Finding the balance between generic and domain-specific knowledge: a parser customization strategy

Myroslava O. Dzikovska, James F. Allen, Mary D. Swift

Computer Science Department
University of Rochester
Rochester, NY, USA, 14627
{myros, james, swift}@cs.rochester.edu

Abstract

Adapting spoken dialogue systems across domains presents a challenge of finding the balance between wide-coverage parsers which can be easily ported but are slow and inaccurate, and domain-specific parsers which are fast and accurate but lack portability. We propose a method for customizing a wide-coverage, domain-independent parser to specific domains. We maintain a domain-independent ontology and define a set of mappings from it into a domain-specific knowledge representation. With this method, we customize the semantic representations output by the parser for reasoning, and we specialize the lexicon for the domain, resulting in substantial improvement in parsing speed and accuracy.

1. Introduction

Developers of spoken dialogue systems for multiple domains are faced with the challenge of finding the optimal balance between domain-independent and domain-specific parsers. There are wide-coverage parsers (*e.g.*, XTag (Doran et al., 1994), LINGO (Copestake and Flickinger, 2000)) that are domain-independent and therefore easy to port to new domains, but they are often not efficient or accurate enough. The typical approach is to hand-craft parsers specifically for each domain (see for example (Goddeau et al., 1994)), but the performance gains in accuracy and efficiency are offset by their lack of portability, requiring additional effort to adapt them to new domains. We propose an alternative approach to address this challenge with a method for customizing a wide-coverage, domain-independent parser developed for spoken dialogue applications to specific domains. We maintain two ontologies: domain-independent for the parser, and domain-specific for the knowledge representation, and define a set of mappings between domain-specific knowledge sources and the semantic representations output by the parser. This method improves upon the generic parser output by specifically tailoring the semantic representations output by the parser for use by the reasoning components in the system. We also use the mappings to specialize the lexicon to the domain, resulting in substantial improvement in parsing speed and accuracy.

The customization method described here was developed in the process of adapting the TRIPS dialogue system (Allen et al., 2001) to several different domains, such as a transportation routing system (Allen et al., 1996) and a medication scheduling adviser. We assume a generic dialogue system architecture (Allen et al., 2000) 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.

Adapting the spoken dialogue system across domains results in tension between the representation of generic vs. specific information in the ontology. To facilitate development when porting the parser to new domains, we want to retain the syntactic and semantic information that is consistent across domains. However, each domain comes with its own semantic information relevant to the application. For example, the representation of physical objects for the transportation domain requires specifying whether an object is suitable cargo for transport, such as different types of food or supplies. In this respect, the distinctions between, say, oranges and potatoes are irrelevant, since they are equally good as cargo. These distinctions become highly relevant in the medical domain, where food-medicine interactions are important. Ideally, we want to customize the ontology to the domain for the most efficient reasoning. This becomes ever more important when using specialized reasoners with pre-defined input representations, for example, a database query system that must have specific template slots filled. Thus our goal is to preserve the language information that is similar across domains, while addressing specialization issues unique to each domain as much as possible, and keeping the development time for custom domain adaptation to a minimum.

The AUTOSEM system (Rosé, 2000) uses a syntactic lexicon COMLEX (Macleod et al., 1994) as a source of reusable syntactic information, and manually links subcategorization frames in the lexicon 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 and used by the reasoners. Rosé shows that her approach speeds up the development process for developing tutoring systems in multiple domains.

Our approach introduces an intermediate layer of abstraction, a generic ontology for the parser (the **LF Ontology**) that is linked to the lexicon and preserved across domains. 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 appli-

cation. 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. At the same time, the domain-independent ontology preserves semantic information consistent across domains that can be used by the Interpretation Manager for reasoning or reference resolution.

This separation between domain-specific and domain-independent ontologies allows us to write mappings in semantic terms without addressing the details of the grammar and subcategorization frames, using a higher level of abstraction. The developers writing the mappings do not need to understand syntactic details such as those in COMLEX subcategorization frames, and can instead use descriptive labels assigned to semantic arguments (*e.g.*, AGENT, THEME, etc.). They can also take advantage of the hierarchical structure in the domain-independent ontology and write mappings that cover large classes of words. Finally, 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.

