*Knowledge Representation and Reasoning*
*BKI312 (2012-2013)*
*Assignments (Series 2)*

*P.Lucas, M. van Otterlo and A. Hommersom*

*december 2012*

## *Introduction to the assignments*

Welcome to the second series of assignments on knowledge representation and reasoning. In this series you will model a probabilistic burglary prediction system, use situation calculus to move some boxes around and you will use proabbilistic explanations for visual information. Thanks go to M. Crosby for his help on the second assignment.

*Additional files* The files that you need are available from the website:

`http://cs.ru.nl/~peterl/teaching/KeR/`

*Marks and Time* This assignment is marked out of a total of 100 percent, and it contributes a total of 20 percent towards your overall grade for the course knowledge representation and reasoning. The distribution over the sub-assignments are as follows:

Assignment 2-1: 35 percent
Assignment 2-2: 35 percent
Assignment 2-3: 30 percent

The estimated amount of time for the average student is roughly four full days of work per student. This varies since most students work in teams of two. The four days are effort asked for, but in the case much more is needed to complete only the required parts, contact the teachers.

*Submission* See the individual descriptions of the assignments for detailed descriptions of what to hand in. We expect all written answers in the form of a small report, including formalizations, pictures, and possibly small code fragments to illustrate your answers. Each assignment needs to be covered in a separate section (or chapter) of your report.

*In addition*: for each of the three assignments, answer the following questions: i) how much time did it take you to finish it?, ii) if you would have to change aspects of the assignment: what would they be and why?

Code files are to be submitted in a zip-file, properly named. All submissions should be done through Blackboard.

Good Luck!

The deadline for submission is

**21st of January 2013**

## *Assignment 2-1: Probabilistic representation and reasoning (and burglars)*

In this exercise, you will learn how to define a Bayesian network using AILog and use the network to answer some simple queries. Please read `http://artint.info/code/ailog/ailog_man_16.html`. It gives an overview of how to specify probabilistic knowledge using AILog. You may also consult `http://artint.info/code/ailog/ailog_code/ch14/leaving.ail` for an example of the 'leaving' network.

1. Read the following story:

   > Mr. Holmes receives a telephone call from his neighbor Dr. Watson stating that he hears a burglar alarm sound from the direction of Mr. Holmes' house. If there is a burglar present (which could happen once every ten years), the alarm is known to go off 95% of the time. Preparing to rush home, Mr. Holmes recalls that Dr. Watson is known to be a tasteless practical joker. There's a 40% chance that Watson is joking and the alarm is in fact off. However, if the alarm is on, Holmes expects Watson to call 80 percent of the time. He decides to first call his other neighbor, Mrs. Gibbons, who, despite occasional drinking problems, is far more reliable. She may not have heard the alarm in 99% of the cases and is thought to erroneously report an alarm when it is in fact off in only 4% of the cases.
   > Mr. Holmes remembers having read in the instruction manual of his alarm system that the device is sensitive to earthquakes and can be triggered by one accidentally in 1 every five cases. A burglary and an earthquake can be seen as independent causes. Other causes which will trigger the alarm do not exist. The incidence rate for earthquakes is about once every 10 years. He realizes that if an earthquake had occurred, it would definitely be on the news. So, he turns on his radio and waits around for a newscast. Of course, sometimes the newscast can be mistaken. This will happen only once per 5000 broadcasts.

2. Draw a Bayesian network (BN) that captures the independencies in the story.

3. Write down the corresponding conditional probability tables (CPTs) and fill them with the respective values.

4. Write an AILog program which defines the probabilistic knowledge associated with the network

5. Use AILog to answer the following queries:

   (a) the prior probability of a burglary

   (b) the probability of a burglary given that Watson called

(c) the probability of a burglary given that Gibbons also reports it

(d) the probability of a burglary given that the newscast reported an earthquake; explain why the probabilities change the way they do.

