Housekeeping with Multiple Autonomous Robots: Representation, Reasoning and Execution

Erdi Aker[1] and **Ahmetcan Erdogan**[2] and **Esra Erdem**[1] and **Volkan Patoglu**[2]
[1]Computer Science and Engineering, [2]Mechatronics Engineering
Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey

Abstract

We formalize actions and change in a housekeeping domain with multiple cleaning robots, and commonsense knowledge about this domain, in the action language $\mathcal{C}+$. Geometric reasoning is lifted to high-level representation by embedding motion planning in the domain description using external predicates. With such a formalization of the domain, a plan can be computed using the causal reasoner CCALC for each robot to tidy some part of the house. We introduce a planning and monitoring algorithm for safe execution of these plans, so that it can recover from plan failures due to collision with movable objects whose presence and location are not known in advance or due to heavy objects that cannot be lifted alone. We illustrate the applicability of this algorithm with a simulation of a housekeeping domain.

1 Introduction

Consider a house consisting of three rooms: a bedroom, a living room and a kitchen as shown in Fig. 1. There are three cleaning robots in the house. The furniture is stationary and their locations are known to the robots a priori. Other objects are movable. There are three types of movable objects: books (green pentagon shaped objects), pillows (red triangular objects) and dishes (blue circular objects). The locations of some of the movable objects are not known to the robots a priori. Some objects are heavy and cannot be moved by one robot only; but the robots do not know which movable objects are heavy. The goal is for the cleaning robots to tidy the house collaboratively in a given amount of time.

This domain is challenging from various aspects:

- It requires representation of some commonsense knowledge. For instance, in a tidy house, books are in the bookcase, dirty dishes are in the dishwasher, pillows are in the closet. In that sense, books are expected to be in the living room, dishes in the kitchen and pillows in the bedroom. Representing such commonsense knowledge and integrating it with the action domain description (and the reasoner) is challenging.
- A robot is allowed to be at the same location with a movable object only if the object is being manipulated (attached, detached or carried); otherwise, robot-robot, robot-stationary object and robot-moveable object collisions are not permitted. Due to these constraints, representing preconditions of (discrete) actions that require (continuous) geometric reasoning for a collision-free execution is challenging. For instance, moving to some part of a room may not be possible for a robot because, although the goal position is clear, it is blocked by a table and a chair and the passage between the table and the chair is too narrow for the robot to pass through.
- When a plan execution fails, the robots may need to find another plan by taking into account some temporal constraints. For instance, when a robot cannot move an object because it is heavy, then the robots may want to compute another plan that postpones moving the heavy object to its goal position in the last four steps by the help of another robot. Representing such planning problems with temporal constraints and solving them are challenging.
- Solving the whole housekeeping problem may not be possible because the formalization gets too large for the reasoner. In that case, we can partition the housekeeping

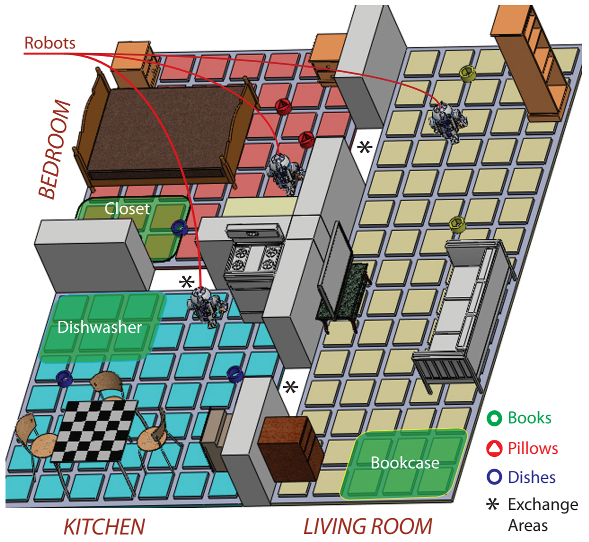

Figure 1: Sample home simulation environment

problem into smaller parts (e.g., each robot can tidy a room of the house). However, then the robots must communicate with each other to tidy the house collaboratively. For instance, if a robot cannot move a heavy object to its goal position, the robot may ask another robot for help. If the robot that cleans kitchen finds a book on the floor, then the robot should transfer it to the robot that cleans the living room, by putting the book in the exchange area between kitchen and living room. Coordination of the robots in such cases, subject to the condition that the house be tidied in a given amount of time, is challenging.

