

# Decision Making in Uncertain Real-World Domains Using DT-Golog \*

**Mikhail Soutchanski and Huy Pham**

Department of Computer Science  
Ryerson University  
245 Church Street, ENG281  
Toronto, Ontario, M5B 2K3, Canada  
{mes, hpham}@scs.ryerson.ca

**John Mylopoulos**

Department of Computer Science,  
University of Toronto,  
Toronto, Ontario, M5S 3G4, Canada  
jm@cs.toronto.edu

## Abstract

DTGolog, a decision-theoretic agent programming language based on the situation calculus, was proposed to ease some of the computational difficulties associated with Markov Decision Processes (MDPs) by using natural ordering constraints on execution of actions. Using DTGolog, domain specific constraints on a set of policies can be expressed in a high-level program to reduce significantly computations required to find a policy optimal in this set. We explore whether the DTGolog framework can be used to evaluate different designs of a decision making agent in a large real-world domain. Each design is understood as combination of a template (expressed as a Golog program) for available policies and a reward function. To evaluate and compare alternative designs we estimate the probability of goal satisfaction for each design. As a domain, we choose the London Ambulance Service (LAS) case study that is well known in software engineering, but remains unknown in AI. We demonstrate that DTGolog can be applied successfully to quantitative evaluation of alternative designs in terms of their ability to satisfy a system goal with a high probability. We provide a detailed axiomatization of the domain in the temporal situation calculus with stochastic actions. The main advantage of this representation is that neither actions, nor states require explicit enumeration. We do an experimental analysis using an on-line implementation of DTGolog coupled with a simulator that models real time actions of many external agents.

## Introduction and Motivation

There is a significant amount of work done in AI related to planning under uncertainty when a certain high level goal must be satisfied with a high probability (Hanks & McDermott 1994; Kushmerick, Hanks, & Weld 1995; Haddawy, Doan, & Goodwin 1995; Boutilier, Dean, & Hanks 1999; Majercik & Littman 2003; Madani, Hanks, & Condon 2003; Younes *et al.* 2005). Most of the proposed approaches are related to solving a decision-theoretic planning problem in relatively small domains (with less than  $10^9$  states). However, there are many practical domains where the task of designing a decision making agent (that guarantees goal satisfaction with a sufficiently high probability) is difficult due to a very large number of the state features and (ground) actions with uncertain effects. In these domains, the main

problem is that state of the art planners cannot scale up to compute (or approximate) an optimal policy due to extremely large size of the state space. Even the task of computing the value of a single policy can be prohibitively difficult in these domains. The second common problem is that in some domains the goal of interest is characterized in terms of quality of an on-going process driven by external agents. To deal with the first problem (scalability), one can try to use a logical representation that avoids explicit state and action enumeration. In addition, one can try to elaborate alternative designs of a decision making agent by goal-means analysis, e.g., using goal regression, a mechanism well studied in AI (Waldinger 1977) (see also (Manna & Waldinger 1987)). By careful refining a goal that must be (at least partially) satisfied into sub-goals, and then by identifying sub-tasks to solve and primitive actions that must be executed to solve these sub-tasks, it is possible to ease to some degree the computational burden of designing such an agent. Indeed, a gradual refinement process can identify useful sequences, loops, conditional or recursive structures of actions that provide together important constraints on the set of policies that need to be considered, and as a consequence, significantly reduce the number of potential policies that ever need to be analyzed. One can imagine also that such analysis can identify where search between alternative actions must concentrate: this can be indicated by nondeterministic choices between actions. In realistic domains, this refinement process can lead to different designs depending on how stakeholder goals will be captured in this process (i.e., how they will be modeled using reward functions) and depending on the degree of nondeterminism. Because a variety of designs will need precise evaluation and comparison, the problem of designing a decision making agent can be reduced to quantitative evaluation of those different designs of an agent which have been elaborated during the goals-means refinement process. To deal with the second problem (representation of an ongoing interaction with external agents), one can build a simulator of exogenous actions and evaluate all identified alternative designs with respect to the same simulator.

