#Practical Assignments (Series 1) Report BKI312 (2014-2015)

Michel Meijerman and Guido Zuidhof (s0723630 & s4160703) **Kunstmatige Intelligentie**

November 2014

Assignment 1-1 Situation calculus and planning

Part 1: Modelling Sokoban

Task 1: Knowledge base

- **Q1:** By introducing a predicate for each of the directions. All connections would be specified as facts, such as `isCon(loc-1-1, loc-1-2)` or `isCon(loc-2-1, loc-1-1)`. Also straight connections are specified as predicates `isStraightH(loc-1-1, loc-1-3)`. Aside from that we define helper functions `isConnected(a, b)` which returns whether a is connected to b, through any direction, and `isStraight(a,b)`, which returns whether two tiles are the same row or column.
- **Q2:** We would introduce a predicate `at(X,Y,S)`, where X is the object (agent or a box), Y is the location, and S is the state. An example instantiation is `at(sokoban, loc-2-3, s0)`. Also a function `isFree(P,S)` which returns whether location P is empty in state S (not occupied by any object).
- **Q3:** By defining facts of the positions of the objects, such as `at(sokoban, loc-1-2, s0),at(a, loc-2-3, s0),at(b, loc-2-2).at(c, loc-2-1, s0).`
- **Q4:** Similar to the initial state (Q3), but this time as a concatenation of the required facts, for example `at(a, loc-1-2, g),at(b, loc-1-3, g),at(c, loc-1-1, g)`, where g is the goal state.

Task 2: Actions

- **Q5:**

```
An object can move if it is an agent, the tile it is moving to is free and
connected.
When an object is moved, the original position becomes free, the object is no
longer at the from position.

% Positive effect axiom of move
        a  = move(object, from, to)
    ->
        at(object, do(move(object, from, to),s))
.



% Negative effect axiom of move
        a = move(object, from, to)
    ->
        ¬at(object, from, do(a,s))
```

```
.

%Possibility axiom of move
        isAgent(object)
    /\
        at(object,from,s)
    /\
        isConnected(from,to)
    /\
        isFree(to, s)
 ->
    Poss(move(object, from, to),s).
```

- Q6:

```
There is a position agentpos, where the agent is. The agent pushes an object,
taking it's place, in order
for this to be valid the object has to be able to move to a position `to`, from
from in a straigt line.
So the `to` and `agentpos` have to be in a straight line for this to happen.

After pushing `object` from `from` to `to` with the agent on `agentpos`, the object
is no longer at from, and the agent is no longer at agentpos.

% Positive effect axiom of push
            a = push(agent, object, agentpos, from, to)
    ->
            at(object, to , do(a, s))
        /\
            at(agent, from , do(a, s))


% Negative effect axiom of push

            a = push(agent, object, agentpos, from, to)
    ->
            ¬at(object, from , do(a, s))
        /\
            ¬at(agent, agentpos , do(a, s))

%Possibility axiom of push
                isAgent(agent)
            /\
                at(object,from,s)
            /\
                isConnected(from,to)
            /\
                isConnected(agentpos,from)
            /\
                at(agent, agentpos, s)
            /\
                isStraigt(agentpos, to)
```

```
            /\
                isFree(to, s)
        ->
            Poss(push(agent,object, agentpos, from, to),s).
```

- **Q7:**

```
%Succesor-state axiom of at

at(object, loc, do(a,s)) ≡
        at(object, loc, s)
    /\
        (
                %Object moved away
                ¬□to{a = move(object, loc, to)}
            /\
                %Something pushed object away
                ¬□to{a = push(_, object, _, loc, _)}
            /\
                %Object pushed something, moving the object too
                ¬□from{a = push(object, _, Loc, _, _)}
        )
\/
    □prev{a = move(object, prev, loc)} %object moved to loc
\/
    □prev{a = push(_, object, _, prev, loc)} %object got pushed to loc
\/
    □to{a = push(object,_ , _, loc, to)} %object pushed something off loc
.
```

Part 2: Implementation

Task 3: Translate axioms

Translating the model of Sokoban proved to be quite challenging. Translating from 'logic' to prolog wasn't all that hard, but in the model we overlooked many things required for correct planning. Often the agent would make moves that were not valid. Note that we have corrected the previous part of this assignment so that it matches the implementation. See `domain-task3.pl` for the implementation of the domain.