- When a plan execution fails, depending on the cause of the failure, a recovery should be made. If a robot collides with a movable object whose presence and location is not known earlier (e.g., a human may bring the movable object into the room while the robot is executing the plan), then the robot may ask the motion planner to find a different trajectory to reach the next state, and continue with the plan. If the robot cannot find a trajectory, then a new plan can be computed to tidy the room. Monitoring executions of plans by multiple cleaning robots, taking into decisions for recovery from failures, is challenging.

We handle these challenges by representing the housekeeping domain in the action description language $\mathcal{C}+$ (Giunchiglia et al. 2004) as a set of "causal laws" (Section 3) and using the causal reasoner CCALC (McCain and Turner 1997) for planning (Section 4), like in (Caldiran et al. 2009a; 2009b; Haspalamutgil et al. 2010; Erdem et al. 2011), in the style of cognitive robotics (Levesque and Lakemeyer 2007). For the first two challenges, we make use of external predicates. We represent commonsense knowledge as a logic program, and use the predicates defined in the logic program as external predicates in causal laws. Similarly, we can implement collision checks as a function in the programming language C++, and use these functions as external functions in causal laws. For the third challenge, we make use of queries. We represent planning problems as queries that allows us to add temporal constraints. For the other challenges related to safe execution of plans, we introduce a planning and monitoring algorithm that solves the housekeeping problem by dividing it into smaller problems and then combining their solutions, that coordinates multiple cleaning robots for a common goal, and that ensures recovery from plan failures if possible (Section 5).

2 An Overview of Action Language $\mathcal{C}+$

We describe action domains in the action description language $\mathcal{C}+$, by "causal laws." Let us give a brief description of the syntax and the semantics of $\mathcal{C}+$; we refer the reader to (Giunchiglia et al. 2004) for a comprehensive description.

We start with a *(multi-valued propositional) signature* that consists of a set σ of *constants* of two sorts, along with a nonempty finite set $Dom(c)$ of *value names*, disjoint from σ, assigned to each constant c. An *atom* of σ is an expression of the form $c = v$ ("the value of c is v") where $c \in \sigma$ and $v \in Dom(c)$. A *formula* of σ is a propositional combination of atoms. If c is a Boolean constant, we will use c (resp. $\neg c$) as shorthand for the atom $c = True$ (resp. $c = False$).

A signature consists of two sorts of constants: *fluent constants* and *action constants*. Intuitively, fluent constants denote "fluents" characterizing a state; action constants denote "actions" characterizing an event leading from one state to another. A *fluent formula* is a formula such that all constants occurring in it are fluent constants. An *action formula* is a formula that contains at least one action constant and no fluent constants.

An *action description* is a set of *causal laws* of three sorts. *Static laws* are of the form

$$\textbf{caused } F \textbf{ if } G \tag{1}$$

where F and G are fluent formulas. *Action dynamic laws* are of the form (1) where F is an action formula and G is a formula. *Fluent dynamic laws* are of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{2}$$

where F and G are as above, and H is a fluent formula. In (1) and (2) the part **if** G can be dropped if G is $True$.

The meaning of an action description can be represented by a "transition system", which can be thought of as a labeled directed graph whose nodes correspond to states of the world and edges to transitions between states. Every state is represented by a vertex labeled with a function from fluent constants to their values. Every transition is a triple $\langle s, A, s' \rangle$ that characterizes change from state s to state s' by execution of a set A of primitive actions.

While describing action domains, we can use some abbreviations. For instance, we can describe the (conditional) direct effects of actions using expressions of the form

$$c \textbf{ causes } F \textbf{ if } G \tag{3}$$

which abbreviates the fluent dynamic law

$$\textbf{caused } F \textbf{ if } \mathit{True} \textbf{ after } c \wedge G$$

expressing that "executing c at a state where G holds, causes F."

