Situational Calculus, Planning and Concistency-based Diagnosis

van Hoorn, W. and Remmits Y.L.J.A.
Radboud University

I. Introduction

This is the first of the two practical assignments in the series of the course Knowledge Representation and Reasoning.

The assignment reflects on our current knowledge of the course and is our first experience with Prolog language. For the first part of the assignment we implemented a representation for the Shokoban domain, where an agent needs to find a path to succesfully push crates on their respective goal locations. The second part of the assignment consists of an implementation of the hitting-set algorithm.

Our implementations are tested on Windows and Linux systems and are present in the .zip file.

II. Assignment 1-1

I. Knowledge base

All locations of the Shokoban domain are represented as variables. The connections between the world's locations are defined. Locations, which have no connection to other locations, are not part of this world.

$$(x, y, d) \tag{1}$$

In predicate 1, location X is connected to location Y in direction D where the direction is one of the four compass directions.

The presence of objects in this world is defined using it's location in a particular state in time. Objects can be agents, crates or keys.

$$On(o, l, s) \tag{2}$$

Locations L in a certain state S, that do not have an object O on it, are clear. We define clear locations by the following predicate.

$$Clear(l, s) \tag{3}$$

The initial state of the Sokuban problem is presented in figure 1. The initial state and the goal state is formalised in the Appendix, section I.

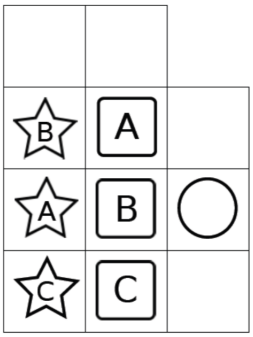

Figure 1: *The Sokoban problem. The grid is a 3x4 rectangle with the upper-right square missing. The agent is represented by the circle. There are three crates (represented by the boxes) A, B and C. The stars represent the goal locations for the respective crates. Source: practical1.pdf*

A goal is reached if all the crates are on their goal locations in a specific state in time. When this state is true is not important. It is important that the crates are on the destined location in the same state, which follows from statement 4.

$$\forall s\ On(blockA, loc1, s) \wedge On(blockB, loc2, s) \wedge On(blockC, loc3, s) \rightarrow Goal(s) \tag{4}$$

II. Actions

To solve the Sokoban we need a transition between state, therefore we define two possible actions with the following effect axioms for the action move (5), (6) and the action push (7), (8) and (9).

$$On(Agent, Loc2, Do(Move(Agent, Loc1, Loc2), S)) \quad (5)$$

$$Clear(Loc1, Do(Move(Agent, Loc1, Loc2), S)) \quad (6)$$

$$On(Agent, Loc2, Do(Push(Agent, Block, Loc1, Loc2, Loc3, Direction), S)) \quad (7)$$

$$On(Block, Loc3, Do(Push(Agent, Block, Loc1, Loc2, Loc3, Direction), S)) \quad (8)$$

$$Clear(Loc1, Do(Push(Agent, Block, Loc1, Loc2, Loc3, Direction), S)) \quad (9)$$

These actions should not be able to occur in any state or all directions. The world has the following constraints.

For an agent it is only possible to move if, within the same state, a connected location is clear of any crates or possible other agents (10).

Secondly, an agent is only able to push a crate (7 & 8) if the location of the crate and the agent are connected in a particular direction while there is also a clear location connected to the crate's location in the same direction (11).

$$\forall A, L1, L2, D, S\ (Agent(A) \wedge On(A, L1, S) \wedge Connected(L1, L2, D) \wedge Clear(L2, S) \rightarrow Poss(Move(A, L1, L2, D), S)) \quad (10)$$

$$\forall A, C, L1, L2, L3, D, S\ (Agent(A) \wedge Crate(C) \wedge On(A, L, S) \wedge Connected(L, L2, D) \wedge On(C, L2, S) \wedge Connected(L2, L3, D) \wedge Clear(L3, S) \rightarrow Poss(Push(A, C, L1, L2, L3, D), S)) \quad (11)$$

Together the effect axioms and possibility axioms do no suffice: they describe how the new situation has been affected by the action executed, but they do not update information unrelated to the specific action. To solve this, succesorstate axioms (12), (13) and (14) are used.
These successorstate axioms are derrived from the effect axioms together with the condition of the move being possible. In addition to this we defined the effect of unrelated actions on the current state.

