Modelling with PRISM of Intelligent System

Zhongdan Ma
Linacre College
University of Oxford

Supervisor: Prof. Marta Kwiatkowska

A thesis submitted for the degree of

MSc in Computer Science

September 2008

Acknowledgement

I am grateful for all the supports I have received whilst researching and writing up this dissertation. Without questions, the first acknowledgement goes to my supervisor Professor Marta Kwiatkowska, who offered incredible insights, enthusiasm, and directions. It is her knowledge and experience in the field of modelling and verification of probabilistic systems, as well as charm that brought me into this exciting project

I am grateful to Dr. David Parker and Dr. Gethin Norman, whose patience and experience save me from struggling with equations and Unix commands alone.

Heartfelt thanks to Nathan Flis, for prove reading my dissertation and feeding me with his birthday cake made by the beautiful wife Anna.

I also appreciate the supports and encouragements from my family, colleagues, house mates and Lubin Chen.

Finally, special thanks to Isaac, for putting up with me.

Any errors remained are mine alone, but there would have been many more without their assistance.

Zhongdan Ma
September 2008

Abstract

Dementia is the progressive decline in cognitive functions such as thinking, remembering, and reasoning due to damage or disease in the brain. People with advanced dementia have difficulty completing the activities of daily living and need assistance from a caregiver. Cognitive assistive technology that helps guide users through these activities allows this elderly population to complete activities of daily living more independently and can relieve some of the stress of the caregiver. The research project of the COACH system is one of the intelligent systems being developed to assist persons with dementia in such activities, in particular hand-washing.

Based on the research of the COACH system, we model the hand-washing problem as Markov decision processes (MDPs) in PRISM, which is one of the stochastic model checking tools and has been used to analyse systems from a wide range of application domains. This project is a case study of modelling and verification of intelligent systems in PRISM. Various data analysis has been applied to investigate properties of the models.

From the properties checking results, we have identified the factors that play important roles. General rules for choosing policies for different users are summarised. Based on lessons learnt from modelling the hand-washing problem, we present the design of a planning system to help people with dementia during the activity of dressing.

Content

Illustrations

Chapter 1
Introduction

1.1 Background

Dementia is the progressive decline in the cognitive functions of thinking, remembering, and reasoning due to damage or disease in the brain. It usually affects older people and becomes more common with age. People with advanced dementia have difficulty completing the activities of daily living and need assistance from a caregiver. However, the dependence on caregivers can lead to feelings of anger and helplessness for the patient. The physical and emotional burden placed on the caregiver is also heavy. Cognitive-assistive technology that helps guide users through these activities allows the population stricken with dementia to complete activities of daily living more independently and can relieve some of the stress of the caregiver. The research project of the COACH system presented in [1], [2], and [5] is one of the intelligent systems being developed to assist persons with dementia in such activities, in particular hand-washing.

PRISM is one of the stochastic model checking tools and has been used to analyse systems from a wide range of application domains, including communication and multimedia protocols, randomised distributed algorithms, security protocols, biological systems and many others [6]. This project is a modelling and verification case study of intelligent systems in PRISM.

This project models the hand-washing problem using DTMCs and MDPs. Based on different assumptions and focused on different system variables, several simplified versions of the hand-washing problem will be analysed, with reduced state space.

The objective of this project is to compare the efficiency of different policies, simulate how the different policies work under different scenarios, and investigate the influence of different factors on the hand-washing problem by modelling the it in PRISM. Also, a new planning system is designed for activities other than hand-washing, demonstrating the versatility of application for the policy.

PRISM is a probabilistic model checking tool which has direct support for three types of probabilistic models: discrete-time Markov chains (DTMCs), Markov decision processes (MDPs) and continuous-time Markov chains (CTMCs).

1.2 Outline

The remainder of this dissertation is outlined as follows. Chapters 2-4 summarize the information that the writer had to learn in order to undertake the work in this dissertation. Chapter 2 introduces the concept of Markov decision processes, and explains how they are used in intelligent systems. The main features of PRISM, the probabilistic model checker used to analyse our models, are described in chapter 3. Chapter 4 describes in detail the COACH system which our application rests on. Chapters 5 and 6 contain the main contribution of this dissertation. In Chapter 5, we explain the simplified model for the hand-washing problem based on COACH's planning system, and discuss properties of both the MDP model and the DTMC model specified in PRISM. Experimental results and analysis are also presented in chapter 5. Supported by our findings in chapter 5, a planning system for the activity of dressing is proposed in Chapter 6. The idea of generalising the use of probabilistic model checking in designing a planning system for more activities of daily living is also presented. Finally, in chapter 7, the achievements and limitations of our contributions are summarised, and suggestions for future related work are included.

Chapter 2
MDP in intelligent systems

With its ability to address stochasticity, decision-theoretic planning is an attractive extension of classical artificial intelligence (AI) planning. As decision-theoretic models, Markov decision processes (MDPs) have been widely used as the underlying models for posing and solving AI planning problems where outcomes are partly random and partly under the control of the decision maker. In this chapter, the general MDP framework, its applications in planning problems and their computational complexity will be discussed. Major information presented in the chapter is sourced form [10].

2.1 The general MDP framework

In this section, we discuss the general MDP framework for basic problem formulation. The framework is general enough to fit most problems in the domain of intelligent systems, and it is compared with the classical view of planning problems. For different problems, the framework can be specified by making different assumptions, which can either simplify the general framework or extend it. Examples of planning problems with underlying MDP framework are given in section 2.2.

A discrete time MDP consists of a finite or countable set S of possible system states, a finite set A of possible events, a real value reward function R, and a description Pr(S'|S, A) of each event's effects in each state.

At any point in time, the system can be in one of the all possible system states, and the system's state changes over time in response to events. For discrete-time system, unit of time is defined as time step or stage. It is assumed that every event takes one time step to complete.

2.1.1 Discrete time Markov chain

MDPs are an extension of Discrete time Markov Chains (DTMCs). An MDP would reduce to a DTMC if the action taken was fixed for each state.

To model the uncertainty of the system, the system's state at some time step t is considered as a random variable S^t, which has a domain of S. The variable S^t does not

depend directly on the value of future variable S^k (k>t), which is known as the assumption of forward causality. Furthermore, we assume that the present state contains enough information to predict the next state. In other words, this assumption (the first order Markov assumption) says that given the state at time *t* is known, transition probabilities to the state at time *t* + 1 are independent of all previous states:

$$Pr(S^{t+1}|S^t, S^{t-1},\ldots,S^0)= Pr(S^{t+1}|S^t)$$

Also the state transition is assumed to be stationary, which means the effects of an event depend only on the state, but not on the time step at which it occurs:

$$Pr(S^{t+1}|S^t) = Pr(S^{t+k+1}|S^{t+k})$$

First order stationary Markov chains are often illustrated using directed graphs, where the arrows describe the probabilities of going from one state to other states.

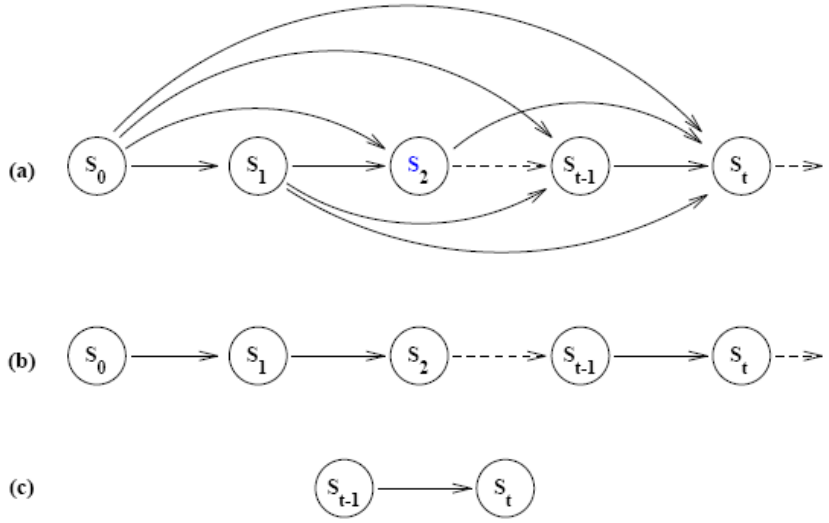

Figure 1 (a) a general stochastic process; (b) a Markov chain; (c) a stationary Markov chain (sourced from [10])

2.1.2 Actions and exogenous events

The difference between MDPs and DTMC is the choice of actions. While a DTMC describes the state transitions of a stochastic system, it does not capture the fact that the agent, or decision maker, can choose an appropriate course of action in order to change the system's state.

Like Markov chains, MDPs possesses the Markov property, which requires that any information necessary to predict the effects of all events is captured in the state. In other words, the effects of an event in a state depend only on that state and not on the prior history:

$$Pr(S^{t+1} \mid A, S^t, S^{t-1}, \ldots, S^0) = Pr(S^{t+1} \mid A, S^t)$$

In MDPs, there are two kinds of events: actions which are instigated by an agent in order to change the system's state; and exogenous events which are not under the agent's control, and whose occurrence may not be fully predictable. No matter whether their occurrence is controlled by the agent or is unpredictable, the effects of both actions and exogenous events can be either deterministic or stochastic. For the deterministic events, a next state can be specified if the the current state and the event are known. In contrast, for the stochastic events, a probability distribution over next states is specified knowing the current state and the event.

A transition function $Pr(S'=s' \mid A=a, S=s)$ captures system dynamics by specifying the probability that performing action a in state s will move the system into state s'. Since an action may have effects on the occurrence of any exogenous events as well as direct effects on the current state, the state transitions distribution can be very complicated due to the events interaction. Instead of specifying the transition functions for actions and exogenous events separately, state transitions are generally viewed as comprised effects of these two events. So, the transition function for an action is usually assumed to account for any exogenous events that might occur when the action is performed [10].

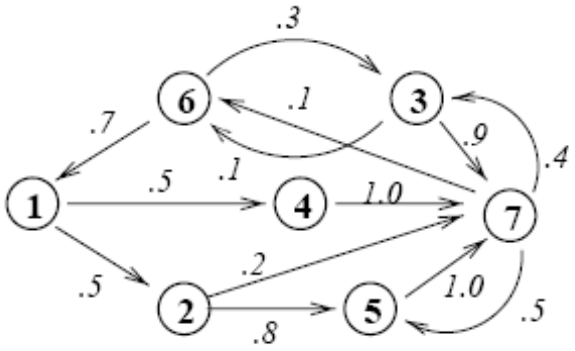

Figure 2: A state transition diagram (sourced from [8])

2.1.3 Observations, policies and reward

The planning system's sensing capability can be modelled by a finite set of observations O. Observations capture all the information available to the decision maker. The observable history is the sequences of actions and observations generated from time step 0 to some time step of interest.

A policy is a strategy telling the decision maker which action to choose. Formally, a policy π is a mapping from the set of observable histories H_O to the set of actions A. In order to evaluate the quality of a policy, a value function is defined. A value

function V is mapped from the set of system histories H_S to reals **R**. Similar to the observation history, the system is a sequence of actions and states generated from time step 0 to some time step of interest. The decision maker prefers system history h to h' when V(h)>V(h'). The value for a policy V_π is the expected value over the possible history caused by actions following this policy.

Since the representation of the system history can be cumbersome, some assumptions are commonly made so that the structure in the value function can be easily identified. Assuming time-separability and additivity, the value function is defined as the combination of positive rewards and negative rewards associated with states or state transitions. A reward can be defined by a function R:S→**R** or R: S◊A→**R** or R: S◊S◊A→**R**. The first function means the reward is associated with a state. The second one means the award is associated with performing an action a in state s. The last one associates the reward with a state transition, in which performing an action a in state s results in state s'. For the reward associated with a state transition, it can either be counted as the reward for the present state s or for the next state s', but not both. A negative reward is also called a cost.

Since the value function has a range of real numbers, but the total reward, including positive and negative, may be infinite, only a few different mappings can be used to combine the rewards into single real numbers.

If the policy is evaluated over a finite number of time steps, the value is the total reward in this finite horizon. Denoting the sum of positive and negative rewards earned at time step t as r_t, the value function of a horizon of length T is:

$$V_\pi = \sum_{t=0}^{T} r_t$$

On the other hand, if the policy is evaluated over an infinite horizon, the value function can be defined as either a cumulative discounted reward or an average reward rate in the limit. The cumulative discounted reward with discount rate $0<\gamma<1$ is defined as follows:

$$V_\pi = \sum_{t=0}^{\infty} \gamma^t r_t$$

The average reward rate in the limit is defined as follows:

$$V_\pi = \lim (1/n) \sum_{t=0}^{n} r_t$$

For different problems, their objectives can be finding an optimal policy, which maximises the value function, or a satisfying policy, whose value function exceeds a given threshold.

2.2 Common planning problems

Various planning problems can be classified using this general MDP framework according to their different modelling assumptions. These are assumptions about the state space, events, observability, value function, objective, etc. These assumptions help to identify the structure of the planning system. The complexity of the policy construction is also determined by the corresponding problem's set of assumptions (discussed in section 2.3).

2.2.1 Fully Observable Markov Decision Processes

Fully Observable Markov Decision Processes (FOMDPs) assume fully observability. In this case, O = S, the set of observation is the set of system states. So the decision maker has full knowledge of the system current state and will know the new state resulting from performing an action. In FOMDPs, a policy is a function of the system state. Under Markov assumption, the present state contains all the system history information relevant to the future.

2.2.2 Partially Observable Markov Decision Processes

In contrast with FOMDPs, system states can not always be determined in Partially Observable Markov Decision Processes (POMDPs). In this case, the decision maker can observe its state of belief but does not know the system state exactly. An observation distribution U(s, o) represents the probability of being in a certain state given an observation. Also a complete POMDP includes a probability distribution over initial states I, which reflects the probability of being in any given initial state.

2.2.3 Deterministic planning

The classical AI planning models, which assumes deterministic actions, can be viewed as a special case of None Observable Markov Decision Processes (NOMDPs). Actions are deterministic means whereby Pr(S'=s'|A=a, S=s) is either 1 or 0. These models assume that the initial state is known and that the executions of actions are prefect. Since the decision maker knows where it is from the initial state and where it will be at after every action, no observability is need. The reward is determined by reaching a goal state. To encourage faster goal achievement, it can either discount the reward or assign a cost to each action.

2.2.4 Probabilistic planning

Extended from deterministic planning, a probabilistic planning problem without feedback has a probability distribution over initial states, stochastic actions, a set of goal states and a probability success threshold. The objective is to find a satisfying policy that reaches any goal state with probability larger than the threshold. If there is no feedback, then the policy can only be in the form of straight line plans and this is a restricted case of NOMDP problem. The problem becomes a restricted case of the general POMDP problem when the actions provide some feedback.

2.3 Comparing the complexity of FOMDP and POMDP

Since we mainly focus on FOMDP and POMDP for the hand-washing problem, computational complexity of solving these two kinds of model will be exam.

Generally MDPs (fully observable) can be solved in polynomial time in the size of the state space. Value iteration and policy iteration are the most common algorithms for solving MDPs. For infinite-horizon problems involving discounted reward, the optimal policy can be shown to be stationary [10].

On the other hand, POMDPs are famous for their computational difficulty. A POMDP can be viewed as “a MDP with an infinite state space consisting of probability distributions over S, each distribution representing the agent's state of belief at a point in time” [10]. It is exponentially hard to find an optimal policy for a POMDP with the objective of maximizing expected total reward or expected total discounted reward over a finite horizon.

2.4 Summary

MDPs provide a mathematical framework for modeling decision-theoretic planning problems. MDPs are attractive as a model for the handwashing problem because its ability of modelling stochastic events and trandoffs between short-term and long-term goals.

POMDP is a generalization of a MDP which allows one to model imprecise information about the current environment state uncertainty in the effect of actions, and conflicting objectives. Hence, as a formal framework for modeling the decision process, POMDP has advantages in accuracy and disadvantages in computational complexity.

Chapter 3
PRISM probabilistic model checker

PRISM is the tool we use for modelling planning systems for activities of daily living. A basic understanding of its theoretical foundations and functionality supported is important for our goal. The majority of the information presented in this chapter is sourced form [6] and [15].

3.1 Overview of PRISM

Probability is not only an important component in the design of algorithms, but it is also a tool for analysing system performance. Since probability can be used to model unreliable or unpredictable behaviour, probabilistic thinking plays an important role in many areas of artificial intelligence, including learning, planning and the evaluation of artificial agents. Probabilistic model checking is a formal verification technique, which is based on the construction of a precise mathematical model of the system which is to be analysed. Properties of the system are then expressed formally in probabilistic temporal logic and automatically analysed against the constructed model. PRISM is one of the stochastic model checking tools and has been used to analyse systems with a wide range of application domains, including communication and multimedia protocols, randomised distributed algorithms, security protocols, biological systems, and many others.

PRISM directly supports three types of probabilistic models, which are Markov decision processes (MDPs), discrete time Markov chains (DTMCs) and continuous time Markov chains (CTMCs). CTMCs are extensions of DTMCs, which allow transitions to occur in continuous time (see chapter 3 for MDPs and DTMCs).

PRISM first parses the model description, which is in PRISM modelling language, and constructs an internal representation of the probabilistic model. The property specification is then parsed and appropriate model checking algorithms can be performed. PRISM supports both qualitative and quantitative probabilistic model checking. For properties which include a probability bound, PRISM reports a true or false result to indicate whether or not each property is satisfied. PRISM can also return quantitative results for properties.

To represent all reachable state space and properties specific to the model, in its implementation, PRISM uses combinations of symbolic data structure such as Multi-

Terminal Binary Decision Diagrams (MTBDDs), which is an extension of Binary Decision Diagrams (BDDs), and conventional explicit storage schemes such as sparse matrices and arrays. These data structures provide space and time efficiency for the storage and analysis of probabilistic models, which often are large and structured, by exploiting their regularity. For more details on the symbolic and explicit approach, please see [17] [18] and PRISM's website.

For reachability analysis, conventional temporal logic model checking and qualitative probabilistic model checking, the underlying computation in PRISM involves graph-theoretical algorithms. For quantitative probabilistic model checking, numerical computation is also involved. When performing numerical computation, PRISM can use one of its three numerical engines. The MTBDD engine is implemented purely in BDDs/MTBDDs; the spare engine uses sparse matrices; and the hybrid engine uses a combination of the two. All of them perform the same calculations but performance in terms of time and space may vary. The MTBDD engine is effective for models with a lot of structure and regularity. The sparse engine outperforms when models are small but model checking takes a long time. The hybrid engine provides the best compromise between time and memory usage.

PRISM uses several iterative numerical methods for different types of property verifications. To solve linear equations, the numerical techniques PRISM supports include the Power method, Jacobi method, Gauss-Seidel method, Backwards Gauss-Seidel method, JOR method, SOR method, and the Backward SOR method. Value iteration is used to solve linear optimisation problems in the analysis of MDPs. For transient analysis of CTMCs, PRISM uses a method called uniformisation.

3.2 The PRISM modelling language

PRISM is a high level, state-based probabilistic modelling language, which is based on the Reactive Modules formalism in [19]. A model in PRISM language is constructed as the parallel composition of its modules, which can interact with each other. The definition of a module contains a number of finite range variables. These variables are called local variables, whose value can be read by all modules but can only be changed by their own module. The values of these local variables determine the state of the module. PRISM also supports global variables, whose values can be read and changed by all modules most of the time. The global state of the whole model is determined by the local state of all modules and values of global variables.

The behaviour of each module is described by commands, each of which comprise a guard and one or more updates:

$$[]\ g \rightarrow \lambda_1 : u_1 + \ldots + \lambda_n : u_n\ ;$$

The guard g is a predicate over all variables in the model. Each update u_i describes a

transition by giving the variables new values. Each expression λ_i is the probabilistic information assigned to a corresponding update. This can be a probability in the case of DTMCs or MDPs, or a rate in the case of CTMCs. If the guard is true, the updates are executed according to their probabilistic information.

PRISM also supports full parallel composition of modules, asynchronous parallel composition of modules and restricted parallel composition of modules. So, transitions can either be performed in an interleaved manner, or simultaneously. Synchronisation can be achieved by labelling commands with actions. In order to avoid synchronising commands of different modules by assigning different values to a global variable, commands labelled with actions are not allowed to update global variables.

Rewards are real values associated with certain states or transitions of the model. State rewards can be specified using multiple reward items, each of the form:

g : r;

The guard g is a predicate over all model variables and the reward r is an expression containing any variables and constants of the model. Transition rewards are represented in a similar form to the state reward, with action label:

[a] g : r;

Noted that [ ] g : r; is a transition reward rather than a state reward. It assigns a reward to all transitions in the model with no action label. If a state or a transition satisfies guards in multiple reward items in one reward structure, its total reward is the sum of the rewards for all these items. Multiple reward structures can be defined in a single PRISM model. A label can be assigned to each of them for the convenient of reward-based property checking. Without labeling, PRISM will use the first reward structure in the model file.

With the support of rewards, a wide range of properties can be specified and analysed in PRISM.

3.3 Property specifications in PCTL

Properties are specified using Probabilistic Computation Tree Logic (PCTL) for MDPs and DTMCs, and Continuous Stochastic Logic (CSL) for CTMCs.

PRISM can either give yes or no answers to assertions (properties with bound), or they can return the expected value of the properties. Different numerical algorithms are used to check different types of properties (see section 3.1). There are 3 principal operators for specifying properties: the P operator, which refers to the probability of an event occurring; the S operator, which refers to steady-state probability; and the R operator, which refers to the expected value of rewards.

The properties specified by the P operator can be of the form:

P bound [pathprop]

which means the probability that path property pathprop is satisfied by the paths from state s meets the bound bound; or of the form:

P query [pathprop]

for which a numerical rather than a Boolean value should return. For DTMCs or CTMCs, query is =?. For MDPs, query can be min=? or max=? because only the minimum or maximum probability of the path property being satisfied can be computed.

The types of path property which can be used inside the P operation include: X, for 'next'; U, for 'until'; F, for 'eventually'; and G for 'always'. The roles of these path property operators are illustrated through their basic syntax in Table 3.1.

Syntax	Meaning
X prop	true if prop is true in its second state
prop_1 U prop_2	true if prop_2 is true in some state of the path and prop_1 is true in all preceding states
prop_1 U time prop_2	true if prop_2 becomes true within time bound time and prop_1 is true in all states before that point
F prop	true if prop eventually becomes true at some point along the path
F time prop	true if prop becomes true within time bound time
G prop	true if prop remains true at all states along the path
G time prop	true if prop remains true in the time bound time

Table 1: Path properties for P operator

For the S operator, the property is of the form:

S bound [prop]

or S bound [prop]

It checks the long-term probability of being in a state which satisfies prop.

The R operator works in a similar fashion to the P and S operators, which is of the form:

R bound [rewardprop]

or R bound [rewardprop]

where bound can be <r, <=r, >r or >=r for an expression r evaluating to a non-negative double, and query is =? for DTMCs or CTMCs, and min=? or max=? for MDPs.

The four types of reward properties are illustrated in Table 2.

Examples of probability-based and reward-based properties specifications can be found in Chapter 5 and 6, where PRISM is used as the tool for analysing the models

for planning systems.

Syntax	Name	Meaning	Notes
F prop	reachability reward	reward cumulated along a path until a state satisfying property prop is reach	state rewards for the prop satisfying state reached are not included
C<=t	cumulative reward	reward cumulated along a path until t time units have elapsed	t is an integer for DTMCs and MDPs, a double for CTMCs
I=t	instantaneous reward	the reward in the state of a path when exactly t time units have elapsed	t is an integer for DTMCs and MDPs, a double for CTMCs
S	steady-state reward	the reward in the long run	currently only available to CTMCs

Table 2: Reward properties for R operator

3.4 Summary

This chapter provided a brief overview of PRISM, and especially of the features we used in modelling the hand-washing problem. In section 3.1 we introduce the stochastic models PRISM supported directly. Implementation details relevant to issues arose throughout our work, including numerical engines used for quantitative probabilistic model checking is presented. Section 3.2 shows how these models can be described in PRISM language. Section 3.3 explains what properties can be model checked and how they can be expressed.

Chapter 4
An MDP-based planning system for the hand-washing problem

The COACH system is a guidance system that assists people with dementia in the activity of hand-washing. Its planning system employs the MDP framework introduced in chapter 3 to make decisions for the user and remind the user what to do next in the task of hand-washing. The design of the planning system is subjected to the constraints of the hand-washing environment, for example the technologies available for capturing the progress in the hand-washing task, and the knowledge of the user's individual characteristics, for example, how people at different stages of dementia cope with the reminders.

Before introducing our model of the hand-washing problem in chapter 5, which uses the research done by the COACH team as the main reference, in this chapter, the research on the COACH system presented in [1] [2] and [3] is discussed in more detail.

4.1 Planning system design

Since the planning system of the hand-washing problem is a part of the assistance system which is formed by the sensing system, the planning system and the prompting system, the system design has to take input and output from the other two parts of the whole system into consideration. In the hand-washing problem, it is desirable but less possible for the system to observe or measure conditions of the user's hands, for example, whether they are dirty or clean, dry or wet, soapy or not. In order to capture the user's actions and progress in hand-washing, the use of computer vision is a logical choice. This sensing system uses computer vision to keep track of the user's hands and relevant objects in the task, such as the soap and the towel. In addition, a sensor to measure the water flow from the tap is also included in the system. Hence, the planning system makes decisions base on the information on the user's hand location relative to the relevant objects and the water flow.