We can formalize that F is a precondition of c by the expression

$$\textbf{nonexecutable } c \textbf{ if } \neg F \tag{4}$$

which stands for the fluent dynamic law

$$\textbf{caused } \mathit{False} \textbf{ if } \mathit{True} \textbf{ after } c \wedge \neg F.$$

Similarly, we can prevent the execution of two actions c and c' by the expression

$$\textbf{nonexecutable } c \wedge c'.$$

Similarly, we can express that F holds by default by the abbreviation

$$\textbf{default } F.$$

We can represent that the value of a fluent F remains to be true unless it is caused to be false, by the abbreviation

$$\textbf{inertial } F.$$

In almost all the action domains, we express that there is no cause for the occurrence of an action A, by the abbreviation

$$\textbf{exogenous } A.$$

3 Representation of Housekeeping Domain

We represent the housekeeping domain in $\mathcal{C}+$ and present it to CCALC for solving reasoning problems, such as planning. In the following, we will describe the housekeeping domain using the language of CCALC.

Fluents and actions We view the house as a grid. The robots and the endpoints of objects are located at grid-points. We consider the fluents `at(TH,X,Y)` ("thing `TH` is at `(X,Y)`") and `connected(R,EP)` ("robot `R` is connected to endpoint `EP`"), and the actions `goto(R,X,Y)` ("robot `R` goes to `(X,Y)`"), `detach(R)` ("robot `R` detaches from the object it is connected to"), and `attach(R)` ("robot `R` attaches to an object"). The fluents are inertial and the actions are exogenous. CCALC allows us to include this information at the very beginning of the action description while declaring fluent constants and action constants.

Direct effects of actions We describe the direct effects of the actions above by causal laws of the form (3). For instance, the following causal law expresses the direct effect of the action of a robot `R` going to location `(X,Y)`:[1]

```
goto(R,X,Y) causes at(R,X,Y).
```

Similarly, we can describe the direct effects of the action of a robot `R` detaching from the endpoints of an object it is connected to:

```
detach(R) causes -connected(R,EP)
   if connected(R,EP).
```

To describe the direct effects of the action of a robot `R` attaching to an endpoint of an object, we introduce an "attribute" `attach_point` of this action to show at which endpoint the robot is attaching. Attributes of actions is a useful feature of CCALC that allows us to talk about various special cases of actions without having to modify the definitions of more general actions. We can formalize the direct effect of attaching a payload ("robot `R` is connected to the endpoint `EP` of an object"):

```
attach(R) causes connected(R,EP)
   if attach_point(R)=EP.
```

Preconditions of actions We describe effects of actions by causal laws of the form (4). For instance, we can describe that a robot `R` cannot go to a location `(X,Y)` if the robot is already at `(X,Y)`, by the causal laws:

```
nonexecutable goto(R,X,Y) if at(R,X,Y).
```

To describe that a robot `R` cannot go to a location `(X,Y)` if that location is already occupied by a stationary object, we need to know in advance the locations of stationary objects in the house. Such knowledge is represented as the "background knowledge" in Prolog. CCALC allows to use the predicates defined as part of background knowledge, in causal laws, as follows:

```
nonexecutable goto(R,X,Y) where occupied(X,Y).
```

[1]To present formulas to CCALC, conjunctions $\wedge$, disjunctions $\vee$, implications $\supset$, negations $\neg$ are replaced with the symbols `&`, `++`, `->>`, and `-` respectively.

where `occupied(X,Y)` describe the locations `(X,Y)` occupied by stationary objects.

In general, the `where` parts in causal laws presented to CCALC include formulas that consist of "external predicates/functions". These predicates/functions are not part of the signature of the domain description (i.e., they are not declared as fluents or actions). They are implemented as functions in some programming language of the user's choice, such as C++. External predicates take as input not only some parameters from the action domain description (e.g., the locations of robots) but also detailed information that is not a part of the action domain description (e.g., geometric models). They are used to externally check some conditions under which the causal laws apply, or externally compute some value of a variable/fluent/action. For instance, suppose that the external predicate `path_exists(R,X,Y,X1,Y1)` (implemented in C++) holds if there is a collision-free path between `(X,Y)` and `(X1,Y1)` for the robot `R`. Then we can express that the robot `R` cannot go from `(X1,Y1)` to `(X,Y)` where `path_exists(R,X,Y,X1,Y1)` does not hold, by a causal law presented to CCALC as follows:

```
nonexecutable goto(R,X,Y) if at(R,X1,Y1)
   where -path_exists(R,X1,Y1,X,Y).
```

Now let us present the preconditions of two other sorts of actions. Consider the action of a robot `R` attaching to an endpoint of an object. This action is not possible if the robot is connected to some endpoint `EP` of an object:

```
nonexecutable attach(R) if connected(R,EP).
```

Note that here we do not refer to the special case of the action of attaching via attributes.