2. The Generic Lexicon

The LF ontology is close in structure to linguistic form, so it can be easily mapped to natural language and used in multiple domains. It classifies entities (i.e., objects, events or properties) primarily in terms of argument structure. Every LF type declares a set of linguistically motivated thematic arguments, a structure inspired by FRAMENET (Johnson and Fillmore, 2000), but which covers a number of areas where FRAMENET is incomplete, such as planning. We use the LF ontology in conjunction with a generic grammar covering a wide range of syntactic structures and requiring minimal changes between domains. For example, adapting the parser from the transportation to the medical domain required adding LF types for medical terms (our generic hierarchy was incomplete in this area) and corresponding vocabulary entries, but we did not need to change the grammar or existing lexical entries, and we continue to use the same lexicon in both domains.

The LF types in the LF ontology are organized in a single-inheritance hierarchy. Obviously, some sort of multiple inheritance is required, because, for example, a person is a living being, but also a solid physical object (as opposed to a formless substance such as water). 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 (Vossen, 1997) 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 (Miller and Schubert, 1988). More importantly, using feature vectors allows us to easily change semantic information associated with a lexical entry, a property utilized during the customization process described below.

Word senses are treated as leaves of the semantic hierarchy. For every word sense in the lexicon, we specify the following information:

- Syntactic features such as agreement, morphology, etc.;
- LF type;
- The subcategorization frame and syntax-semantics mappings.

To illustrate, consider the verb *load* in the sense *to fill the container*. The LF type definition for *LF_LOAD* is shown in Figure 1. It specifies generic type restrictions on the arguments which are then propagated in the lexical entries. Intuitively, it defines a loading event in which an intentional being (AGENT) loads a movable object (THEME) into another physical object that can serve as a container (TO-LOC). The lexicon entry for *load* is linked to LF_Load and contains two possible mappings from the syntax to the LF: one in which the THEME is realized as direct object, corresponding to *load the oranges into the truck*, and another in which the THEME is realized as prepositional complement, corresponding to *load the truck with oranges*. The restrictions from the THEME argument are propagated into the lexicon, and the parser makes use of them as follows: only objects marked as *(mobility movable)* are accepted as a direct object or prepositional *with* complement of *load*.

```
(define-type LF_LOAD
 :sem (situation (aspect dynamic)
                 (cause agentive))
 :arguments
  (AGENT (phys-obj (intentional +)))
  (THEME (phys-obj (mobility movable)))
  (TO-LOC (phys-obj (container +)))
)
```

Figure 1: The LF type definition for LF_LOAD. In the lexicon, feature vectors from LF arguments are used to generate selectional restrictions based on mappings between subcategorization frames and LF arguments

The parser produces a flattened and unscoped logical form using reified events (Davidson, 1967). A simplified representation showing the semantic content of *Load the oranges into the truck* is shown in Figure 2[1]. For every entity, the full type is written as LF-parent*LF-form, where the LF-parent is the type defined in the LF ontology, and the LF-form is the canonical form associated with the word, for example, LF_VEHICLE*truck.

3. The KR customization

To produce domain-specific KR representations from the generic LF representations, 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

[1]For simplicity, we ignore speech act information in our representations.

(TYPE e LF_LOAD*load)
(AGENT e *YOU*) (THEME e v1) (TO-LOC e v2)
(TYPE v1 LF_FOOD*orange)
(TYPE v2 LF_VEHICLE*truck)

Figure 2: The LF representation of the sentence *load the oranges into the truck.*

```
(a)
(LF-to-frame-transform load-transform
       :pattern (LF_LOAD LOAD)
       :arguments (AGENT :ACTOR)
                  (THEME :CARGO)
                  (TO-LOC :VEHICLE))

(b) (define-class LOAD
     :isa ACTION
     :slots
       (:ACTOR AGENT)
       (:CARGO COMMODITY)
       (:VEHICLE (OR TRUCK HELICOPTER)))

(c) [LOAD
      :ACTOR [PERSON +YOU+]
      :CARGO [ORANGE V1]
      :VEHICLE [TRUCK V2]]
```

Figure 3: LF-to-frame-transform. (a) The transform for LF_LOAD type; (b) the definition of LOAD class that the transform maps into; (c) The KR frame that results from applying this transform to the load event representation in Figure 2.

slots, and a representation that has predicates with positional arguments. The KR ontology must have subtype support, and for the lexicon specialization process described in the next section, type restrictions on the arguments of frames/predicates, though it need not be so in the most general case.