The design of the prompting system is more flexible than the sensing system. In other words, there are many ways to provide the user with reminders of what to do next in the task. In the COACH system, six different task reminders (water on/off, use soap, wet hands, rinse hands, dry hands) at three levels of specificity (general, moderate,

specific) are used. These prompting reminders can be tailored for each user and for different activities. Different wording, volume and recorded voice can be used to achieve maximum responsiveness. Also, multimedia systems, such as video, can be deployed as a part of the assistive system to provide reminders.

As a key part of the task assistive system, the planning system models the hand-washing activity in a Markov decision process. As discussed in chapter 3, by adopting the MDP framework, different aspects of the hand-washing problem can be modelled in terms of states and state transitions, observations, actions, and rewards. Observations are limited by the information provided by the sensing system, and actions are the outputs of the prompting system. The MDP model for the hand-washing problem can be extended to a POMDP model, which can capture partial observability, for example the noise of input and hidden variables that describe the user's behaviour. The MDP model is capable of taking into account both uncertainty in the effects of its actions and tradeoffs between short-term and long-term goals. Extending to a POMDP model, it also has the ability to estimate the user's characteristic over time. Differences in three versions of the model for the hand-washing problem will be discussed in the next section.

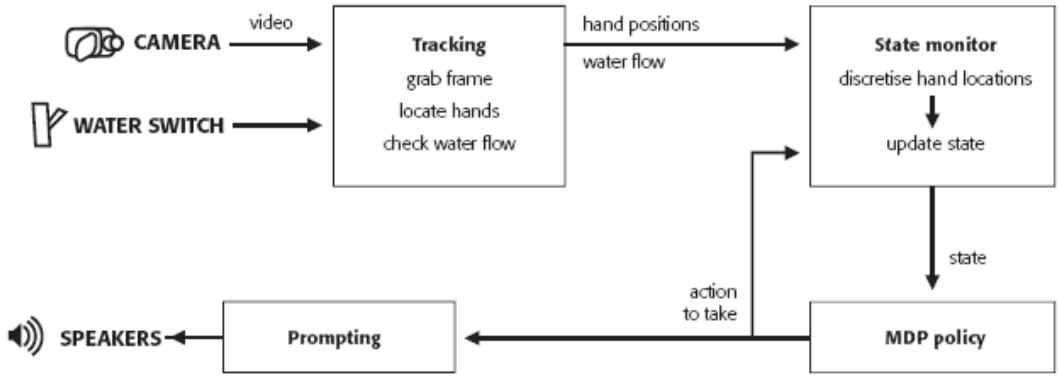

Figure 3: The COACH system [20]

4.2 System variables

To describe the hand-washing problem, the task is divided into several sub-tasks, which are called plan steps. Each plan step is a critical step in the hand-washing activity. In the MDP model described in [3] and the POMDP model described in [2], the plan steps capture the actions that the user has completed. The plan graph for these two models is shown in Figure 4. Each node in this plan steps graph is labelled as an action of the user. It is important to note that the action is only the label of a state and not the state itself. The action alone cannot represent a unique situation in the hand-washing activity. For example, plan steps E and D in figure 4 both appear as 'use

soap', but they are two different steps and represent different progressions the user has made in the hand-washing activity. It does not contradict the Markov assumption, which says that the states transition only depends on the current state, because plan step E and D represent different states even if they have the same label.

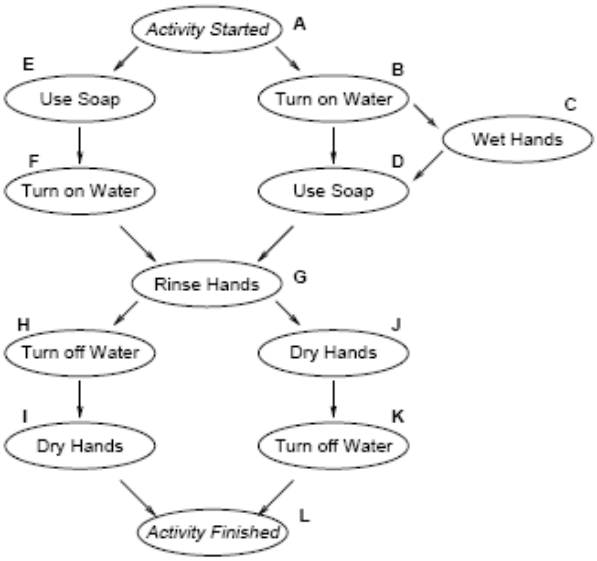

Figure 4: Plan graph 1 of COACH [2]

To solve this confusion, [1] and [8] present the same plan step graph in a different way by labelling each plan step node as the condition of the user. In the plan step graph from [8] (Figure 5), the arcs between nodes are labelled with the actions which cause the transitions between the nodes.

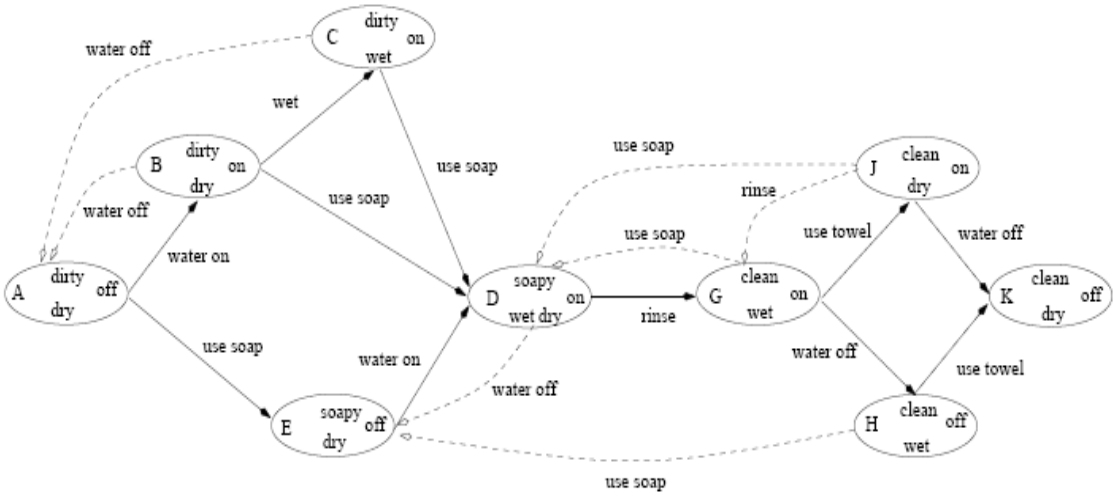

Figure 5: Plan graph of COACH [8]

As mentioned above, there are two key observables of the hand-washing problem. The variable used to describe the water flow during the hand-washing task is binary, representing the water flow from the tap off and on respectively. For the hand location, each value represents a possible region for the position of the user's, for example at sink, at soap, at towel, at taps or at water. In [1], the hand location variable is extended to capture the locations of both hands, for example, one at sink and one at tap, instead of assuming both hands are always at the same location. In the MDP model, perfect observability is assumed. On the other hand, the POMDP model assumes that there is 25% noise in the observation of function that detects hand location [2].

There are three kinds of actions available to the planning system according to [1], [2], and [3]. The system can give a prompt, call the caregiver, or do nothing at all. Presuming that the hand-washing task will be completed successfully with the caregiver's intervention, the action of calling the caregiver ends the planning system's attempt to assist the user to finish the task. There are 6 prompts which are reminders of a certain subtask, at 3 levels of specificity. It is assumed that the more specific the prompt is, the more likely the user will response to this prompt, but at the same time it is more likely the user will become annoyed less independent during the task. For this reason, each system action is associated with a cost, which is explained in section 4.4.

In order to give a full picture of the task, other variables are used to capture the status of the hand-washing activity, such as the maximum plan step completed, whether the current plan step is repeated, and the progress within the current plan step. Since the model has Markov property, which means that the future states are independent of the past states (given the present state), history variables are introduced into the model to provide a summary of the system history relevant to prediction of the user's behaviour. These history variables are the number of prompts issued during the current plan step, the type of the last prompt, the specificity level of the last prompt, and the number of time steps waited since the last prompt.

4.3 Modelling the user's behaviour

One notable difference between the MDP model in [2] [3] and the POMDP model in [3] [1] is the use of user variables, which reflect the aspects of the user's mental state which have an impact on their actions pattern in the hand-washing task. These factors in the user's individual characteristics include his or her overall level of dementia, the responsiveness to the prompts, and the awareness of how to do the task. These variables are not observable, however, the POMDP model can estimate a particular user's characteristic factors over time.

In the MDP model, without the system's ability to estimate the user's characteristic, the dynamic associated with the user's actions pattern were "manually specified by one of the authors using prior knowledge of the domain, gained through extensive observation of a professional caregiver guiding ten subjects with dementia through hand-washing" [3].

In the POMDP model, the dementia level variable "has some prior distribution set based on the population of users [1]". The model then estimates a particular user's level of dementia by watching his/her hand-washing behaviour over multiple sequences.

4.4 Costs and rewards

Both the MDP model and POMDP model have similar cost and reward structure. The design of the reward function is to "promote user independence, completion of overall task, and minimal regression by the user [3]".

A large reward is associated with the completion of the hand-washing task. Also smaller rewards are given for the completion of subtasks, in another word, reaching another maximum plan step for the first time. This design is to encourage progress even when the task is unlikely to be finished.

The cost for a prompt is proportional to its level of specificity. The more specific the prompt is, the more it costs. Otherwise, as already stated, giving prompts too often, especially specific prompts, could make the user feel less independent or annoyed. The costs for the three levels of prompts are 3, 5, and 7 according to [3] and 3, 5, 7, according to [2].

In these models, the cost for calling the caregiver is set high. Since the aim for the planning system is to help the user to finish the task as independently as possible, it is desirable to limit human intervention. Calling the caregiver is therefore penalised. However, if all else fails, this action is still an important option for the system. If the cost associated with calling the caregiver is set too high, the system can always get a higher total reward by doing nothing and will never call the caregiver even if the probability of finishing the task without the caregiver's intervention is very low.

According to [3], the values shown in Table 3 were determined in an iterative process. Through a series of simulated trials (see section 4.5), the reward function was successively altered until desired performance was attained. Also an ongoing research project of fine-tuning the reward function is reported in [2].

Action	*planStep*	*MPSrepeat*	Reward Value
None	-	-	0
Minimal detail level	-	-	-3
Moderate detail level	-	-	-5
Specific detail level	-	-	-7
Call caregiver	-	-	-1000
-	*A,B,C,D,E,F,G,H,J*	No	3
-	*I,K*	No	300
-	-	Yes	0

Table 3: Definition of reward function (sourced form [3])

4.5 Policies for the planning system

The research done in [1], [2], and [3] mainly focuses on constructing optimal policies for the MDP planning system and the POMDP planning system. The optimal policies aim to maximise the long term discounted expected reward, as defined in the above section. These solutions, which are usually of high computational complexity, are computed offline.

For all versions of the planning system for the hand-washing problem, the optimal policies were computed offline due to the computational complexity of both MDP model and POMDP model for a planning system of such large scale, as mentioned in Chapter 2.

According to [2], the MDP model has 25,090,560 states and the POMDP model has 50,181,120 states. The optimal policy for the MDP model was solved by the SPUDD algorithm and an ADD with 3,284 internal nodes and 18 leaves was used to represent the optimal policy for the MDP model. The optimal value function was an ADD with a total of 139,443 internal nodes and 106, 328 leaves. For the POMDP model, since its scale is beyond the reach of any exact solution techniques, the Perseus-ADD, a new point-based approximation solution technique based on the Perseus algorithm [12] approximation algorithm developed by the COACH team, is used to solve the POMDP only for a specific set of belief point.

Some other simple heuristic policies were also developed for comparison. These include the Nil policy, which never gives a prompt and the call-caregiver policy, which always calls the caregiver.

For the POMDP model, there are two more alternative policies which can be

compared with the optimal policy. One alternative policy is that has a fixed set of thresholds on the belief distribution and attempts to prompt when the user is not aware. The other one assesses the most likely state given the current belief, and then acts according to the optimal policy of the MDP model.

4.6 Evaluation

Although clinical trial is the most reliable method of evaluating the system, it can only be applied when the system has been proved safe. Survey and simulation are two alternatives. The simulation described in [2] and [3] used an actor to play the role of the user. The actor washed hands with the help of either human care giver or the prompting system. Than the simulation results were analysed by the research team. Furthermore, the process was recorded and a group of participants, who have experience in care giving, rated the efficiency of a series of simulated trials by watching video and answering questionnaires.

Satisfactory results from both simulation experiment and clinical trial are reported in [1]. However, simulation can only assess the system in a small amount of scenarios and the risk and cost associated with clinical trial are very high.

4.7 Summary

In this chapter, the MDP based planning system assisting people with dementia through hand-washing is discussed. Detailed descriptions of design of plan steps, modelling of the user's behaviour, specification of cost and reward, as well as computation of optimal policy are given from section 4.1 to section 4.5. As seen from the evaluation process discussed in section 4.6, the lack of an efficient tool to assess the planning system might be a bottle neck of the system development. In next chapter, the ability of a probabilistic model checker to model the hand-washing problem is investigated since probabilistic model checking is possible an efficient alternative to evaluate the planning system.

Chapter 5
Modelling the hand-washing problem with PRISM

The COACH's planning system discussed in Chapter 4 is modelled in a decision theoretic fashion as a MDP or a POMDP, both of which were introduced in Chapter 2. The planning system has been evaluated through simulation experiments and trials with actors. The preparation for clinical trials was also reported in [1]. However, these experiments/tests can only expose the planning system in a limited number of situations. In order to analyse the planning system in a more systematic way, to extend its exposure to a wider range of situations, and to test its performance for different user characteristics, we analysed the hand-washing problem with the methodology of probabilistic model checking (Chapter 3).

Based on the research of the COACH team in [1], [2], and [3], we modelled a simplified version of the hand-washing problem and its planning system with PRISM. Having already established the PRISM model for the hand-washing problem, model checking was applied to compare the performances of different policies for different user characteristics in different scenarios.

5.1 Modelling the hand-washing problem in PRISM

The simplified model of the hand-washing problem is based mainly on the work in [2], with most of the assumptions described in chapter 5. While the COACH's planning system is extended from MDP to POMDP with the ability to model the non-observability of individual user characteristics, we first modelled the interaction of the planning system and the user in a MDP framework because PRISM does not directly support POMDP.

In the MDP version of the COACH planning system [3], the user characteristic is specified manually and all users at different mental stages are assumed to have the same behaviour in the hand-washing task. In the POMDP version, a particular user's characteristic is assumed unchanged and can be estimated over time. However, this assumption might not hold because the user characteristic can vary over time even for the same user. If the characteristic for the same user changes, either because the user is in different mode or different health condition, the POMDP planning system potentially loses its advantage over the MDP planning system.

Although we are not able to model the hand-washing problem in POMDP with PRISM, we still want to analyse the hand-washing problem, in particular the planning system, for different user characteristics. Moreover, since we want to relax the assumption that the characteristic for a particular user is unchanged over time, MDP is more attractive as a model than POMDP because of the latter's computational complexity. In order to capture different user characteristics in our hand-washing model, a few parameters are defined to represent the user's awareness and responsiveness (section 5.1.2). In the COACH's MDP model, these parameters are fixed and in the POMDP model, these are hidden variables which can be estimated. In our MDP hand-washing model, these parameters can be either constants or variables, so the planning system has different abilities in matching its policy to different user characteristics, which are analysed in section 5.3.2.

Furthermore, some policies are specified as simple heuristic functions for the hand-washing problem. With a specified policy, non-deterministic is removed from the MDP model and it becomes a DTMC model. We built some DTMC models with different heuristic policies for the following reasons:

First, unlike the optimal policy computed for the MDP model, the heuristic policy does not take all variables of the model into account. In the DTMC model, the planning system makes decisions solely on the variables included in the heuristic function so that the variables used to capture the user characteristics are irrelevant to the decisions made. This answers for the assumption that the user characteristics are unobservable.

Secondly, there are few problems with the policy of the MDP model. Obviously it is difficult to compare the details of the policies of the MDP models with different parameters because the state space of these models is large. Also it is not easy to examine the policy of one MDP model for another MDP model which describes the same hand-washing problem but with different user characteristics. Comparatively, with the DTMC models, we can compare the same policy's performance when different users' characteristics are specified. Another problem is that since PRISM does currently not support discounted reward, in order to compute the discounted award, the number of time steps has to be defined as a variable explicitly (see section 5.1.7). Consequently, the MDP model is not stationary (see Chapter 2), which is how we usually assume it to be.

Last but not least, MDP solutions focus on expected values. Without knowing the exact user's characteristics, either optimal solution for a certain characteristic can be generated, or, assuming all types of characteristic are modelled, the solution which has the highest expected performance for all types of user characteristic can be computed using the MDP model. However, good average performance does not necessarily mean satisfactory performance for all cases. It is possible that it will

perform well when some characteristics are seen, but poorly for some other user characteristics. In the DTMC model, heuristic policy can be specified as a randomised algorithm in order to achieve satisfactory performance for any user characteristics.

5.1.1 Structure of the PRISM model

We modelled the hand-washing problem in PRISM as a model containing seven modules. The states' transitions are specified in seven phases and each module represents one phase. The reasons for using seven phases to represent one signal time step are as follows:

First of all, the state space of the hand-washing problem is large and it is easier to model different parts of the problem in several modules. PRISM language is a high level language; the system dynamic can be specified structurally as the interaction between modules. Different parts of the hand-washing problem, for example, the decision maker, the user's behaviour, and the hand-washing environment, should be modelled in several modules.

In each time step, system prompts affect the user's action and the user's action determine the plan step. Refer to actions and exogenous events in chapter 2, where the combined transition is difficult to specify, even though it is possible. Dividing one single time step into phases simplifies the model by assuming they occur in isolation. This arrangement simplifies the interaction of events which seem to occur simultaneously.

Also, each time step has to be divided into several phases because synchronisation cannot be used to control the updates from different parts of the hand-washing problem taking place at the same time. This is due to the fact that global variables cannot be changed in synchronised commands and modules cannot change the local variables of other modules. Variables like the waited time can be changed by a user moving his or her hands, or the system giving prompts. Since hand movement and system prompts are modelled as separate modules, the waiting time can only be a global variable and can only be modified in different modules in order. These system dynamics make enforcing transitions specified in different modules by synchronisation impossible at times.

Finally, using the 7-phase structure can simplify the model by eliminating some system history variables (section 5.1.5). At each time step the whole system is updated in a few phases; the phase that occurs earlier can access the old value of some variables which are updated in the same time step but in the later phase. For example, one module needs to know the last prompt; putting this module in the phase before issuing the new prompt, this module can assess the information on the last prompt

more easily. So, with the help of the 7-phase structure, the model contains fewer system history variables and is thus simplified.

Phase 0 models the decision-making process of the planning system. Phase 1-4 models the observation of the user's action. Other system variables, including the history variables are updated in Phase 5 and 6. For each single time step, the system goes through some or all of these 6 phases.

Phase 0 is the module for the policy of the planning system. Phase 1 models the effect of the given prompt on the transition of hand location. If there is no prompt given, then hand location is updated in phase 2. Similarly, if hand location is 'at tap', water flow change is modelled in phase 3 or 4, depending on whether a related prompt is given. Plan step and number of time steps waited are determined in phase 5. In phase 6, MPS are updated according to the plan step and current MPS.

5.1.2 Plan step vs. hand location and water flow

The user's action is captured and represented by two variables, which are the hand location variable and the water flow variable. The transition of the plan steps of the hand-washing problem is caused by the user's action. Hence, the plan step transition is a function of both the user's hand location and the water flow. Figure 6 shows the plan steps graph of the simplified hand-washing model, which is a modified version of Figure 5, with more arrows indicating the plan steps transition.

Dividing the plan steps into 6 levels, MPS indicates the maximum plan step reached. The higher the level is, the closer it is to finishing the task. Plan step 0 is at MPS 0, which is the lowest level and indicates the beginning plan step for the task. Plan steps 1, 2, 4 are at level 1, which means their MPS equals 1. Plan step 3 has MPS 2 and plan step 5 has MPS 3. With MPS equal to 4, plan steps 6 and 7 are the closest plan steps to the final step, which is plan step 8 with MPS 5.

For each plan step, the neighbour step with a shorter distance to the final step is a correct next step. In other words, one plan step might have more than one correct next step. Reaching a correct next step from the current plan step is known as making progress in the task. Any plan step transitions other than staying in the same plan step or making progress are called regressions.

Making progress differs from an increment of MPS in two ways. Transition between plan steps from the same MPS level can also be a progression. For example, moving from plan step 1 to plan step 2 is making progress, but they are in the same MPS level. Furthermore, progress can be made repeatedly but a MPS increment can be

made only once in one hand-washing event. For example, if in a hand-washing event, the plan step changes from 0 to 1 and then 1 to 0 repeatedly, it is making progress every time the plan step changes from 0 to 1, but the MPS becomes 1 after the first transition and never increases again. The design of MPS is to reward reaching a plan step which is closer to the finishing step for the first time, but not to reward making the same type of progress repeatedly. More details on the reward will be given in section 5.1.5.

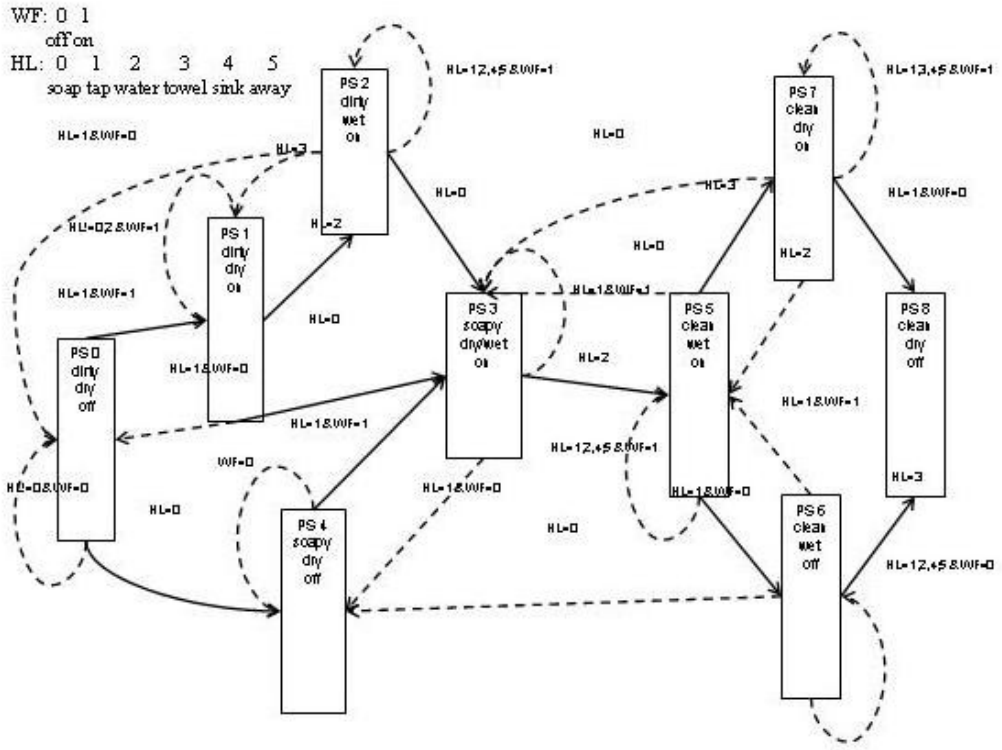

Figure 6: Plan steps transition diagram

5.1.3 System action

In our simplified model, there are 7 actions available. The system actions include doing nothing or giving one of the 6 prompts. The different prompts representing 6 movements are: doing nothing, turning the water on, turning the water off, using the soap, rinsing hands, wetting hands, and drying hands. How the user reacts to the system prompts depends on the current plan step, the last action of the user, the user's awareness, and responsiveness. We assume that the system prompt can increase the probability of the user doing the action mentioned in the prompt in the next time step.

Unlike the COACH system, the simplified model does not include the action of "call

caregiver". Calling the caregiver should only occur when the planning system decides to stop trying to finish the task without human intervention. The simplified model does not need to include this termination action because a cost/reward threshold can be used to determine whether the task should be terminated.

5.1.4 Dynamic of hand location and water flow

The variable of the hand location has 6 possible values represented by 6 regions, namely: away, at sink, at soap, at towel, at tap, and at water. In the model, it is assumed that in each time step, the user's hand can move from one region to any of other 5 regions or stay in the same one. Also, it is assumed that the water flow can only be changed when the user's hand location is at the tap.

The user characteristic is represented by the parameters of user awareness and responsiveness, which reflect on the dynamic of the hand location and water flow. When the planning system doesn't give any prompt, the hand location and the water flow transitions only depend on the current plan step and the user's awareness. When the system gives a prompt, the hand location and the water flow transitions depend on the current plan step, the prompt, the user's awareness and the user's responsiveness. The parameters for user's awareness and responsiveness include HLCC, the hand-location-correct coefficient, HLSC, the hand-location-same coefficient, WFCC, the water-flow-correct coefficient, HLE, the hand-location effect, and WFE, the water-flow effect. HLE and WFE indicate the responsiveness of the user, which reflect on the hand location dynamic and water flow dynamic respectively when a correct prompt is given. HLSC is not related to awareness or responsiveness directly, but it helps to specify a certain type of user behaviour.

In this simplified model, system actions that do not make sense are eliminated. In other words, the planning system should never give incorrect prompts. For each plan step, any action that makes progress is a correct action. Similarly, any hand location or water flow that shows signs of making progress is a correct hand location or correct water flow for the current plan step. Since 'at sink' is the position that hand movements most frequently cross, this location is generally considered as a correct hand location for most of plan steps. Other correct hand locations for each plan step are labelled in red in Figure 7.