Also this action is not possible if the robot and the endpoint are not at the same location `(X,Y)`:

```
nonexecutable attach(R) & attach_point(R)=EP
   if -[\/X \/Y | at(R,X,Y) & at(EP,X,Y)].
```

In the last line above, the negated expression stands for a disjunction of conjunctions `at(R,X,Y) & at(EP,X,Y)` over locations `(X,Y)`.

Finally, we can describe that a robot `R` cannot detach from an object if it is not connected to any endpoint:

```
nonexecutable detach(R)
   if [/\EP | -connected(R,EP)].
```

the expression in the last line above stands for a conjunction of `-connected(R,EP)` over endpoints `EP`.

Ramifications We describe two ramifications of the action of a robot `R` going to a location `(X,Y)`. If the robot is connected to an endpoint of an object, then the location of the object changes as well:

```
caused at(EP,X,Y)
   if connected(R,EP) & at(R,X,Y).
```

Furthermore, neither the robot nor the endpoint are at their previous locations anymore:

```
caused -at(TH,X,Y) if at(TH,X1,Y1)
   where X\=X1 ++ Y\=Y1.
```

Here `TH` denotes a "thing" which can be either a robot or an endpoint.

Constraints We ensure that two objects do not reside at the same location by the constraint

```
caused false if at(EP,X,Y) & at(EP1,X,Y)
   where EP \= EP1.
```

and that a robot is not connected to two endpoints by the constraint

```
caused false
   if connected(R,EP1) & connected(R,EP)
   where EP \= EP1.
```

Some objects OBJ are large and have two endpoints EP and EP1 one unit from each other. To be able to pick these objects, we ensure that the endpoints of the objects are located horizontally or vertically, and one unit apart from each other by the constraint:

```
caused false if at(EP1,X1,Y1) & at(EP2,X2,Y2)
   where belongs(EP1,OBJ) & belongs(EP2,OBJ)
   & Dist is sqrt((X1-X2)^2 + (Y1-Y2)^2)
   & EP1 \= EP2 & Dist \= 1.
```

Here belongs/2 is defined externally in predicate.

Finally, we need to express that a robot cannot move to a location and attach to or detach from an endpoint of an object at the same time.

```
nonexecutable goto(R,X,Y) & attach(R).
nonexecutable goto(R,X,Y) & detach(R).
```

Commonsense knowledge To clean a house, the robots should have an understanding of the following: tidying a house means that the objects are at their desired locations. For that, first we declare a "statically determined fluent" at_desired_location(EP) describing that the endpoint of an object is at its expected position in the house. A statically determined fluent is much like a "derived predicate": it is defined in terms of other fluents. We define at_desired_location(EP) as follows:

```
caused at_desired_location(EP) if at(EP,X,Y)
   where in_place(EP,X,Y).
default -at_desired_location(EP).
```

The second causal law expresses that normally the movable objects in an untidy house are not at their desired locations. The first causal law formalizes that the endpoint EP of an object is at its desired location if it is at some "appropriate" position (X,Y) in the right room. Here in_place/3 is defined externally.

After defining at_desired_location/1, we can introduce a "macro" to define tidy:

```
:- macros
   tidy -> [/\EP | at_desired_location(EP)].
```

Finally, the robots need to know that books are expected to be in the bookcase, dirty dishes in the dishwasher, and pillows in the closet. Moreover, a bookcase is expected to be in the living-room, dishwasher in the kitchen, and the closet in the bedroom. We describe such background knowledge externally as a Prolog program. For instance, the external predicate in_place/3 is defined as follows:

```
in_place(EP,X,Y) :- belongs(EP,Obj),
  type_of(Obj,Type), el(Type,Room),
  area(Room,Xmin,Xmax,Ymin,Ymax),
  X>=Xmin, X=<Xmax, Y>=Ymin, Y=<Ymax.
```

Here belongs(EP,OBJ), type_of(OBJ,Type) describes the type Type of an object Obj that the endpoint EP belongs to, and el(Type,Room) describes the expected room of an object of type Type. The rest of the body of the rule above checks that the endpoint's location (X,Y) is a desired part of the room Room.