$$Poss(a, s) \rightarrow (On(Agent, Loc, Do(a, s)) \leftrightarrow a = Move(Agent, Loc1, Loc) \vee a = Push(Agent, Crate, Loc1, Loc, Loc2, Direction) \vee (On(Agent, Loc, s) \wedge \neg(a = Move(Agent, Loc, Loc2)) \wedge \neg(a = Push(Agent, Crate, Loc, Loc1, Loc2, Direction)))) \quad (12)$$

$$Poss(a, s) \rightarrow (On(Crate, Loc, Do(a, s)) \leftrightarrow a = Push(Agent, Crate, Loc1, Loc2, Loc, Direction) \vee (On(Block, Loc, s) \wedge \neg(a = Push(Agent, Crate, Loc1, Loc, Loc2, Direction)))) \quad (13)$$

$$
\begin{aligned}
&Poss(a,s) \rightarrow \\
&(Clear(Loc, Do(a,s)) \leftrightarrow \\
&a = Move(Agent, Loc, Loc2) \vee \\
&a = Push(Agent, Crate, Loc, \\
&Loc1, Loc2, Direction) \vee \\
&(Clear(Loc,s) \wedge \\
&\neg(a = Move(Agent, Loc1, Loc)) \wedge \\
&\neg(a = Push(Agent, Crate, Loc1, \\
&Loc2, Loc, Direction)))
\end{aligned}
\tag{14}
$$

Using this domain and the instance, formalised in the Appendix, section I, coded in prolog, the predefined planner derived a plan of length 14.

III. Different Instances

We were asked to also build other instances (shown in figure 2 & 3) to further test our domain. In both instances we succeed to find an optimal plan. These instances and all other files are available in the .zip file.

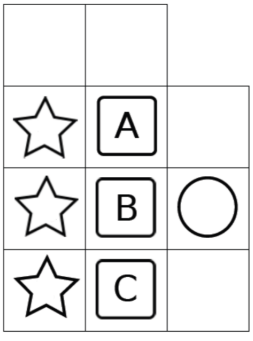

Figure 2: *Clear stars indicate universal goal locations.*

In Figure (2) an altered domain is shown. Here all crates are allowed to end in any of the goal states. Using the domain of the previous task and an altered initial state instance, coded in prolog, the predefined planner derived a plan of length 8.

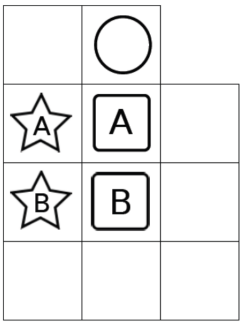

Figure 3: *An instance where the agent visits all locations in the world.*

In Figure (3) the domain is altered in such way that the agent must visit all locations in the world in its resulting plan. Using the domain of the previous task and an altered initial state instance, coded in prolog, the predefined planner derived a plan of length 11.

IV. 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.

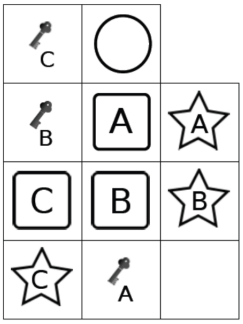

Figure 4: *The agent is required to obtain crate specific keys to unlock the crates before it is able to push.*

We have added a new action pickUpKey. PickUpKey(A, K, Loc, Loc2) has the following possibility axiom:

$$\begin{aligned}\forall A, K, Loc, Loc2, S(Agent(A) \wedge \\ Key(K) \wedge On(A, Loc, S) \wedge \\ connected(Loc, Loc2, D) \wedge \\ On(K, Loc2, S) \rightarrow \\ Poss(PickUpKey(A, K, Loc, Loc2), S))\end{aligned} \tag{15}$$

This new action led the successorstate axioms for on(X,Y,S) and clear(Loc, S) to change.

$$\begin{aligned}Poss(a, s) \rightarrow \\ (On(Agent, Loc, Do(a, s)) \leftrightarrow \\ a = Move(Agent, Loci, Loc) \vee \\ a = Push(Agent, Crate, Loc1, \\ Loc, Loc2, Direction) \vee \\ a = PickUpKey(Agent, Key, Loc1, Loc) \vee \\ (On(Agent, Loc, s) \wedge \\ \neg(a = Move(Agent, Loc, Loc2)) \wedge \\ \neg(a = Push(Agent, Crate, Loc, \\ Loc1, Loc2, Direction)) \wedge \\ \neg(a = PickUpKey(Agent, Block, \\ Loc, Loc1)))\end{aligned} \tag{16}$$

$$\begin{aligned}Poss(a, s) \rightarrow \\ (Clear(Loc, Do(a, s)) \leftrightarrow \\ a = Move(Agent, Loc, Loc2) \vee \\ a = Push(Agent, Crate, Loc, \\ Loc1, Loc2, Direction) \vee \\ a = PickUpKey(Agent, Key, Loc, Loc2) \vee \\ (Clear(Loc, s) \wedge \\ \neg(a = Move(Agent, Loc1, Loc)) \wedge \\ \neg(a = Push(Agent, Crate, Loc1, \\ Loc2, Loc, Direction)))\end{aligned} \tag{17}$$