Task 4: The Planning Problem in Figure 1

Implementing the problem in figure one was pretty straightforward. It was just a matter of listing the facts that make up the world and adding some helper functions.

Excerpts from this task specification:

Agent declaration

```
isAgent(agent).
```

World structure declaration

```
isCon(loc-1-1, loc-2-1).
isCon(loc-2-1, loc-3-1).
. . .
isCon(loc-3-2, loc-3-3).

isStraightH(loc-1-1, loc-1-3).
isStraightH(loc-2-1, loc-2-3).
. . .
isStraightH(loc-1-3, loc-3-3).
```

Helper functions

If A is connected to B, B is also connected to A

```
isConnected(A,B) :-
    isCon(A,B);
    isCon(B,A).
```

A is straight to B (same collumn or row)

```
isStraight(A,B) :-
    isStraightH(A,B); %Straight connections go both ways
    isStraightH(B,A);
    isConnected(A,B). %When direcly connected it's always straight
```

Position P is free if nothing is at it

```
isFree(P, S) :-
  not(at(_,P,S)).
```

Starting state declaration

```
at(agent, loc-2-4,s0).
at(a, loc-2-3,s0).
at(b, loc-2-2,s0).
at(c, loc-1-2,s0).
```

Goal state declaration

```
goal(S) :-
  at(a, loc-3-3,S),
  at(b, loc-3-2,S),
```

```
  at(c, loc-1-1,S).
```

Task 5: Crates go to Any Goal Location

This was an easy change.
We added 3 facts, declaring the a, b, and c as boxes.

```
box(a).
box(b).
box(c).
```

Then in the goal state we want `Q`, `R` and `T`, which should be boxes, to be at the goal locations.

```
  at(Q, loc-1-2, S), box(Q),
  at(R, loc-1-3,S), box(R),
  at(T, loc-1-1,S), box(T).
```

Task 6: Inverse Problem

A nice puzzle. In the end this problem satisfied the conditions

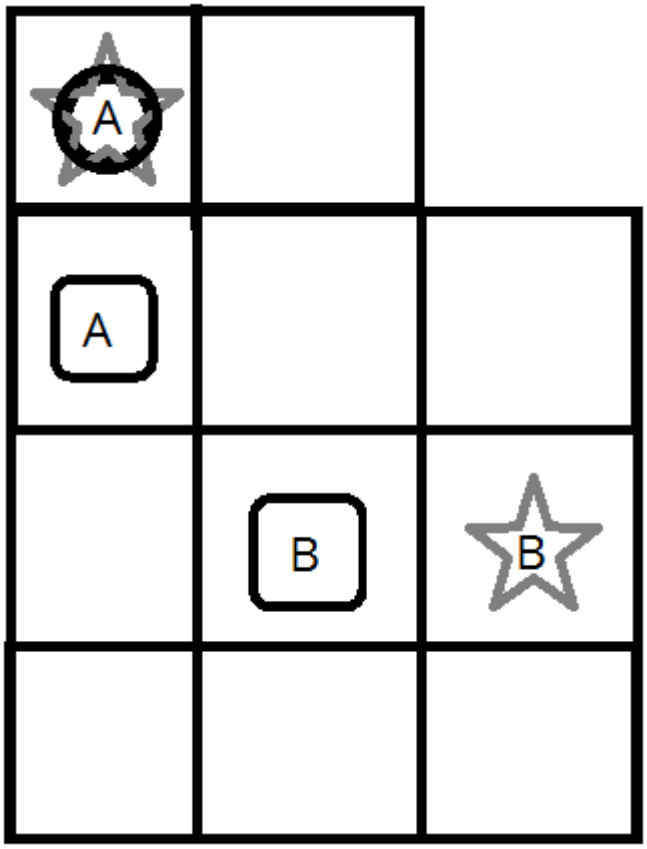
Starting state

```
at(agent, loc-1-4,s0).
at(a, loc-1-3,s0).
at(b, loc-2-2,s0).
```

Goal state

```
goal(S) :-
  at(a, loc-1-4, S),
  at(b, loc-3-2,S).
```

Artistic MS Paint impression



The agent has to move all the way around B to push it A. After that it only has to push B to have visited the last spot.