Note that the expected location of an object depends on where it is: for instance, the expected location of a book on the floor of a kitchen is the exchange area between kitchen and living room (so that the robot whose goal is to tidy the living room can pick it up and put it in the bookcase); on the other hand, the expected location of a plate on the floor of kitchen is the dishwasher. Therefore, area/5 describes either an exchange area (if the object does not belong to the room where it is at) or a deposit area (if the object belongs to the room where it is at). Note also that knowledge about specific objects in a room as well as specific deposit and exchange areas are not common knowledge to all robots; each robot has a different knowledge of objects in their room.

4 Reasoning about Housekeeping Domain

Given the action domain description and the background and commonsense knowledge above, we can solve various reasoning tasks, such as planning, using CCALC. However, the overall planning problem for three cleaning robots may be too large (considering the size of the house, number of the objects, etc.). In such cases, we can divide the problem into three smaller planning problems, assigning each robot to tidy a room of the house in a given amount of time.

Consider the housekeeping domain described above. The goal is for the cleaning robots to tidy the house collaboratively in a given amount of time: Robot 1 is expected to tidy the living room, Robot 2 the bedroom, and Robot 3 the kitchen.

Suppose that the locations of some of the movable objects are known to the robots a priori, as shown in Table 1. Robot 1 knows that there are two books, comics1 and novel1, on the living room floor. Robot 2, on the other hand, knows that there are two pillows, redpillow1 and bluepillow1, and a plate, plate1, on the bedroom floor. The robots also know where to collect the objects, as shown in Table 1. For instance, Robot 2 knows that, in the bedroom, the closet occupies the rectangular area whose corners are at (5,0), (5,3), (7,0), (7,3). Robot 2 also knows that the objects that do not belong to bedroom, such as plate1 of type dish, should be deposited to the exchange area between bedroom and kitchen, that occupies the points (3,7)-(5,7).

Planning problems for each robot are shown in Table 2. For instance, in the living room, initially Robot 1 is at (3,2), whereas the books comics11 and novel11 are located at (1,2) and (6,3). The goal is to tidy the room and make sure that the robot is free (i.e., not attached to any objects), in at most k steps. Here free is a macro, like tidy.

Table 1: Background knowledge for each robot.

	Robot 1 in Living Room	Robot 2 in Bedroom	Robot 3 in Kitchen

Object: Type	`comics1 : book`
	`novel1 : book`
		`redpillow1 : pillow`
		`bluepillow1 : pillow`
		`plate1 : dish`
			`pan1 : dish`
			`spoon1 : dish`

Deposit Area	`(13,15,2,5)`	`(5,7,0,3)`	`(0,2,0,3)`

Exchange Area	`kitchen: (11,12,0,0)`
	`bedroom: (0,0,3,4)`
		`kitchen: (3,4,7,7)`
		`living-room: (7,7,3,4)`
			`bedroom: (0,0,3,4)`
			`living-room: (3,4,7,7)`

Table 2: Planning problems for each robot.

	Robot 1 in Living Room	Robot 2 in Bedroom	Robot 3 in Kitchen

Initial State	`at(r1,3,2) at(comics1,1,2)`
	`at(novel1,6,3)`
		`at(r2,5,6) at(redpillow1,2,5)`
		`at(bluepillow1,3,6) at(plate1,6,3)`
			`at(r3,1,5) at(pan1,3,1)`
			`at(spoon1,3,6)`

Goal	`tidy, free`	`tidy, free`	`tidy, free`

Table 3: Execution of the plans computed by CCALC for each robot. The rows that are not labeled by a time step are not part of these plans, but are implemented at the low-level.