(e) write down the most probable explanation for the observed evidence

6. Suppose now getting more information about burglaries. Mr. Holmes house is in a residential area consisting of 10,000 houses, in which (only) the burglars Joe, William, Jack, and Averall are active. Each day a burglar decides whether he wants to work or not, and on average this happens only 5 days a week. However, some of them will only burgle if some specific colleages will join them, given by the following knowledge:

```
needs(joe, []).
needs(william, []).
needs(jack, [joe]).
needs(averall, [jack, william]).
```

i.e., Joe and William do not need anyone in particular, Jack only burgles if Joe is around, and Averall only burgles if both Jack and William are there. To avoid loneliness in the cold nights, the burglars will only work if they form a group of at least two. In addition, all active buglars on one night stay together (so they will never split-up in two groups). Finally, if they decide to burgle, then they will burgle 3 houses a night. Model this situation in AILog and derive the (new) probability that there is a burglary in Holmes' house.

7. Submit your assignment consisting of the BN and its CPTs, the AILog program and the AILog output for the queries.

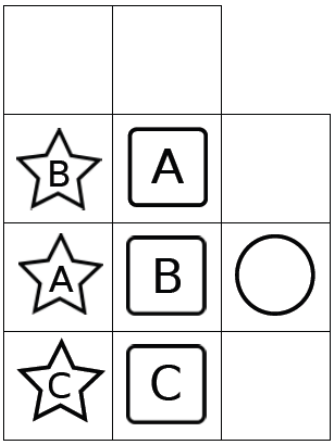


Figure 1: Example Sokoban problem. The grid is a 3x4 rectangle with the upper-right square missing. The agent is represented by the circle. There are three crates (represented by the boxes) A, B and C. The stars represent the goal locations for the respective crates.

## *Assignment 2-2: Planning and situation calculus (and boxes)*

This assignment deals with modeling and programming using the situation calculus.

### *Introduction*

This assignment is about the Situation Calculus and planning. It will evaluate your skills in formalising, implementing and testing a planning problem. Part 1 is a written exercise that requires you to formalise a planning problem using Situation Calculus. Part 2 requires you to implement the model and verify its correctness using a planner. In Part 3, you will extend the model and its implementation to deal with additional aspects of the environment. Note that where sometimes Golog is mentioned, all required tasks only deal with the situation calculus fragment. The nonrequired extensions may ask for specific Golog constructs.

### *Part I: Modeling a warehouse (Sokoban)*

The first part of the assignment requires you to develop a model for a planning problem. You will need to formalise the domain using the Situation Calculus. You will then use the axioms you defined to infer a plan for a simple instance of the problem.

*Problem description* You need to model a simple Sokoban warehouse domain. In this domain, an agent moves around a grid world pushing boxes into desired locations. The agent can move to any orthogonally connected empty grid square. Additionally, there are crates in the domain that an agent can push. An agent can push a crate only

in a straight line and only if the space behind it is empty. There may only be one crate (or agent) in any grid location at any given time. Figure 1 shows an example problem. It contains eleven grid squares, three crates and one agent. Each crate has a respective goal location that it must be pushed to. We will refer to locations numbered so that the bottom left location is `loc1-1` and the two top-right locations are `loc2-4` and `loc3-3`.

*Knowledge base* The first step in the creation of a model is the design of the knowledge base, i.e. the structures that will hold information about the environment that the planner can use when it chooses an action. The initial model should include information about the grid world, the crates and the location of the player. You should defne a set of predicates that can encode every state of the problem. Some of them will be atemporal predicates, which don't change as time progresses, and some will be fluent predicates whose values depend on the current situation. Briefly comment all predicates you introduce.

**Q1**: Specify how you would show which locations are connected. This should include the direction in which the locations are connected as this will help when defining the push action.

**Q2**: Explain how to keep track of the position at which the agent and crates are located at any particular moment and which locations are empty.