First consider the case that no prompt is given. The total probability that the user hand location at the next time step will be correct is equal to the HLCC. For example, if there is only one correct hand location in the current plan step, then HLCC is the probability of the user moving his or her hands to this correct region. If there are n correct hand locations, the user is assumed to move hands to each correct hand location with probability HLCC/n. If the user's current hand location is not a correct

hand location, then with probability HLSC, the hand location will remain the same, which means the user's hands don't move. So, the more aware the user is, the higher the parameter HLCC is. In the case that the user is not aware and doesn't do the right thing, the higher the parameter HLSC, the less hand movements the user makes.

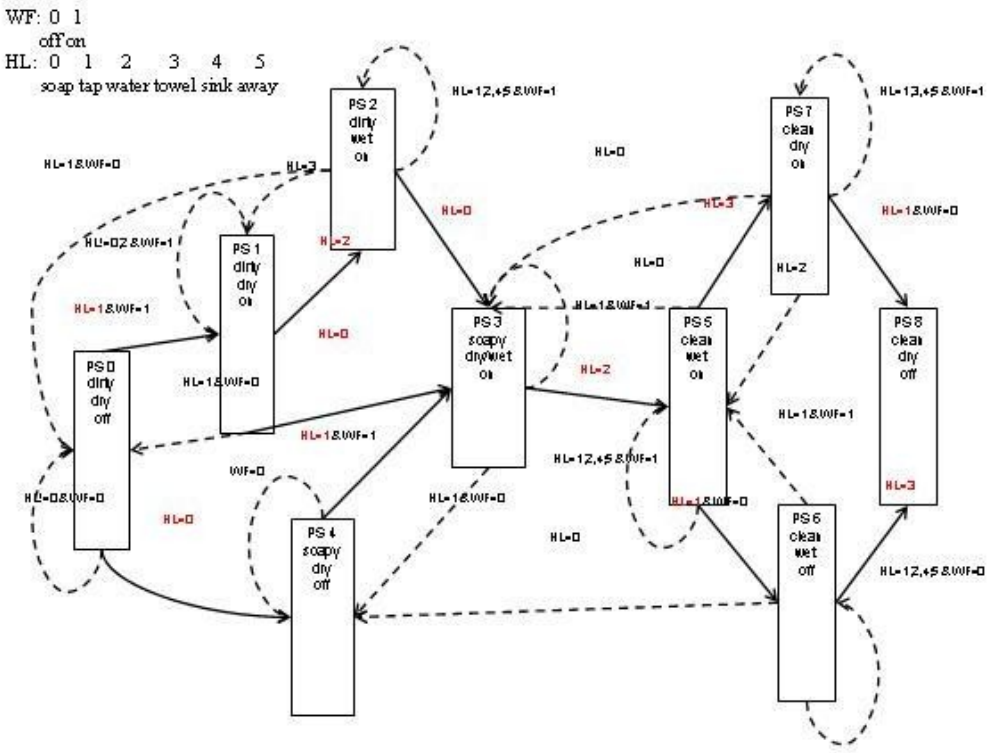

Figure 7: Correct hand locations (other than 'HL=4') for each plan step

One basic assumption is that the planning system's action of giving a prompt only has an effect on the user's action in the same time step. In other words, a prompt only affects the hand location and water flow of the next time step; the transition in hand location or water flow does not depend on the prompt given previously.

When a correct prompt is given, since one hand location corresponds to one user action, from all correct hand locations for that plan step, there should be only one correct hand location associated with the given prompt. For example, 'at water' is the only correct hand location when the prompt 'rinse hands' is given. HLE is the probability that the correct hand location of the prompt will be observed in the next time step. In other words, HLE is the probability that the user responds to the prompt correctly. The more responsive the user is, the higher the HLE is.

If the user is not responsive to the prompt, then there is still total probability HLCC that other correct hand location will be observed. Each of the correct hand locations

other than the prompted one is equally likely to be observed. Also, wrong hand locations are equally likely to be observed with total probability (1-HLE)*(1-HLCC).

Similar to the definition of correct hand locations, correct water flow for the current plan step is the water flow of any neighbour plan steps closer to the finishing plan step. For example, 'water on' for plan step 4 is the only correct water flow because its only arrow pointing forward points to plan step 3, in which the water flow is on (refer to Figure 6 or Figure 7). So the water flow of the current plan step is irrelevant to the correct water flow, which only depends on the water flow of the next correct step.

If the hand location is 'at tap', then it is possible that water flow can be changed. Without a prompt, if only one of 'water on' and 'water off' is the correct water flow, the correct one will be observed with probability WFCC. If both 'water on' and 'water off' are correct, then they are equally likely to be observed.

Given that hands are already on the tap, the prompts other than water on/off don't affect the water flow dynamic. But other prompts at the first place affect the probability of the hand location to be observed as 'at tap'.

Since we want to simplify the planning system by eliminating system actions that do not make sense, only correct prompts will be given. So the system can give the prompt of 'water on' only when the current water flow is off and water should be on for the correct next step. Similarly, the prompt of 'water off' can only be given when the current water flow is on and the water should be off for the correct next step.

When a correct prompt about water flow is given, with probability WFE*WFCC, the water flow will be changed into the correct water flow in next time step. WFE is a coefficient equal to or larger than 1, as the prompt should never have a negative impact, which reduces the probability of correct water flow change. The maximum value of WFE is limited by WFCC, as WFE*WFCC should never be greater than 1.

In the DTMC model, the awareness and responsiveness parameters are specified as constants and they have no effect on the decisions made by the planning system. In the MDP model, when policy is generated for particular user characteristics, these parameters are defined as constant. When policy is generated for a group of user characteristic models, these parameters can be defined as variables. The distribution of values for each of these variables is estimated as the probability of the occurrence of each user characteristic.

It would be better if the awareness could be represented in one single parameter, but for the convenience of model construction, it is separated into HLCC and WFCC. So, when we check the properties for different user characteristics, it does not make sense to have values for HLCC and WFCC which have contradictory meanings in terms of awareness, for example a small value for HLCC but a large value for WFCC. This

also applies to responsiveness, which is separated into HLE and WFE. Since model checking will only apply to the values of HLCC, WFCC, HLE and WFE which make sense, 5 types of user characteristic are defined. These are users with low-awareness-low-responsiveness, high-awareness-low-responsiveness, medium-awareness-and-responsiveness, low-awareness-high-responsiveness, and high-awareness-high-responsiveness. For more accurate and comprehensive model checking, it is possible to define more types to achieve a better coverage of different situations.

5.1.5 History variables

One of the most important system history variables in this simplified model is the amount of time waited. Modified from its definition in [8], the amount of time waited in this model is defined as the count of successive time steps in which the user does not make any progress. This is the case when no hand movement is detected, or a regression in terms of plan step transition is observed (do you mean 'not observed'?). The amount of time waited will be reset when the user makes a progress or the planning system gives a prompt. If a correct hand location is observed but the plan step is unchanged, it is also progress so the amount of time waited will be reset as well.

Here we assume that the pattern of user's behaviour is only defined by awareness and responsiveness. In other words, a small value of responsiveness means the user does not follow instruction very well or that the user responds to instruction very slowly. The same goes for the awareness measurement. Under these assumptions, the variable of amount of time waited can capture the user's behaviour for the planning system. The longer the time waited, the more likely the user is not aware or not responsive.

With the inclusion of a variable for in amount of time waited capturing whether a progress is made in each time step, the progress variable in [8] does not need to be included in this simplified model.

5.1.6 Policies

The system's action, in the form of a prompt, changes the probability of reaching a certain plan step by influencing the user's action. Since the user's actions are observed as changes in hand location and water flow, the impact of the system's prompts on the user's actions are observed as their effect on the transition of the hand location and the water flow. One basic assumption of this prompting system is that giving a correct prompt can increase the probability of the user to do the right thing.

A policy is a strategy that decides which prompt the system should give at each time step. Since it is the case that for each plan step, not all possible user's actions lead to a correct next plan step, not all possible prompts make sense. In fact, for each plan step there is at least one correct user's action for the correct next plan step (represented by a solid arrow in the plan step graph, Figure 7). So, for each plan step, there is at least one correct prompt that can increase the probability of making progress.

In the MDP model, the policy module represents a group of non-deterministic choices on what the planning system should do. At each time step, the planning system can choose from all correct prompts or do nothing. The non-deterministic choices are resolved in the DTMC model, in which the policy module specifies a strategy for each situation observed, including the planning system's actions of giving some correct prompts or doing nothing.

When a quantitative property of a MDP model is checked by PRISM, the policy which maximises or minimises the value of this property is generated during the model checking. This optimal policy is the decision made during the path of generating the maximum reward. However due to the large state space, it is difficult to analyse each of these policies in detail.

On the other hand, heuristic policies are developed and DTMC models are built. A policy can be a function of any system variables except the awareness and responsiveness parameters because they are assumed to be unobservable. The heuristic policies are specified with the system variables that reflect the user's characteristics most directly.

The number of time steps waited is an important signal for the user's awareness and responsiveness. The heuristic policy structure uses the maximum wait, 'maxwait', as a threshold on when to give a prompt. If no prompt has been given and the hand location has not changed for a number of time steps, the planning system will give a correct prompt with probability 'p_p'. The larger the 'p_p' is and the smaller the 'maxwait' is, the more aggressive the planning system isThe meaning of different values for 'p_p' and 'maxwait' are shown in Table 4.

p_p=0	never give prompts
maxwait=0, p_p=1	always give prompts
maxwait>0, p_p=1	always give prompts when threshold is met
maxwait>0, 0<p_p<1	give prompts w/ prob p_p when threshold is met, if threshold is not met, never give prompts

Table 4: Parameters in heuristic policy

Compared with the MDP model, or the POMDP model without the ability to estimate

the user characteristic because of the relaxation of the assumption on consistence of a particular user's characteristic, whose optimal policy focuses on expected values, the heuristic policy can achieve satisfactory performance for all types of user characteristic. This is shown in Figure 8.

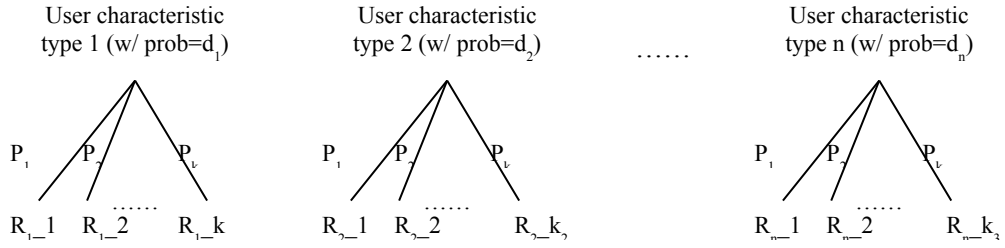

Figure 8: Policies and expected rewards for different types of user characteristic

In Figure 8, P_x, x=1,2,…k denotes all possible determinant policies. Each P_x is a series of choices of the system's actions, which does not involve probability distribution. In each MDP model for a user characteristic of type m∈{1,2,…n}, the optimal policy $P_{x(m)}$, x(m) ∈{1,2,…k} is the determinant policy whose expected reward $R_m\_x(m)$ is the highest among all possible determinant policies:
$R_m\_x(m) = \max R_m\_x$ for any x=1,2,…k.
However, since the user characteristic type m is unobservable to the planning system, we may wrongfully employ the optimal policy for type m1 to type m2. In this case, $R_{m2}\_x(m1) <= R_{m2}\_x(m2)$, because $R_{m2}\_x(m2)=\max R_{m2}\_x$ for any x = 1, 2…k. In fact, it is possible that the expected reward we get by applying the optimal policy for type m1 to type m2, which is $R_{m2}\_x(m1)$, will be very small.

In order to find the optimal policy for all user types, the probability of encountering each user type p_m is estimated. We denote the optimal policy for all user type as P_o, o∈{1,2,…k}. Then,
$\sum^{m=1,2,\ldots n} R_m\_o \Diamond p_m = \max \sum^{m=1,2,\ldots n} R_m\_i \Diamond p_m$ for all i=1, 2…k.
Although $P_{x(o)}$ is the policy with the highest expected reward, it is possible that its performance for a certain type of user characteristic is poor, because there is no guarantee of the expected reward for this particular type of user characteristic $R_m\_x(o) \Diamond p_m$.

For the purpose of securing a satisfactory expected reward, randomised algorithms can be employed in defining the heuristic policy. This is shown by a simple example in Figure 9.

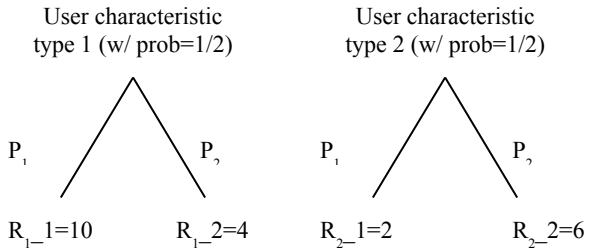

Figure 9: An simple example on randomized heuristic policy

In the case shown in Figure 9, if we know that the user characteristic type is 1 or 2, the optimal policy is P_1 or P_2 respectively. For the average case, since
$R_1\_1◊p1+ R_2\_1◊p2=10◊1/2+2◊1/2=6$, and
$R_1\_2◊p1+ R_2\_2◊p2=4◊1/2+6◊1/2=5$, the optimal policy is P_1.
Using this optimal policy, if the user type is 2, then the expected reward is 2.
If we want to maximise the minimum expected reward, then policy P_2 should be adopted. The minimum expected reward is 4, when the user characteristic is of type 1. The expected reward for all cases is then 5. To balance the goal of achieving high expected reward and high minimum expected reward, a randomized policy can be used. For example, if we choose P_1 with probability 1/2 and P_2 with probability of 1/2, then the expected reward in the case of type 1 and 2 are 7 and 4 respectively. So the minimum expected reward is still 4. However, the average expected reward is 5.5, which is higher than 5.

In modelling the hand-washing problem, it is impractical to find out all possible policies. So, the randomized policies we use here use a simple heuristic structure, and the performances of each of them are evaluated with the help of probabilistic model checking in section 5.3.1.

5.1.7 Cost and reward

Since the PRISM model does not support a negative reward, two different cost and reward structures are used in this simplified hand-washing model. The first one separates positive reward and negative reward, which means the cost. By having two rewards representing the positive value and negative value, the similar reward structure as in [2] can be implemented. However, the total reward can be represented by the sum of its positive part and negative part only in DTMC models. Without non-deterministic choices, expected value can be calculated (refer to chapter 2). For MDP, because of the existence of non-deterministic choices, the reward is calculated as its maximum value or minimum value. The maximum value for positive reward might

not be obtained from the same path when the negative reward is at its minimum value. So it does not make sense to add up maximum positive reward and minimum negative reward as the maximum total reward. In order to evaluate and find the optimal policy of the MDP model, the second reward structure assigns a single value to represent the total reward.

For the first reward structure, since the simplified model has only seven-system action, of which the 6 prompts are at one specificity level, the average cost for the three levels of prompts in [3] is used as the cost of each prompt (refer to section 5.4). For the second one, since no 'cost' can be computed, the reward structure modified from [2] defines the cost of giving prompts differently. Instead of penalising the planning system for giving a prompt by assigning a negative reward, a positive reward is given for each time the planning system decides not to give any prompt. In this case, the expected reward is higher but it also represents the idea of addressing both long-term and short-term objectives. Both of these reward structures assign small rewards for making progresses and a big reward for finishing the task.

All of these rewards are discounted rewards (refer to Chapter 2). A discount rate between 0 and 1 is used to calculate the cumulated rewards. The discount rate represents the cost in time.

$$\text{reward} = \text{discount}^{\text{time\_step}}$$

Each reward is weighted by the discount rate to the power of the number of time step when the reward is assigned to a state or a transition. For a larger discount rate the long-term rewards are preferred to the short-term rewards and vice versa. The reward structures specified in PRISM language can be found in Appendix I and II.

Since all variables in the PRISM models are defined with a range, a maximum value for the variable "time_step" has to be defined. This is the cap value which the count for total time step cannot exceed. Since the time axis has to be limited to finite time, and the time_step stops increasing when the cap value is reached, using a small cap value might in fact create errors in calculating the discounted cumulated rewards.

5.2 Simulation and model checking for debug

Using the PRISM simulator, sample paths through a model can be generated. This tool is very useful for debugging models. The paths can be generated either by manual exploration or automatically. Figure 10 and Figure 11 show screenshots of using the simulator of the GUI version of PRISM.

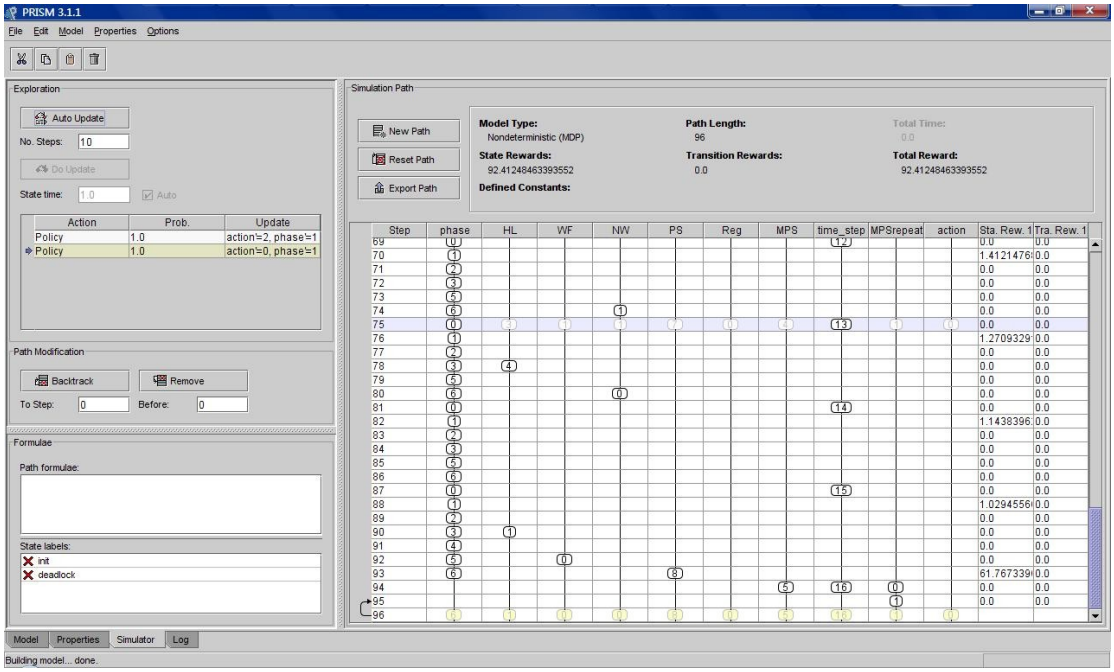

Figure 10: Simulation of the MDP model for the hand-washing problem with GUI version of PRISM

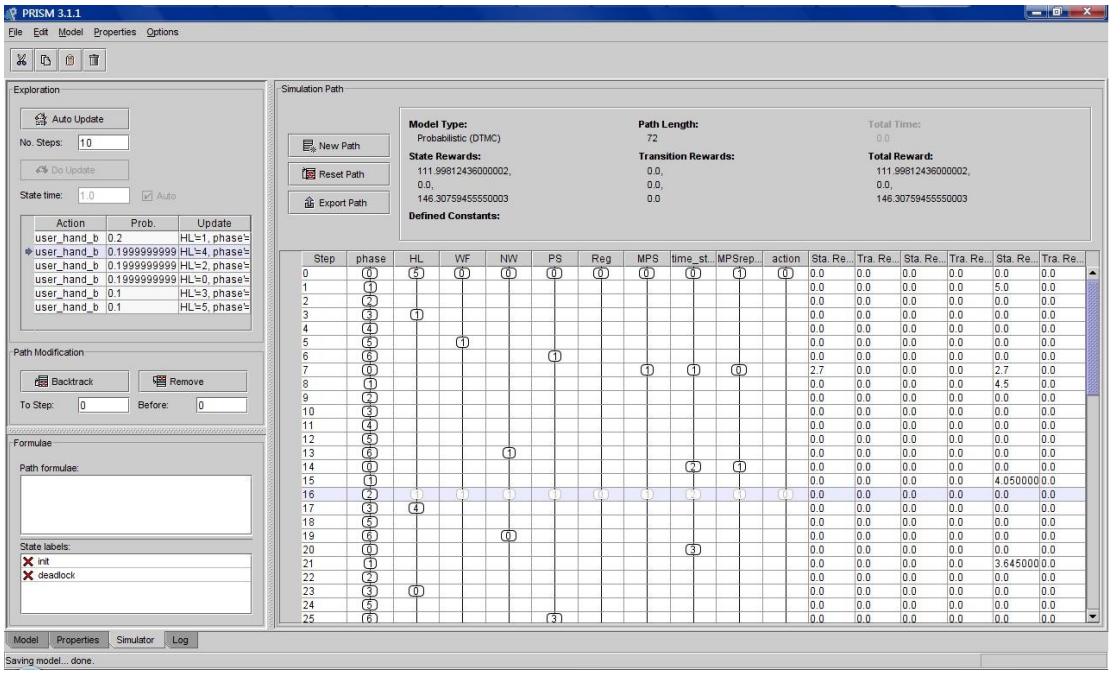

Figure 11: Simulation of the DTMC model for the hand-washing problem with GUI version of PRISM

To investigate the correctness of the models, properties are checked against the underlying assumptions of the problem modelled. Checking properties of the DTMC model in which the planning system will never give a prompt, the dynamics in the hand-washing problem without the influence of actions of the planning system can be analysed.

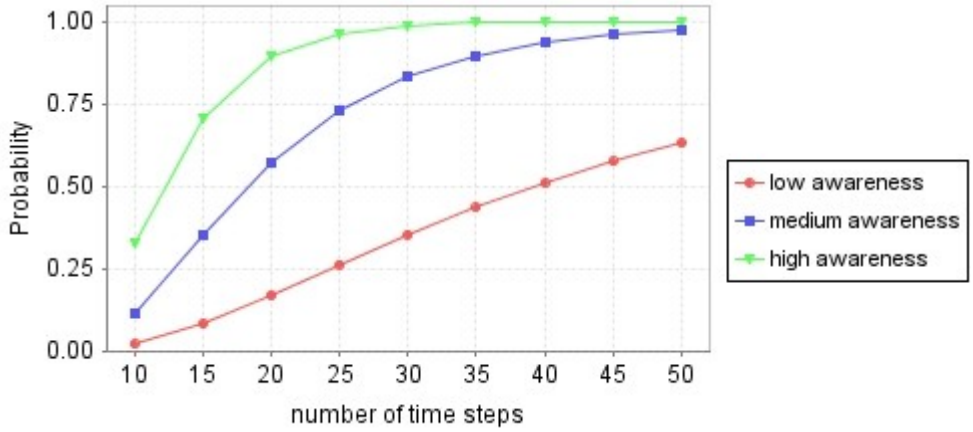

Figure 12: Probability of finishing the hand-washing task within a period of time for different user characteristic

Figure 12 shows the probability that the task can be finished within a number of time steps. This property expressed in PRISM language is:
P=? [time_step < total_time U MPS = 5]
For different user characteristics, this probability varies. Within a certain number of time steps, the user with a characteristic of higher awareness is more likely to finish the task without the help of the planning system.

Also, the expected number of time steps to finish the task can be checked. To specify this property in PRISM, a reward of time step has been defined in the model. Since the model uses seven sub-steps to model one-time steps, which are defined as phases, the number of time steps can be counted as:
rewards "time"
phase = 0 : 1;
endrewards
and the property to check is: R{“time”}=? [F MPS=5].

For analyzing the model in which no reminder will be prompted, the expected time step of finishing the task can be used instead of the cost and reward defined in section 5.1.7. Without any cost associated with giving a prompt, the discounted reward can only reflect whether the task has been finished and how fast it has been finished. Thus, checking the probability of finishing the task with a certain number of time steps and the expected number of time steps to finish the task can be used as a straightforward method to analyse these properties. However, this does not apply to the models when system prompts and their associated costs are involved.

The expected numbers of time steps to finish the task are 49.6, 19.9, and 13.6 for user awareness at the low level, medium level, and high level respectively.

Furthermore, a path can be expressed in a property specification with the U operator (see section 3.3). Although this specification can be miscellaneous, the probability of

the occurrence of any specific scenario can be checked. For example, if we want to check the probability that the user's hand location is 'at sink' at the first time step, and is away at the second and third time step, the property's expression in PRISM is:
P =? [(((time_step=0)| ((time_step=1)&(HL=4))) | ((time_step=2) & ((phase<3)|(HL=5)))) | (time_step>2) U ((time_step=3)& (HL=5))]

5.3 Experiments

We are mainly interested in checking the discounted rewards for different heuristic policies of the DTMC models and the optimal policies of the MDP models for the hand-washing problem. The discounted reward should be a fair measurement to evaluate the performance of each policy. For simplification, instead of evaluating the policies for every single user characteristic modelled with our assumptions, 5 typical users' characteristics are considered. Type 1 represents characteristic of low awareness and low responsiveness; type 2 means high awareness and low responsiveness; type 3 means medium awareness and responsiveness; type 4 means low awareness and high responsiveness; and finally, type 5 means high awareness and high responsiveness. For details of the user characteristic, please see section 5.1.4.

Since our models are not very large and model checking takes a long time, the spare engine (see Chapter 3, section 3.1) is used. The difference in time and memory usage between the sparse engine and the hybrid engine is significant here.

