pass, but Slaughter was never able to shed the title of "acting governor." Following his term as governor, Slaughter became a Baptist lay minister and served on the first board of trustees of Georgetown College. He died September 19, 1830 and was buried in his family's cemetery."

After running the program on the above text, we obtain:

"Gabriel Slaughter ([REDACTED] His family moved to Kentucky from Virginia when he was very young. He became a member of the Kentucky militia, serving throughout his political career. He received a citation from the state legislature in recognition of his service at the Battle of New Orleans. After spending a decade in the state legislature, Slaughter was elected lieutenant governor, serving under Charles Scott. With the War of 1812 looming at the end of his tenure, Slaughter ran for governor against Isaac Shelby, the state's first governor and a noted military leader. Shelby beat Slaughter soundly. Four years later, Slaughter was again elected as lieutenant governor, serving under George Madison. [REDACTED] He sought to be sworn in as governor, but public sentiment turned against him when he replaced Shelby's son-in-law with John Pope as Secretary of State. Pope was an unpopular figure in Kentucky and, after his appointment, some in the General Assembly began to call for a special election to replace Slaughter. The measure did not pass, but Slaughter was never able to shed the title of "acting governor." Following his term as governor, Slaughter became a Baptist lay minister and served on the first board of trustees of Georgetown College. [REDACTED]"

V. Implementation: Compiler and pipeline for web content sanitization

LAIR was specifically designed for redaction of real-world corpora of text in natural language, the perennial example being the World Wide Web. To show the feasibility of our approach, we needed to both circumvent the problem of misspellings commonly occurring in text corpora from uncontrolled sources and to test how much efficiency and proper sanitization can be garnered by using a current top-of-the line semantic parser based on FrameNet.

We built a **LAIR** compiler integrated with the LTH shallow semantic parser [10]; currently, the compiler and parser are not decoupled, and parts of the compiler will need to be reimplemented if other semantic parsers are to be used. We implemented a software pipeline for the sanitization of web pages. We implemented a standard statistics-based spelling corrector to correct spelling errors that may confuse the parser. To ensure further robustness, all input text is split into sentences that are further tokenized to ensure compliance with the parser.

All changes to the text are logged in a changeholder state and are reversed on a sentence-by-sentence basis if they are not redacted by the **LAIR** program. Exceptions include certain dynamic and style sheet features on web pages that are prohibitively hard to reverse. An overview of the pipeline is given in Figure 2.

VI. Experimental experience and results

While **LAIR** is a potentially powerful tool for sanitization purposes, the efficiency of the language is highly contingent on the accuracy and speed of the underlying semantic parser. To demonstrate feasibility using the current generation of semantic parsers employing the FrameNet data, we performed an ad-hoc experiment sanitizing pages on the world wide web using the pipeline of Section V.

We compiled a subjective list of potentially "dangerous" frames alluding to such activity that may be seen in Figure 3. We subsequently wrote the small **LAIR**-program of Figure 4 that first loops through all frames and censors any frame deemed offensive. To avoid inappropriately censoring the very common "Observable_bodyparts" frame (that refers to sentences concerning bodyparts such as arms or legs, but also sexual organs), we made a whitelist of bodyparts deemed "safe". The "Observable_bodyparts" frames were only redacted if the (unique) "Body_part" frame element they contain was *not* on the whitelist. The program was specifically designed to only consider semantic information

