Subsumption-Based Matching: Bringing Semantics to Goals

Yolanda Gil
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
gil@isi.edu

Pedro A. González
Departamento de Informática y Automática
Universidad Complutense de Madrid
Ciudad Universitaria
28040 Madrid, Spain
pagoncal@dia.ucm.es

Abstract

Matching a posted goal against a library of rules is a task common in many AI systems. There are matching algorithms that can perform this process with reasonable efficiency. However, they are based on the syntactic features of goals instead of their semantic meaning. Representing the semantic meaning of goals can support additional features in matching algorithms and further reasoning about goals. This paper presents a matching algorithm that uses a semantic goal representation based on description logic. Each goal is translated into a description, and matching relies on the reasoning performed by a classifier to determine which rules unify with the posted goal. An extension of the matcher relaxes the posted goal to retrieve rules that *almost match* the goal based on the subsumption hierarchies of the domain ontology and the goals. The matching algorithm has been implemented using LOOM as the underlying description logic, and is used routinely as a component of the EXPECT problem-solving architecture. We show how it classifies and retrieves the methods in EXPECT's domains, and how the relaxed matching mode can be used to support knowledge acquisition.

1 Introduction

Matching a posted goal against a library of rules is a task common in many AI systems, including problem solvers, planners, and production systems. Two predicates match if their names are the same and if each argument matches in turn, where two constants match only if they are equal and a variable matches another variable or a constant. This approach to matching is based on syntactic features and treats the predicates and their arguments solely as tokens. These are impoverished representations compared to the knowledge representation systems that are currently available, such as description logics. Another shortcoming of these matching algorithms is their "all-or-nothing" nature, i.e., the complete expression of the posted goal must be matched *exactly* when they return a result. When an exact match cannot be found, they do not return anything.

This abstract presents a matching algorithm that uses a semantic goal representation based on description logic. Each goal is translated into a description, and matching relies on the reasoning performed by a classifier to determine which rules unify with the posted goal. Our work stems from the EXPECT project [Swartout and Gil, 1995; Gil, 1994; Gil and Melz, 1996] (and its predecessor system EES [Swartout *et al.*, 1991]), an architecture for developing knowledge-based systems that is tightly coupled with LOOM [MacGregor, 1988; 1991], a description logic system. EXPECT represents domain objects and classes in LOOM, as well as the goals of the methods to manipulate those objects. We have also extended the EXPECT matcher to work in a relaxed mode and retrieve rules that *almost match* a posted goal based on the subsumption hierarchies of the domain ontology and the goals. This relaxed matching mode can be used to support knowledge acquisition.

The EXPECT matching algorithm is used routinely as a component of the EXPECT problem-solving architecture, which we have used to implement transportation planning and air campaign planning decision aids. The EXPECT matcher classifies and retrieves the methods in these and other domains.

2 Bringing Semantics to Goals

In EXPECT, goals are expressed as verb clauses with an action name and several roles (as in a case grammar). The arguments of the goal are typed. The simplest type is an instance of a concept defined in the domain model. For example, the goal of transporting a package to a location can be expressed as a verb with two roles, i.e., the verb `transport` with a direct object role `obj` filled by an instance of a package, and a second role `to` filled by an instance of a location. We express this goal in EXPECT as `(transport (obj (inst-of package)) (to (inst-of location)))`. Previous versions of EXPECT represented it with the LOOM concept `(defconcept transport-package-to-location :is (:and transport (:the obj package) (:the to location)))`, where restrictions on the types of the arguments are represented with restrictions on fillers of the roles, resembling other approaches such as [Yen *et al.*, 1991].

However, this kind of definition is not enough to represent EXPECT goal arguments. Besides instances, the types of goal arguments in EXPECT include concepts, extensional sets, and intensional sets. Using concepts as parameter types is useful to make goal expressions more explicit. For example, the goal of computing the factorial of a number can be expressed as `(compute (obj (spec-of factorial)) (of (inst-of number)))`[1] with goal instances such as `(compute (obj (spec-of factorial)) (of 3))`. Notice that the goal could also be stated as `(compute-factorial-of 3)`, which computationally would yield the same result but its representation is not very explicit. Goal parameters can also be extensional or intensional sets. Sets are used to express goals such as transporting a set of objects to a location, as in `(transport (obj (set-of (inst-of package))) (to (inst-of location)))` and in `(transport (obj (p1 p2 p3)) (to (inst-of location)))`.

