# Constructing custom semantic representations from a generic lexicon

Myroslava O. Dzikovska, Mary D. Swift, James F. Allen
Computer Science Department
University of Rochester
Rochester, NY, USA, 14627
{myros, swift, james}@cs.rochester.edu

*Abstract* Language input to practical dialogue systems must be transformed into a semantic representation that is customized for use by the back-end domain reasoners. At the same time, we want to keep front-end system components as domain independent as possible for easy portability across multiple domains. We propose a transparent way to achieve domain specificity from a broad-coverage domain-independent parser. We maintain a domain-independent ontology and define a set of mappings from it into a domain-specific knowledge representation. We use the mappings to customize the semantic representations output by the parser for the reasoners, and to specialize the lexicon to the domain, which improves parsing speed and accuracy. This method facilitates our approach to instances of semantic type coercion common in our domains by combining lexical representations with domain-specific constraints on interpretation.

## 1 Introduction

Most dialogue systems are developed for specific domains to maximize performance efficiency. Back-end system components use a knowledge representation tailored to application needs, and the language input must be converted into that representation. This is traditionally achieved by linking the lexical definitions directly to the concepts in the domain-specific ontology. This linking is also commonly used to bring in domain-specific selectional restrictions to increase parsing efficiency. Adapting the system to a new domain requires relinking the lexicon to the new ontology. We propose an alternative method that is an easy, transparent way to achieve domain

specificity from a broad-coverage, deep parser. In our approach, we maintain two ontologies: domain-independent for the parser and domain-specific for the knowledge representation, and we define a set of mappings between domain-specific knowledge sources and the semantic representations generated by the parser. Our method allows us to easily obtain domain-specific semantic representations without modifying the lexicon or grammar. We also use the mappings to specialize the lexicon to the domain, resulting in substantial improvement in parsing speed and accuracy. In this paper, we describe our customization method and illustrate how it facilitates our approach to semantic type coercion by combining lexical representations with domain-specific constraints on interpretation.

The customization method described here was developed in the process of adapting the TRIPS dialogue system [2] to several different domains, including a transportation routing system [3] and a medication scheduling system [7]. We assume a dialogue system architecture [1] that includes a speech module, a parser, an interpretation manager (responsible for contextual processing and dialogue management), and a back-end application responsible for the general problem-solving behavior of the system. Our philosophy is to keep all components as domain-independent as possible for easy portability, and to develop tools to facilitate component customization to different domains. In particular, our goal is to develop a parser and grammar that can handle language input from different application domains, but still retain speed and efficiency.

## 2 Background

A common approach to customizing parsers for different domains uses corpora to train a probabilistic model. For example, the TINA parser [16] in the multi-domain dialogue system GALAXY [8] provides a mechanism for semi-automatically learning a probabilistic model from a training corpus. For each new domain, training data must be collected and annotated. Similarly, parsers for information extraction use probabilistic models trained on text corpora to learn subcategorization frames and selectional restrictions in a given domain (*e.g.*, [17]).

When suitable training corpora are not available, the domain ontology can be used to customize a wide-coverage parser. One such system, AUTOSEM [15], uses COMLEX [11] as a source of reusable syntactic information. The subcategorization frames in the lexicon are manually linked to the domain-specific knowledge representation. The linking is performed directly

from syntactic arguments (*e.g.*, subject, object) to the slots in a frame-like domain representation output by the parser. Rosé's approach speeds up the process of developing tutoring systems in multiple domains. In a similar approach, McDonald [12] maps the output of a partial parser to the semantic representation for information extraction to improve parsing speed and accuracy.

Another issue that needs to be addressed in porting dialogue systems between domains is linking the lexical entries to the domain semantic representation. TINA lexical entries specify the frames to which the words are linked, and GALAXY requires that all system components use a shared ontology. Therefore, for new domains the system lexicon needs to be re-linked to the new ontology. The AUTOSEM architecture makes the re-linking easier by separating the re-usable syntactic information in COMLEX from the links to the domain ontology.