We use two basic transform types to map generic representations produced by the parser into the KR representation: LF-to-frame-transforms, shown in Figure 3, and LF-to-predicate-transforms, shown in Figure 4.

The LF-to-frame transforms convert LF types into KR frame structures by specifying the KR frame that the LF type maps into, and how the arguments are transformed into the frame slots. These transforms can be simple and name the slot into which the value is placed, or more elaborate and specify the operator expression that is applied to the value. The LF-to-predicate transforms are used to convert the frame-like LF structures into predicates with positional arguments. They specify a KR predicate that an LF type maps into and the expression that is formed.

After the parser produces the logical form, the Interpretation Manager decides which transform to apply to a given LF with the following algorithm:

- Find all transforms that are consistent with the LF or its ancestors;

```
(a)
(LF-to-pred-transform load-transform
 :pattern (LF_LOAD
           (LOAD *AGENT *THEME *TO-LOC)
          ))

(b) (define-predicate LOAD
       :isa ACTION
       :arguments
       (1:AGENT 2:COMMODITY
        3:(OR TRUCK HELICOPTER)))

(c) (AND (LOAD +YOU+ V1 V2)
         (COMMODITY V1) (TRUCK V2))
```

Figure 4: LF-to-predicate-transform. (a) The transform for LF_LOAD type; (b) the definition of LOAD predicate that the transform maps into; (c) The KR formula that results from applying this transform to the load event representation in Figure 2.

- Select the most specific transform that applies, that is, the transform that uses only the roles realized in this particular LF representation, that has all obligatory mappings filled, and for which the types of the LF arguments are consistent with the type restrictions on the class arguments;
- If there are several candidates, choose the transform that uses the most specific LF, and, if there are several for the same LF, the transform that maps into the most specific KR class;
- Apply the transform to the LF type and all its arguments to produce the new representation.

For example, the parser produces the logical form in Figure 2 for *load the oranges into the truck*. The Interpretation Manager determines that the most specific transform consistent with the arguments is the `load-transform`. If the back-end reasoners use the frame representation, then we use an LF-to-frame transform and obtain the frame shown in Figure 3. Alternatively, a system using predicates with positional arguments as its representation uses an LF-to-predicate transform and obtains the (simplified) representation shown in Figure 4.

Our examples show the simplest versions of the transforms for exposition purposes. The actual implementation permits a variety of constructs that we cannot illustrate due to space limitations, including the application of operators to arguments, default transforms that apply to LF arguments if no mapping is specified in LF-to-frame transform, and the use of the lexical forms in transforms when the KR uses similar terms. For example, from the point of view of the language ontology, medication names have similar distributions across syntactic contexts, and therefore are represented as leaves under the LF_DRUG type, *e.g.*, LF_DRUG*prozac, LF_DRUG*aspirin. The KR ontology makes pragmatic distinctions between them (*e.g.*, prescription vs. over-the-counter medicines), but uses the names as

leaf types in the hierarchy. We can write a single template mapping for all LF_DRUG children that does the conversion based on the lexical form specified in the entry. This allows us to convert the generic representation produced by the parser to a representation that uses the concepts and formalism suited to the domain.

4. Specializing the lexicon

We use the mappings described above in a post-processing stage to customize the generic parser output for the reasoners. We also use the mappings 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. For each transform, we create a new sense definition identical to the old generic definition plus a new feature *KR-TYPE* in the 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 belong. We then propagate type information into the syntactic arguments, making tighter selectional restrictions in the lexicon. We also increase the preference values for the senses for which mappings were found.[2] This allows us to control the parser search space better and obtain greater parsing speed and accuracy.