Goals can also contain descriptions of objects. Any legal LOOM expression for class definitions can be used to specify the type of an argument. For example, the goal to transport a package whose contents include some fragile object to a location can be expressed as `(transport (obj (inst-of (and package (some contents fragile-object)))) (to (inst-of location)))`, where `contents` is a role of the concept `package`.

EXPECT translates goal expressions to LOOM definitions, following an algorithm described in [Gil and González, 1996]. For example, the EXPECT goal `(compute (obj (spec-of factorial)) (of (5 7)))` is translated into:

```
(defconcept compute-factorial-of-numbers
    :is (:and compute
              (:the obj (:and concept-description factorial))
              (:the of (:and number extensional-instance-set
                             (:filled-by instance-name 5)
                             (:filled-by instance-name 7)))))
```

Notice that this translation is done automatically while in other approaches, such as COMET [Mark *et al.*, 1992] and LaSSIE [Devanbu *et al.*, 1991], it is done manually.

3 Subsumption-Based Matching

Given a library of methods, we express the goals that they can achieve as a LOOM concept as we explained in the previous section. When a goal is posted, the matcher expresses it as a LOOM expression, and uses the classifier to determine which methods have goals that subsume it. In [Gil and González, 1996], we summarize the algorithm for matching a posted goal against a library of goals achieved by methods available to the system. The algorithm returns a list of methods that match the goal and the bindings for the variables of each method, ordered according to the specificity of their corresponding descriptions. EXPECT's problem solver tries the most specific one first. For example, suppose that we have the following method (expressed in EXPECT's grammar) to double a number by multiplying the number by 2:

```
(defmethod double-number
 :goal (double (obj (?n is (inst-of number))))
 :result-type (inst-of number))
 :method-body (multiply (obj ?n) (by 2)))
```

When a goal such as `(double (obj 100))` is posted, the matcher (1) retrieves `double-number` and (2) specifies that the binding of `?n` is `100`. EXPECT then expands the method body by substituting the binding as `(multiply (obj 100) (by 2))` and posting it as a goal.

The concept definitions for goals use a subset of LOOM's representation language. Although reasoning with LOOM's full representation language is not complete, EXPECT's matcher uses a subset that is complete.

4 Relaxed Goal Matching

Our relaxed goal matcher is an extension to our approach that finds methods that almost match the posted goal. Since a goal expression can be relaxed in many ways (e.g., a different action name, different parameter names, alternative parameter types,...), any method available in the library would almost match the posted goal in the extreme. The relaxed matcher works in an interactive mode where the user specifies which parts of the posted goal expression can be relaxed. An alternative would be to use a similarity metric that exploits proximity within the subsumption hierarchy (extended with heuristics as in MRL [Koehler, 1994]). We use the interactive mode because we built the relaxed matcher to support users in extending a knowledge base with a knowledge acquisition tool [Gil, 1994]. To define a new method, a user can try to find an existing one that the user considers similar to the new method that he or she wants to define, and use the commands provided by EXPECT's knowledge acquisition interface to modify the retrieved method.

In order to support the search for similar methods the relaxed matcher defines some additional concepts called *action patterns*. Action patterns are LOOM concepts that represent goals with the same action and parameter names and do not specify the types of the parameters. For example, if the method library contains a method to achieve the goal `(find (obj (set-of (inst-of seaport))) (of (inst of location)))`, the matcher will define the pattern `(find (obj) (of))` and the pattern `(find)`. These patterns are turned into concepts that effectively impose additional structure on the goal hierarchy which will be used by the relaxed matcher to guide the search of related methods. The patterns support navigation from a pattern to other patterns that are close in the subsumption hierarchy.

The relaxed matcher works essentially as follows. Given a posted goal that cannot be matched, the user can expand its pattern to obtain the patterns accessible from it. When the user accepts one of these accessible patterns, the process iterates with the patterns accessible from them. At any point, if a pattern accepted by

[1]spec-of is short for specialization-of, and inst-of is short for instance-of.

the user has a method associated with it, the method is presented to the user. The user can also ask the relaxed matcher to compare the posted goal with a retrieved goal. The matcher compares two goals by showing how the action names, parameter names and parameter types relate to one another. To show the relationship between two concepts the relaxed matcher first finds their most specific subsumer c. It then presents the *specialization chain* of each concept with respect to c, which is the ordered set of concepts that must be traversed through the is-a links to reach c from the concept. The user can also browse the goal hierarchy through a graphical interface.