While AUTOSEM re-uses syntactic information across domains, it does not provide a way to re-use common semantic properties of words. In our approach, we introduce an intermediate layer of abstraction: a generic ontology for the parser (the **LF Ontology**) that is linked to the lexicon and preserved across domains. In this way, we preserve basic semantic features associated with lexical entries (*e.g.*, whether a word represents an event or an object) as well as some general selectional restrictions that do not change across our domains (*e.g.*, the verb *cancel* takes an action or event as an object argument). The parser uses this ontology to supply meaning representations of the input speech to the interpretation manager, which handles contextual processing and dialogue management and interfaces with the back-end application. The domain-specific ontology used for reasoning (the **KR ontology**) is localized in the back-end application. We then customize the communication between the parser/interpretation manager and the back-end application via a set of mappings between the LF and KR ontologies (section 4).

Our method of separating domain-specific and domain-independent ontologies has a number of advantages. First, it allows developers to write mappings in semantic terms at a higher level of abstraction, so there is no need to address the details of the grammar and subcategorization frames such as those used in COMLEX. Developers can instead use descriptive labels for semantic arguments, such as AGENT, THEME, etc. Second, it allows developers to take advantage of the hierarchical structure of the domain-independent ontology and write mappings that cover large classes of words (see example in section 4). Third, the mappings are used to convert the generic representation into the particular form utilized by the back-end

application, either a frame-like structure or a predicate logic representation, without changing the grammar rules, as described in [6]. Finally, the lexicon is specialized to the domain via the mappings, which both improves parsing speed and accuracy and provides a transparent way to map lexical forms to domain-specific meanings.

## 3 Domain-independent representation

Entries in the generic lexicon are linked to the LF ontology, a domain-independent ontology for the parser. The LF ontology is kept as general as possible so it can be used in multiple domains. The LF ontology consists of a set of representations (LF types) that classify entities corresponding to (classes of) lexical items in terms of argument structure and selectional restrictions, with a hierarchical structure inspired by FRAMENET [9]. Every LF type declares a set of thematic arguments with selectional restrictions. The LF ontology is used in conjunction with a unification-based grammar that covers a wide range of syntactic structures.

The LF types are organized in a single-inheritance hierarchy. We implement multiple inheritance via semantic feature vectors associated with each LF type. The features correspond to basic meaning components and are based on the EuroWordNet [18] feature system with some additional features we have found useful across domains. While the same distinctions can be represented in a multiple inheritance hierarchy, a feature-based representation makes it easy to implement an efficient type-matching algorithm based on [13]. More importantly, using semantic feature vectors allows us to easily augment semantic information associated with a lexical entry during the customization process and the semantic coercion operations described below. Our choice of the LF types and semantic features included in the LF ontology is linguistically motivated, an approach similar to Lascarides and Copestake [10], who try to encode the knowledge "appropriate as a locus for linguistically-relevant generalizations" in their lexical semantic hierarchy.

Word senses are treated as leaves of the semantic hierarchy. The following information is specified for every word sense in the lexicon:

- Syntactic features such as agreement, morphology, etc.;
- LF type;
- The semantic feature vector (mostly inherited from LF type definition)
- The subcategorization frame and syntax-semantics mappings.

```
(define-type LF_CONSUME
 :semfeatures (Situation (aspect dynamic) (cause agentive))
 :arguments   (AGENT (Phys-obj (intentional +) (origin living)))
              (THEME (Phys-obj (form substance))))

(define-type LF_DRUG
 :semfeatures (Phys-obj (form substance)))
```

Figure 1: LF type definitions for LF_CONSUME and LF_DRUG.

To illustrate, consider the definition for the verb *take* in the sense *to consume substances*, as in *take aspirin*. The LF type definition for the *consume* sense of *take* is shown in Figure 1. It specifies a generic semantic feature vector associated with the type and selectional restrictions on its arguments. Intuitively, LF_Consume defines a dynamic event in which an intentional living being (AGENT) consumes some substance. The lexicon entry for *take* (shown in Figure 2) is linked to the LF type definition by mapping the syntactic roles to the semantic argument labels in the LF type. The selectional restrictions specified in the LF type arguments are propagated into the lexicon. When trying to create a verb phrase, the parser checks that the semantic feature vector specified in the argument restriction matches the semantic feature vector associated with the noun phrase. Thus, only noun phrases marked as substances in their feature vectors *(form substance)* are accepted as direct objects of verb *take* in the *consume* sense.