Time Step	Robot 1 in Living Room	Robot 2 in Bedroom	Robot 3 in Kitchen
1	`goto(r1,6,3)`	`goto(r2,6,3)`	`goto(r3,3,1)`
2	`attach(r1,novel1)`	`attach(r2,plate1)`	`attach(pan1)` - FAILURE (Heavy object)
			Re-task planning
3	`goto(r1,13,3)`	`goto(r2,7,3)`	`goto(r3,3,6)`
4	`detach(r1)`	`detach(r2)`	`attach(r3,spoon1)`
5	`goto(r1,1,2)`	`goto(r2,3,6)` - FAILURE (Unknown object)	`goto(r3,0,2)`
		Re-motion planning	
6	`attach(r1,comics1)`	`attach(r2, bluepillow1)`	`detach(r3)`
	Get ready to help r3		
7	help r3	`goto(r2,5,2)`	`goto(r3,3,1), goto(r1,4,1)`
8	help r3	`detach(r2)`	`attach(r3,pan1), attach(r1,pan2)`
9	help r3	`goto (r2,2,5)`	`goto(r3,0,1), goto(r1,1,1)`
10	help r3	`attach(r2,redpillow1)`	`detach(r3), detach(r1)`
	Get ready to continue plan		
11	`goto(r1,13,2)`	`goto(r2,7,1)`	-
12	`detach(r1)`	`detach(r2)`	-
	Final check for previously unknown or relocated objects		
13	-	`goto(r2,4,5)`	-
14	-	`attach(r2,yellowpillow1)`	-
15	-	`goto(r2,6,1)`	-
16	-	`detach(r2)`	-

The planning problem for Robot 1 is presented to CCALC as a query as follows:

```
:- query
maxstep :: 0..k;
0: at(r1,3,2), at(comics1,1,2), at(novel1,6,3);
maxstep: tidy, free.
```

CCALC finds a shortest plan, Plan 1, for this problem:

```
0:  goto(r1,6,3)
1:  attach(r1,attach_point=novel1)
2:  goto(r1,13,3)
3:  detach(r1)
4:  goto(r1,1,2)
5:  attach(r1,attach_point=comics1)
6:  goto(r1,13,2)
7:  detach(r1)
```

While CCALC computes such a plan, CCALC calls a motion planner (based on Rapidly exploring Random Trees (RRTs) (Lavalle 1998)) to check the preconditions of the action of a robot going to a location without any collisions.

```
Algorithm 1: Planning and Monitoring
Input: An action description $\mathcal{D}$, a goal $g$, and a nonnegative integer $k$
$s \leftarrow$ getState;   // Obtain the current state $s$ by sensors
while $s$ does not satisfy $g$ do
    // Find a plan $\mathcal{P}$ of length $\leq k$ to reach the goal $g$ from $s$ under constraints $F$, so that $\mathcal{P}$ involves $n$ robots only
    $n := 1$;   $F := true$;
    $plan, \mathcal{P} \leftarrow$ findP($\mathcal{D}, n, s, g, F, k$);
    if $plan$ then
        // Execute the plan $\mathcal{P}$ starting at state $s$ and, if it fails, return the cause of failure ($result$) and the next state ($s'$) to-be-reached
        $result, s' \leftarrow$ execute($\mathcal{P}, s$);
        while $result \neq$ NO-FAILURE do
            // Recover from the failure
            if $result ==$ UNKNOWN-OBJECT then
                $c \leftarrow$ getConfig;   // Obtain the current configuration $c$ of the robot by sensors
                // Find a new trajectory $\pi$ from $c$ to $s'$
                $traj, \pi \leftarrow$ findT($c, s'$);
                if $traj$ then
                    $result \leftarrow$ Follow the trajectory $\pi$; if it fails, return the cause of failure;
                    if $result ==$ NO-FAILURE then
                        $result, s' \leftarrow$ execute($\mathcal{P}, s'$);
                else
                    // No trajectory: Find a new plan $\mathcal{P}$
                    $plan := false$;
                    while $\neg plan$ do
                        $s \leftarrow$ Find a safe state closeby;
                        $plan, \mathcal{P} \leftarrow$ findP($\mathcal{D}, n, s, g, F, k$);
                    $result, s' \leftarrow$ execute($\mathcal{P}, s$);
            else if $result ==$ OBJECT-NOT-FOUND then
                // Find a new plan $\mathcal{P}$
                $s \leftarrow$ getState;
                $plan, \mathcal{P} \leftarrow$ findP($\mathcal{D}, n, s, g, F, k$);
                if $plan$ then
                    $result, s' \leftarrow$ execute($\mathcal{P}, s$);
            else
                // $result$ = HEAVY-OBJECT
                // Find a new plan $\mathcal{P}$ with 2 robots
                $n := 2$;   $i := 4$;
                $s \leftarrow$ getState;
                $plan := false$;
                while $\neg plan$ and $i \leq k$ do
                    $F \leftarrow$ "the helper robot does not move before the last $i^{th}$ step";
                    $plan, \mathcal{P} \leftarrow$ findP($\mathcal{D}, n, s, g, F, k$);
                    $i := i + 1$;
                $result, s' \leftarrow$ execute($\mathcal{P}, s$);
    $s \leftarrow$ getState;
```