We also added an extra fluent HasKey, which is used as a precondition for the Push action. This new fluent has the following successorstate axiom.

$$\begin{aligned}Poss(a, s) \rightarrow \\ (HasKey(Agent, Key, Do(a, s)) \leftrightarrow \\ a = PickUpKey(Agent, Key, Loc, Loc2) \vee \\ hasKey(Agent, Key, s).\end{aligned} \tag{18}$$

As mentioned the preconditions of Push also change. Two predicates are added: HasKey, indicating if the agent has the key and RightKey, indicating that the agent has the right key for the specific crate.

$$\begin{aligned}\forall A, C, L1, L2, L3, D, S \\ (Agent(A) \wedge Crate(C) \\ \wedge RightKey(K, C) \wedge hasKey(A, K, S) \\ \wedge On(A, L, S) \wedge Connected(L, L2, D) \\ \wedge On(C, L2, S) \wedge Connected(L2, L3, D) \\ \wedge Clear(L3, S) \rightarrow \\ Poss(Push(A, C, L1, L2, L3, D), S))\end{aligned} \tag{19}$$

Using this domain and an altered initial state instance, coded in prolog, the predefined planner derived a plan of length 9.

V. General Questions

V.1 Sitcalc Expressivity

The Shokoban domain is dynamic and it is important to distinguish what does and what does not change. How to specify what does not change in logical systems when actions are applied, is the frame problem. Situalional calculus and STRIPS lend themselves to be a solution to this frame problem because they seperate the specification the effects of actions from the task of reasoning about these actions. For the Shokoban domain we defined effect axioms and successorstate axioms, which are this seperation.

After specifying what does not change in the Shokoban domain, it is fairly easy to extend our model for different situations. You could simply change the initial state and or add a new effect axiom with a corresponding successorstate axiom.

V.2 Related Work

[Erdem et al., 2015] proposed a general execution monitoring framework for cognitive factories that is intergrated with novel synergistic diagnostic reasoning method which utilizes hypothetical reasoning, geometric reasoning and

learns from experiences. It ensures that an computed optimal plan is executed in a reliable and fault tolerant manner, even under change.

They experimented over reasonably-sized factory scenarios and the usefulness of the framework is shown in (i) allowing repair actions during replanning and (ii) to learn from experience. It can detect multiple interacting faults and utilizes feasiblity checks.

Concluding, the algorithm is applicable in dynamic domains.

III. Assignment 1-2

I. Conflict-set Generation

The definition of a conflict set is as follows: A conflict set is a set of components that, if we assume that they are all normal, the observations imply an inconsistency.

I.1 Problem 1

$CS = [a1]$
Problem 1 consists of two unconnected AND gates since the both inputs of a1 are true, the expected outcome is also true. If we assume a1 to be normal, the obervations imply an inconsistency, namely the observed output is false where we would expect true.

I.2 Problem 2

$CS = [a1, a2]$
Problem 2 consists of two AND gates, for which the output of the first gate is connected to the first input of the second gate. If we assume a1 and a2 to be normal, with inputs in1(a1) = true and in2(a1) = false, observations impy an inconsistency, namely the observed output is true where we would expect false.

I.3 Problem 3

$CS = [a1, o1, a2]$
Problem 3 consists of two AND gates and an OR gate. The outputs of the two AND gates form the inputs of the OR gate. If we assume all three gate to be normal and the inputs all true, the observations imply an inconsistency, namely the observed output is false where we expect it to be true.

I.4 Fulladder

$CS = [a1, x1, a2, r1, x2]$
This problem represents an one-bit full adder. A circuit which can be used for the addition of two bits, with carry in and carry out bits. If we assume all the gates to be normal with the input in1(fa) = 1, in2(fa) = 0, carryin(fa) = 1, we expect the following outputs out(fa) = 0 and carryout(fa) = 1. The observations imply an inconsistensy, namely the output is outputs out(fa) = 1 and carryout(fa) = 0

I.5 tp

TP substracts the HS form the components, which are the components assumed to be abnormal. Since HS is empty it does not substract any component. This yields all components which are assumed to be normal. These components are added to the system discription and the observations. Then tp tries to prove that this yields an inconsistency, by proving the negated formula.

II. Data Structure

A hitting-set tree is a tree for which an intersection of the node label (the conflict set) and it's child's edge label (the child's hitting-set, or path) returns an non-empty set.

III. Hitting-set Algorithm

Our hitting-set algorithm takes 5 arguments: the System Description, the set of Components, the Observations, a Hitting Set Tree of form node(HittingSet, ConflictSet, Children) (to be formed) and Hitting Set containing all elements already in the hitting-set of the parent(initially empty). First we use tp/5 t compute the conflict set(CS). By making use of the predicate makeHSTreeLoop we loop over all elements of CS, generating a childnode in the

tree for each element in CS. Then recursively we do the same for each child.