The parser uses the LF types to construct a general semantic representation (the base logical form) of the input language, which is a flattened and unscoped logical form using reified events [5]. A base logical form for *take aspirin* is shown in Figure 3. Note the examples show only the parts of our semantic representation relevant to customization. We use a richer logical form in the system, and omit details such as speech acts and information from determiners and quantifiers due to space limitations.

## 4 Constructing domain-specific representations

To produce domain-specific KR representations from the base logical form, we developed a method to customize parser output. The current system supports two knowledge representation formalisms often used by reasoners: a frame-like formalism where types have named slots, and a representation that has predicates with positional arguments. For this paper, we assume a

```
(take
  :lf LF_CONSUME*take
  :semfeatures (Situation (aspect dynamic) (cause agentive)))
  :subject (NP (Role AGENT)
               (Restriction (Phys-obj (intentional +) (origin living))))
  :object (NP (Role THEME) (Restriction (Phys-obj (form substance)))))

(aspirin
  :lf LF_DRUG*aspirin
  :semfeatures (Phys-obj (form substance) (origin artifact)))
```

Figure 2: Lexicon entries for *consume* sense of *take* and for *aspirin*.

```
(TYPE e LF_CONSUME*take) (AGENT e +YOU+) (THEME e v1)
(TYPE v1 LF_DRUG*aspirin)
```

Figure 3: LF representation of *take aspirin*.

frame representation used by the reasoners.

We use LF-to-frame transforms to convert from base logical form into a frame representation. These transforms specify the KR frame that the LF type maps to, the mappings between LF arguments and KR slots, and additional functions that can be applied to arguments during the transform process. These transforms can be simple and name the slot into which the value is placed, as in Figure 4, or more elaborate and specify an operator expression that is applied to the value, which we use, for example, for semantic type coercion, described in section 6.

The Interpretation Manager takes the parser's base logical form, determines the most specific transform consistent with it, and uses the transform to convert the base representation into the domain knowledge representation.

To illustrate, consider the base logical form for *take aspirin* in Figure 3. The applicable transform is `takemed-transform` (Figure 4a), and when the Interpretation Manager applies it, the frame shown in Figure 4c is the result. The details of the process, and the way transforms are used to adapt the base logical form to different possible formalisms used by reasoners are described in [6]. Note that this single transform covers a class of words. For example, the same transform also covers *have* used in the *consume* sense, as

```
(a) (LF-to-frame-transform takemed-transform
     :pattern (LF_CONSUME TAKEMED)
     :arguments (AGENT :ACTOR)
                (THEME :MEDICATION))

(b) (define-class TAKEMED
     :isa ACTION
     :slots (:ACTOR PERSON)
            (:MEDICATION MEDICINAL-SUBSTANCE))

(c) [TAKEMED
     :ACTOR [PERSON +YOU+]
     :MEDICATION [MEDICINAL-SUBSTANCE V1]
```

Figure 4: LF-to-frame-transform. (a) Transform for LF_CONSUME type; (b) Definition of KR class TAKEMED that the transform maps into; (c) The KR frame that results from applying the transform in (a) to the consume event representation in Figure 3.

in *have an aspirin every day.* Transforms can also utilize a lexical form of a word. For example, medication names are all grouped as leaves under the LF_Drug type in our ontology, but the transform for drugs uses the lexical form of the item transformed to determine the correct KR class name for the mapping.

## 5 Lexicon Specialization

We use the transforms described above in a post-processing stage to customize the generic parser output for the reasoners. We also use them in a pre-processing stage to specialize the lexicon, which speeds up parsing and improves semantic disambiguation accuracy by integrating the domain-specific semantic information into the lexicon and grammar.

We pre-process every entry in the lexicon by determining all possible transforms that apply to its LF type. For each transform, we create a new sense definition identical to the existing generic definition plus a new feature *kr-type* in its semantic vector. The value of *kr-type* is the KR ontology class that results from applying this transform to the entry. Thus, we obtain a (possibly larger) set of entries which specify the KR class to which they

(kr-type MEDICINAL-SUBSTANCE) => (phys-obj (form substance))

Figure 5: Feature inference rule to derive a feature vector for kr-type MEDICINAL-SUBSTANCE

belong. We then propagate type information into the syntactic arguments, making tighter selectional restrictions in the lexicon. This allows us to control the parser search space better and obtain greater parsing speed and accuracy.