5 Monitoring the Cleaning Robots

For each cleaning robot, a plan is computed and executed by Algorithm 1. First, each robot obtains the current state s of the world from the sensor information, by the function `getState`. After that, each robot computes a plan $\mathcal{P}$ of length less than or equal to a given nonnegative integer k, from the observed state s to a goal state (that satisfies the given goal g) in a world described by a given action domain description $\mathcal{D}$. Plans are computed using CCALC as described in the previous section, by the function `findP`.

Once a plan $\mathcal{P}$ is computed, each robot starts executing it. However, a plan execution may fail, for instance, due to the unpredictable interference of human. Human may relocate an object which is to be carried by robot, or bring a new movable object to the room. Since the robots are unaware of these dynamic changes, they may collide with these obstacles. Furthermore, while most of the moveable objects are carried with only one robot, some of these objects are heavy and their manipulation requires two robots. The robots do not know in advance which objects are heavy, but discover a heavy object only when they attempt to move it.

When such incidents occur, robots can identify the cause of the failure and act accordingly. When a robot collides with an unknown movable object while following its planned trajectory for a transition $\langle s, A, s' \rangle$ (i.e., the cause of the failure is *UNKNOWN-OBJECT*), the robot tries to calculate a new, collision-free trajectory π to reach the next state s' from its current configuration. Such a trajectory π is computed by the function `findT`, which implements a motion planner based on RRTs. If no such trajectory is calculated by the motion planer, the robot goes to a safe state s (possibly the previous state) and asks CCALC to find a new plan $\mathcal{P}$ to reach the goal from s taking into account the recently discovered moveable objects.

When a plan fails because a robot attempts to carry an object which is relocated by another robot (i.e., the cause of the failure is *OBJECT-NOT-FOUND*), the robot observes the world and asks CCALC to find a new plan $\mathcal{P}$ to reach the goal from the current state s taking into account the recently relocated moveable objects.

When a plan fails because a robot attempts to manipulate a heavy object (i.e., the cause of the failure is *HEAVY-OBJECT*), the robot asks for assistance from other robots so that the heavy object can be carried to its destination. However, in order not to disturb the other robots while they are occupied with their own responsibilities, the call for help is delayed as much as possible. With the observation that the manipulation of the heavy object takes 4 steps (get to the heavy object, attach to it, carry it, detach from it), this is accomplished by asking CCALC to find a new plan $\mathcal{P}$ that manipulates the heavy object within the last $i = 4, 5, 6, ...$ steps of the plan only. Once such a plan is computed, one of the robots who are willing to help gets prepared (e.g., detaches from the object it is carrying, if there is any) and goes to the room of the robot who requests help. (Currently, task allocation is done randomly.)

Note that if a human brings some objects into the room, and these objects are not in collision course with the robots, they cannot be discovered until the robot observes the world. Therefore, after the execution of the plans, a final observation instance is triggered comparing current state of the room with its goal state. If there are any discrepancies, then the robots ask for new plans to reach the goal.

In Algorithm 1, plans are executed by the function `execute`. This function does the following for each transition $\langle s, A, s' \rangle$ of the history of the plan. If A is a primitive action of the form `goto(R,X,Y)`, then it computes a trajectory and follows the trajectory. If A is a primitive action of some other form (e.g., `attach(R)` or `detach(R)`), then the action is executed by the function `execute`. If A is a concurrent action of the form $\{A_1, A_2\}$ to be executed by two robots, then first it calls for help and finds an available robot. After that, it finds how each action A_i is to be executed and sends the relevant information to the helper robot so that both actions are executed concurrently.