**Q3**: Using the symbols you just defined, write down the initial state of the problem depicted in Figure 1.

**Q4**: Using the symbols you just defined, describe how to specify the set of goal states of the problem depicted in Figure 1.

*Actions* The agent can move from the space it occupies to any adjacent empty space. Alternatively, it can push a crate in a straight line as long as the space behind the crate is empty. For example, in Figure 1 the only push action the agent can perform (in the initial state) is to push crate B left into `loc1-3` which would leave the crate in `loc1-3` and the agent in `loc2-3`. Formalise the following actions in terms of possibility axioms and effect axioms. You can omit universal quantifers.

**Q5**: The agent can move to an adjacent empty space.

**Q6**: The agent can push a crate that it is next to into an empty space behind the crate. Note that a crate can only be pushed in a straight line and only into the square directly behind it. This moves the agent into the space the crate originally occupied. You should not need to use any arithmetic operations here. You can make use of the direction information you encoded with your connected predicate.

Effect axioms alone are not sufficient: they describe how the new situation has been affected by the action executed, but they do not update information unrelated to the specific action, which may (or may not, if not updated) remain the same.

A successor-state axiom defines the state of a fluent, based on its state in the previous situation and the new action executed. At a high level, it formalises the idea that a fluent will be true if the most recent action makes it true, or if it was already true in the previous situation, and the most recent action has not changed its state.

**Q7**: Write the successor-state axioms for the fluents in your model.

**Q8**: In the lectures we have seen two extensions of the situation calculus: i) procedures in Golog, ii) probabilistic effects and fluents. In the last two decades many extensions of situation calculus have appeared in the literature. On the slides there are lists of several other possible and desired extensions but there are many more. Find a single paper (e.g. in Google scholar, the library, etc.) that describes another extension of the situation calculus and describe a) what the extension is and which problem it solves, b) the technical-logical solution (brief, you do not have to put formal details here, but the general idea suffices), and c) for which (kinds of) applications it is useful. Of course, provide the right bibliographic reference for the paper you found.

### *Part II: Implementation*

The second part of the assignment is centred on the implementation of the model you developed in Part 1. Once we have translated the axioms into rules that a planner can understand, we can work on more complex instances of the problem. The conversion is fairly straightforward, mostly a translation of logical symbols into ASCII characters, as we shall see in this section.

*A planner and the Situation Calculus* You can use the planner in `planner.pl`. We also provide will find two examples implementing a simple blocks world, `sample-blocks.pl` and `sample-blocks-domain.pl`.

To show the differences and similarities between situation calculus and the language read by the planner, we compare two (simplified) versions of the blocks world example. What follows are the possibility and successor-state axioms for the move action within the blocks world. Following the general conventions, we have predicates starting with an uppercase letter and variables in lower-case, quantified.

$$\forall x, y, s. Clear(x, s) \wedge Clear(y, s) \rightarrow Poss(Move(x, y), s)$$

In Golog, the opposite is true; predicates begin with lower-case letters

and variables with capitals. Quantifiers are dropped. Logical connectives change: the implication symbol is now `:-`. A comma represents a conjunction, while disjunctions are marked by semi-colons. The end of a rule is marked by a dot. The following statement means 'if What is clear and Where is clear in state S, then it is possible to move from What to Where in state S':

```
poss(move(What, Where), S) :-
clear(What, S), clear(Where, S).
```

In logic, a successor-state axiom is guarded by the predicate that verifies if the action is possible.

$$Poss(a, s) \rightarrow On(x, y, Result(a, s)) \leftrightarrow Move(x, y) \vee (On(x, y, s) \wedge a \neq Move(y, z))$$

This is done automatically by the planner or Golog interpreter, and can be dropped. Moreover, we are interested in the planning task, so we keep only one direction of the iff in the formula above: the $\leftarrow$. The resulting Golog axiom is (where the word `results` is used instead of `do` in this version):

```
on(Block, Support, results(A, S)) :- A = move(Block, Support);
        on(Block, Support, S), not(A = move(Block, _)).
```

where the semicolon ; is a disjunction, and the underscore is an anonymous variable that unifies with anything.

