SOW-BKI329 Representation and Interaction (2017-2018)
Practical Assignment 1

J. Kwisthout and P.C. Groot

1 Assignment 1: Consistency-based diagnosis

Model-based reasoning is one of the central topics of knowledge representation and reasoning in artificial intelligence. The second part of this assignment aims at testing your understanding of Prolog and model-based reasoning by implementing the hitting-set algorithm that you have seen during the lecture.

1.1 Working with trees in Prolog

In order to implement the hitting-set algorithm, we need to be able to represent trees in Prolog. The first task helps to get used to working with such data structures in a logic programming environment.

Exercises In contrast to imperative programming languages, Prolog does not contain constructs for "real" data structures in the language. However, using *terms*, it is possible to represent any data structure by a compound term that constructs such a data structure. While you have seen some basic data structures before (e.g. lists), the following exercise illustrates this further using tree structures in Prolog. You do not need to submit these warm-up exercises and you may ask the instructors for help, if necessary.

(a) Consider for example a binary tree which we can build up using the following two terms (some examples are given below):

- a constant `leaf` which represents a leaf node;
- a function `node` with arity 2, which, given 2 nodes (its children), returns a tree.

Define a predicate `isBinaryTree(Term)`, which is true if and only if `Term` represents a tree. For example, `isBinaryTree(Term)` should be true if `Term = leaf`. Test this on compound terms such as:

- `leaf` (true)
- `node(leaf)` (false)
- `node(leaf,leaf)` (true)
- `node(leaf,node(leaf,leaf))` (true)

(b) Define a predicate `nnodes(Tree, N)`, which computes the number of nodes N of a given `Tree`, e.g.

```
?- nnodes(leaf,N).
N = 1.

?- nnodes(node(leaf,node(leaf,leaf)),N).
N = 5.
```

(c) Extend the representation of the tree so that each node (and leaf) is labelled with a number. Adapt your definition of `isBinaryTree` and `nnodes` to reflect this representation.

(d) Define a predicate `makeBinary(N, Tree)` which gets some number $N \geq 0$ and returns a tree where the root node is labelled by N. Furthermore, if a node is labelled by $K > 0$, then it has children that are labelled by $K - 1$. If a node is labelled by 0, then it does not have children.

(e) Now extend the representation of your tree so that each node can have an *arbitrary* number of children using lists. Also define a `nnodes` predicate for these kind of trees.

1.2 Implementation of the hitting-set algorithm

The *hitting-set algorithm* acts as the core of consistency-based diagnosis, and has been discussed during the course. In these tasks, you will implement this algorithm in Prolog.

Task 1: Generate conflict sets To get started, perform the following exercise.

- Download `tp.pl` and `diagnosis.pl` from blackboard.
- In `tp.pl`, scroll down to the bottom and inspect the definition of `tp/5`.
- In `diagnosis.pl`, inspect the definitions of the diagnostic problems in the file. Formulas are represented by Prolog terms where constants (and functions) are interpreted as predicates, with additional operators `~` (*not*), `,` (*and*), `;` (*or*), `=>` (*implies*), `<=>` (*iff*), and quantification $\{$`all`, `or`$\}$ `X:`f, where `X` is a (Prolog) variable and f is a term which contains `X`. Since `,` and `;` are also Prolog operators, it is often required to put brackets around these terms. For example, the formula $\forall_x(P(x) \vee Q(x))$ can be represented by the term `(all X:(p(X) ; q(X)))`.

Experiment with `tp/5` and determine at least three conflict sets for the diagnostic problems. For one of these provide a proof that it is indeed a conflict set, and give an informal explanation how `tp/5` computes those sets.

Task 2: Define your data structure Define a Prolog representation for hitting-set trees. Program a corresponding predicate `isHittingSetTree(Tree)` and convince yourself that this predicate is true if and only if `Tree` is a hitting-set tree. Guarantee that the edge labels on a path from the root to another node are distinct. In the following, the `tp/5` program will deliver these labels (conflict sets) for you.

Task 3: Implementation Use this representation to develop a Prolog program of the hitting-set algorithm. The input to the program is a diagnostic problem; the output is the set of all *minimal* hitting sets (i.e. diagnoses).

You may use standard Prolog functions such as `subset` and `append`. Other useful predicates may include `var(X)` to test whether X is a variable or `=..` to construct and deconstruct a term into symbols. See the SWI-Prolog manual for more details.
Evaluate your program using the given diagnostic problems and determine the minimal diagnoses. Explain why the results are correct. Also, reflect on your code (such as: What are the limitations? How can it be improved? What are the problems encountered? Do the (optional) optimizations help? What can you say about its complexity?)

(not required) Testing the performance Optionally, add some of the optimizations to prune the search space as described in [1] (see Google scholar or the library for the paper). Modify the example problems to become larger and more complex and investigate the time and space limits of your hitting-set implementation and Prolog (again up to a 10% bonus on your grade; note that the final grade cannot exceed 100).

References

[1] R. Reiter (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, **32**, 57–95.