Consider the following example. Given the definition of the verb *load* and LF_Load in Figure 1, and the definitions in Figure 3, the algorithm proceeds as follows:

- As part of generating the lexical entry for the verb *load*, the system fetches the definition of *LF_load* and the semantic vectors for it and its arguments;
- Next, the system determines the applicable LF-to-frame-transform, `load-transform`;
- Based on the transform, *KR-type load* is added to the feature vector of *load*;
- Since the mapping specifies that the LF argument THEME maps to KR slot *CARGO*, and the class definition contains the restriction that cargo should be of class *COMMODITY*, *KR-type commodity* is added to the feature vector of the THEME argument. Similar transforms are applied to the rest of the arguments.

As a result, in the lexicon we obtain a new definition of *load* with 2 entries corresponding to the same two usages described in section 2, but with stricter selectional restrictions. Now suitable objects or prepositional complements of *load* must be not only movable, but also identified as belonging to class COMMODITY in our domain. Since similar transforms were applied to nouns, oranges, people and other cargoes will have a *KR-type* value that is a subtype of COMMODITY inserted in their semantic feature vectors.

As a result of the specialization process, the number of distinct lexical entries will increase because there is no one-to-one correspondence between the LF and KR ontologies, so several transforms may apply to the same LF depending on the semantic arguments that are filled. A new entry is created for every possible transform, but during parsing the selectional restrictions propagated into the entries will effectively select the correct definitions. The Interpretation Manager thus knows the correct KR types assigned to all entities in the logical form output by the parser and the corresponding transforms, and only needs to apply them to convert the LF expression into the form used by the back-end reasoners.

	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

	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 for the test set.

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 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.[3]

At the same time, the amount of work involved in domain customization is relatively small. The generic 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

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

[3] For the purpose of this evaluation, we considered correct choices in which a different pronoun, article or tense form were substituted, *e.g.*, *can I tell my doctor* and *could I tell my doctor* were considered equivalent. However, equally grammatical substitutions of a different word sense, *e.g., drive the people* vs. *get the people* were counted as errors.

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, this represents an improvement over hand-crafting custom lexicons for every domain.

5. Conclusion

The customization method presented here allows the use of a lexicon and grammar with generic syntactic and semantic representations for improved domain coverage and portability, while facilitating the specialization of the lexicon and the representation produced by the parser to the needs of a particular domain. With this method we can produce specialized grammars for more efficient and accurate parsing, and allow the parser, in cooperation with Interpretation Manager, to produce semantic representations optimally suited for specific reasoners within the domain.

6. Acknowledgments

We would like to thank Carolyn Rosé for her feedback on this article. 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.

7. References

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

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

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

Ann Copestake and Dan Flickinger. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation*, Athens, Greece.

Donald Davidson. 1967. The logical form of action sentences. In Nicholas Rescher, editor, *The Logic of Decision and Action*, pages 81–95. University of Pittsburgh Press, Pittsburgh. Republished in Donald Davidson, *Essays on Actions and Events*, Oxford University Press, Oxford, 1980.

Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. 1994. XTAG system – a wide coverage grammar for English. In *Proceedings of the 15th. International Conference on Computational Linguistics (COLING 94)*, volume II, pages 922–928, Kyoto, Japan.

D. Goddeau, E. Brill, J. Glass, C. Pao, M. Phillips, J. Polifroni, S. Seneff, and V. Zue. 1994. Galaxy: A human-language interface to on-line travel information. In *Proc. ICSLP '94*, pages 707–710, Yokohama, Japan, September. URL http://www.sls.lcs.mit.edu/ps/SLSps/icslp94/galaxy.ps.

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

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

Stephanie A. Miller and Lenhart K. Schubert. 1988. Using specialists to accelerate general reasoning. In Tom M. Smith, Reid G.; Mitchell, editor, *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 161–165, St. Paul, MN, August. Morgan Kaufmann.

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

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