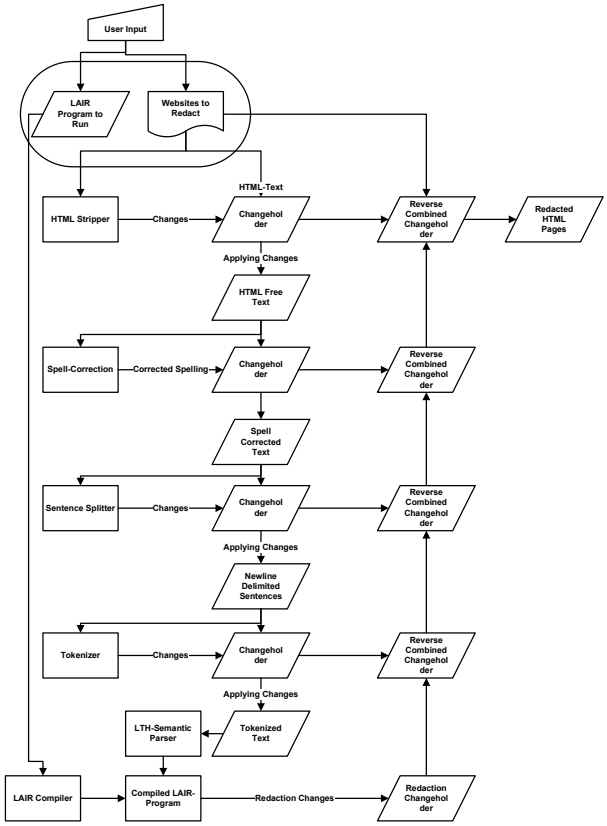


Figure 2. Flowchart of pipeline for web page redaction using **LAIR**.

Abusing	Addiction
Arson	Bearing_arms
Catastrophe	Cause_harm
Committing_crime	Corporal_punishment
Crime_scenario	Criminal_investigation
Criminal_process	Damaging
Dead_or_alive	Death
Destroying	Dying
Endangering	Excreting
Experience_bodily_harm	Fighting_activity
Hostile_encounter	Imprisonment
Intoxicants	Intoxication
Kidnapping	Killing
Offenses	Prison
Rape	Revenge
Robbery	Severity_of_offense
Shoot_projectiles	Shooting_scenario
Terrorism	Theft
Weapon	

Figure 3. Frames names selected as "offending" in the experiment

instead of syntactical as we wanted to test the capabilities of semantic sanitization. We could have augmented the program with traditional black-list based word recognition and text substitution (both possible in **LAIR**), but chose not to do so as this can be accomplished by existing tools.

We selected a corpus of 20 web pages for analysis (an extra page was selected, but crashed the LTH parser

```
foreach framename in offendingframes do
        redactframes := namedframes(frames,framename);
        framesubst(redactframes,
                '(sentence deleted due to inappropriate content)');
end;

foreach curframe in namedframes(frames,'Observable_bodyparts') do
        safe := false
        foreach bodypart in safeparts do
                if name(getfe(curframe,'Body_part')) = bodypart
                then
                        safe := true;
                end;
        end;
        if not safe
        then
              singleframesubst(curframe,
              '(sentence deleted due to possible nudity)');
        end;
end;
```

Figure 4. Program used in the experiment for sanitizing explicit content

irreparably); every page was selected with an aim to contain both potentially offending as well as innocent content in four broad categories: Mixed content (5 pages), sexually explicit content (4 pages), newspaper and encyclopedia articles (9 pages), and foul language/insults (2 pages). The pages contained a total of 1654 sentences.

For each sentence in the input we manually (and subjectively) assessed whether the sentence contained mention of sexually explicit or violent content, or content related to criminal investigation, incarceration or drug use. The annotated sentences were then manually compared to the output of the **LAIR** pipeline.

For each of the four categories, we computed the standard measures of precision and recall defined as:

$$\text{Precision} = \frac{\text{offending sentences redacted}}{\text{sentences redacted}}$$

$$\text{Recall} = \frac{\text{offending sentences redacted}}{\text{offending sentences}}$$

and the usual F1-score defined by $F1 = (2 \cdot \text{Precision} \cdot \text{Recall})/(\text{Precision} + \text{Recall})$. Finally, we computed the accuracy measure, defined as the percentage of sentences classified correctly as either offending or innocent.

The results are shown in Table I.

Category	Precision	Recall	F1	Accuracy
Sexual content	0,75	0,53	0,62	0,70
News/enc. articles	0,56	0,93	0,70	0,91
Foul language/insults	0,74	0,41	0,53	0,79
Mixed	0,32	0,73	0,45	0,84
Entire corpus	0,58	0,67	0,62	0,84

Table I
RESULTS FOR THE TEST CORPUS.

In general, news and encyclopedia articles fared best on recall, but rather poorly on precision, a trend reversed

for the remaining categories. We attribute this to a lack of frame types in FrameNet having explicitly to do with actions of a sexually explicit nature, and to the fact that foul language and insults often occurred in texts as part of a frame whose main semantic content was unrelated. Had we in addition used black-list based text substitution, we would expect to obtain better results. The generally unimpressive recall, but passable precision suggest that the state of current semantic parsers are not adequate for using **LAIR** for safety-critical purposes, but that **LAIR** may at least cull some questionable content and could conceivably be used to supplement traditional methods for automated sanitization.