5.3.1 MDP model and optimal policy

In order to generate an optimal policy for a MDP model, the second reward structure defined in section 5.1.7 is used. The optimal policy is the one that maximizes the expected value of this cumulated discounted reward. The first reward structure is not available for the MDP model since the maximum positive reward and minimum negative reward may come from different paths (section 5.1.7). The optimal policy is generated by PRISM when the property regarding the maximum value of this reward is checked: R{"total_2"} max =? [F MPS=5].

By checking the minimum value of the expected reward, the worst case can be analysed. This is specified in PRISM language as:
R{"total_2"} min =? [F MPS=5].

The results of the property checking under different assumptions of user characteristics are shown in Table 5. Also, note that the maximum expected reward and the minimum expected reward of the MDP model are the upper bound and lower

bound for expected rewards associated with any policy, either determinate or randomized.

Expected reward:	max	min
Low awareness low responsiveness	189.134	136.813
High awareness low responsiveness	255.119	209.439
Medium awareness medium responsiveness	230.671	192.333
Low awareness high responsiveness	230.770	142.690
High awareness high responsiveness	248.152	217.548
Characteristic unknown	224.781	179.766

Table 5: Maximum and minimum expected rewards for different user characteristic

When the characteristic type is unknown, it is assumed that these 5 typical characteristic types have covered all the cases and each of them is equally likely to occur. With this additional assumption, the optimal policy of the MDP model for the unknown characteristic can be generated.

Under the basic assumption of the planning system's ability of assisting people, no matter what policy is used, it should not have a negative impact on the hand-washing activity. So the minimum probability of finishing the task within a finite number of time steps is checked. This property specified in PRISM language is:
P min = ? [time_step < total_time U MPS=5]
Figure 13 shows the result. Comparing Figure 13 with Figure 12 (section 5.2), the difference is unnoticeable. One implication is that the model does not violate the assumption of no negative impact. The other implication is that the policy that has the minimum probability of finishing the task with a particular number of time steps might be the policy of doing nothing.

However, generally it is not easy to investigate what exactly a particular policy does because the state space is large even in version 3.2 common line based PRISM, the transition matrix generated from the MDP model regarding the property checked can be exported. Also the MDP model is not stationary because the number of time steps is defined as a variable for the purpose to computing discounted reward. In order to eliminate the effect of the variable of time step on the policy and compare different policies in a more systematic way, the DTMC model with heuristic policies are studied in next section.

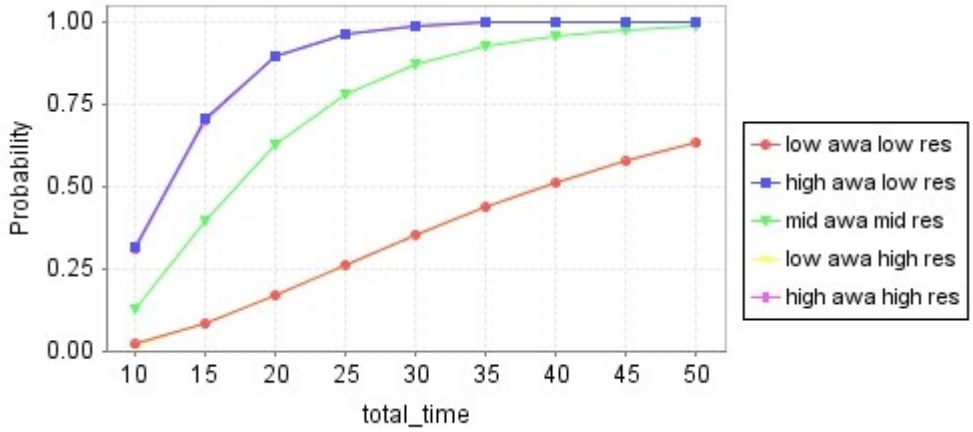

Figure 13: Minimum probability of finishing the hand-washing task within a period of time

5.3.1 DTMC model with heuristic policy

In this section, properties of the DTMC model which implement the simple heuristic policy structure are checked. Changing the parameter 'maxwait' and 'p_p' of policy structure, different policies can be generated. The policy with smaller value of 'maxwait' and larger number of 'p_p' is more aggressive in terms of more likely to give a prompt at each time step. For details of the heuristic policy structure, see section 5.1.6.

5.3.1.1 Significant factors

Before investigating the policies, first of all, the reward structure, which is the mechanism of evaluating their performance, is analysed in this part of the experiment. Two different reward structures are used, one is directly adopted from [2], which is in the form of the total of one positive part and one negative part, and the other one is a modification to represent the total reward in one single value (see section 5.1.7). If these two rewards are consistent, then for a particular user type, when a policy has a higher first reward than another policy, its second reward should also be higher; similarly, for a particular reward, if its first reward for one type of user is higher than another type, its second reward should be higher as well. However, this consistency does not hold for all scenarios, which will be shown in this part of the experiment. Although the focus is on the rewards for finishing the task, there are some other factors which can affect the result of the evaluation. These factors include the discount rate used in computing discounted reward and the cap value for counting time steps.

The reward-based properties checked in this section include:
R{"positive_1"}=? [F MPS=5]

R{“negative_1”}=? [F MPS=5]
R{“total_2”}=? [F MPS=5]
The first reward property is asking for the reward ‘positive_1’, which is the positive part of the first reward structure, cumulated until finishing the hand-washing task. The second reward property is about the negative part of the first reward structure and the third reward property is about the total reward defined by the second reward structure.

In order to compare the difference in rewards when different numbers are used as the cap values for the total time steps, the expected rewards when different policies are used, with different cap value of the total time steps, are shown in Figure 14, Figure 15, Figure 16, Figure 17 and Figure 18. Each of these figures shows the case for a different user characteristic type. The second reward structure (see section 5.1.7) is used and the discount rate equals 0.95. The heuristic policies have a fixed value of p_p at 1 and different policies are generated by varying the value of maxwait. These straightforward policies are at different levels of aggressiveness: the smaller the value of maxwait is, the more often the planning system gives a prompt (see section 5.1.6).

The larger the cap value is, the more accurate the discounted reward function in section 5.1.7 is. From Figure 14 - 18 we can see that, for every user characteristic type, the reward for each policy decreases as the cap value for total time increases. Since the rewards obtained after the total time reaches its maximum value are not being discounted further, a small cap value results in a larger cumulated discounted reward. Also, the function of cumulated discounted reward converges for a larger cap value of time. The differences in the reward curves for max time=50 and max time=100 are almost undifferentiable. The maximum differences between rewards obtained when max time = 50 and max time = 100 are 5.70063%, 0.00015%, 0.05901%, 5.63519% and 0.00015% for these 5 user characteristic types respectively. If the cap value is too low, the result might not be accurate since the cumulated discounted rewards calculated in this case might give a wrong idea of which policy performs better. For example, in the case of user characteristic type of low awareness high responsiveness (Figure 17), when the max time is 50 or 100, the expected rewards decrease as the maxwait of the policies increases. In other words, policies with lower maxwait have higher expected rewards than policies with higher maxwait. However, this is not true when the max time is 10, 20 or 30, where policies with higher maxwait might have higher expected rewards. Throughout the following experiments, the total_time of 50 is used.

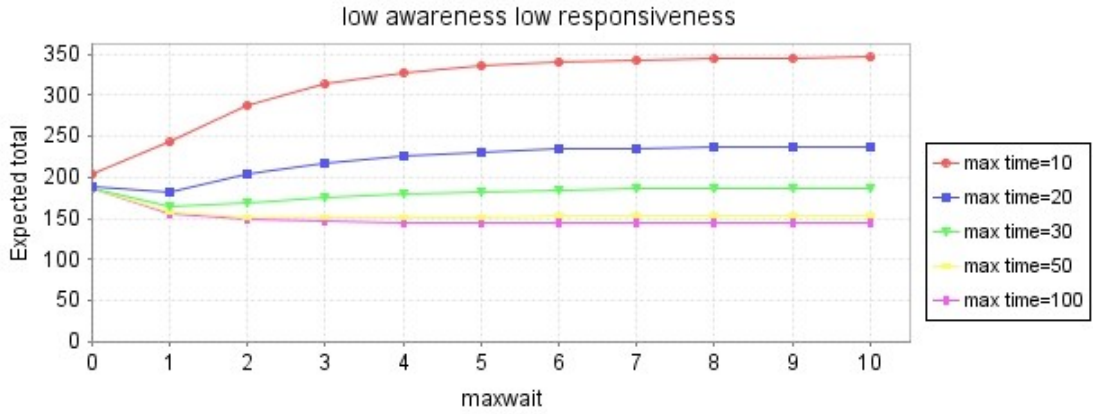

Figure 14: Reward vs. policy, with different cap values of total time (user characteristic type: low awareness low responsiveness)

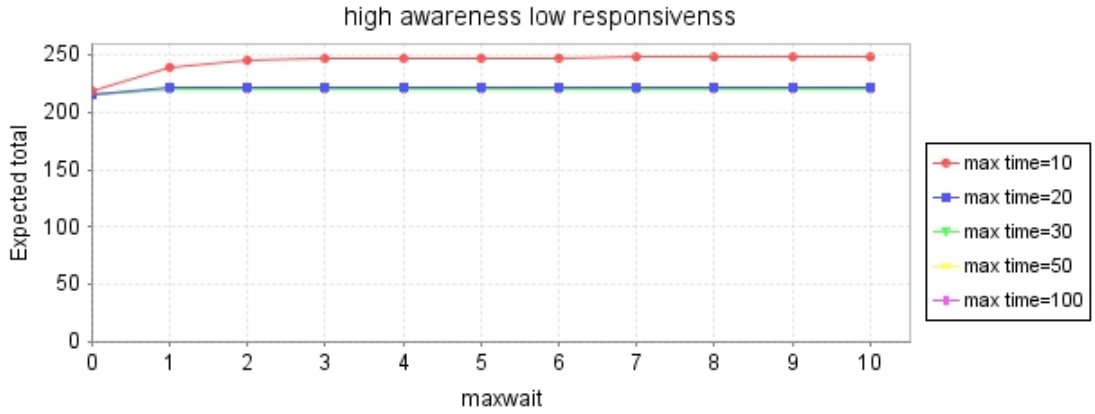

Figure 15: Reward vs. policy, with different cap values of total time (user characteristic type: high awareness low responsiveness)

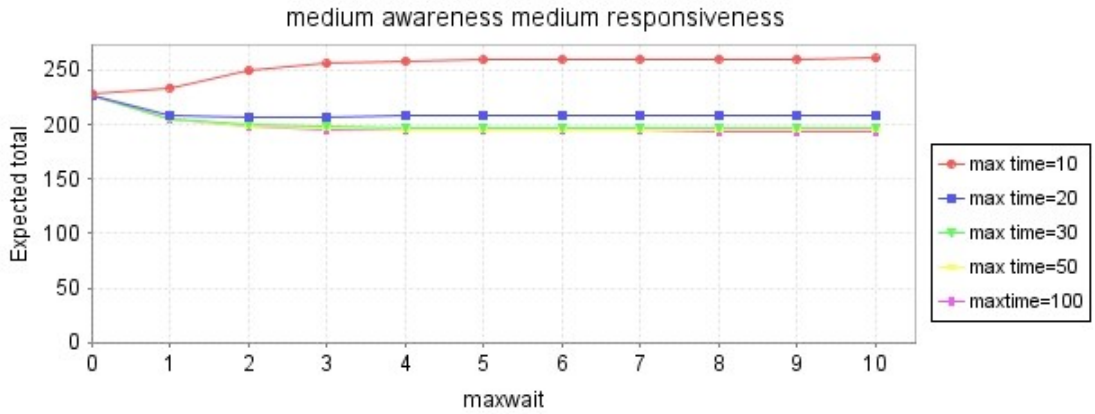

Figure 16: Reward vs. policy, with different cap values of total time (user characteristic type: medium awareness medium responsiveness)

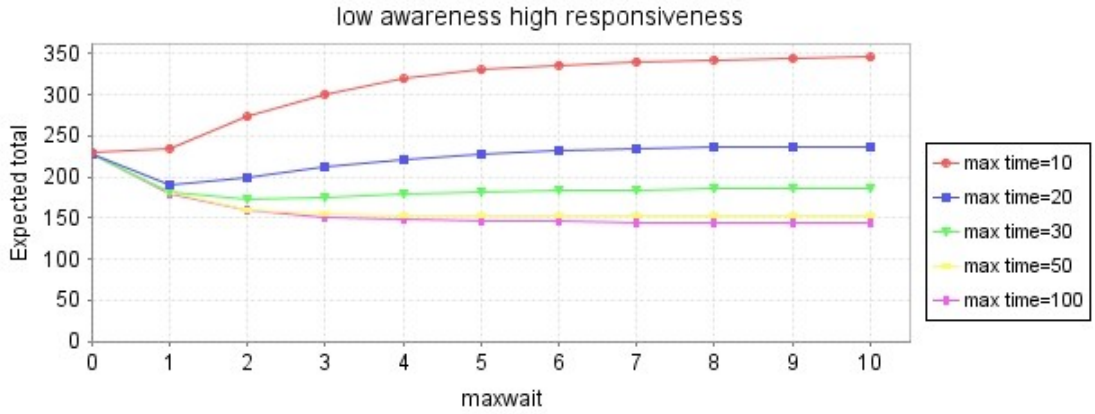

Figure 17: Reward vs. policy, with different cap values of total time (user characteristic type: low awareness high responsiveness)

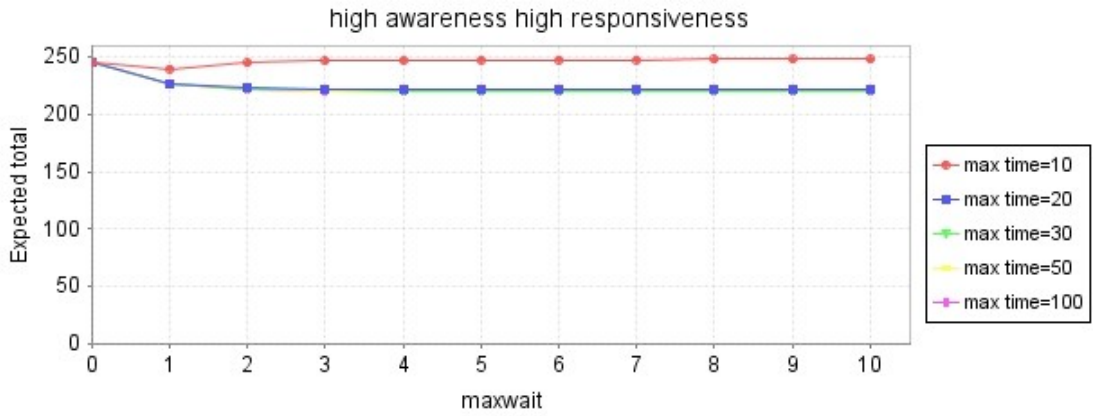

Figure 18: Reward vs. policy, with different cap values of total time (user characteristic type: high awareness high responsiveness)

Not only the cap value of the total time, but also the discount rate used in calculating the cumulated rewards affects the model checking results. Figure 19-23 show the rewards defined by the second reward structure when different values are assigned to the discount rate. Again, different policies with different values of "maxwait" but fixed value 1 of "p_p" are used and the cases of different user characteristic types are considered.

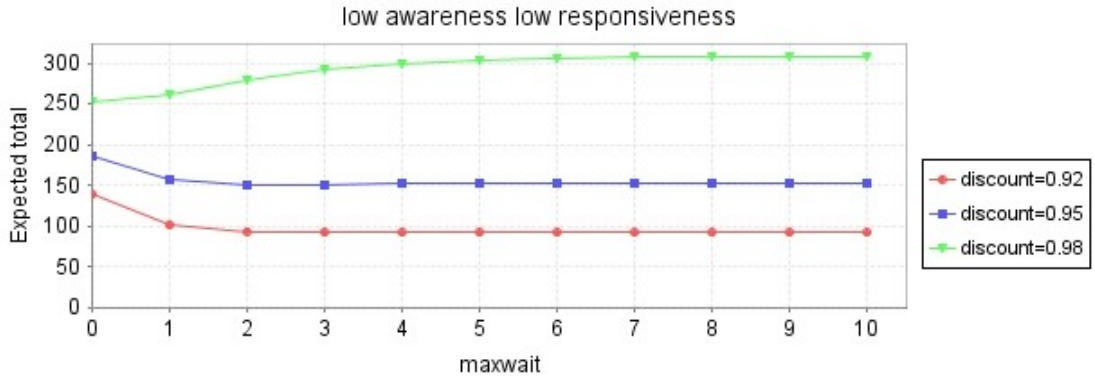

Figure 19: Reward vs. policy, with different discount rates (user characteristic type: low awareness low responsiveness)

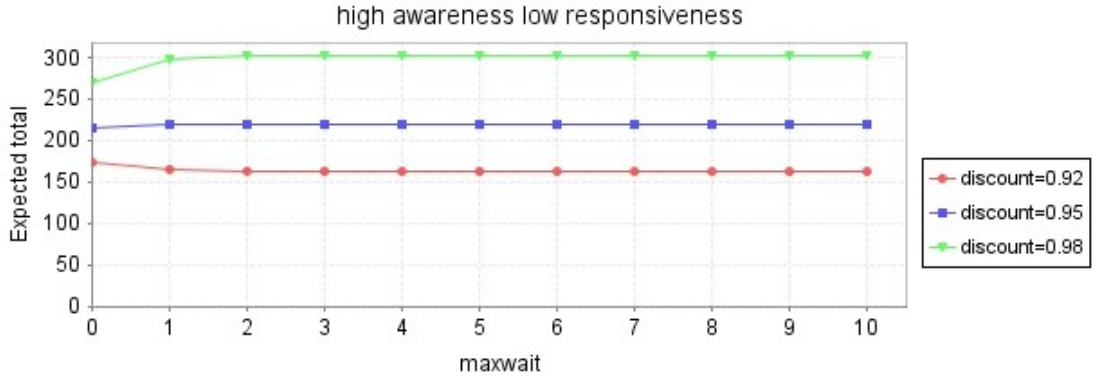

Figure 20: Reward vs. policy, with different discount rates (user characteristic type: high awareness low responsiveness)

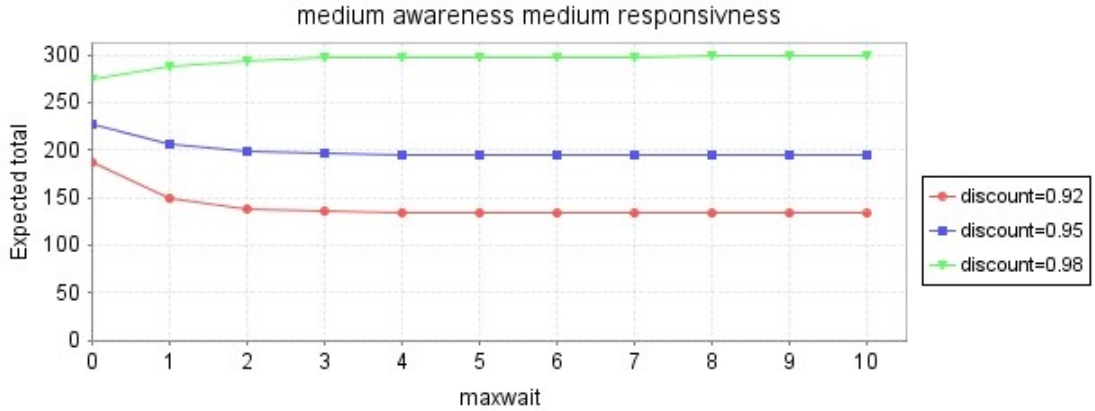

Figure 21: Reward vs. policy, with different discount rates (user characteristic type: medium awareness medium responsiveness)

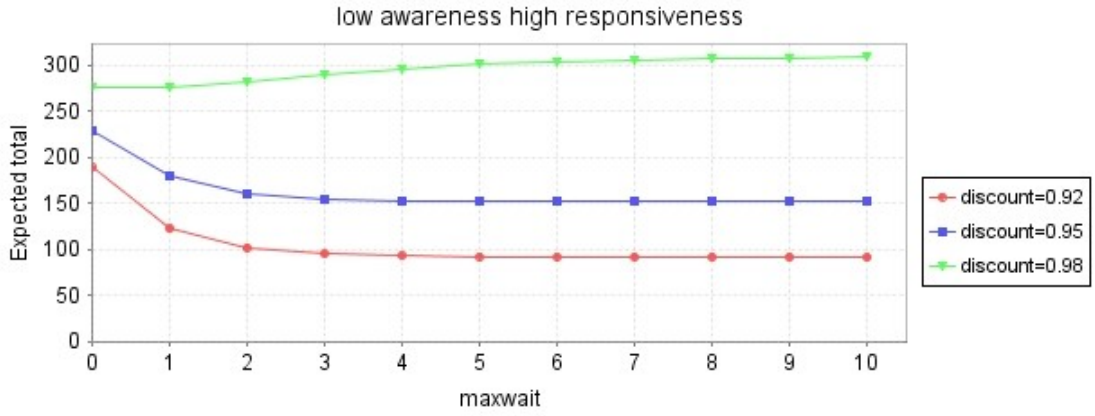

Figure 22: Reward vs. policy, with different discount rates (user characteristic type: low awareness high responsiveness)

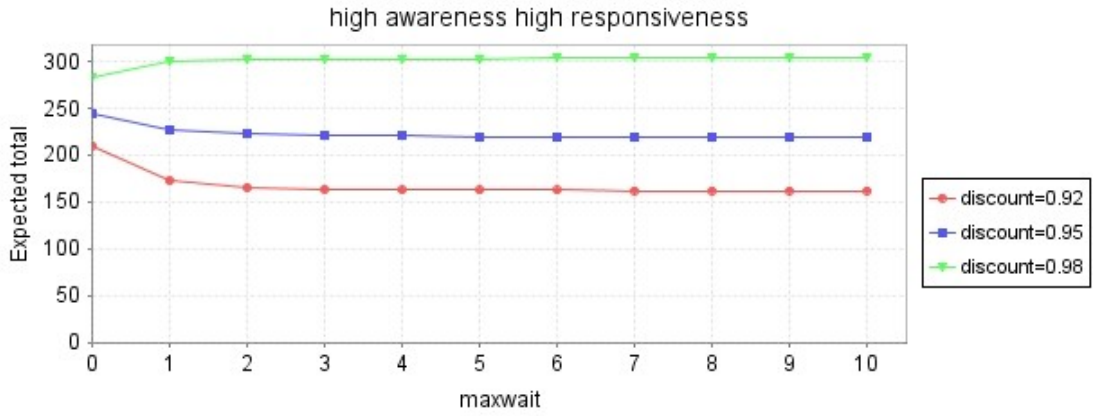

Figure 23: Reward vs. policy, with different discount rates (user characteristic type: high awareness high responsiveness)

According to the formula of the discounted reward in section 5.1.7, the discounted reward is a monofonic function of the discount rate. So the cumulated discounted reward, which is the sum of all discounted rewards along the path, is also a monofonic function of the discount rate. In other words, the larger the discount rate is, the larger the total reward is. From Figure 19 to Figure 23 we can see that the levels of the reward curves increase as the discount rate increases.

Moreover, using different discount rates can change the preference in different policies for a particular type of user. For example, for user characteristic type of high awareness high responsiveness (Figure 23), when the discount rate is 0.92 or 0.95, policy with maxwait=0 is the best policy among the 11 policies. However, when the discount rate is 0.98, this policy has the lowest expected cumulated discounted reward. This is because different values of the discount rate encourage different tradeoffs between short-term and long-term goals. When the discount rate is small, the short-term goal is more important because the reward obtained from achieving the long-term goal would be low. So when the discount rate is lower, the most aggressive

policy of all these 11 policies, which give a correct prompt at every time step, is preferred. Using the more aggressive policy, it is more likely that rewards can be obtained in a shorter period of time. By contrast, the least aggressive policy is preferred for the larger discount rate because time is less important and the planning system can afford to wait for a longer time in the hope of the user making progress without a prompt, which can reduce the cost of giving a prompt.
Fixing the discount value to be 0.95, the two reward structures are compared through a series of experiments, whose results are shown in Figure 24-28. The reward "total_1", which is defined by the first reward structure, is the difference of the positive reward and the negative reward. The reward "total_2" is defined by the second reward structure, which, instead of penalising the actions of giving a prompt by the assignment of costs, a positive reward is given when no reminder is prompted. So, reward "total_2" is larger than reward "total_1' in all cases. Ideally, if reward "total_2" equals to "total_1" plus an offset, these two reward structures are consistent.

However, from Figure 25, we can see that the above is not true. For the case of high awareness low responsiveness user characteristic, reward 'total_1' claims policy with maxwait=0 is prefer to policy with maxwait=1, while reward 'total_2' shows the performance of policy with maxwait=1 is better. Contradictory results indicate that the two reward structures handle the tradeoffs between costs differently. For the second reward structure, since more positive rewards are assigned to sub goals, the final goal, which is finishing the task, is weighed less important than that in the first reward structure. As a consequence, instead of giving a prompt at each time step, less aggressive policies are preferred. By comparing Figure 27 with Figure 25, it is shown that if a reward of 200 instead of 300 is assigned to the final goal in the first reward structure, the results from two reward structures become consistent.