Part 3: Extending the domain

Task 7: Unlocking the Crates

Thanks to our sane code (without any dynamic programming optimizations required) this was easy to add.

Changes to the existing domain from previous tasks:

New fluent `hasKey` introduced

```
hasKey(Key, result(A,S)) :-
    %Already has the key
    hasKey(Key, S)
;
    %Key was picked up
    A = pickup(_, Key)
.
```

Possibility axiom of push updated

`hasKey(Object, S)` was added to the possibility axiom of push, so in order to push an object a key for it is required.

Possibility axiom of new pickup action added

In hindsight a possible optimization might be checking whether the key has not already been picked up.

```
poss(pickup(Agent, Key), S):-
  isAgent(Agent),
  keyAt(Key, Pos),
  at(Agent, Pos, S).
```

Changes to task description

Besides updating the positions of the agent and the boxes, the location of the keys had to be specified

```
keyAt(a, loc-2-1).
keyAt(b, loc-1-3).
keyAt(c, loc-1-4).
```

Part 4: General Questions

Task 10: Sitcalc expressivity

Using situation calculus had it's advantages. An advantage was the ease of planning. The black box that was `planner.pl` was usable after creating the domain and taskspecification. Extending the domain (with the keys) was very straightforward, it involved adding less than 10 lines of code and we got it right the first time.

A big disadvantage of using situation calculus is the pain that is debugging. Now this may be because we are using Prolog here (instead of your run-of-the-mill imperative or functional language), but in my experience of using STRIPS in Python it was a lot easier to find out where a mistake was made in the problem specification.

Using STRIPS one only has to define pre and post conditions for actions. Whereas in situation calculus one has to define every possible way a fluent can be true. Which is harder to get right in my experience.

Task 11: Related work

The paper we chose is a study by Al-Mouadhen et al.[1]. In this paper a framework is presented that integrates the planning of high-level robot actions with common-sense domain knowledge in a domestic environment. The high level actions are modelled in a novel way called the *Semantic Action Model (SAM)*. An algorithm then creates a planning domain from this model in PDDL format, which is fed to the planner. The authors note that any STRIPS-family planner could have been used instead of PDDL.

It is concluded that the use of a 'common-sense robot knowledge-base' makes the robot more flexible in completing its task.

Assignment 1-2 Consistency-based diagnosis

Exercises

```
isBinaryTree(X):-
    isLeaf(X);
    isBinaryNode(X).


isBinaryNode(X):-
    X=node((Y,Z),(Number) ),
    isBinaryTree(Y),
    isBinaryTree(Z).

isLeaf(leaf(Number)).

nnodesBinary(Tree, N):-
    isLeaf(Tree), N=1;
    Tree = node(X,Y),
    nnodes(X,M), nnodes(Y,L),N is 1+M+L.

makeBinary(N,Tree):-
    N=0, Tree = leaf(0);
    N>0, N1 is N-1, Tree = node( (makeBinary(N1,  T), makeBinary(N1, T)),N).

isTree(node(X,Y)):-
    isNode(X).

isNode(X):-
    X=[];
    X=[H|T], isTree(H), isNode(T).

nnodes(node(Tree,_),N):- nnodes(Tree,N1), N is N1 + 1.
nnodes([],0).
nnodes([H|T],N):-
nnodes(H,N1),nnodes(T,N2), N is N1 + N2.
```

Implementation of the hitting-set algorithm

Task 12: Generate conflict sets

The conflict sets we found are:

Problem1 `{a1}` and `{a2}`. Problem2 `{}` Problem3 `{a1,o1,a2}`, `{a2,o1}`, `{a1,01}` Fulladder `{a2,x2,x1}`, `{a2,rl,x1}`, `{a1,x1,x2}`

Working of `tp/5`

The theorem prover `tp/5` has as input `SD`, `COMP`, `OBS` and `HS`. SD is the system description and is a list

of logical formulas. `COMP` is the set of all components, `OBS` a set of all observations. `HS` is the set of all components that are assumed to be abnormal. The output is a conflict set `CS`.

`tp/5` first computes which components are assumed to be normal, by subtracting the components in `HS` from the set of all components. With this new set it builds a theory and tests it. It tests the truth of the negated formula and collects the components of the proof. This set minus the `HS` and duplicates is returned as `CS`.