Beyond the constraints of the experiment above, a qualitative look at **LAIR** redaction reveals that the system shows both invigorating successes in cases where traditional redaction would fall short, as well as disappointing failures to redact, largely due to parser shortcomings. Successes include proper names such as "Slaughter" and "Battle" are left unredacted when scrutinizing for violent content, unless they appear in a semantic frame of such content. Likewise, common single words with ambiguous meaning (e.g. "hit") are treated depending on their semantic content, leading to sentences such as "The mother had been hit by a car only days after giving birth" to be flagged as a "Cause_Harm" frame and a candidate for redaction of violent content. However, the reliance of **LAIR** on the underlying semantic parser inevitably leads to failures: Unknown slang is often left unredacted, e.g. in the sentence "Not content with the reaming they give consumers …", the word "reaming", slang for a violent sex act, is not properly identified.

VII. Conclusion and future work

We have presented **LAIR**: A language for specifying actions based on frame semantics to automatically sanitize text. **LAIR** may be seen as a first domain specific language for manipulating natural language text by professionals with knowledge of (a specific framework for) natural language semantics. It is conceivable that **LAIR** would benefit from a more declarative SQL-like syntax; if so, the language could possibly specify redactions depending on general entities, not just frames and frame elements, hence be parameterized over the method of semantic analysis. We aim to investigate this in future work.

Finally, the efficiency of **LAIR** is extremely contingent on the performance of the underlying semantic parser. The current crop of FrameNet-based semantic parsers have very high memory footprint and somewhat questionable stability, leading to fairly inefficient redaction (on the order of several minutes for parser startup and gigabytes of space usage, irrespective of document size). More efficient semantic parsers would lead to greater practical usability of **LAIR** and other such languages. For possible industrial applications, running **LAIR** atop a highly efficient semantic parser and performing a larger-scale experiment using **LAIR** programs in conjunction with existing syntax-based methods for sanitization will be necessary.

References

[1] Sanitization (sensitive information). Wikipedia, retrieved May 1. 2009.

[2] AiREDACT. Aptitude Solutions Inc., November 2008. http://www.aptitudesolutions.com/Product-aiRedact.html.

[3] e-Redact. Footprint Solutions Limited, November 2008. http://www.e-redact.co.uk.

[4] RapidRedact. Onstream Systems, November 2008. http://www.rapidredact.com.

[5] V. T. Chakaravarthy, H. Gupta, P. Roy, and M. K. Mohania. Efficient techniques for document sanitization. In J. G. Shanahan, S. Amer-Yahia, I. Manolescu, Y. Zhang, D. A. Evans, A. Kolcz, K.-S. Choi, and A. Chowdhury, editors, *CIKM*, pages 843–852. ACM, 2008.

[6] C. B. Charles, C. J. Fillmore, and J. B. Lowe. The berkeley framenet project. In *in Proceedings of the COLING-ACL*, pages 86–90, 1998.

[7] K. Erk and S. Pado. Shalmaneser - a flexible toolbox for semantic role assignment. In *Proceedings of LREC 2006*, Genoa, Italy, 2006.

[8] R. Ge and R. Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 9–16, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[9] A.-M. Giuglea and A. Moschitti. Shallow semantic parsing based on framenet, verbnet and propbank. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *ECAI*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 563–567. IOS Press, 2006.

[10] R. Johansson and P. Nugues. Semantic structure extraction using nonprojective dependency trees. In *Proceedings of SemEval-2007*, Prague, Czech Republic, June 23-24 2007.

[11] M. Petruck. Frame semantics. In *Handbook of Pragmatics. John Benjamins*, pages 1–13. John Benjamins, 1996.

[12] J. Ruppenhofer, M. Ellsworth, M. R. L. Petruck, C. R. Johnson, and J. Scheffczyk. FrameNet II: Extended theory and practice. Technical report, ICSI, 2005.

[13] L. Shi and R. Mihalcea. Putting pieces together: Combining framenet, verbnet and wordnet for robust semantic parsing. In A. F. Gelbukh, editor, *CICLing*, volume 3406 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2005.