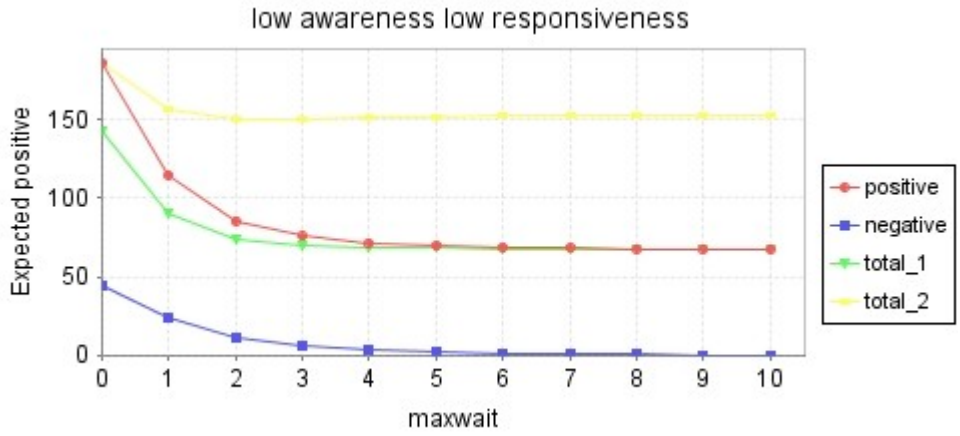

Figure 24: Reward vs. policy, with different reward structures (user characteristic type: low awareness low responsiveness)

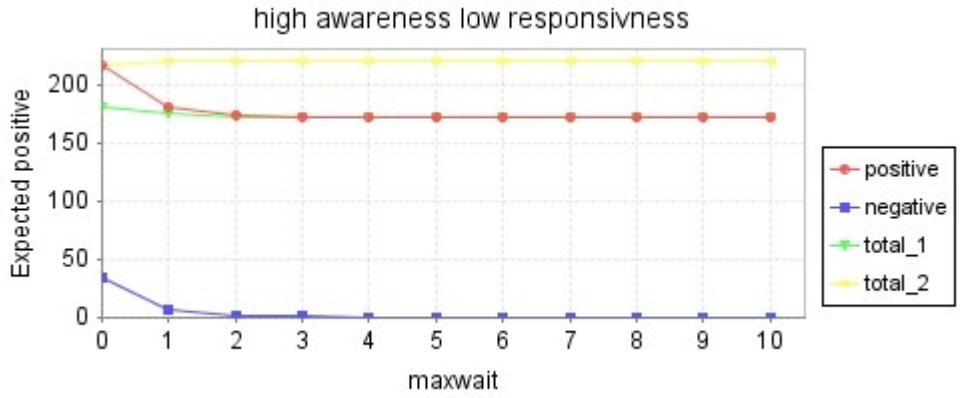

Figure 25: Reward vs. policy, with different reward structures (user characteristic type: high awareness low responsiveness)

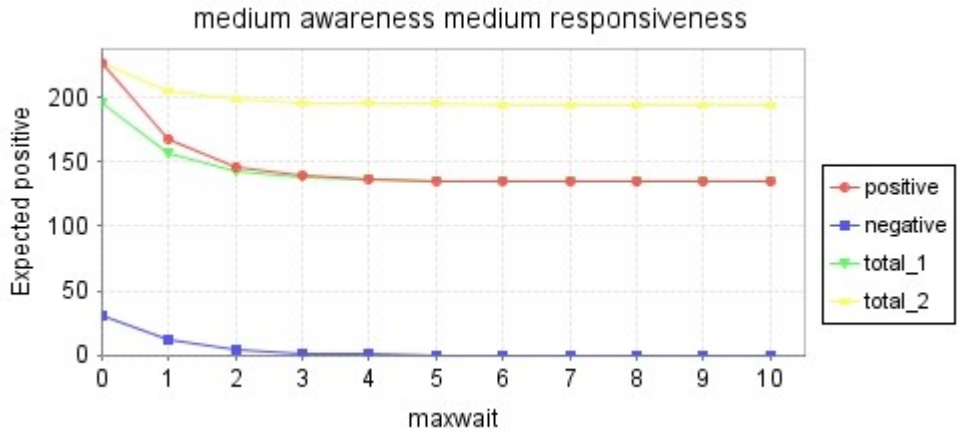

Figure 26: Reward vs. policy, with different reward structures (user characteristic type: medium awareness medium responsiveness)

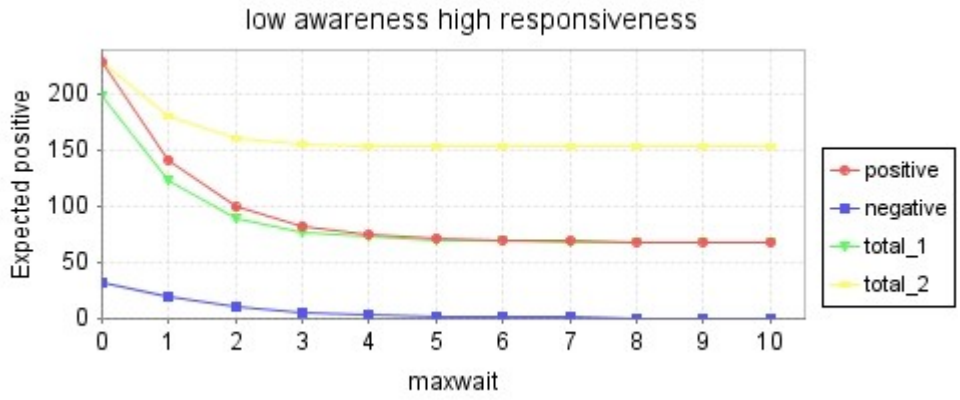

Figure 27: Reward vs. policy, with different reward structures (user characteristic type: low awareness high responsiveness)

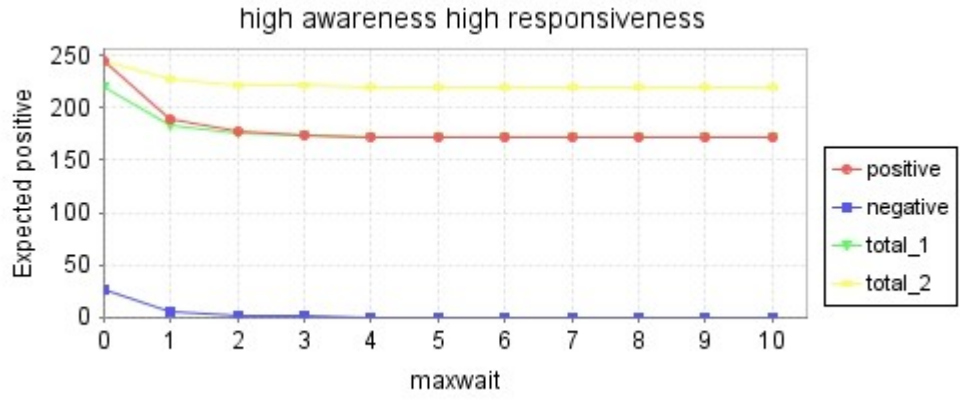

Figure 28: Reward vs. policy, with different reward structures (user characteristic type: high awareness high responsiveness)

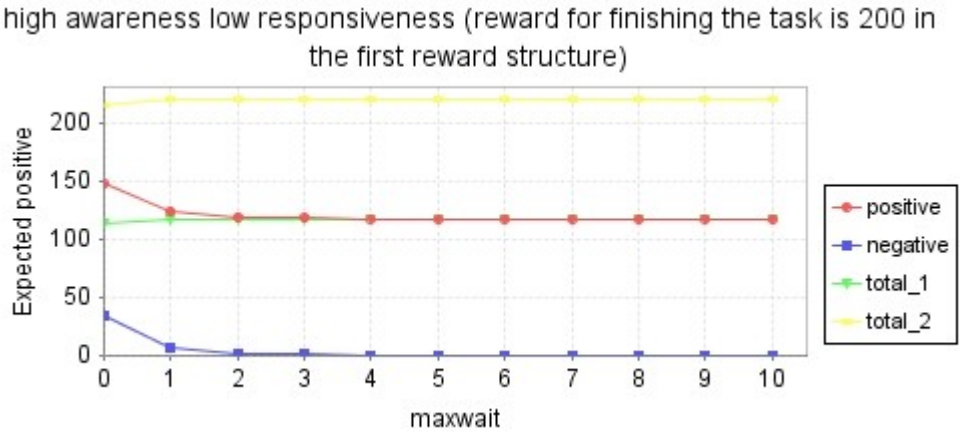

Figure 29: Reward vs. policy, comparing the two reward structures (user characteristic type: high awareness low responsiveness)

5.3.1.2 Performance analysis

In this section, the performances of different policies for different user characteristic types are evaluated. With the same setting of the system factors, different user types may prefer different policies. The discount rate is assigned to be 0.95, the cap value of total time 50 is used throughout the remaining experiments.

First, we look at two types of user characteristic with the most interesting properties: high awareness/low responsiveness and low awareness/high responsiveness. Aggressive policies which prompt the user all the time might not suit the type of high awareness/low responsiveness because the probability that the user can make progress without a prompt is high and the probability that the user responsiveness to a prompt is low. On the other hand, users with low awareness/high responsiveness might need system prompts more often. This is shown is Figure 30 and Figure 31, in which the expected rewards when policies with different values of maxwait and p_p are plotted

for these two user characteristic types. Here, the second reward structure is used.

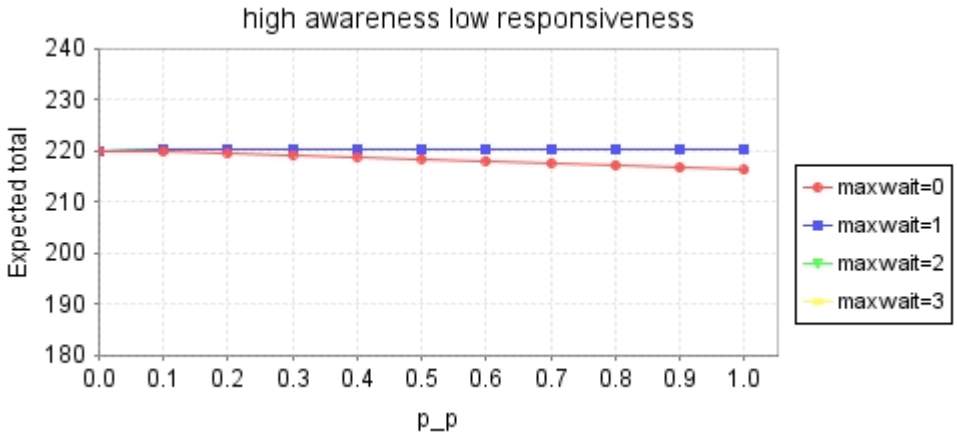

Figure 30: Reward vs. policy, for high awareness low responsiveness user characteristic

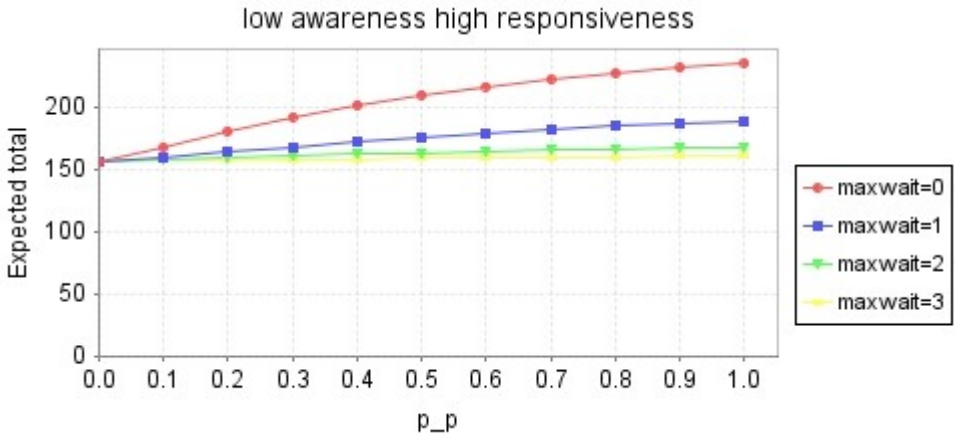

Figure 31: Reward vs. policy, for low awareness high responsiveness user characteristic

In Figure 30, for high awareness low responsiveness characteristic type, the maximum expected reward, 220.43, is achieved when the policy with maxwait=1 and p_p=0.9 is employed. In Figure 31, for low awareness/high responsiveness characteristic type, the maximum expected reward, 216.27, is achieved when the policy with maxwait=0 and p_p=1 is employed. This result is consistent with the argument that more responsive and less aware users need more prompts.

5.4 Discussion

In this chapter, the modelling of the hand-washing problem in PRISM and the model checking are explained in details. Prior knowledge of the hand-washing problem, which is mainly learnt from the research done by the COACH team, the additional

assumptions on the hand-washing problem made for simplification, and details of our simplified MDP and DTMC models for the hand-washing problem are discussed in section 5.1. In section 5.2, using simulation and property checking for debugging the models was discussed. Then, in section 5.3, the MDP model for the hand-washing problem and the DTMC models with heuristic policies were analysed with more in-depth probabilistic property checking.

Although it was shown in section 5.2 and 5.3 that the MDP and DTMC models are able to capture the essence of the hand-washing problem, the lack of data and prior knowledge in the domain of assisting people with dementia limits overall accuracy and comprehensiveness. For example, we assume that the effect of each prompt lasts for one time step and we did not consider the fact that the user's response time might vary. As a consequence, some of the assumptions on the hand-washing problem discussed in this chapter, upon which our models built, are too naive to be true. On the other hand, simplifying the hand-washing problem provides an advantage for analysing the relations between each factor in the hand-washing problem. Section 5.3.2 discussed how the tradeoffs between short term and long term goals, the method of assigning cost and reward, and the difference in user characteristics can all affect the policy preference.

In section 5.2.1, by analysing the worst-case scenarios, we showed that if the user characteristic has been wrongly estimated, the performance of the planning system can be poor. To avoid such cases, if the user characteristic is unobservable, while the optimal policy focuses on maximising the expected value, a randomised policy can be used to ensure satisfactory performance for any user type, as argued in section 5.1.6.

Since the number of time steps has to be defined as a variable in the models, the MDP model is not stationary, which is a basic assumption for most intelligent systems. There are three alternative way to solve this problem. Firstly, since we are interested in properties of the model in a finite time line, it is possible to calculate the cumulated reward without discount. However, the tradeoffs between short-term and long-term goals are not reflected in the model which does not take time into account. Since the positive reward for finishing the task eventually will be the same if the discount rate is 1, the planning system will always choose to do nothing in order to minimise the negative reward and achieve the maximum total reward. The second alternative is using the expected time of finishing the task as the evaluation of planning system's performance. In the reverse of the first alternative, the planning system can simply give a prompt every time to achieve the minimum expected time since there is no cost for giving a prompt. So the best way of solving this problem is to add a functionality of PRISM, which enables the calculation of cumulated discounted reward. With this functionality, time does not need to be defined explicitly in the model so that the system is stationary. This functionality should be easy to realise since the probabilistic model checking is based on PCTL.

Compared with the method used to evaluate the COACH system, the experiments discussed in this chapter show that probabilistic model checking is a systematic, efficient, and economy way of evaluating a planning system. Unlike carrying out simulation, the model is exposed to almost all possible scenarios during probabilistic model checking. Although clinical trial is the most reliable way to evaluate the system, the cost and risk are very high. So we propose the use of probabilistic model checking as an alternative evaluation method of clinical trial for planning systems guiding people with dementia through activities of daily living at their early design stage. In the next chapter, the design and evaluation of a planning system modified to address the dressing problem will be discussed, in which probabilistic model checking plays an important role.

Chapter 6
Planning systems for other activities of daily living

Inspired by the planning system to assist people with dementia with the task of hand washing, as well as the usefulness of probabilistic model checking in modelling problems in this domain and in system evaluation as discussed in Chapter 5, a naïve planning system for the activity of dressing has been designed. In this chapter, we will go through the motivation of applying the planning framework to activities other than hand washing, the model for the activity of dressing, and the model checking with PRISM.

6.1 The activity of dressing

The planning system for the hand-washing problem can be generalised to other activities in daily living to help people with dementia. There are many activities that can be modelled in similar ways as the hand-washing problem. The planning system designed for people with dementia should be designed for the activities which are easy to observe by sensors and significant in affecting the user's independence.

Here, we chose the problem of clothes-changing to analyse, in which a camera should be sufficient to monitor the progress. Without the requirement for some sophisticated sensing system, we can assume that the technologies needed to implement the planning system for assisting people with dementia in the activity of dressing are available.

The basic idea for monitoring the progresses in the dressing is that the planning system can identify which pieces of clothes the user is wearing with the help of computer vision. Technologies used in the COACH system to monitor hand-washing should be able to handle this problem. Although it might be difficult to keep track of a piece of clothing using computer vision because it is not a rigid body, as most vision tracking system assume their target to be [24], colour recognition can be used to solve this problem. Assuming each piece of clothing has a different colour, the planning system can identify which piece is worn on top. This assumption can be easily satisfied in real life. Since colour is easier to identify than other important elements in the dressing environment, for example motion, the requirement for the vision system is not too high.

Defining the dressing problem with plan steps, with the ability of observing the top layer, the planning system can identify all the layers the user has put on. Details of plan steps are in section 6.2.

6.2 Modelling the activity of dressing

The activity of dressing is modelled as an MDP. The activity status is defined by system variables including plan step and colour observed. The colour observed indicates which piece of clothing is the top layer worn by the user. Each piece of clothing is associated with one unique colour. For example, if a jacket is grey and a shirt is green, green is observed when the user is wearing the shirt. After the user puts the jacket on, only grey is observed no matter what colour of clothes the user wears under the jacket. Skin colour is observed if the user does not have clothes on.

As a starting point for modelling the dressing problem, it is assumed that 3 pieces of clothes are used in this activity. Initially, the user is in pyjamas, and the final goal of the task of dressing is to put on two layers, a shirt first and a jacket on top. During the activity, there are many possible statuses in terms of the combination of clothes worn. To reflect the reality as well as to limit the number of possible combinations, it is assumed that the jacket can only be worn as the top layer. So, there are 10 possible user statuses, which are: wearing pyjamas, wearing no clothes, wearing a shirt, wearing a shirt and a jacket, wearing pyjamas with a shirt on top, wearing pyjamas and a jacket, wearing pyjamas and a shirt and a jacket, wearing a shirt with pyjamas and a jacket, and wearing a shirt with pyjamas on top.

Figure 32 shows the plan step transition caused by the user's action. The user's action can be affected by the planning system. The user's possible actions include doing nothing, taking off the pyjamas, taking off the shirt, taking off the jacket, putting on the pyjamas, putting on the shirt, and putting on the jacket. These actions result in transition of the plan steps and can be observed by identifying the colour of the top layer. The planning system can give prompts, reminding the user what to do next. For the goal defined above, the system's actions include doing nothing and giving one of the five prompts, which are taking the pyjamas off, taking the shirt off, taking the jacket off, putting on the shirt, and putting on the jacket.

The transition probability also depends on the user's awareness and responsiveness. If the awareness is high, the probability that the user do the right thing at each step is high. If the responsiveness is high, the increase in probability of doing the right thing with the system's prompt is larger. For details of these assumptions, please refer to Chapter 5.

Other system variables include maximum plan step, maximum plan step repeat, and total time are defined for the same reasons and in a similar fashion as in the model for the hand-washing problem.

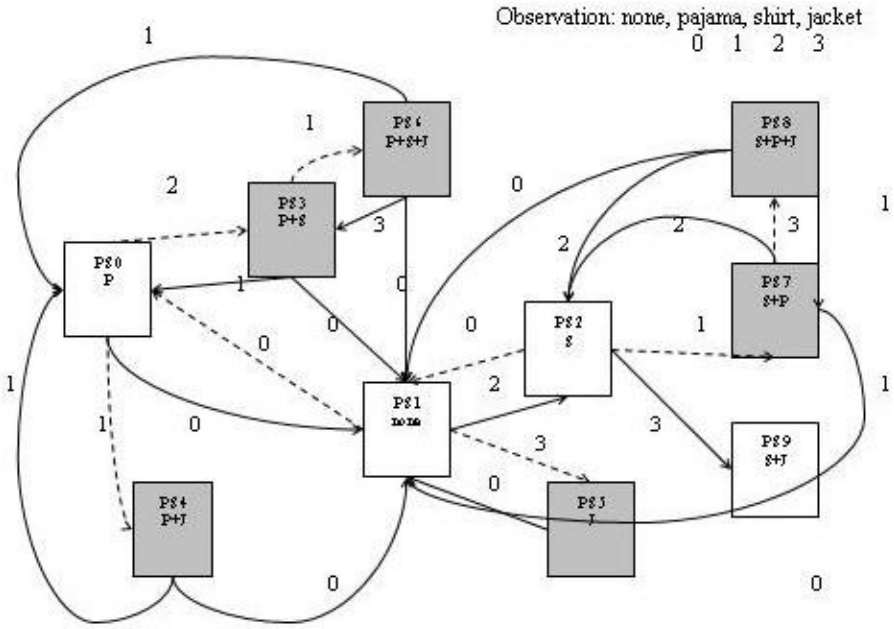

Figure 32: Plan-step diagram for the clothes-change problem (self loop eliminated)

The second reward structure defined in Chapter 5, section 5.1.7 is applied to this model for two reasons. Firstly, without enough prior knowledge in the domain of care-giving, relying on assumptions made for hand-washing, which is a similar to dressing, is a better strategy than making up unjustifiable assumptions. Second, as discussed in Chapter 5, the first reward structure is not suitable for MDP models.

The structure of the PRISM for the activity of dressing is similar to that of the hand-washing problem. For the same reasons discussed in Chapter 5 section 5.1.1, the dressing model consists of 6 modules, which are 5 basic phases and one additional step to specify a user type at the beginning of the task. The module for phase 0 contains the choices of system actions. In phase 1, the user's action is modelled as the effect of the system actions, and results in the top layer observed, represented by its colour. The system variables, including plan step and maximum plan step are updated determinately in phases 2 and 3. And then the system starts phase 0 of the next time step.

The main difference between the dressing problem and the hand-washing problem is that some of the plan steps in the dressing problem are undesirable. In the hand-washing problem, every plan step is necessary for finishing the task. In comparison, in the hand-washing problem, some plan steps are reachable only when the user does the

wrong things. All undesirable plan steps are marked grey in Figure 32.

6.3 Evaluation of the model for dressing in PRISM

Similar to experiments in Chapter 5, we are interested in evaluating policies for users with different characteristics. Because of the difference in modelling of the dressing problem and the hand-washing problem, properties related to entering the undesired plan steps can also be analysed. Also, a program has been developed to modify the optimal solution generated by PRISM, enforcing the stationary assumption.

6.3.1 Property checking

Firstly, the maximum and minimum cumulated discounted rewards are checked for different user characteristic types. Since awareness and responsiveness are used to model the user characteristic, the types are defined as low awareness/low responsiveness, low awareness/high responsiveness, high awareness/low responsiveness, and high awareness/high responsiveness.

Since there are tradeoffs between accuracy and complexity, the constant ‘total_time’ which is the upper bound of the variable ‘time_step’ is analysed. As discussed in Chapter 5, section 5.3, the larger the total time is, the more accurate the cumulated discounted reward is. On the other hand, with a smaller value of total time, a model with smaller state space and lower computational complexity can be built. From the experiment results shown in Figure 33, we can see that any value greater than 10 would be a reasonable choice in terms of accuracy. Also, taking complexity into consideration, a value of 20 is chosen. The maximum value is for the expected reward of deploying the best policy. The minimum value is the lower bound of the expected reward, no matter which possible policies the planning system uses. The properties checked are expressed in PRISM language as:
R {"total"} max =? [F MPS = 3]
and
R {"total"} min =? [F MPS = 3]

In the plan steps for hand-washing, although not every step is necessary for finishing the task, for example, it is possible for the user to skip the step of wetting hands before using the soap, entering a plan step for the first time always makes progress. By comparison, entering some plan steps even for the first time cannot cause a regression. For example, if the user puts a jacket on with the pyjamas, the user is going further away from reaching the goal.