Task 13: Define your data structure

All requirements are met by this data structure `isHittingTreeSet/1`.

```
isHittingSetTree(Tree):-
Tree = node([],_);
Tree = node([X],_), isHittingSetTree(X);
Tree = node([H|T],_), isHittingSetTree(H), isHittingSetTree(T).
```

Task 14: Implementation

```
diagnoses(SD,OBS,COMPS,D):- hittingTree(SD,OBS,COMPS,[],CS),paths(CS,D).

conflictSet(SD,OBS,COMPS,HS,CS):- tp(SD,OBS,COMPS,HS,CS). %get an conflictset CS
with help of the tp funtion.

hittingTree(SD,OBS,COMPS,HS,Node):-
    %if no conflictset is found, make an empty node (leaf).
    not(conflictSet(SD,OBS,COMPS,HS,CS)), Node = node([],[]);
    %make a note with label CS and find its children.
    conflictSet(SD,OBS,COMPS,HS,CS),
        CS = [H|T], hittingTree2([H|T], SD,OBS,COMPS,HS,Children), Node =
node(Children, CS).

%use recursion to find all children of a node.
hittingTree2([X|XS],SD,OBS,COMPS,HS,Children) :-
    hittingTree(SD,OBS,COMPS,[X|HS],Child),
    hittingTree2(XS,SD,OBS,COMPS,HS,T), Children = [Child|T].
hittingTree2([],SD,OBS,COMPS,HS,Children) :- Children = [].

%find the paths = diagnoses.
paths(node([],Path), Path).
paths(node([H],X), [X|P]) :- paths(H,P).
paths(node([H|T],X),[X|P]):- paths(H,P).
```

Querying This code can be queried by first loading the problem domain, and then calling (for problem 1)

```
problem1(SD,OBS,COMPS), diagnoses(SD,OBS,COMPS,D).
```

Found diagnoses

`D = {a1}, {a2}` for problem1 `D = {a1,a2}` for problem2 `D = {a1, o1, a2}, {a2, o1}` for problem3 `D = {a1, x1, a2, r1, x2}, {a2, x2, x1}, {x1, x2}` for the fulladder.

These are all correct diagnoses, they are however not all minimal diagnoses. An improvement to our program would be to filter the non minimal diagnoses (those which are a subset of other diagnoses). We decided to keep it simple, but pruning the hitting set tree would increase it's execution speed. But given that it already was pretty fast and HS trees in general make for a quick enumeration of hitting sets, this would add unnecessary complexity.

Reflection

Assignment 1-1

How long did it take you to finish it

Hard to say, as hardly all the time spent sitting behind a computer is the time working at it. Does time thinking about it in the shower count? Anyway, we spent some 3 'werkcolleges', and besides that another 8 hours on this assignment, of which 1 and a half hour on polishing the report and putting the code in the required structure to be handed in.

If you would have to change aspects of the assignment: what would they be and why?

- Not as rigid code hand-in requirements. We had one `sokoban#.pl` file per task, which was around 100 lines long. Splitting this into the required `instance-task#.pl` and `domain-task#.pl` didn't make much sense.
- More of a build-up in the task. Sokoban isn't the easiest problem, as it involves a `push` task which changes the position of multiple entities. Perhaps it could start with a task where the goal is simply pathfinding, and work from there.

Assignment 1-2

How long did it take you to finish it

We spent 2 'werkcolleges' on this task, and besides that a good part of the weekend and the whole tuesday so that would be around 18 hours? However it would have been a lot less if we didn't lose most of the code for the second part monday night.

If you would have to change aspects of the assignment: what would they be and why?

Also for this part we would have liked some more of a build-up in the task. A good start would be dividing task 14 in some subtasks so there is some more structure. Also it was not easy to find out how to test the code before it was all finished.

References

- [1] Ahmed Al-Moadhen, Renxi Qiu, Michael Packianather, Ze Ji, Rossi Setchi, *Integrating Robot Task Planner with Common-sense Knowledge Base to Improve the Efficiency of Planning*, Procedia

Computer Science, Volume 22, 2013, Pages 211-220, ISSN 1877-0509.