CS 376 2013

Assignment 3: Creating a Search/Planning Domain

Design Notes - Version 1.0

Design Notes for Modeling a Domain

When you want to define a new domain you need to create the following directories and files. Your top-level directory contains all the planning/search code and the "domains.d" directory. Each domain has a directory in "domains.d". For example, if you want to set up the 8 puzzle domain, you would create a domain directory called something like "eightPuzzle.d" or "eightPuzzle". In the domain directory, you need the following files:

1. "ontology.pl", which describes the domain ontology according to the conventions specified in the lecture notes and the rules listed in the next section.

2. "ops.pl", which describes the operator schemas which generate the edges in the search tree.

In the domain directory, you need the following directory:

1. "problems.d", which is where you will put your problem files.

Rules for Encoding "ontology.pl" Files

1. All static predicates, derived predicates, and metaLevel predicates have to be explicitly listed.

2. Fluent predicates, primitive predicates, and objectLevel predicates are defined as being not static, not derived, and not metaLevel predicates respectively. If a predicate is not listed as static then it is assumed to be fluent, if it is not listed as derived then it is assumed to be primitive, if it is not listed as metaLevel then it is assumed to be objectLevel.

3. Derived predicates can only be defined in the same language as goals.

4. Meta-level predicates are implemented in Prolog and cannot reference states or goals.

5. All predicates, appearing as the head of any clause in the ontology file, need to be declared as dynamic.

Rules for Writing Goal Language Expressions

Both goals and operator preconditions are written in what we call the "goal language". These "goal" expressions can include any of the four types of predicates (static, fluent, derived, or meta-level). You should be aware that because these expressions are evaluated in a Prolog-like manner the order that the terms appear in the expression makes a difference both in how expensive they are to evaluate and even whether they will be evaluated correctly.

Negative literals containing variables in a goal expression must appear after all those variables have been bound by positive literals. For example, assuming that neither A nor B is bound, the goal expression "*not(on(A,B)), clear(A), on(B,table)"* will always fail as long as there is some block on another one. However, the expression "*clear(A), on(B, table), not(on(A,B))"* will return A and B bound to blocks that satisfy that expression (assuming there is such a pair of blocks).

Positive literals can be arranged in any order and they will be correctly evaluated. However, their order in the expression can affect the cost of their evaluation. To reduce the evaluation cost they should be arranged in descending order of restrictiveness. In other words, the positive literal that has the fewest sets of bindings that will make it true should appear before those that have more sets of bindings.

Note on Producing State Descriptions

A state description in Prolog is a list of positive primitive literals. This list must be in **ordset** order. **"ordset"** is a prolog library and stands for ordered set. The planner expects all state descriptions to be ordsets. This means the initial state description must be made into an ordset list using *list_to_ord_set/2* from the SWI ordset library.