*Task 1: Translate Axioms* After reading the sample files and the included documentation, make a copy of the `domain-template.pl` file and rename it `domain-task1.pl`. Translate the axioms of your model and save them in this file.

*Simple Experiments* The goal of the following three exercises is to learn the language accepted by the planner, and for you to test the correctness of the model. Each task has at least one solution, and all plans do not exceed 15 actions in length; if the planner fails to find a plan, there might be something incorrect in the model. For each task, make a new copy of the file `instance-template.pl`, and rename it `instance-task#.pl`, where # is the number of the task. Any comment or description can go inside the `.pl` source file. Make sure to include both `instance-task#.pl` and `domain-task#.pl` for each task.

*Task 2: The Planning Problem in Figure 1* Implement and test the problem shown in Figure 1.

*Task 3: Crates go to Any Goal Location* Rewrite the problem so that the crates are allowed to end in any of the goal locations in Figure 1. Implement and test on the same problem.

*Task 4: Inverse Problem* Find an initial state and goal specification for which the agent visits every location in the grid world in the resulting plan. The agent does not need to revisit its starting location but must visit the goal locations of crates and the crates' initial locations. You may change the number of crates, their locations and their goal locations as well as the agent's starting location (but not the size or shape of the grid world). The goal specification should only include the goal locations of crates (as before). Test whether the resulting plan satisfies the requirements.

### *Part III: Extending the domain*

In this section you will extend the original problem to include new actions, action effects and goals. Only the Golog implementation of the axioms is required, but make sure the code is properly commented when defining new predicates. For each task, make a new copy of the file `domain-template.pl`, and rename it `domain-task#.pl`. Any comment or description should go inside the `.pl` source file. Make sure to include both `instance-task#.pl` and `domain-task#.pl` for each task.

Each task is a separate extension to the basic problem (Part 2: Tasks 1 and 2) so that, for example, in Tasks 6 you should not include the rest action of Task 5a.

**Choose at least one of the following extensions task 5a or 5b:**

*Task 5a: Hard Work* In this version of the problem the agent cannot push a crate twice in a row, but must move or rest at least once in between pushes. Add a rest action to the domain definition and test it against the original problem setup. (The rest action is needed to ensure that a plan exists of length 15.) You may need to modify your original actions and introduce new predicate symbols to express whether an agent is tired/rested. You may want to use a `rest(X)` action (where `X` is the agent's location) as the planner does not deal well with actions with no parameters.

*Task 5b: Unlocking the Crates* In this version of the domain each crate starts locked to the ground. The agent cannot push the boxes until it has found and picked up the key for each crate. Add a pickup action that lets the agent pick up a key that is in its current location. You should not add any more actions; instead, update your definitions

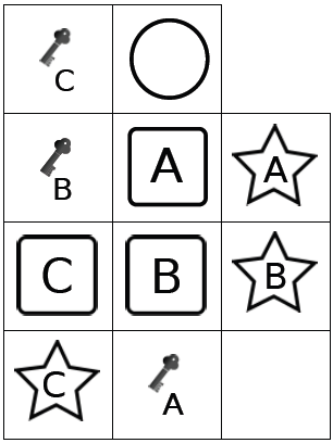


Figure 2: Problem for task 6. A key's label corresponds to the crate it unlocks.

for the push action so that they require the agent to have picked up the correct key. Once implemented, test on the problem shown in Figure 2.

*Task 6: Number of Actions* For this final exercise you should add the stipulation that the final plan has either an odd or an even number of steps. Since we are only interested in the parity, you do not need to introduce arithmetic operations to achieve this. Run the experiment for (at least) the problem shown in Figure 1 requiring an even and then an odd number of actions. (The agent may still move after pushing all the crates to their respective goal locations.)