Our algorithm does run and does terminate. Unfortunately the HST it gives back, is not a full tree. Since we have no time, before the deadline, to find our fault, we have settled for this incomplete algorithm.

We assume that, appart form some minor mistake, our algorithm is able to compute a full hitting-set tree. This tree can be used to find all hitting-sets by looking at all leaf nodes(all nodes with an empty set of children). Since each node has three arguments: hitting-set HS, conflict set CS and a Children list, we can look at the first argument to get this hitting set for each leaf. The predicate findHS returns HSS, a set of hitting sets, which contains the hitting sets of all leaf nodes.

The next step is to compute the minimal hitting-sets from this. This would be the sets that aren't supersets of any other set in HSS. We did not get this far.

We might me able to have makeHSTree only compute minimal hitting-sets by pruning efficiently but we have not figured out how to do this. It might be possible to check for reoccuring edge labels between branches.

The time complexity of our non-optimized hitting-set algorithm consists of the total number of conflict sets C times all the elements in those conflict sets E, so we believe it is $\mathcal{O}(C * E)$.

III.1 Minimal Diagnosis

Using the given diagnostic problems we manually solved it to verify our algorithm, but since our algorithm still has bugs in it, we hereby present the minimal diagnosis. (i) Problem 1: $\{a1, a2\}$. (ii) Problem 2: $\{a1\}, \{a2\}$. (iii) Problem 3: $\{o1\}, \{a1, a2\}$. (iv) Fulladder: $\{x1\}, \{x2, r1\}$ or $\{x1\}, \{x2, a2\}$. These sets are minimal because, within their problem, all presented sets are subset-minimal and are valid hitting-sets.

IV. Reflection

We do not think a 50% devision between the assignments is fair. We spend (more than) the required amount of 36 hours on the assignments(37). We spend much more of this time on assignment 1. This is the assignment that requires a larger part in the report and notes than the second assignment. Also assignment 1 costs a lot of time since you have to first get used to prolog. We would propose a 60%, 40% division of grade. In addition we would like to see the available grades per subtask. For example, we saw around us that some people getting stuck on the keys extention and it would be better to have some kind of indication of how important such an expansion is, in contrast to the whole assignment. We felt like the availabe seminars were quite little, especially one student assistent does not seem to be enough. We met up with Diede outside the office hours to get extra help we couldn't get in he seminars, which does not seem fair, for she does not get paid for this.

References

[Erdem et al., 2015] Erdem, E., Patoglu, V., and Saribatur, Z. G. (2015). Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2007–2013. IEEE.

V. Appendix

I. Task 1

I.1 Initial world

Connect(loc1-1, loc1-2, N)
Connect(loc1-2, loc1-1, S)
Connect(loc1-2, loc1-3, N)
Connect(loc1-3, loc1-2, S)
Connect(loc1-3, loc1-4, N)
Connect(loc1-4, loc1-3, S)

Connect(loc2-1, loc2-2, N)
Connect(loc2-2, loc2-1, S)
Connect(loc2-2, loc2-3, N)
Connect(loc2-3, loc2-2, S)
Connect(loc2-3, loc2-4, N)
Connect(loc2-4, loc2-3, S)

Connect(loc3-1, loc3-2, N)
Connect(loc3-2, loc3-1, S)
Connect(loc3-2, loc3-3, N)
Connect(loc3-3, loc3-2, S)

Connect(loc1-1, loc2-1, E)
Connect(loc2-1, loc1-1, W)
Connect(loc2-1, loc3-1, E)
Connect(loc3-1, loc2-1, W)

Connect(loc1-2, loc2-2, E)
Connect(loc2-2, loc1-2, W)
Connect(loc2-2, loc3-2, E)
Connect(loc3-2, loc2-2, W)

Connect(loc1-3, loc2-3, E)
Connect(loc2-3, loc1-3, W)
Connect(loc2-3, loc3-3, E)
Connect(loc3-3, loc2-3, W)

Connect(loc1-4, loc2-4, E)
Connect(loc2-4, loc1-4, W)

I.2 Initial empty locations

Clear(loc1-1, s0)
Clear(loc1-2, s0)
Clear(loc1-3, s0)
Clear(loc1-4, s0)

Clear(loc2-4, s0)

Clear(loc3-1, s0)
Clear(loc3-3, s0)

I.3 Initial object locations

Crate(blockA)
Crate(blockB)
Crate(blockC)
On(blockA, loc2-3, s0)
On(blockB, loc2-2, s0)
On(blockC, loc2-1, s0)

Agent(a)
On(a, loc3-2, s0)

I.4 Goalstate

$$On(blockA, loc1-2, S) \wedge On(blockB, loc1-3, S) \wedge On(blockC, loc1-1, S) \rightarrow Goal(S) \quad (20)$$