Table 3 shows the execution of plans by Robots 1–3. Robot 1 executes Plan 1 safely without any collisions, but goes to kitchen at time step `7` to help Robot 3 to move a heavy object to its goal position. Robot 2 starts executing its plan, but fails at time step `5` due to a collision with a pillow `bluePillow1` unknown to Robot 2 a priori. Fortunately, the motion planner finds another trajectory to reach the next state and Robot 2 continues with the execution of the rest of the plan after following the trajectory. After the completion of the plan at time step `13`, Robot 2 checks whether there is any other pillow to be put into the closet. After discovering another pillow `yellowPillow1`, Robot 2 asks for a new plan to move it to the closet, and safely executes that plan as well. Robot 3 on the other hand starts executing a plan, but at time step `2`, finds out that the pan `pan1` he wants to move is too heavy. Then Robot 3 goes to a safe state and asks for help to carry the heavy object to its goal position.

We showed the execution of these plans with a simulation, where the housekeeping domain is modeled using VRML 2.0 and the execution of the actions is implemented in Matlab/Simulink. Video clips illustrating this simulation can be found at `http://krr.sabanciuniv.edu/cogrobo/demos/housekeeping/`.

6 Conclusion

We formalized a housekeeping domain with multiple cleaning robots, in the action description language $\mathcal{C}+$, and solved some housekeeping problem instances using the reasoner CCALC as part of a planning and monitoring framework.

While representing the domain, we made use of some utilities of CCALC: external predicates are used to embed geometric reasoning in causal laws, and attributes of actions are used to talk about the special cases of actions. To represent commonsense knowledge and background knowledge, we made use of external predicates/functions and macros in causal laws. While computing plans for each robot, taking into account temporal constraints, we made use of queries.

The planning and monitoring algorithm integrates planning with motion planning; if a plan failure occurs, it identifies the cause of the failure and finds a recovery accordingly. We showed the applicability of the planning and monitoring framework with a simulation of a housekeeping domain.

A related work on the use of AI planning for housekeeping with multiple robots is (Lundh, Karlsson, and Saffiotti 2008), which considers a domain with no concurrency, no obstacles and no human intervention. On the other hand, it considers heterogenous robots with different functionalities (e.g., sensors, actuators), and take their functionalities into account while planning. The extension of our approach to handle collaborations of such heterogenous robots is part of our ongoing work.

Since the focus of this work on the use of causal reasoning for planning and coordination of multiple robots in a decentralized way, important aspects of this domain, such as communication of the robots and interaction of robots for task allocation have been dealt in a simple way at the low-level. The investigation of the use of an Agent Communication Language, such as KQML (Finin et al. 1994) or FIPA ACL (O'Brien and R. 1998), and the use of a method for deciding for compatible agents, such as Contract Net Protocols (Smith 1980), is also part of our ongoing work.

7 Acknowledgments

We thank anonymous reviewers for their useful comments. This work has been partially supported by Sabanci University IRP Grant.

References

Caldiran, O.; Haspalamutgil, K.; Ok, A.; Palaz, C.; Erdem, E.; and Patoglu, V. 2009a. Bridging the gap between high-level reasoning and low-level control. In *Proc. of LPNMR*.

Caldiran, O.; Haspalamutgil, K.; Ok, A.; Palaz, C.; Erdem, E.; and Patoglu, V. 2009b. From discrete task plans to continuous trajectories. In *Proc. of ICAPS Workshop BTAMP*.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. of ICRA*. To appear.

Finin, T. W.; Fritzson, R.; McKay, D. P.; and McEntire, R. 1994. KQML as an agent communication language. In *Proc. of CIKM*, 456–463.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153:49–104.

Haspalamutgil, K.; Palaz, C.; Uras, T.; Erdem, E.; and Patoglu, V. 2010. A tight integration of task and motion planning in an execution monitoring framework. In *Proc. of AAAI Workshop BTAMP*.

Lavalle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. Technical report.

Levesque, H., and Lakemeyer, G. 2007. Cognitive robotics. In *Handbook of Knowledge Representation*. Elsevier.

Lundh, R.; Karlsson, L.; and Saffiotti, A. 2008. Autonomous functional configuration of a network robot system. *Robotics and Autonomous Systems* 56(10):819–830.

McCain, N., and Turner, H. 1997. Causal theories of action and change. In *Proc. of AAAI/IAAI*, 460–465.

O'Brien, P., and R., N. 1998. FIPA – Towards a standard for software agents. *BT Technology Journal* 16:3:51–59.

Smith, R. G. 1980. The Contract Net Protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Computers* 29(12):1104–1113.