*(not required) Various Extensions* There are several immediate ways to make things even more interesting. These are not required, but highly encouraged. Completing any of them will be beneficial for your final grade for the assignments, to various degrees (with a maximum of 10 points).

- Extend the model with new exciting effects or new actions. Some blocks may glide multiple grid positions unexpectedly, or sometimes we can jump over a block to get to another position more easily, or ....
- With some small effort, you can translate your code to AILog (see the slides for examples) where you can extend it with probabilistic and/or decision-theoretic aspects and let AILog compute plans and their likelihood to succeed.
- Extend your model with Golog-constructs. For example a solution can be programmed using `while` *there is an object still not on its place* `do` *something*. More elegantly, one could add procedures to,

for example, *make room* for something else: in order to push some block to its position, one may need to first clear an exit which might take up some actions first.

- Model a larger level of the original game (Google for Sokoban). Originally, the game involved several rooms and corridors. You would need to extend the model to incorporate maybe some additional elements, and surely one needs to be careful with planning since solutions for large levels might become very large. (Extending the model with some Golog-predicates to guide the search for a plan might help too to make planning more efficient).
- Implement a more efficient planner.
- ...(and many other possibilities)

## *Assignment 2-3: Visual representations and reasoning*

In this final task, you are going to experiment with logical knowledge representation of visual information and the accompanying reasoning styles to find out what is actually depicted. You have lots of freedom to choose your own domain, i.e. you own images.

In the lecture we have seen a simple example of aerial sketch map recognition. The aim of this assignment is to model (probabilistic) visual features, logical axioms about the visual domain, and to reasoning about specific pictures. Especially the rules defining how image objects come about from scene objects is important.

*Picking a domain* There are lots of different domains to choose from. In the lecture we have seen examples of houses (based on windows and doors), kitchen design, general architectural design, aerial sketch maps, shapes (e.g. a puppet) made out of basic shapes (such as squares and triangles) and so on. But, one can also think of cartoon-like pictures, maps, line drawings and so on. Note that – because you are using the power of logic – you can make models a bit more interesting with more complex definitions. For example, a stick figure consists of a `torso`, two `leg` objects on each side, and two `arm` objects on each side. Defining *spatial* predicates such as `leftof(X,Y)` and `above(X,Y)` may be useful for some domains.

You can choose any domain (but choose a fun or nice one, of course). The main **requirement** is that it should not be much simpler than the example used in the slides (the aerial sketch maps by Poole at the end of the vision-lecture), in terms of number of rules and components. You should be able to argue that your application is roughly (at least) as complex as that one. Another **requirement** is that there is considerable uncertainty in the rules.

*Modeling the domain in AILog* Your domain (as well as the pictures-represented-as-sets-observations) needs to be modeled in AILog. Look up the manual of AILog to see which other features you can use. An easy way to have multiple pictures in the database is to annotate picture observations with the picture number; for example `line(l1,picture1)`. That way, one can query about specific pictures by querying all observations for a specific picture. Another way could be to use `observe` in AILog.

*Inference on specific, hand-chosen instances* Inference on the pictures in your domain consists of supplying the observations belonging to a specific picture, and computing explanations for them. A **requirement** is that your input pictures contain enough ambiguities to

ensure that *multiple* explanations exist for some instances.

*What to do* So, summarizing, you need to choose a domain and create or gather some images. Then, you need to model your domain of pictures and implement it in AILog. Finally, you need to (ask AILog to) compute explanations for the pictures, i.e. interpretations of what is on the pictures. Show for some well-chosen pictures which explanations are possible, and what the most likely explanations are for the picture (just like the examples on the slides). Additionally, you can show some interesting queries, or funny interpretations, or unexpected results as well. Of course, you need to describe and specify your domain completely in your report. Hint: start small (just some small aspects of your domain, and test implementation and inference on these, and then make things larger and more complex).