5 Conclusion

We have presented a system that represents goals explicitly in a description logic and uses these representations to match goals to problem-solving methods. The main advantages of our approach to subsumption-based matching are:

- Goals can be matched with methods to achieve them based on their semantic meaning, instead of their syntactic structure. This is particularly useful when there are several alternatives are available to achieve a goal, because it provides an understanding how each alternative relates exactly to the goal.
- When a goal cannot be matched with any of the methods available, it is possible to try to re-express the goal by reformulating it into a semantically equivalent set of subgoals that can be matched [Swartout and Gil, 1995]. In other systems, this kind of subgoal expansion has to be explicitly stated.
- When no method matches a posted goal (or its equivalent goal expressions,) it is possible to try an approximate match based on the meaning of the goal expression to retrieve methods that "almost match" the posted goal.
- Goals can be expressed in a more flexible manner. The arguments of goals can be definitions of classes and objects. In other systems only the name of a type can be given. Also, the arguments can be given in any order since they can be identified by their case role. In other representations the order of the arguments is fixed.

The explicit representation of goals can support additional types of reasoning besides matching. In EXPECT, we use the goal representations to do static analysis of problem-solving knowledge [Swartout and Gil, 1995; Gil, 1994], natural language generation [Swartout *et al.*, 1991], and knowledge acquisition [Gil, 1994; Gil and Melz, 1996].

Acknowledgements

We are indebted to current and past members of the EXPECT project for many useful discussions and for the design and implementation of previous versions of the EXPECT matcher that inspired this work, in particular to Bill Swartout, Ramesh Patil, Cécile Paris, Vibhu Mittal, Bing Leng, and Marcelo Tallis. We would also like to thank the members of the LOOM project for their continuing support. We also thank Kevin Knight, Craig Knoblock, Bob McGregor, Eric Melz, Tom Russ, and Milind Tambe for their suggestions to improve the clarity of this paper. We gratefully acknowledge the support of the Defense Advanced Research Projects Agency under contract number DABT63-91-C-0025. Pedro A. González also received support from the Spanish Committee of Science and Technology (CICYT) under contract number TIC92-0058.

References

[Devanbu *et al.*, 1991] P. Devanbu, R. J. Brachman, P. G. Selfridge, and B. W. Ballard. LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 34:35-49, 1991.

[Devanbu and Litman, 1991] P. T. Devanbu and D. J. Litman. Plan-based terminological reasoning. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, 1991.

[Gil, 1994] Y. Gil. Knowledge refinement in a reflective architecture. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, 1994.

[Gil and Melz, 1996] Y. Gil and E. Melz. Explicit representations of problem-solving strategies to support knowledge acquisition. To appear in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, August 1996.

[Gil and González, 1996] Y. Gil and P. A. González. Subsumption-Based Matching: Bringing Semantics to Goals. *Unpublished manuscript.*

[Koehler, 1994] J. Koehler. An application of terminological logics to case-based reasoning. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany, May 1994.

[MacGregor, 1988] R. MacGregor. A deductive pattern matcher. In *Proceedings of the 1988 National Conference on Artificial Intelligence*, St Paul, MN, August 1988.

[MacGregor, 1991] R. MacGregor. The evolving technology of classification-based knowledge representation systems. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* Morgan Kaufmann, San Mateo, CA, 1991.

[Mark *et al.*, 1992] W. Mark, S. Tyler, J. McGuire, and J. Schlossberg. Commitment-based software development. *IEEE Transactions on Software Engineering* 18(10):870–885, 1992.

[Swartout *et al.*, 1991] W. R. Swartout, C. L. Paris, and J. D. Moore. Design for explainable expert systems. *IEEE Expert* 6(3):58–64, 1991.

[Swartout and Gil, 1995] W. R. Swartout and Y. Gil. EXPECT: Explicit representations for flexible acquisition. *In Proceedings of the Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, 1995.

[Yen *et al.*, 1991] J. Yen, R. Neches, and R. MacGregor. CLASP: Integrating term subsumption systems and production systems. *IEEE Transactions on Knowledge and Data Engineering*, 3(1):25-32, 1991.