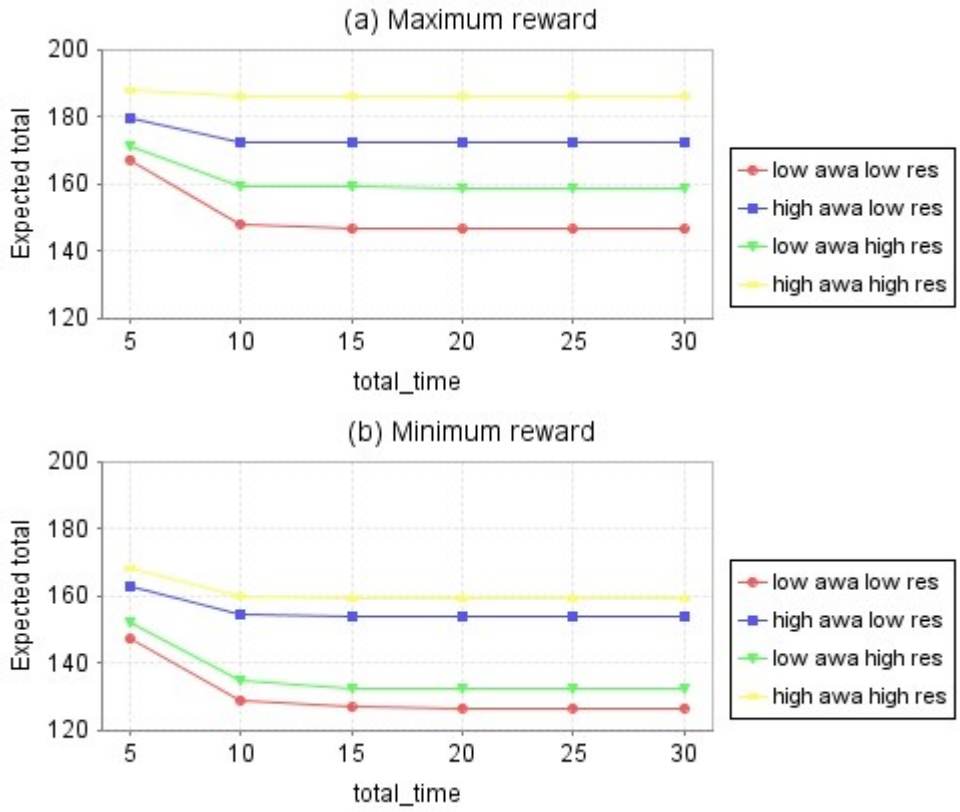

Figure 33: Cumulated discounted rewards for 4 types of user characteristic (a: maximum reward; b: minimum reward)

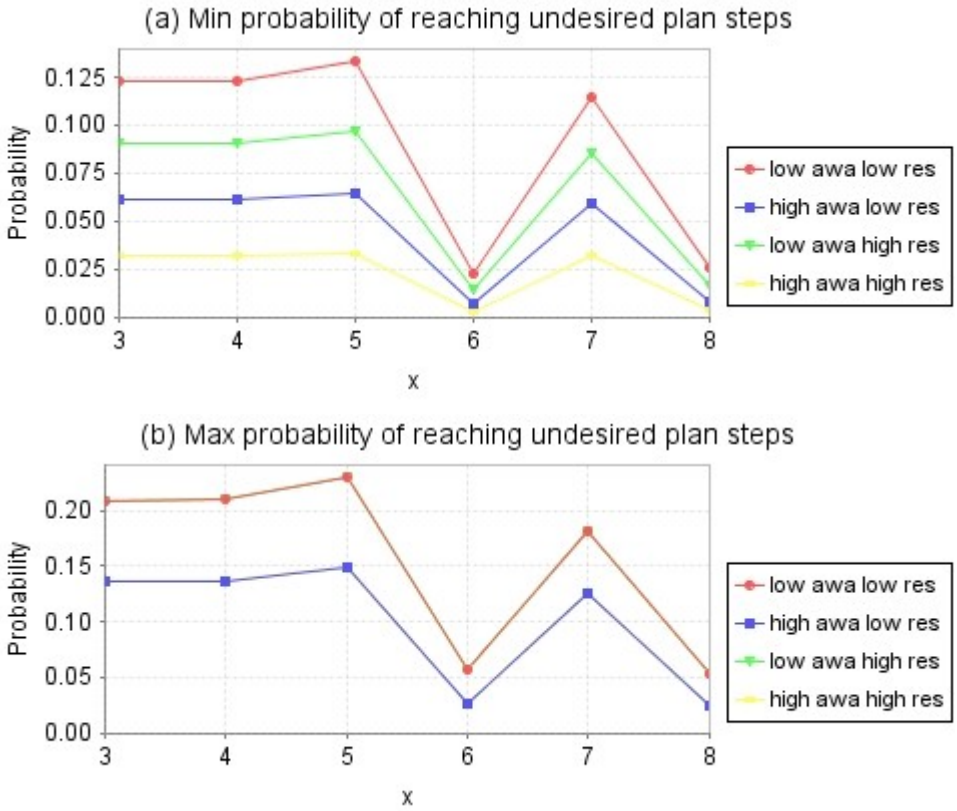

Figure 34: Probability of reaching undesired plan steps (a: minimum; b: maximum)

Figure 34 shows the maximum and minimum probability of reaching undesired plan steps. In PRISM language, the specifications of these properties are:
P min = ? [true U PS= x]
and
P max = ? [true U PS= x]
where x is the index of plan step interested.

Only plan steps 3- 8 are undesired as shown in Figure 32. Since plan step 0, which is the initial step, plan step 1, 2 and the final step 9 are necessary steps for finishing the task, their probabilities are 1. Plan steps 6 and 8 are less likely to be entered. Referring to Figure 32, plan step 6 must be entered from plan step 3, which is another undesired plan step. Not linking to a necessary plan step makes the possibility of entering plan step 6 low. The probabilities for plan step 8 are low for the same reason.

From Figure 34(b) we also notice that the differences in maximum probabilities between low awareness/low responsiveness and low awareness/high responsiveness, and between high awareness/low responsiveness and high awareness/low responsiveness are tiny. This is because in the worst case, the planning system gives very few prompts, so the probabilities only depend on the user's awareness but not on responsiveness.

Note that using a better policy can decrease the possibility of entering undesired plan

steps, but the policies achieving minimum values for each undesired plan step might not be the same. For example, if the policy optimising the probability of entering plan step 3 is used, the probability of entering plan step 4 might not achieve its minimum value. As we have shown, the event of entering one plan step is not independent of events of entering some other plan steps, so it does not make sense to add these probabilities up. To calculate the minimum and maximum probabilities of entering any of the undesired plan steps (Table 6), these properties are checked:

P min = ? [true U ((((PS = 3)|(PS = 4)) | ((PS=5) | (PS=6)))| ((PS=7) | (PS=8)))]

and

P max = ? [true U ((((PS = 3)|(PS = 4)) | ((PS=5) | (PS=6)))| ((PS=7) | (PS=8)))]

	min	max
Low awareness low responsiveness	0.401	0.578
High awareness low responsiveness	0.224	0.431
Low awareness high responsiveness	0.312	0.578
High awareness high responsiveness	0.123	0.431

Table 6: Probability of entering any undesired plan steps

6.2.2 Modifying the optimal policy

As discussed in Chapter 5, the stationary assumption cannot be made because the number of time steps has to be defined as a variable for the purpose of calculating cumulated discounted award, which is a function of the number of time steps. In order to show that the maximum cumulated discounted award is not achievable under the assumption of a stationary system, a program is written to modify policies generated for the MDP model by PRISM (see Appendix IV). The program is written in Java, which is an objected oriented programming language. It takes a text file of the transition matrix generated by PRISM and a text file describing the state space as inputs, and the output is a text file of a new transition matrix.

Assuming that changes in time do not vary the actions chosen by the planning system, transitions for states which have different values for ‘time_step’ but the same values for all other variables, are exactly the same. So, we choose the transitions for states with time_step=20 as the transitions for states with different values of time_step. Twenty is a relatively large number in terms of time steps, so we believe that states defined by all variables other than time_step are reachable at this time. The transition matrix has the format of:

state1 state2 probability

in which state1 and state2 are integers representing the states. For example:

0 1 0.25

0 2630 0.25

0 5635 0.25

0 8264 0.25
1 11269 1
2 11272 1

The meaning of the state is defined in the state space file, which has the format of:
state:(variable1,variable2,…)
For example:
(phase,a,rb,PS,MPS,MPSrepeat,time_step,top,r,prompt)
0:(-1,6,11,0,0,1,0,1,11,0)
1:(0,6,11,0,0,1,0,1,11,0)

Referring to the state space, the states defined by different values of time_step whose values for all other variables are identical are grouped together. Then the transitions from the states in the same group are changed according to the transitions when time_step=20. For example, two of the lines in the state space file are as follow:
2494:(0,6,11,4,0,0,1,9,11,0)
19828:(0,6,11,4,0,0,1,20,11,0)
Since the eighth variable is time_step, states 1234 and 4321 should have the same transition according the assumption of stationary. If there are two lines as follow in the transition matrix file:
2494 11269 1
19828 11272 1
then the first line should be changed into:
2494 11272 1
Since we only change the choices made by the planning system (states in which phase=0), the state transitions interested all have probability 1.

A DTMC Model can be constructed in PRISM through direct specification of the modified transition matrix. A file containing information about the initial state of the model as well as the state space file should be imported. Since importing state reward and transition reward is not supported by the current version of PRISM, only probability-based properties can be checked.

The natural language translation for
P = ? [time_step<5 U PS=9]
is “the probability that PS is eventually equal to 9 and time_step remains less than 5 until that point”. Since we know that PS is eventually equal to 9 for certain, this property can also be translated to “the probability of PS is equal to 9 within 5 time steps.”

We compared this probability for the DTMC model constructed by importing the transition matrix generated by PRISM, which specified the optimal policy regarding cumulated discounted reward and for the DTMC model constructed by importing the modified transition matrix, as well as the maximum and minimum probability for the

MDP model. Table 7 shows the result.

	L awa L res	H awa L res	L awa H res	H awa H res
DTMC (optimal)	0.595	0.796	0.753	0.937
DTMC (modified)	0.528	0.711	0.682	0.831
MDP max	0.660	0.855	0.760	0.941
MDP min	0.454	0.629	0.454	0.629

Table 7: Probability of finishing the task within 5 time steps

Note that the directly generated policy optimises the cumulated discounted reward, not the probability of finishing the task within 5 time steps, so that the probabilities for the DTMC model with the optimal policy are less than their maximum values. Although the policies generated by the Java program satisfy the stationary assumption, they might not be the optimal ones under this assumption. As discussed in Chapter 5, the optimal policy under the stationary assumption can be easily found if PRISM supports discounted rewards.

The idea of modifying the transition matrix to generate interesting policies can potentially be applied to model problems which are only partially observable. Similar to what we are doing with the variable time_step, influence of the variables which are unobservable to the planning system can be removed. However, as discussed above, there is no guarantee that the policies generated are optimal under the assumption of partial observability.

6.4 Discussion

In this chapter, we proposed a planning system to help people with dementia with the activity of dressing. The problem of dressing is modelled as an MDP in PRISM. The nature of the dressing problem is analysed by checking properties related to its plan steps.

Also, a Java program has been developed to modify the optimal policy generated by PRISM. The program outputs a transition matrix in plain text, which can be imported into PRISM to construct a DTMC model. This program can be used to eliminate the effect of time in the policy to make the model to be stationary. However, the best way to solve the stationary problem is to support a discounted reward in PRISM. It is also feasible to use this program to make the model satisfying the assumption of partial observability by removing the effect of unobservable variables in the policy. The result will be a feasible policy, even if it is not the optimal one.

The planning system designed here can serve as an example of modelling activities in daily living in PRISM. It is possible to model other activities like brushing teeth,

making tea, and using the toilet, for which planning systems can be designed with the support of different sensing systems. With a probabilistic model checker, models can be assessed against the assumptions of the domain and system requirements. Since the parameters can be easily changed, and both worst-case and best-case scenarios can be analysed, modelling with a probabilistic model checker is potentially a useful first step in designing a planning system to help people with dementia through activities in daily living.

Chapter 7
Conclusion and future work

From the probability modelling checking in Chapter 5 for the hand-washing problem, we can see that the discount rate and the reward structure have significant influence on the preference in policies. This is because they represent the objectives in trading off between long-term and short-term goals. So, fine-tuning rewards is critical for the development of a planning system. By identifying the relationships between factors in the hand-washing problem, the experiment results from Chapter 5 show that even when the user's behaviour cannot be fully observed or modelled accurately, general rules can be applied in choosing a policy for the user.

The planning system helping people with dementia during the activity of dressing is presented in Chapter 6. Modelled the dressing problem successfully, it is shown that probabilistic model checking is useful in early stage of planning system design.

The program presented in Chapter 6, which can modify the policies generated by PRISM, is useful for generating feasible policy. It can provide feasible policy for stationary system when time is defined as a variable in the system. It can be modified to generate feasible policy under the assumption of partially observability.

From modelling intelligent systems in PRISM, we found that the functionality of PRISM can be further improved in several ways. Firstly, it could support cumulated discounted reward by enabling the use of time in specification of cost/reward. Secondly, if state rewards and transition rewards can be imported explicitly, then reward-based properties can be checked for models constructed through direct specification of their transition matrix.

There are some other limitations of this work. Because of the lack of prior knowledge in the domain of care giving, although probability model checking is able to cover every possible scenario, the scenarios chosen to be discussed and presented might not be the scenarios with most concerns. Also the simplified models might be too naïve and not accurate enough to expose all interesting properties of the hand-washing and dressing problems.

Other than MDP, it is possible to model one activity of daily living as a CTMC in PRISM. Modelling as a CTMC, the user's action can be modelled as a smooth state transition with a rate. This does not only model the user action more accurately, it also can improve the planning system performance by taking the speed of the user's action into account.

As future work, MDP based planning systems can be designed for more activities in daily living. With the help of probabilistic model checking, best-case and worst-case scenarios can be analysed. So even without prior knowledge in the domain of care giving or results form early stage clinical trials, the system can be analysed by simply changing the value of each parameter. It is also possible to gain a better understanding of relevant factors in the system form analysing parameters of the model. The cost for carrying out model checking is low, and this early stage analysis can help to set a higher starting point for later clinical trials.

Bibliography

[1] J. Hoey, A. Bertoldi, P. Poupart, and A. Mihailidis. **Assisting Persons with Dementia during Handwashing Using a Partially Observable Markov Decision Process.** In *Proceedings of the International Conference on Vision Systems (ICVS)*, Biefeld, Germany, 2007.

[2] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. **A Decision-Theoretic Approach to Task Assistance for Persons with Dementia**. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1293-1299, Edinburgh, Scotland, 2005.

[3] J. Boger, J. Hoey, P. Poupart, C. Boutilier, G. Fernie, and A. Mihailidis. **A Planning System Based on Markov Decision Processes to Guide People with Dementia Through Activities of Daily Living**. Manuscript, 2005.

[4] NHS's website
http://www.nhsdirect.nhs.uk/articles/article.aspx?articleId=124#
Reviewed on July 17th 2008.

[5] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. **PRISM: A Tool for Automatic Verification of Probabilistic System**. In H. Hermanns and J. Palsberg (editors) *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of LNCS, pages 441-444, Springer. March 2006.

[6] **PRISM Manual** *version 3.2*. Available from PRISM's website:
http://www.prismmodelchecker.org/

[7] M. Kwiatkowska. **Quantitative Verification: Models, Techniques and Tools.** In *Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 449-458, ACM Press. September 2007.

[8] J. Hoey. **MDP Handwashing Model**.
http://www.cs.toronto.edu/~jhoey/papers/zmj.pdf. February 2006.

[9] J. Bates, J. Boote, and C. Beverly. **Psychosocial interventions for people with a dementing illness: A systematical review.** Journal of Advanced Nursing, 45(6):644-658, 2004.

[10] C. Boutilier, T. Dean, and S. Hanks. **Decision-Theoretic Planning: Structural Assumptions and Computational Leverage.** *J. of Artificial Intelligence Research,* 11:1-94 (1999).

[11] J. Hoey, R. St-Aubin, A. Hu, and C. Boutillier. **SPUDD: Stochastic Planning using Decision Diagrams.** in *Proceedings of the Fifteenth International Conference*

on Uncertainty in Artificial Intelligence, Stockholm, 1999, pp. 279-288.

[12] M. Spaan and N. Vlassis. **Perseus: Randomized point-based value iteration for POMDPs.** *Journal of Artificial Interlligence Research,* 24:195-220, 2005.

[13] N. Kushmerick, S. Hanks, and D. Weld. **An Algorithm for Probabilistic Planning.** *Artificial Intelligence*, 76, 239-286, 1995.

[14] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. **Towards robotic assistants in nursing homes: challenges and results.** *Rob. Auton. Syst.*, vol. 42, pp. 271-281, 2003.

[15] M. Kwiatkowska, G. Norman and D. Parker. **Stochastic Model Checking.** *In M. Bernardo, J. Hillston, editors, Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)* Vol. 4486 of LNCS (Tutorial Volume), pages 220-270. Springer, 2007.

[16] K. Etessami, M. Kwiatkowska, M. Vardi and M. Yannakakis. **Multi-Objective Model Checking of Markov Decision Processes.** *In O. Grumberg, M. Huth, editors, Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)* Vol. 4424 of LNCS, pages 50-65. Springer, 2007.

[17] D. Parker. **Implementation of Symbolic Model Checking for Probabilistic Systems.** *PhD thesis, University of Birmingham,* 2002.

[18] M. Kwiatkowska, G. Norman and D. Parker. **Probobilistic symbolic model checking with PRISM: A hybrid approach.** *Int. Journal on Software Tools for Technology Transfer, 6(2):128-142,* 2004.

[19] R. Alur and T.A. Henzinger. **Reactive modules.** *Formal Methods in System Design: An International Journal, 15(1):7-48, Jul.* 1999.

[20] A. Mihailidis, J. Boger, M. Canido and J. Hoey. **The use of an intelligent prompting system for people with dementia.** *Designing for seniors: innovations for graying times, Volume 14, Issue 4. ACM New York,* 2007.

[21] M. Kwiatkowska, G. Norman and D. Parker. **Modelling and Verification of Probabilistic Systems.** *Lecture notes of the course on Probabilistic Model Checking taught as part of the Workshop on Mathematical Models and Techniques for Analysing Systems, University of Montr´eal, September 30–October 4, 2002.*

[22] H. Hoenig, D. Taylor Jr and F. Sloan. **Does Assistive Technology Substitute for Personal Assistance Among the Disabled Elderly?** *Am. J. Public Health; 93:330-337,* 2003.

[23] W. S. Lovejoy. **A Survey of Algorithmic Methods for Partially Observed Markov Decision Processes.** *Annals of Operations Research, 28:47-66,* 1991.

[24] Yi Ma, Stefano Soatto, Jana Kosecka, and S.Shankar Sastry. **An Invitation to 3-D Vision**. *Springer-Verlag New York, Inc.*, 2004.