When specializing the lexicon to the medical domain, given the definition of the verb *take* and LF_Consume in Figure 1, and the definitions in Figure 4, the system determines the applicable LF-to-frame-transform, `takemed-transform`, and adds *(kr-type takemed)* to the feature vector of *take*. Next, it applies the argument mappings from the transform. For example, the mappings specify that the LF argument THEME maps to KR slot :medication, and therefore should be restricted to medicinal substances. Since THEME is realized as a direct object of *take*, *(kr-type medicinal-substance)* is added to the semantic vector in the object restriction. Similar transforms are applied to the rest of the arguments.

As a result, a new definition of *take* with stricter selectional restrictions is added to the lexicon, and suitable objects of *take* must not only be substances, but also identified as medicines. Similarly, the definition of *aspirin* is specialized using a domain rule that maps drug names into medicinal substances. This rule sets the kr-type feature of *aspirin* to be a subclass of KR type MEDICINAL-SUBSTANCE.

In the process of specialization, the parser uses a feature inference mechanism we have implemented in our system. In the generic lexicon, it is used to express dependencies between feature values. For example, if something is marked as a human being *(origin human)*, it is also a solid object *(form solid-object)*. For the KR specialization process we add rules that declare dependencies between the values of *kr-type* feature and the values of domain independent features. It is used in particular in the way we handle semantic type coercion, described in the next section.

The LF to KR mapping includes the feature inference rule in Figure 5, which indicates that the *form* feature of all words identified as medicinal substances in our domain should be set to *substance*. This does not make a difference in this example because *aspirin* is already marked with *(form substance)*, but it will have the impact when other types such as *prescription* are coerced into medications, as described in section 6.

| | Generic | Transportation | Medical |
|---|---|---|---|
| # of senses | 1947 | 2028 | 1954 |
| # of KR classes | - | 228 | 182 |
| # of mappings | - | 113 | 95 |

Table 1: Some lexicon statistics in our system

### 5.1 Evaluation

Lexicon specialization considerably speeds up the parsing process. We conducted an evaluation comparing parsing speed and accuracy on two sets of 50-best speech lattices produced by our speech recognizer: 34 sentences in the medical domain and 200 sentences in the transportation domain. Table 1 describes the lexicon and ontologies used in these domains. The results presented in Table 2 show that lexicon specialization considerably increases parsing speed and improves disambiguation accuracy. The times represent the average parsing time per lattice, and the errors are the number of cases in which the parser selected the incorrect word sequence out of the alternatives in the lattice.[1]

The improvement comes from two sources. The tighter selectional restrictions limit the search space and help to correctly disambiguate according to our domain knowledge. In addition, our parser has preference values associated with different senses, and correspondingly with constituents that use those senses. We increase the preference values for specialized entries, so they are tried first during parsing, which helps the parser find an interpretation for in-domain utterances faster.[2]

The amount of work involved in domain customization is relatively small. The lexicon and grammar stay essentially the same across domains, and a KR ontology must be defined for the use of back-end reasoners anyway. We need to write the transforms to connect the LF and KR ontologies, but as their number is small compared to the total number of sense entries in the lexicon and the number of words needed in every domain (see Table 1), this represents an improvement over hand-crafting custom lexicons for every domain.

[1] Choices in which a different pronoun, article or tense form were substituted, *e.g.*, *can/could I tell my doctor* were considered equivalent, but grammatical substitutions of a different word sense, *e.g.*, *drive/get the people* were counted as errors.

[2] Unspecialized entries have a lower preference, so parses for out of domain utterances can be found if no domain-specific interpretation exists.

| | Transportation | Medical |
|---|---|---|
| # of sentences | 200 | 34 |
| Time with KR (sec) | 4.35 (870) | 2.5 (84) |
| Time with no KR (sec) | 9.7(1944) | 4.3 (146) |
| Errors with KR | 24%(47) | 24% (8) |
| Errors with no KR | 32% (65) | 47% (16) |

Table 2: Average parsing time per lattice in seconds and sentence error rate for our specialized grammar compared to our generic grammar. Numbers in parentheses denote total time and error counts.

## 6 Coercion rules

Certain semantic type coercions are frequent in our domains. For example, in our medical adviser domain, the word *prescription* frequently appears in contexts that require a word for (a type of) medication, as in (1) *Which prescriptions do I need to take?* Intuitively, this is understood to mean (2) *Which medications specified by my prescriptions do I need to take?* We have adopted a practical approach to such coercions by applying a domain-specific operator to the mismatched argument to produce an entity of the coerced type. While this is a restricted approach compared to, *e.g.*, [14], [10], we adopt it as a transparent method of handling our domain-specific coercions in the most efficient way for our system.

The first problem is for the parser to recognize (1) as a valid utterance in spite of the semantic type mismatch, since *prescription* is not semantically typed as a consumable substance, which is required for an argument of *take* in its *consume* sense. An approach frequently taken by robust parsers (*e.g.*, [15]) is to relax the constraints in case of a type mismatch. However, if we need to construct a semantic representation for this utterance that is suitable for use by the back-end reasoners, the problem is deeper than just finding a parse tree. If the literal meaning of *prescriptions* is used during the interpretation process, the query asking for prescription objects consumed by the user (Figure 6) would be sent to the medication knowledge base, eliciting an empty result, since *prescriptions* are not recognized in the knowledge base as consumable objects.[3]

[3]Arguably, reasoning about prescriptions as consumables could be implemented in the knowledge base. However, in our system the context information needed to resolve some instances of coercion is localized in the intention recognition module, and using operators handled by the intention recognition is a way to take it into account. Such implementa-

(a) ASK ?y
(SET-OF ?y ?x (PRESCRIPTION ?x))
(TAKEMED v123) (:ACTOR v123 +USER+) (:MEDICATION v123 ?x)

(b)
(SET-OF ?y ?x (PRESCRIBED-MEDICATION (PRESCRIPTION ?x)))

Figure 6: (a) Query for *Which prescriptions do I need to take* with no type coercion; (b) representation for *prescriptions* coerced to *medication*.

A correct interpretation needs a semantic representation for (1) that resembles the semantic representation for (2). For this, additional information must be interpolated - in particular, the relation that holds between prescriptions and medications. We accomplish this in our framework with a library of coercion rules. To handle the semantic type coercion of *prescription* to *medication*, we define the following coercion rule, which uses a prescribed-medication operator (known to the knowledge base) to declare that prescriptions can be coerced into medications, in Figure 7a.

(a) declare-coercion-operator prescribed-medication
:arguments prescription
:return medication

(b) (prescription
:semfeatures (Phys-obj (information +) (kr-type PRESCRIPTION))
:lf LF_Information-object*prescription
:coercion ((operator prescribed-medication)
(semfeatures (phys-obj (kr-type medicinal-substance)
(form substance))))))

Figure 7: (a) Operator to coerce prescriptions to medications. (b) Lexical entry for *prescription* with coercion feature generated from operator in (a)

During the lexicon specialization process, information from coercion rules is propagated into the lexical entries that have domain-specific mappings. When the lexicon is specialized for the medical domain, the lexical entry for *prescription* has a nonempty coercion feature (Figure 7b).

The coercion feature specifies the coercion operator that will be applied during the interpretation process, and a set semantic features derived from

tion differences between components with different reasoning capabilities provide another justification for the need to specialize parser output for different application back-ends.

the operator returns type medicinal-substance (with the help of a feature inference rule in Figure 5), allowing the coerced NP *prescriptions* to be accepted in parsing contexts that require substances. For every noun phrase with a nonempty coercion feature, the grammar produces an additional NP representation for that item with a feature that indicates that a coercion has occurred, and the original semantic features replaced by the features specified by the coercion. During the transform process, the Interpretation Manager applies the specified coercion rule, resulting in the (simplified) representation for *prescriptions* in Figure 6b. This coerced representation is then plugged into the query in Figure 6 in place of (PRESCRIPTION ?x). The new query is sent to the medication database and results in the correct answer: a set of medications.

Adding the information about possible coercion rules into the lexical entries gives us an edge in efficiency, because only the coercions relevant to our domain are attempted, and the interpretation manager can apply the correct rule immediately. At the same time, the declarative coercion rules can also be used in reference resolution. For example, consider the sequence of utterances: *I have a new prescription. When do I need to take it?* The obvious referent for *it* is *prescription*, but there is a type mismatch, because *take* in this context prefers an argument that is a medicinal substance. The PHORA system [4] can use the declarative coercion rules in the resolution process. When a type mismatch is encountered, it searches the library of coercion rules for an operator with suitable argument types to perform the coercion and attempts to resolve the pronoun.