Appendix I
The PRISM file of MDP model for the hand-washing problem

```
mdp

//scheduler
global phase : [0..6] init 0;
//phase 1-4: User action, sensor detect
//phase=1: if prompt has effect on hand location, update handlocation and go to phase 3, otherwise go to phase 2
//phase=2: update hand location for non-effective prompt or no prompt
//phase=3: if prompt has effect on water flow, update water flow and go to phase 5, otherwise go to phase 4
//phase=4: update water flow
//phase 5: update activity status (planstep etc.)
//phase=6: update other variables


//HandLocation: HL
//soap, tap, water, towel, sink, away
global HL : [0..5] init 5;

//waterFlow: WF
//on, off
global WF : [0..1] init 0;

//handLocationCorrectConst
const double HLCC;

//handLocationSameConst
const double HLSC;

//waterFlowCorrectConst
const double WFCC;

//handLocation Effect
//HLCC/2 < HLE < 1
const double HLE;
```

```
const double WFE;

const int maxwait;

const int total_time;

//number of time intervals without plan step change or progress
global NW: [-1..maxwait] init 0;

module update_activity_a
//PlanStep: PS
//A B C D E G H J K
PS : [0..8] init 0;


//Regression:
Reg: [0..1] init 0;

        [] phase = 5 & PS = 0 & HL= 0 & WF = 0 -> (PS'=4) & (phase'=6) & (Reg'=0) & (NW'=0);
    //planstep A, handLocation soap, Waterflow off -> planstep E

    [] phase = 5 & PS = 0 & HL= 1 & WF = 0 -> (PS'=0) & (phase'=6) & (Reg'=0) & (NW'=0);
    //planstep A, handLocation tap, Waterflow off -> planstep A
    [] phase = 5 & PS = 0 & HL= 1 & WF = 1 -> (PS'=1) & (phase'=6) &  (Reg'=0)& (NW'=0);
    //planstep A, handLocation tap, Waterflow on -> planstep B
    [] phase = 5 & PS = 0 & HL= 2 & WF = 0 -> (PS'=0) & (phase'=6) &  (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep A, handLocation water, Waterflow off -> planstep A
    [] phase = 5 & PS = 0 & HL= 3 & WF = 0 -> (PS'=0) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep A, handLocation towel, Waterflow off -> planstep A
    [] phase = 5 & PS = 0 & HL= 4 & WF = 0-> (PS'=0) & (phase'=6) &  (Reg'=0)& (NW'=0);
    //planstep A, handLocation sink, Waterflow off -> planstep A
    [] phase = 5 & PS = 0 & HL= 5 & WF = 0 -> (PS'=0) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep A, handLocation away, Waterflow off -> planstep A

    [] phase = 5 & PS = 1 & HL= 0 & WF = 1-> (PS'=3) & (phase'=6)&   (Reg'=0)& (NW'=0);
    //planstep B, handLocation soap, Waterflow on -> planstep D
    [] phase = 5 & PS = 1 & HL= 1 & WF = 0 -> (PS'=0) & (phase'=6) & (Reg'=1)& (NW'=0);
    //planstep B, handLocation tap, Waterflow off -> planstep A
    [] phase = 5 & PS = 1 & HL= 1 & WF = 1 -> (PS'=1) & (phase'=6) &  (Reg'=0) &
(NW'=min(NW+1, maxwait));
```

```
    //planstep B, handLocation tap, Waterflow on -> planstep B
    [] phase = 5 & PS = 1 & HL= 2 & WF = 1 -> (PS'=2) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep B, handLocation water, Waterflow on -> planstep C
    [] phase = 5 & PS = 1 & HL= 3 & WF = 1 -> (PS'=1) & (phase'=6) &  (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep B, handLocation towel, Waterflow on -> planstep B
    [] phase = 5 & PS = 1 & HL= 4 & WF = 1 -> (PS'=1) & (phase'=6) &  (Reg'=0)& (NW'=0);
    //planstep B, handLocation sink, Waterflow on -> planstep B
    [] phase = 5 & PS = 1 & HL= 5 & WF = 1 -> (PS'=1) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep B, handLocation away, Waterflow on -> planstep B

    [] phase = 5 & PS = 2 & HL= 0 & WF = 1-> (PS'=3) & (phase'=6)& (Reg'=0) & (NW'=0);
    //planstep C, handLocation soap, Waterflow on -> planstep D
    [] phase = 5 & PS = 2 & HL= 1 & WF = 0 -> (PS'=0) & (phase'=6) & (Reg'=1)
&(NW'=min(NW+1, maxwait));
    //planstep C, handLocation tap, Waterflow off -> planstep A
    [] phase = 5 & PS = 2 & HL= 1 & WF = 1 -> (PS'=2) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep C, handLocation tap, Waterflow on -> planstep C
    [] phase = 5 & PS = 2 & HL= 2 & WF = 1 -> (PS'=2) & (phase'=6) & (Reg'=0)
&(NW'=min(NW+1, maxwait));
    //planstep C, handLocation water, Waterflow on -> planstep C
    [] phase = 5 & PS = 2 & HL= 3 & WF = 1 -> (PS'=1) & (phase'=6) & (Reg'=1)
&(NW'=min(NW+1, maxwait));
    //planstep C, handLocation towel, Waterflow on -> planstep B
    [] phase = 5 & PS = 2 & HL= 4 & WF = 1 -> (PS'=2) & (phase'=6) & (Reg'=0)
&(NW'=min(NW+1, maxwait));
    //planstep C, handLocation sink, Waterflow on -> planstep C
    [] phase = 5 & PS = 2 & HL= 5 & WF = 1 -> (PS'=2) & (phase'=6)  & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep C, handLocation away, Waterflow on -> planstep C

    [] phase = 5 & PS = 3 & HL= 0 & WF = 1-> (PS'=3) & (phase'=6)&   (Reg'=0)&
(NW'=min(NW+1, maxwait));
    //planstep D, handLocation soap, Waterflow on -> planstep D
    [] phase = 5 & PS = 3 & HL= 1 & WF = 0 -> (PS'=4) & (phase'=6) &
(Reg'=0)&(NW'=min(NW+1, maxwait));
    //planstep D, handLocation tap, Waterflow off -> planstep E
    [] phase = 5 & PS = 3 & HL= 1 & WF = 1 -> (PS'=3) & (phase'=6) &  (Reg'=0)&
(NW'=min(NW+1, maxwait));
    //planstep D, handLocation tap, Waterflow on -> planstep D
    [] phase = 5 & PS = 3 & HL= 2 & WF = 1 -> (PS'=5) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep D, handLocation water, Waterflow on -> planstep G
```

```
    [] phase = 5 & PS = 3 & HL= 3 & WF = 1 -> (PS'=3) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep D, handLocation towel, Waterflow on -> planstep D
    [] phase = 5 & PS = 3 & HL= 4 & WF = 1 -> (PS'=3) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep D, handLocation sink, Waterflow on -> planstep D
    [] phase = 5 & PS = 3 & HL= 5 & WF = 1 -> (PS'=3) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep D, handLocation away, Waterflow on -> planstep D

    [] phase = 5 & PS = 4 & HL= 0 & WF = 0 -> (PS'=4) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep E, handLocation soap, Waterflow off -> planstep E
    [] phase = 5 & PS = 4 & HL= 1 & WF = 0 -> (PS'=4) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep E, handLocation tap, Waterflow off -> planstep E
    [] phase = 5 & PS = 4 & HL= 1 & WF = 1 -> (PS'=3) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep E, handLocation tap, Waterflow on -> planstep D
    [] phase = 5 & PS = 4 & HL= 2 & WF = 0 -> (PS'=4) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep E, handLocation water, Waterflow off -> planstep E
    [] phase = 5 & PS = 4 & HL= 3 & WF = 0 -> (PS'=4) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep E, handLocation towel, Waterflow off -> planstep E
    [] phase = 5 & PS = 4 & HL= 4 & WF = 0-> (PS'=4) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep E, handLocation sink, Waterflow off -> planstep E
    [] phase = 5 & PS = 4 & HL= 5 & WF = 0 -> (PS'=4) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep E, handLocation away, Waterflow off -> planstep E

        [] phase = 5 & PS = 5& HL= 0 & WF = 1-> (PS'=3) & (phase'=6)&  (Reg'=1)&
(NW'=min(NW+1, maxwait));
    //planstep G, handLocation soap, Waterflow on -> planstep D
    [] phase = 5 & PS = 5 & HL= 1 & WF = 0 -> (PS'=6) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep G, handLocation tap, Waterflow off -> planstep H
    [] phase = 5 & PS = 5 & HL= 1 & WF = 1 -> (PS'=5) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep G, handLocation tap, Waterflow on -> planstep G
    [] phase = 5 & PS = 5 & HL= 2 & WF = 1 -> (PS'=5) & (phase'=6) & (Reg'=0) & (NW'=0);
    //planstep G, handLocation water, Waterflow on -> planstep G (keep washing)
   [] phase = 5 & PS = 5 & HL= 3 & WF = 1 -> (PS'=7) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep G, handLocation towel, Waterflow on -> planstep J
    [] phase = 5 & PS = 5 & HL= 4 & WF = 1 -> (PS'=5) & (phase'=6) & (Reg'=0) & (NW'=0);
    //planstep G, handLocation sink, Waterflow on -> planstep G
    [] phase = 5 & PS = 5 & HL= 5 & WF = 1 -> (PS'=5) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep G, handLocation away, Waterflow on -> planstep G
```

```
    [] phase = 5 & PS = 6 & HL= 0 & WF = 0 -> (PS'=4) & (phase'=6) & (Reg'= 1)&
(NW'=min(NW+1, maxwait));
    //planstep H, handLocation soap, Waterflow off -> planstep E
        [] phase = 5 & PS = 6 & HL= 1 & WF = 0 -> (PS'=6) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
        //planstep H, handLocation tap, Waterflow off -> planstep H
    [] phase = 5 & PS = 6 & HL= 1 & WF = 1 -> (PS'=5) & (phase'=6) & (Reg'=1) &
(NW'=min(NW+1, maxwait));
    //planstep H, handLocation tap, Waterflow on -> planstep G
    [] phase = 5 & PS = 6 & HL= 2 & WF = 0 -> (PS'=6) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep H, handLocation water, Waterflow off -> planstep H
    [] phase = 5 & PS = 6 & HL= 3 & WF = 0 -> (PS'=8) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep H, handLocation towel, Waterflow off -> planstep K
    [] phase = 5 & PS = 6 & HL= 4 & WF = 0-> (PS'=6) & (phase'=6) & (Reg'=0) & (NW'=0);
    //planstep H, handLocation sink, Waterflow off -> planstep H
    [] phase = 5 & PS = 6 & HL= 5 & WF = 0 -> (PS'=6) & (phase'=6) &  (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep H, handLocation away, Waterflow off -> planstep H
    [] phase = 5 & PS = 7 & HL= 0 & WF = 1-> (PS'=3) & (phase'=6)&  (Reg'=1)&
(NW'=min(NW+1, maxwait));
    //planstep J, handLocation soap, Waterflow on -> planstep D
    [] phase = 5 & PS = 7 & HL= 1 & WF = 0 -> (PS'=8) & (phase'=6) &  (Reg'=0)& (NW'=0);
    //planstep J, handLocation tap, Waterflow off -> planstep K
    [] phase = 5 & PS = 7 & HL= 1 & WF = 1 -> (PS'=7) & (phase'=6) & (Reg'=0)& (NW'=0);
    //planstep J, handLocation tap, Waterflow on -> planstep J
   [] phase = 5 & PS = 7 & HL= 2 & WF = 1 -> (PS'=5) & (phase'=6)&(Reg'=1) &
(NW'=min(NW+1, maxwait)) ;
        //planstep J, handLocation towel, Waterflow off -> planstep K
    [] phase = 5 & PS = 7 & HL= 3 & WF = 1 -> (PS'=7) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep J, handLocation towel, Waterflow on -> planstep J
   [] phase = 5 & PS = 7 & HL= 4 & WF = 1 -> (PS'=7) & (phase'=6) & (Reg'=0) & (NW'=0);
    //planstep J, handLocation sink, Waterflow on -> planstep J
   [] phase = 5 & PS = 7 & HL= 5 & WF = 1 -> (PS'=7) & (phase'=6) & (Reg'=0) &
(NW'=min(NW+1, maxwait));
    //planstep J, handLocation water, Waterflow on -> planstep G
    //[] phase = 5 & PS = 7 & HL= 3 & WF = 0 -> (PS'=8) & (phase'=6) & (Prog'=0)& (Reg'=0)&
(NW'=0);


    [] phase =5& PS=8-> (phase'=6);
```

```
endmodule

module update_activity_b
//Maximum plan step: MPS
//A, BCE, D, G, HJ, K
//0, 1,   2, 3, 4,  5
MPS : [0..5] init 0;

time_step: [0..total_time] init 0;

//MPSrepeat
//1 means repeat, 0 mean it's the first time to reach this MPS
MPSrepeat : [0..1] init 1;
    [] phase = 6 & ((PS)=(1)|(PS)=(2)|(PS)=(4))& MPS = 0-> (MPS' = 1) & (phase'=0) &
(MPSrepeat'=0) & (time_step' = min(time_step+1, total_time)) ;
    [] phase = 6 & PS = 3 & MPS = 1 -> (MPS'= 2)& (phase'=0)& (MPSrepeat'=0) &(time_step' =
min(time_step+1, total_time));
    [] phase = 6 & PS = 5 & MPS = 2 -> (MPS'=3)& (phase'=0)& (MPSrepeat'=0) &(time_step' =
min(time_step+1, total_time));
    [] phase = 6 & ((PS)=(6)|(PS)=(7))& MPS = 3 -> (MPS' = 4) & (phase'=0)& (MPSrepeat'=0)
&(time_step' = min(time_step+1, total_time));

    [] phase = 6 & PS=0  -> (phase'=0) & (MPSrepeat'=1) &(time_step' = min(time_step+1,
total_time));
    [] phase = 6 & ((PS)=(1)|(PS)=(2)|(PS)=(4))& ((MPS)=(1)|(MPS)=(2)|(MPS)=(3)|(MPS)=(4))->
(phase'=0) & (MPSrepeat'=1) &(time_step' = min(time_step+1, total_time));
    [] phase = 6 & PS = 3 & ((MPS)=(0)|(MPS)=(2)|(MPS)=(3)|(MPS)=(4))-> (phase'=0) &
(MPSrepeat'=1) &(time_step' = min(time_step+1, total_time));
    [] phase = 6 & PS = 5 & ((MPS)=(0)|(MPS)=(1)|(MPS)=(3)|(MPS)=(4))-> (phase'=0) &
(MPSrepeat'=1) &(time_step' = min(time_step+1, total_time));
    [] phase = 6 & ((PS)=(6)|(PS)=(7))& ((MPS)=(0)|(MPS)=(1)|(MPS)=(2)|(MPS)=(4))-> (phase'=0)
& (MPSrepeat'=1) &(time_step' = min(time_step+1, total_time));
    [] phase = 6 & PS = 8 & MPS=4  -> (MPS'=5)& (MPSrepeat'=0) &(time_step' = min(time_step+1,
total_time));
    [] phase= 6 & PS=8 & MPS=5 -> (MPSrepeat'=1);
endmodule



module user_hand_a
//nothing:
    [] phase = 1 & action = 0 -> (phase'=2);
//Water On:
```

```
    //planstep 0 (A) | 4 (E) , handLocation 1 (tap)
    [] phase = 1 & action = 1 & PS = 0 ->
        HLE : (HL' = 1) & (phase'=3)
        + (1-HLE)*HLCC/2 :(HL'=0)&(phase'=3)
        + (1-HLE)*HLCC/2 : (HL'=4)&(phase'=3)
        + (1-HLE)*(1-HLCC)/3: (HL'=2)&(phase'=3)
        + (1-HLE)*(1-HLCC)/3: (HL'=3)&(phase'=3)
        + (1-HLE)*(1-HLCC)/3:  (HL'=5)&(phase'=3);

    [] phase =1 & action = 1 & PS = 4 ->
        HLE : (HL' = 1) & (phase'=3)
        + (1-HLE)*HLCC/2:(HL'=2)&(phase'=3)
        + (1-HLE)*HLCC/2 : (HL'=4)&(phase'=3)
        + (1-HLE)*(1-HLCC)/3:  (HL'=0)&(phase'=3)
        + (1-HLE)*(1-HLCC)/3: (HL'=3)&(phase'=3)
        + (1-HLE)*(1-HLCC)/3: (HL'=5)&(phase'=3);



//water Off:
    //planstep 5 (G) | 7 (J) , handLocation 1 (tap)
    [] phase = 1 & action = 2 & PS = 5 ->
        HLE : (HL' = 1) & (phase'=3)
        + (1-HLE)*HLCC/2 : (HL' = 3) & (phase'=3)
        + (1-HLE)*HLCC/2 : (HL' = 4) & (phase' = 3)
        + (1-HLE)*(1-HLCC)/3:  (HL'=0) & (phase'=3)
        + (1-HLE)*(1-HLCC)/3:  (HL'=2) & (phase'=3)
        + (1-HLE)*(1-HLCC)/3:  (HL'=5) & (phase'=3);

    [] phase = 1 & action = 2 & PS = 7 ->
        HLE : (HL' = 1) & (phase'=3)
        + (1-HLE)*HLCC : (HL' = 4) & (phase' = 3)
        + (1-HLE)*(1-HLCC)/4: (HL'=0) & ( phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=2) & ( phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=3) & ( phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=5) & ( phase'=3);

//use soap:
    //planstep 0 (A) | 1 (B) | 2 (C), handLocation 0 (soap)
    [] phase = 1 & action = 3 & PS = 0 ->
        HLE : (HL' = 0) & (phase' = 3)
        + (1-HLE)*HLCC/2  : (HL' = 1 ) & (phase' = 3)
        + (1-HLE)*HLCC/2  : (HL' = 4) & (phase' = 3)
        + (1-HLE)*(1-HLCC)/3 : (HL' = 2) & ( phase' = 3)
```

```
        + (1-HLE)*(1-HLCC)/3 : (HL' = 3) & ( phase' = 3)
        + (1-HLE)*(1-HLCC)/3 : (HL' = 5) & ( phase' = 3);

    [] phase = 1 & action = 3 & PS = 1 ->
        HLE : (HL' = 0) & (phase' = 3)
        + (1-HLE)*HLCC/2 : (HL'= 2) & (phase' = 3)
        + (1-HLE)*HLCC /2 : (HL'=4) & (phase' = 3)
        + (1-HLE)*(1-HLCC)/3 : (HL' = 1) & (phase' = 3)
        + (1-HLE)*(1-HLCC)/3 : (HL' = 3) & (phase' = 3)
        + (1-HLE)*(1-HLCC)/3: (HL' = 5) & (phase' = 3);

    [] phase = 1 & action = 3 & PS = 2 ->
        HLE  : (HL'=0) & (phase' = 3)
        + (1-HLE)*HLCC : (HL' = 4) & (phase' = 3)
        + (1-HLE)*(1-HLCC)/4: (HL'=1) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=2) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=3) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=5) & (phase'=3);



//rinse hands:
    //planstep 3 (D), handLocation 2 (water)
    [] phase = 1 & action = 4 & PS = 3->
        HLE: (HL'=2)& (phase'=3)
        + (1-HLE)*HLCC: (HL'=4) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=0) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=1) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=3) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4: (HL'=5) & (phase'=3);



//wet hands:
    //planstep 1 (B), handLocation 2 (water)
    [] phase = 1 & action = 5 & PS = 1->
        HLE: (HL'=2) & (phase'= 3)
        + (1-HLE)*HLCC/2: (HL'=0) & (phase'=3)
        + (1-HLE)*HLCC/2: (HL'=4) & (phase'=3)
        + (1-HLE)*(1-HLCC)/3 : (HL'=1) & (phase'=3)
        + (1-HLE)*(1-HLCC)/3 : (HL'=3) & (phase'=3)
        + (1-HLE)*(1-HLCC)/3  : (HL'=5) & (phase'=3);
```

```
//dry hands:
    //planstep 5 (G) | 6 (H), handLocation 3 (towel)
    []phase = 1& action =6 & PS =5 ->
        HLE: (HL'=3) & (phase'=3)
        + (1-HLE)*HLCC/2: (HL'=1) & (phase' = 3)
        + (1-HLE)*HLCC/2: (HL'=4) & (phase'=3)
        + (1-HLE)*(1-HLCC)/3   : (HL'=0) & (phase'=3)
        + (1-HLE)*(1-HLCC)/3   : (HL'=2) & (phase'=3)
        + (1-HLE)*(1-HLCC)/3   : (HL'=5) & (phase'=3);

    []phase = 1& action =6 & PS =6 ->
        HLE: (HL'=3) & (phase'=3)
        +(1-HLE)*HLCC: (HL'=4) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4 : (HL'=0) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4 : (HL'=1) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4  : (HL'=2) & (phase'=3)
        + (1-HLE)*(1-HLCC)/4  : (HL'=5) & (phase'=3);


endmodule

module user_hand_b
//do nothing:
    //planstep 0 (correct location 0, 1, 4)
    [] phase = 2 & PS=0 & (HL = 0 | HL = 1 | HL = 4)->
        HLCC/3: (HL'=0)&(phase'=3)
        + HLCC/3 :(HL'=1)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        + (1-HLCC)/3: (HL'=2)&(phase'=3)
        + (1-HLCC)/3: (HL'=3)&(phase'=3)
        +(1-HLCC)/3: (HL'=5)&(phase'=3);

    [] phase = 2  & PS=0 & HL = 2 ->
        (1-HLCC)*HLSC : (HL'=2)&(phase'=3)
        + HLCC/3 : (HL'=0)&(phase'=3)
        + HLCC/3 :(HL'=1)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        + (1-HLCC)*(1-HLSC)/2: (HL'=3)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=5)&(phase'=3);

    [] phase = 2  & PS=0 & HL = 3 ->
        (1-HLCC)*HLSC : (HL'=3)&(phase'=3)
        + HLCC/3 : (HL'=0)&(phase'=3)
```

```
        + HLCC/3 :(HL'=1)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        + (1-HLCC)*(1-HLSC)/2: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=5)&(phase'=3);

    [] phase = 2 & PS=0 & HL = 5 ->
        (1-HLCC)*HLSC : (HL'=5)&(phase'=3)
        + HLCC/3 : (HL'=0)&(phase'=3)
        + HLCC/3 :(HL'=1)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        + (1-HLCC)*(1-HLSC)/2: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=3)&(phase'=3);

    //planstep 1  (correct location 0, 2, 4)
    [] phase = 2 & PS=1 & ((HL) = (0) |(HL) = (2) |(HL) = (4))  ->
                HLCC/3 : (HL'=0)&(phase'=3)
        + HLCC/3 :(HL'=2)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        + (1-HLCC)/3: (HL'=1)&(phase'=3)
        + (1-HLCC)/3: (HL'=3)&(phase'=3)
        +(1-HLCC)/3: (HL'=5)&(phase'=3);

    [] phase = 2 & PS=1  & HL = 1 ->
        (1-HLCC)*HLSC : (HL'=1)&(phase'=3)
        + HLCC/3 :(HL'=4)&(phase'=3)
        + HLCC/3 : (HL'=2)&(phase'=3)
        + HLCC/3: (HL'=0)& (phase'=3)
        + (1-HLCC)*(1-HLSC)/2: (HL'=3)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=5)&(phase'=3);


    [] phase = 2 & PS=1  & HL = 3 ->
(1- HLCC)*HLSC : (HL'=3)&(phase'=3)
+ HLCC/3: (HL'=0)& (phase'=3)
        + HLCC/3 :(HL'=4)&(phase'=3)
        + HLCC/3 : (HL'=2)&(phase'=3)
        + (1-HLCC)*(1-HLSC)/2: (HL'=1)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=5)&(phase'=3);

    [] phase = 2  & PS=1 & HL = 5 ->
        (1-HLCC)*HLSC : (HL'=5)&(phase'=3)
        + HLCC/3: (HL'=0)& (phase'=3)
                + HLCC/3 :(HL'=4)&(phase'=3)
        + HLCC/3: (HL'=2)&(phase'=3)
```

```
        + (1-HLCC)*(1-HLSC)/2: (HL'=1)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=3)&(phase'=3);

//planstep 2 (correct location 0, 4)
[] phase = 2  & PS=2 & (HL = 0 | HL = 4) ->
        HLCC/2 :(HL'=0)&(phase'=3)
        + HLCC/2 : (HL'=4)&(phase'=3)
        + (1-HLCC)/4: (HL'=1)&(phase'=3)
        + (1-HLCC)/4: (HL'=2)&(phase'=3)
        + (1-HLCC)/4: (HL'=3)&(phase'=3)
        +(1-HLCC)/4: (HL'=5)&(phase'=3);

[] phase = 2  & PS=2 & HL = 1 ->
        (1-HLCC)*HLSC: (HL'=1)&(phase'=3)
        +HLCC/2 :(HL'=0)&(phase'=3)
        + HLCC/2 : (HL'=4)&(phase'=3)
        + (1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3)
        + (1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2  & PS=2 & HL = 2 ->
        (1-HLCC)*HLSC: (HL'=2)&(phase'=3)
        +HLCC/2 :(HL'=0)&(phase'=3)
        + HLCC/2 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2  & PS=2 & HL = 3 ->
        (1-HLCC)*HLSC: (HL'=3)&(phase'=3)
        +HLCC/2 :(HL'=0)&(phase'=3)
        + HLCC/2 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2  & PS=2 & HL = 5 ->
        (1-HLCC)*HLSC: (HL'=5)&(phase'=3)
        +HLCC/2 :(HL'=0)&(phase'=3)
        + HLCC/2 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3);
```

```
                // planstep 3 (correct location 2, 4)
[] phase = 2 & PS=3 & HL = 0 ->
      (1-HLCC)*HLSC : (HL'=0)&(phase'=3)
      + HLCC/2 :(HL'=2)&(phase'=3)
      + HLCC/2 : (HL'=4)&(phase'=3)
      + (1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
      + (1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2 & PS=3 & HL = 1 ->
      (1-HLCC)*HLSC : (HL'=1)&(phase'=3)
      + HLCC/2 :(HL'=4)&(phase'=3)
      + HLCC/2 : (HL'=2)&(phase'=3)
      + (1-HLCC)*(1-HLSC)/3: (HL'=0)& (phase'=3)
      + (1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2  & PS=3 & ((HL)=(2)|(HL)=(4))->
      HLCC/2 : (HL'=2)&(phase'=3)
      + HLCC/2 :(HL'=4)&(phase'=3)
      + (1-HLCC)/4: (HL'=0)&(phase'=3)
      + (1-HLCC)/4: (HL'=1)&(phase'=3)
      + (1-HLCC)/4: (HL'=3)&(phase'=3)
      +(1-HLCC)/4: (HL'=5)&(phase'=3);

[] phase = 2 & PS=3 & HL = 3 ->
      (1-HLCC)*HLSC : (HL'=3)&(phase'=3)
      + HLCC/2 :(HL'=4)&(phase'=3)
      + HLCC/2 : (HL'=2)&(phase'=3)
      + (1-HLCC)*(1-HLSC)/3: (HL'=0)& (phase'=3)
      + (1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2  & PS=3  & HL = 5 ->
      (1-HLCC)*HLSC : (HL'=5)&(phase'=3)
      + HLCC/2 :(HL'=4)&(phase'=3)
      + HLCC/2 : (HL'=2)&(phase'=3)
      + (1-HLCC)*(1-HLSC)/3: (HL'=0)& (phase'=3)
      + (1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3);

//planstep 4 (correct location 1, 2, 4)
[] phase = 2  & PS=4 & HL = 0 ->
      (1-HLCC)*HLSC : (HL'=0)&(phase'=3)
```

```
        +HLCC/3 :(HL'=1)&(phase'=3)
        +HLCC/3 :(HL'=2)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=3)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=5)&(phase'=3);

[] phase =2 & PS=4 & (HL = 1 | HL = 2 | HL = 4) ->
        HLCC/3 :(HL'=1)&(phase'=3)
        +HLCC/3 :(HL'=2)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        + (1-HLCC)/3: (HL'=0)&(phase'=3)
        + (1-HLCC)/3: (HL'=3)&(phase'=3)
        + (1-HLCC)/3: (HL'=5)&(phase'=3);

[] phase = 2 & PS=4 & HL = 3 ->
        (1-HLCC)*HLSC : (HL'=3)&(phase'=3)
        +HLCC/3 :(HL'=1)&(phase'=3)
        +HLCC/3 :(HL'=2)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=1)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=5)&(phase'=3);

[] phase = 2  & PS=4 & HL = 5 ->
        (1-HLCC)*HLSC : (HL'=5)&(phase'=3)
        +HLCC/3 :(HL'=1)&(phase'=3)
        +HLCC/3 :(HL'=2)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=3)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=0)&(phase'=3);

//planstep 5 (correct location 1, 3, 4)
[] phase = 2 & PS=5 & HL = 0 ->
        (1-HLCC)*HLSC : (HL'=0)&(phase'=3)
        +HLCC/3 :(HL'=1)&(phase'=3)
        +HLCC/3 :(HL'=3)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=5)&(phase'=3);

[] phase = 2  & PS=5 & (HL = 1 | HL = 3 | HL = 4) ->
        HLCC/3 :(HL'=1)&(phase'=3)
        +HLCC/3 :(HL'=3)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        +(1-HLCC)/3: (HL'=0)&(phase'=3)
```

```
        +(1-HLCC)/3: (HL'=2)&(phase'=3)
        +(1-HLCC)/3: (HL'=5)&(phase'=3);

[] phase = 2  & PS=5 & HL = 2 ->
        (1-HLCC)*HLSC : (HL'=2)&(phase'=3)
        +HLCC/3 :(HL'=1)&(phase'=3)
        +HLCC/3 :(HL'=3)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=0)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=5)&(phase'=3);

[] phase = 2  & PS=5 & HL = 5 ->
        (1-HLCC)*HLSC : (HL'=5)&(phase'=3)
        +HLCC/3 :(HL'=1)&(phase'=3)
        +HLCC/3 :(HL'=3)&(phase'=3)
        + HLCC/3 : (HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=0)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/2: (HL'=2)&(phase'=3);

//planstep 6 (correct location 3, 4)
[] phase = 2  & PS=6 & HL = 0 ->
        (1-HLCC)*HLSC : (HL'=0)&(phase'=3)
        +HLCC/2 :(HL'=3)&(phase'=3)
        +HLCC/2 :(HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2 & PS=6 & HL = 1 ->
        (1-HLCC)*HLSC : (HL'=1)&(phase'=3)
        +HLCC/2 :(HL'=3)&(phase'=3)
        +HLCC/2 :(HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=0)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2 & PS=6 & HL = 2 ->
        (1-HLCC)*HLSC : (HL'=2)&(phase'=3)
        +HLCC/2 :(HL'=3)&(phase'=3)
        +HLCC/2 :(HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=0)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);
```

```
[] phase = 2  & PS=6 &(HL = 3 | HL = 4) ->
      HLCC/2 :(HL'=3)&(phase'=3)
      +HLCC/2 :(HL'=4)&(phase'=3)
      + (1-HLCC)/4: (HL'=0)&(phase'=3)
      + (1-HLCC)/4: (HL'=1)&(phase'=3)
      + (1-HLCC)/4: (HL'=2)&(phase'=3)
      + (1-HLCC)/4: (HL'=5)&(phase'=3);

[] phase = 2  & PS=6 & HL = 5 ->
      (1-HLCC)*HLSC : (HL'=5)&(phase'=3)
      +HLCC/2 :(HL'=3)&(phase'=3)
      +HLCC/2 :(HL'=4)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=0)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=1)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3);

//planstep 7 (correct location 1, 4)
[] phase = 2 & PS=7 & HL = 0 ->
      (1-HLCC)*HLSC  : (HL'=0)&(phase'=3)
      +HLCC/2 :(HL'=1)&(phase'=3)
      +HLCC/2 :(HL'=4)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2  & PS=7 & (HL = 1 | HL = 4) ->
      HLCC/2  :(HL'=1)&(phase'=3)
      +HLCC/2 :(HL'=4)&(phase'=3)
      + (1-HLCC)/4: (HL'=0)&(phase'=3)
      + (1-HLCC)/4: (HL'=2)&(phase'=3)
      + (1-HLCC)/4: (HL'=3)&(phase'=3)
      + (1-HLCC)/4: (HL'=5)&(phase'=3);

[] phase = 2  & PS=7 & HL = 2->
      (1-HLCC)*HLSC : (HL'=2)&(phase'=3)
      +HLCC/2 :(HL'=1)&(phase'=3)
      +HLCC/2 :(HL'=4)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=0)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3)
      +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

[] phase = 2 & PS=7 & HL = 3 ->
      (1-HLCC)*HLSC  : (HL'=3)&(phase'=3)
      +HLCC/2 :(HL'=1)&(phase'=3)
```

```
        +HLCC/2 :(HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=0)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=5)&(phase'=3);

    [] phase = 2  & PS=7 & HL = 5 ->
        (1-HLCC)*HLSC  : (HL'=5)&(phase'=3)
        +HLCC/2 :(HL'=1)&(phase'=3)
        +HLCC/2 :(HL'=4)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=0)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=2)&(phase'=3)
        +(1-HLCC)*(1-HLSC)/3: (HL'=3)&(phase'=3);

endmodule

module user_water_a
//hand not on tap, cannot change water flow:
    [] phase = 3 & (HL=0 | HL=2 | HL=3|HL=4|HL=5) -> (phase'=5);
//Water On:
    [] phase = 3 & action = 1 & HL = 1 & (PS = 0| PS = 4) ->
        WFE*WFCC : (WF'=1) & (phase' = 5)
        + (1-WFE*WFCC) : (WF'=0) & (phase' = 5);

    //[] phase = 3 & action =1 & HL=1 & (PS=1|PS=2|PS=3|PS=5|PS=6|PS=7|PS=8) ->(phase'=4);

//Water Off:
    [] phase = 3 & action = 2 & HL=1 & (PS = 5|PS=7)->
        WFE*WFCC : (WF'=0) &(phase'=5)
        +(1-WFE*WFCC) : (WF'=1) & (phase'=5);



//other actions:
    [] phase = 3 & HL=1 & (action = 0 | action= 3 | action = 4|action=5|action=6) ->(phase'=4);
endmodule

module updateWF_nothing
    //WF = 0 is correct for next step at plan step 6,7
    [] phase = 4 & (((PS)=(6)|(PS)=(7)))  ->
        WFCC : (WF'=0) & (phase'=5)
        +(1-WFCC) : (WF'=1) & (phase'=5);

    //WF = 1 is correct for next step at plan step 1,2,3,4
    [] phase = 4 & (((PS)=(1)|(PS)=(2)|(PS)=(3)|(PS)=(4)))->
        WFCC : (WF'=1 ) & (phase'=5)
```

```
        +(1-WFCC): (WF'=0) & (phase'=5);

    //either WF=1 or WF=0 are correct for next step at 0,5
    [] phase = 4 & (((PS)=(0)|(PS)=(5))) ->
        0.5: (WF'=1) & (phase'=5) + 0.5: (WF'=0) & (phase'=5);
endmodule

//MDP
module Policy
//action
action: [0..6] init 0;
//0   1 2  3   4    5  6
//none on off soap rinse wet dry

    //ps 0, on, soap
        [] phase = 0 & PS=0 -> (action'=1) & (phase'=1) &(NW'=0) ;
        [] phase = 0 & PS=0 -> (action'=3) & (phase'=1) &(NW'=0) ;
        [] phase = 0 & PS=0 -> (action'=0) & (phase'=1) ;
        //ps 1, wet, soap
        [] phase = 0 & PS=1 -> (action'=3) & (phase'=1) &(NW'=0) ;
        [] phase = 0 & PS=1 -> (action'=5) & (phase'=1) &(NW'=0) ;
        [] phase = 0 & PS=1 -> (action'=0) & (phase'=1) ;
        //ps 2 use soap
    [] phase = 0 & PS=2 -> (action'=3) & (phase'=1) &(NW'=0) ;
    [] phase = 0 & PS=2 -> (action'=0) & (phase'=1);
    //ps 3, rinse
    [] phase = 0 & PS = 3->  (action'=4) & (phase'=1)&(NW'=0) ;
    [] phase = 0 & PS = 3-> (action'=0) & (phase'=1);
    //ps 4, water on
    [] phase = 0 & PS = 4 ->(action'=1) & (phase'=1)&(NW'=0) ;
    [] phase = 0 & PS = 4 ->(action'=0) & (phase'=1);
    //ps 5, use towel or water off
        [] phase = 0 & PS=5 -> (action'=6) & (phase'=1)&(NW'=0) ;
    [] phase = 0 & PS=5 -> (action'=0) & (phase'=1);
    [] phase = 0 & PS=5 -> (action'= 2) & (phase'=1)&(NW'=0) ;
    //ps 6, use towel
        [] phase = 0 & PS=6-> (action'=6) & (phase'=1)&(NW'=0) ;
    [] phase = 0 & PS=6-> (action'=0) & (phase'=1);
    //ps 7, water off
    [] phase = 0 & PS=7-> (action' = 2) & (phase'=1)&(NW'=0) ;
    [] phase = 0 & PS=7-> (action'=0) & (phase'=1);
    //[] phase = 0 & PS=8 -> phase'=6;
endmodule
```

```
const double discount;

rewards  "total_2"
      phase = 0 & MPSrepeat = 0 & PS !=8 : 3*func(pow, discount,time_step);
      PS = 8 & MPS = 4  : 300*func(pow, discount, time_step);
            phase = 1 &  action = 0 : 5*func(pow, discount, time_step);
endrewards
```