# 7 Conclusion

Our method of parser customization allows us to maintain a domain-independent lexicon and grammar for improved domain coverage and portability, and at the same time provides a straightforward mechanism for constructing custom semantic representations that are optimally suited for specific domain reasoners. Our lexicon specialization process improves parsing speed and accuracy by using custom domain knowledge to boost domain-specific word senses, tighten selectional restrictions on arguments and reduce search space during parsing. We also use the domain specific information to handle semantic type coercions common in our domains in a computationally efficient manner.

## 8 Acknowledgments

This material is based upon work supported by the Office of Naval Research under grant number N00014-01-1-1015 and the Defense Advanced Research Projects Agency under grant number F30602-98-2-0133. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of ONR or DARPA.

## References

[1] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. An architecture for a generic dialogue shell. *NLENG: Natural Language Engineering, Cambridge University Press*, 6(3):1–16, 2000.

[2] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. Towards conversational human-computer interaction. *AI Magazine*, 22(4):27–38, 2001.

[3] James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. A robust system for natural spoken dialogue. In *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL'96)*, 1996.

[4] Donna K. Byron. *Resolving Pronominal Reference to Abstract Entities*. PhD thesis, University of Rochester, 2002.

[5] Donald Davidson. The logical form of action sentences. In Nicholas Rescher, editor, *The Logic of Decision and Action*, pages 81–95. University of Pittsburgh Press, Pittsburgh, 1967.

[6] Myroslava Dzikovska, James F. Allen, and Mary D. Swift. Finding the balance between generic and domain-specific knowledge: a parser customization strategy. In *Proceedings of LREC 2002 Workshop on Customizing Knowledge for NLP applications*, 2002.

[7] G.M. Ferguson, J.F. Allen, N.J. Blaylock, D.K. Byron, N.W. Chambers, M.O. Dzikovska, L. Galescu, X. Shen, R.S. Swier, and M.D. Swift. The medication advisor project: Preliminary report. Technical Report 766, Computer Science Dept., University of Rochester, May 2002.

[8] D. Goddeau, E. Brill, J. Glass, C. Pao, M. Phillips, J. Polifroni, S. Seneff, and V. Zue. Galaxy: A human-language interface to on-line travel information. In *Proc. ICSLP '94*, pages 707–710, Yokohama, Japan, September 1994.

[9] Christopher Johnson and Charles J Fillmore. The FrameNet tagset for frame-semantic and syntactic coding of predicate-argument structure. In *Proceedings ANLP-NAACL 2000*, Seattle, WA, 2000.

[10] Alex Lascarides and Ann Copestake. Pragmatics and word meaning. *Journal of Linguistics*, 34(2):387–414, 1998.

[11] Catherine Macleod, Ralph Grishman, and Adam Meyers. Creating a common syntactic dictionary of English. In *SNLR: International Workshop on Sharable Natural Language Resources*, August 1994.

[12] David D. McDonald. The interplay of syntactic and semantic node labels in partial parsing. In H. Bunt and M. Tomita, editors, *Recent Advances in Parsing Technology*, pages 295–323. Kluwer Academic Publishers, 1996.

[13] Stephanie A. Miller and Lenhart K. Schubert. Using specialists to accelerate general reasoning. In Tom M. Smith and Reid G. Mitchell, editors, *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 161–165, August 1988.

[14] James Pustejovsky. *The Generative Lexicon.* The MIT Press, Cambridge, Massachuesetts, 1995.

[15] Carolyn Rosé. A framework for robust semantic interpretation. In *Proceedings 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, 2000.

[16] Stephanie Seneff. TINA: A natural language system for spoken language applications. *Computational Linguistics*, 18(1):61–86, 1992.

[17] Takehito Utsuro and Yuji Matsumoto. Learning probabilistic subcategorization preference by identifying case dependencies and optimal noun class generalization level. In *Proceedings of 5th ANLP Conference*, 1997.

[18] Piek Vossen. EuroWordNet: a multilingual database for information retrieval. In *Proceedings of the Delos workshop on Cross-language Information Retrieval*, March 1997.