Appendix II
The PRISM file of DTMC model for the hand-washing problem

```
dtmc

//scheduler
global phase : [0..6] init 0;


//HandLocation: HL
//soap, tap, water, towel, sink, away
global HL : [0..5] init 5;

//waterFlow: WF
//on, off
global WF : [0..1] init 0;

//handLocationCorrectConst
const double HLCC;

//handLocationSameConst
const double HLSC;

//waterFlowCorrectConst
const double WFCC;

//handLocation Effect
//HLCC/2 < HLE < 1
const double HLE;

const double WFE;

const int maxwait;

const int total_time;

//number of time intervals without plan step change or progress
global NW: [-1..maxwait] init 0;
```

```
module update_activity_a
//(same as in the MDP model, Appendix I, omitted here)
endmodule

module update_activity_b
//(same as in the MDP model, Appendix I, omitted here)
endmodule


module user_hand_a
//(same as in the MDP model, Appendix I, omitted here)
endmodule

module user_hand_b
//(same as in the MDP model, Appendix I, omitted here)
endmodule

module user_water_a
//(same as in the MDP model, Appendix I, omitted here)
endmodule

module updateWF_nothing
//(same as in the MDP model, Appendix I, omitted here)
endmodule

//probability to give the correct prompts
const double p_p=1;

//wait for a certain amount of time then give correct prompts with probability p_p
module Policy_1
//action
action: [0..6] init 0;
        [] phase = 0 & PS = 0 & NW = maxwait-> p_p/2: (action'=1) & (phase'=1)&(NW'=0) +
p_p/2: (action'=3) & (phase'=1)&(NW'=0) + (1-p_p): (action'=0) & (phase'=1);

        [] phase = 0 & PS = 1 & NW = maxwait-> p_p/2: (action'=3) & (phase'=1)&(NW'=0) +
p_p/2: (action'=5) & (phase'=1)&(NW'=0) + (1-p_p): (action'=0) & (phase'=1);

        [] phase = 0 & PS = 2 & NW = maxwait-> p_p: (action'=3) & (phase'=1) &(NW'=0)+ (1-p_p):
(action'=0) & (phase'=1);

        //ps 3, rinse
        [] phase = 0 & PS = 3 & NW = maxwait->p_p:  (action'=4) & (phase'=1)&(NW'=0)+ (1-p_p):
```

```
(action'=0) & (phase'=1);

        //ps 4, water on
        [] phase = 0 & PS = 4& NW = maxwait -> p_p: (action'=1) & (phase'=1)&(NW'=0)+ (1-p_p):
(action'=0) & (phase'=1);

        //ps 5, use towel or water off (which one is better depends on user behavior)
        [] phase = 0 & PS = 5 & NW = maxwait->p_p/2:  (action'=6) & (phase'=1)&(NW'=0)+ p_p/2:
(action' = 2) & (phase'=1)&(NW'=0)+  (1-p_p): (action'=0) & (phase'=1);

        //ps 6, use towel
        [] phase = 0 & PS = 6 & NW = maxwait->p_p:  (action'=6) & (phase'=1)&(NW'=0)+ (1-p_p):
(action'=0) & (phase'=1);

        //ps 7, water off
        [] phase = 0 & PS = 7 & NW = maxwait -> p_p: (action' = 2) & (phase'=1)&(NW'=0)+ (1-
p_p): (action'=0) & (phase'=1);

        [] phase = 0 & PS!= 8 & NW< maxwait -> (action'=0) & (phase'=1);
        //[] phase = 0 & PS = 8 -> phase'=6;

endmodule

const double discount;

rewards "positive"
    phase = 0 & MPSrepeat = 0 & PS!=8 : 3*func(pow, discount,time_step);
        PS=8 & MPS=4 : 200*func(pow, discount,time_step);
endrewards

rewards  "negative"
    phase = 1& action != 0 : 5*func(pow, discount,time_step);
endrewards


rewards  "total_2"
     phase = 0 & MPSrepeat = 0 & PS !=8 : 3*func(pow, discount,time_step);
     PS = 8 & MPS = 4  : 300*func(pow,discount,time_step);
     phase = 1 &  action = 0 : 5*func(pow, discount, time_step);
endrewards

rewards "time"
        phase = 5 : 1;
endrewards
```

Appendix III
The PRISM file of MDP model for the dressing problem

```
mdp

global phase: [0..4] init 0;
//const int maxwait;
const int total_time;
//global NW: [0..maxwait] init 0;

const int a;
const int rb;

module plan_step
PS: [0..9] init 0;
[]phase = 2 & PS = 0 & top = 0 -> (PS'=1) &( phase'=3);
[]phase = 2 & PS = 0 & top = 1 -> (PS'=0) & (phase'=3);
[]phase = 2 & PS = 0 & top = 2 -> (PS'=3) & (phase'=3);
[]phase = 2 & PS = 0 & top = 3 -> (PS'=4) & (phase'=3);

[]phase = 2 & PS = 1 & top = 0 -> (PS'=1) & (phase'=3);
[]phase = 2 & PS = 1 & top = 1 -> (PS'=0) & (phase'=3);
[]phase = 2 & PS = 1 & top = 2 -> (PS'=2) & (phase'=3);
[]phase = 2 & PS = 1 & top = 3 -> (PS'=5) & (phase'=3);

[]phase = 2 & PS = 2 & top = 0 -> (PS'=1) & (phase'=3);
[]phase = 2 & PS = 2 & top = 1 -> (PS'=7) & (phase'=3);
[]phase = 2 & PS = 2 & top = 2 -> (PS'=2) & (phase'=3);
[]phase = 2 & PS = 2 & top = 3 -> (PS'=9) & (phase'=3);

[]phase = 2 & PS = 3 & top = 0 -> (PS'=1) & (phase'=3);
[]phase = 2 & PS = 3 & top = 1 -> (PS'=0) & (phase'=3);
[]phase = 2 & PS = 3 & top = 2 -> (PS'=3) & (phase'=3);
[]phase = 2 & PS = 3 & top = 3 -> (PS'=6) & (phase'=3);

[]phase = 2 & PS = 4 & top = 0 -> (PS'=1) & (phase'=3);
[]phase = 2 & PS = 4 & top = 1 -> (PS'=0) & (phase'=3);
[]phase = 2 & PS = 4 & top = 3 -> (PS'=4) & (phase'=3);
```

```
[]phase = 2 & PS = 5 & top = 0 -> (PS'=1) & (phase'=3);
[]phase = 2 & PS = 5 & top = 3 -> (PS'=5) & (phase'=3);

[]phase = 2 & PS = 6 & top = 0 -> (PS'=1) & (phase'=3);
[]phase = 2 & PS = 6 & top = 1 -> (PS'=0) & (phase'=3);
[]phase = 2 & PS = 6 & top = 2 -> (PS'=3) & (phase'=3);
[]phase = 2 & PS = 6 & top = 3 -> (PS'=6) & (phase'=3);

[]phase = 2 & PS = 7 & top = 0 -> (PS'=1) & (phase'=3);
[]phase = 2 & PS = 7 & top = 1 -> (PS'=7) & (phase'=3);
[]phase = 2 & PS = 7 & top = 2 -> (PS'=2) & (phase'=3);
[]phase = 2 & PS = 7 & top = 3 -> (PS'=8) & (phase'=3);

[]phase = 2 & PS = 8 & top = 0 -> (PS'=1) & (phase'=3);
[]phase = 2 & PS = 8 & top = 1 -> (PS'=7) & (phase'=3);
[]phase = 2 & PS = 8 & top = 2 -> (PS'=2 )& (phase'=3);
[]phase = 2 & PS = 8 & top = 3 -> (PS'=8 )& (phase'=3);
endmodule

module update
MPS: [0..3] init 0;
MPSrepeat: [0..1] init 1;
time_step: [0..total_time] init 0;
[]phase = 3 & (PS= 0 |  PS =3 | PS=4 |PS=5 | PS=6| PS=7 | PS=8) -> (phase'=0) & (time_step' =
min(time_step +1, total_time)) & ( MPSrepeat'= 1);
[]phase = 3 & MPS=0 & PS=1 -> (MPS' = 1) & (phase'=0)& (time_step' = min(time_step +1,
total_time)) & (MPSrepeat'= 0);
[]phase = 3 & PS=1 & (MPS =1 | MPS = 2) -> (phase'=0) & (time_step' = min(time_step + 1,
total_time)) & (MPSrepeat'= 1) ;
[]phase = 3 & MPS=1 & PS=2 -> MPS' = 2 & (phase'=0) & (time_step' = min(time_step +1,
total_time)) & (MPSrepeat'= 0);
[]phase = 3 & MPS = 2  & PS=2 ->  (phase'=0) & (time_step' = min(time_step +1, total_time)) &
(MPSrepeat'= 1);
[]phase = 3 & PS=9 -> MPS' = 3;

endmodule

module user_action
//put on P, put on T, put on J, take off P, take off T, take off J,
//user_act: [0..5] init 0;
//none, P, T, J
top: [0..3] init 1;
//PS=0
```

```
[]phase =1 & PS = 0 & prompt=0 ->
        a/10: (top'=0) & (phase'=2)
        + (1-a/10)/3: (top'=1) & (phase'=2)
        + (1-a/10)/3: (top'=2) & (phase'=2)
        + (1-a/10)/3: (top'=3) & (phase'=2);

[]phase =1 & PS = 0 & prompt=3 ->
        r*a/100: (top'=0) & (phase'=2)
        + (1-r*a/100)/3: (top'=1) & (phase'=2)
        + (1-r*a/100)/3: (top'=2) & (phase'=2)
        + (1-r*a/100)/3: (top'=3) & (phase'=2);
//PS=1
[]phase =1 & PS = 1 & prompt=0 ->
        a/10: (top'=2) & (phase'=2)
        + (1-a/10)/3: (top'=0) & (phase'=2)
        + (1-a/10)/3: (top'=1) & (phase'=2)
        + (1-a/10)/3: (top'=3) & (phase'=2);

[]phase =1 & PS = 1 & prompt=1 ->
        r*a/100: (top'=2) & (phase'=2)
        + (1-r*a/100)/3: (top'=0) & (phase'=2)
        + (1-r*a/100)/3: (top'=1) & (phase'=2)
        + (1-r*a/100)/3: (top'=3) & (phase'=2);
//PS=2
[]phase =1 & PS = 2 & prompt=0 ->
        a/10: (top'=3) & (phase'=2)
        + (1-a/10)/3: (top'=0) & (phase'=2)
        + (1-a/10)/3: (top'=1) & (phase'=2)
        + (1-a/10)/3: (top'=2) & (phase'=2);

[]phase =1 & PS = 2 & prompt=2 ->
        r*a/100: (top'=3) & (phase'=2)
        + (1-r*a/100)/3: (top'=0) & (phase'=2)
        + (1-r*a/100)/3: (top'=1) & (phase'=2)
        + (1-r*a/100)/3: (top'=2) & (phase'=2);
//PS=3
[]phase =1 & PS = 3 & prompt=0 ->
        a/20: (top'=0) & (phase'=2)
        + a/20: (top'=1) & (phase'=2)
        + (1-a/10)/2: (top'=2) & (phase'=2)
        + (1-a/10)/2: (top'=3) & (phase'=2);

[]phase =1 & PS = 3 & prompt=4 ->
        r*a/200: (top'=1) & (phase'=2)
```

```
        + a/20: (top'=0) & (phase'=2)
        + (1-r*a/200-a/20)/2: (top'=1) & (phase'=2)
        + (1-r*a/200-a/20)/2: (top'=3) & (phase'=2);

//PS=4
[]phase =1 & PS = 4 & prompt=0 ->
        a/20: (top'=0) & (phase'=2)
        + a/20: (top'=1) & (phase'=2)
        + 1-a/10: (top'=3) & (phase'=2);

[]phase =1 & PS = 4 & prompt=5 ->
        r*a/200: (top'=1) & (phase'=2)
        + a/20: (top'=0) & (phase'=2)
        + (1-r*a/200-a/20): (top'=3) & (phase'=2);

//PS=5
[]phase =1 & PS = 5 & prompt=0 ->
        a/10: (top'=0) & (phase'=2)
        + (1-a/10): (top'=3) & (phase'=2);

[]phase =1 & PS = 5 & prompt=5 ->
        r*a/100: (top'=0) & (phase'=2)
        + (1-r*a/100): (top'=3) & (phase'=2);
//PS=6
[]phase =1 & PS = 6 & prompt=0 ->
        a/30: (top'=0) & (phase'=2)
        + a/30: (top'=1) & (phase'=2)
        + a/30: (top'=2) & (phase'=2)
        + (1-a/10): (top'=3) & (phase'=2);

[]phase =1 & PS = 6 & prompt=5 ->
        r*a/300: (top'=2) & (phase'=2)
        + a/30: (top'=0) & (phase'=2)
        + a/30: (top'=1) & (phase'=2)
        + (1-r*a/300-2*a/30): (top'=3) & (phase'=2);
//PS=7
[]phase =1 & PS = 7 & prompt=0 ->
        a/20: (top'=0) & (phase'=2)
        + a/20: (top'=2) & (phase'=2)
        + (1-a/10)/2: (top'=1) & (phase'=2)
        + (1-a/10)/2: (top'=3) & (phase'=2);

[]phase =1 & PS = 7 & prompt=3 ->
        r*a/200: (top'=2) & (phase'=2)
```

```
        + a/20: (top'=0) & (phase'=2)
        + (1-r*a/200-a/20)/2: (top'=1) & (phase'=2)
        + (1-r*a/200-a/20)/2: (top'=3) & (phase'=2);

//PS=8
[]phase =1 & PS = 8 & prompt=0 ->
        a/30: (top'=0) & (phase'=2 )
        + a/30: (top'=1) & (phase'=2)
        + a/30: (top'=2) & (phase'=2)
        + (1-a/10): (top'=3) &( phase'=2);

[]phase =1 & PS = 8 & prompt=5 ->
        r*a/300: (top'=1) & (phase'=2)
        + a/30: (top'=0) & (phase'=2)
        + a/30: (top'=1) & (phase'=2)
        + (1-r*a/300-2*a/30): (top'=3) &( phase'=2);

endmodule

module policy_mdp
r: [11..13] init 11;

// nothing, put on T, put on J, take off P, take off T, take off J
//0       1       2       3       4       5
prompt: [0..5] init 0;
[]phase = 0 & PS = 0 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=0 -> prompt'=3 & phase'=1 & r'=rb;
[]phase = 0 & PS=0 -> prompt'=3 & phase'=1 & r'=rb+1;

[]phase = 0 & PS = 1 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=1 -> prompt'=1 & phase'=1 & r'=rb;
[]phase = 0 & PS=1 -> prompt'=1 & phase'=1 & r'=rb+1;


[]phase = 0 & PS =2 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=2 -> prompt'=2 & phase'=1& r'=rb;
[]phase = 0 & PS=2 -> prompt'=2 & phase'=1 & r'=rb+1;


[]phase = 0 & PS = 3 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=3 -> prompt'=4 & phase'=1& r'=rb;
[]phase = 0 & PS=3 -> prompt'=4 & phase'=1 & r'=rb+1;
```

```
[]phase = 0 & PS = 4 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=4 -> prompt'=5 & phase'=1& r'=rb;
[]phase = 0 & PS=4 -> prompt'=5 & phase'=1 & r'=rb+1;


[]phase = 0 & PS = 5 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=5 -> prompt'=5 & phase'=1& r'=rb;
[]phase = 0 & PS=5 -> prompt'=5 & phase'=1 & r'=rb+1;


[]phase = 0 & PS = 6 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=6 -> prompt'=5 & phase'=1& r'=rb;
[]phase = 0 & PS=6 -> prompt'=5 & phase'=1 & r'=rb+1;


[]phase = 0 & PS = 7 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=7 -> prompt'=3 & phase'=1& r'=rb;
[]phase = 0 & PS=7 -> prompt'=3 & phase'=1 & r'=rb+1;


[]phase = 0 & PS = 8 -> prompt'=0 & phase'=1;
[]phase = 0 & PS=8 -> prompt'=5 & phase'=1& r'=rb;
[]phase = 0 & PS=8 -> prompt'=5 & phase'=1 & r'=rb+1;

endmodule


const double discount=0.82;

rewards " positive"
PS=9 & MPS=2 : 300*func(pow, discount, time_step);
phase=3 & MPSrepeat = 0 & PS!=9: 3*func(pow, discount,time_step);
endrewards

rewards "negative"
phase=0 & prompt!=0: 5*func(pow, discount, time_step);
endrewards

rewards "total"
      phase = 0 & MPSrepeat = 0& prompt=0 & PS !=9: 3*func(pow, discount, time_step);
      PS = 9 & MPS = 2  : 300*func(pow, discount,  time_step);
            phase = 1 & prompt=0 : 5*func(pow, discount, time_step)
endrewards
```

Appendix IV
A Java program to modify the optimal policy generated by PRISM

```
import java.io.*;
import java.util.Hashtable;
import java.util.Enumeration;
import java.util.LinkedList;
import java.util.StringTokenizer;

public class stationary {
	static Hashtable<String, tr> tra;
	static LinkedList<st> sta;
	private stationary(){
		tra = new Hashtable<String, tr>();
		sta = new LinkedList<st>();
	}

	public static void main(String[] args){
		new stationary();
		String line = read_tra();
		read_sta();
		System.out.println(Integer.toString(tra.size()));

		for (int i=0; i<sta.size(); i++){
			if (sta.get(i).time_step == 5 & tra.contains(sta.get(i).name)){
				int goalstate = tra.get(sta.get(i)).st2;
				for (int j=0; j<sta.size(); j++){
					if (st.same(sta.get(i), sta.get(j))&
tra.contains(sta.get(j).name)){
						tra.get(sta.get(j).name).st2 = goalstate;
					}
				}
			}
		}

		write_tra(line);
		System.exit(0);
	}
```

```
public static void write_tra(String first){
        try{
                BufferedWriter out = new BufferedWriter(new FileWriter("test.tra"));

                out.write(first);
                for (Enumeration<tr> e = tra.elements() ; e.hasMoreElements() ;) {

                        out.write('\n');
                        tr temp = new tr();
                        temp = e.nextElement();
            out.write(Integer.toString(temp.st1)+' '
                        +Integer.toString(temp.st2)+' '
                        + Double.toString(temp.p)+'\r');
                }
           out.close();
        }catch (Exception e){
                System.out.println(e);
        }
        System.out.println("writeOK");
}

public static String read_tra()         {
        String first = null;
        try {
        BufferedReader input = null;
        //read file
        input = new BufferedReader(
                        new FileReader(
                                        "C:\\Users\\workspace\\tra.txt"));
        String line = null;
        //a while loop to read all lines in this file

        first = input.readLine();
        while ((line = input.readLine()) != null) {
                                                //to read one line
                StringTokenizer StringTokens = new StringTokenizer(line);
                String name = null;
                tr a = new tr();
                        name = StringTokens.nextToken();
                        a.st1=Integer.parseInt(name);
                        a.st2 = Integer.parseInt(StringTokens.nextToken());
                        a.p =  Double.parseDouble(StringTokens.nextToken());
                tra.put(name,a);
```

```
		}

	} catch (FileNotFoundException e) {
		System.out.println(e.getMessage());
	} catch (IOException e) {
		System.out.println(e.getMessage());
	}
	System.out.println("tra ok");
	return first;

	}

	public static void read_sta(){
		try {
		BufferedReader input = null;
		//read file
		input = new BufferedReader(
				new FileReader(
						"C:\\Users\\workspace\\sta.txt"));
		String line = null;
		//a while loop to read all lines in this file

		input.readLine();

		while ((line = input.readLine()) != null) {
								//to read one line
			//StringTokenizer StringTokens = new StringTokenizer(line);

			st a = new st();
			String name = "";

			int i=0;
			while ( line.charAt(i)!= ':'){
				name = name + line.charAt(i);
				i++;
			}
			i++;
			i++;
			String temp = "";
			while ( line.charAt(i)!= ','){

				temp = temp + line.charAt(i);
				i++;
			}
```

```
//System.out.println(temp);
if (Integer.parseInt(temp) == 0){
		a.name= name;
		i++;
		temp = "";
		while ( line.charAt(i)!= ','){
			temp = temp + line.charAt(i);
			i++;
		}
		a.a = Integer.parseInt(temp);

		i++;
		temp = "";
		while ( line.charAt(i)!= ','){
			temp = temp + line.charAt(i);
			i++;
		}
		a.rb = Integer.parseInt(temp);

		i++;
		temp = "";
		while ( line.charAt(i)!= ','){
			temp = temp + line.charAt(i);
			i++;
		}
		a.PS = Integer.parseInt(temp);

		i++;
		temp = "";
		while ( line.charAt(i)!= ','){
			temp = temp + line.charAt(i);
			i++;
		}
		a.MPS = Integer.parseInt(temp);

		i++;
		temp = "";
		while ( line.charAt(i)!= ','){
			temp = temp + line.charAt(i);
			i++;
		}
		a.MPSrepeat = Integer.parseInt(temp);

		i++;
```

```
                        temp = "";
                        while ( line.charAt(i)!= ','){
                                temp = temp + line.charAt(i);
                                i++;
                        }
                        a.time_step = Integer.parseInt(temp);

                        i++;
                        temp ="";
                        while ( line.charAt(i)!= ','){
                                temp = temp + line.charAt(i);
                                i++;
                        }
                        a.top = Integer.parseInt(temp);

                        i++;
                        temp ="";
                        while ( line.charAt(i)!= ','){
                                temp = temp + line.charAt(i);
                                i++;
                        }
                        a.r = Integer.parseInt(temp);

                        i++;
                        temp = "";
                        while ( line.charAt(i)!= ')'){
                                temp = temp + line.charAt(i);
                                i++;
                        }
                        a.prompt = Integer.parseInt(temp);

                        sta.add(a);
                }
        }

        } catch (FileNotFoundException e) {
                System.out.println(e.getMessage());
        } catch (IOException e) {
                System.out.println(e.getMessage());
        }
        System.out.println("read sta ok");
        }

}
```