We discuss an expressive framework that provides a tool for reasoning about different designs. This framework, a decision-theoretic extension of Golog (DTGolog), was first introduced in the context of designing efficient controllers for mobile robots (Boutilier *et al.* 2000; Soutchanski 2001). Later, it was extended to the multi-person games (Finzi & Lukasiewicz 2004), was successfully adapted to program robots playing soccer (Ferrein, Fritz, & Lakemeyer 2005),

\*A preliminary short version of this paper has been accepted (with another title) as a poster by the 17th European Conference on AI, Riva del Garda, Italy (Aug 28th - Sept 1st, 2006) and will be also published in the proceedings of the ECAI06 W/Sh on Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds.

and was also extended with preferences to personalize Semantic Web services (Fritz & McIlraith 2006). DTGolog is somewhat similar in spirit to Hierarchical Task Network (HTN), but relies on problem-specific search control knowledge expressed as Golog programs. In addition, HTN plans can be encoded in generalized form as Golog programs, and consequently, one can build domain-specific (but task independent) Golog programs similar to HTN “methods” (task-decomposition templates) (Sirin *et al.* 2004).

All previous approaches applied DTGolog to the task of computing a policy in a finite horizon decision-theoretic planning problem. To the best of our knowledge, there were no attempts to use the DTGolog framework as a tool for quantitative evaluation of alternative designs of a decision making agent functioning in a large-scale domain characterized by on-going interaction with many external agents. DTGolog is a natural choice for this type of problems because domain specific constraints elaborated during the refinement process can be easily expressed in Golog. Golog provides all standard programming constructs as well as several non-deterministic choice constructs that can be used to specify alternative decisions to be resolved at the moment of decision making. Goals of stakeholders can be captured using rewards functions, but because mapping of qualitative goals of many agents to quantitative rewards is not unique and can lead to several reward functions, the decision maker is faced with several different designs that represent goals of stakeholders differently. The semantics of DTGolog (based on directed value iteration) guarantees that the nondeterministic choices (if any), mentioned in a program, will be resolved to compute an optimal policy (optimal in the set of all policies that are specified by a Golog program). We would like to do extensive analysis of applicability of DTGolog to evaluation of designs using a real case study (each design is represented as a combination of a Golog program and a reward function). In particular, we explore a well-known case study London Ambulance Service Computer Aided Dispatch System (LAS-CAD) that received a significant attention in the software engineering literature, but remains unknown to the AI community (The Communications Directorate 1993; Kramer & Wolf 1996; Letier & van Lamsweerde 2000; 2004). It is an excellent example of a problem with probabilistic goals. This case study comes from an investigation into a software development project that failed in October 1992 with dramatic consequences. We suggest this case study as a grand challenge for research on planning under uncertainty.

We try to keep our presentation as generic as possible to indicate how our modeling framework can be applied or extended to other decision-making environments. In particular, we clearly show the main features of DTGolog model that can be applied to other domains as well (Reiter 2001; Soutchanski 2003): actions, fluents, precondition axioms, successor-state axioms, initial database, transition probabilities, rewards, Golog procedures for expressing natural constraints on decision making. We illustrate all these main features of the specification framework using the case study.

The main contributions of our paper are the following. First, we developed an extensive logical formalization of

a non-trivial domain. Second, we demonstrated that DTGolog is well suited to the task of evaluation of alternative designs of a decision making agent. Third, we did experimental analysis of three different designs of a decision making agent using the same simulator for a fair comparison. The methodology for quantitative evaluation of designs (in terms of probability of goal satisfaction) has been proposed in (Letier & van Lamsweerde 2004), but it is not based on MDPs and it has never been implemented. All the other previous work in requirements engineering related to the LAS case study, approached this case study in qualitative terms only (Kramer & Wolf 1996; Wang & Lespérance 2001; You 2003).

## Background

### Markov Decision Processes

Formally speaking, a fully observable MDP  $M = \langle \mathcal{S}, \mathcal{A}, \text{Pr}, R \rangle$  comprises the following components.  $\mathcal{S}$  is a finite set of *states* of the system being controlled. The state changes over time, possibly in response to the occurrence of exogenous actions (including nature’s actions) and/or actions on the part of the decision maker. A finite set  $\mathcal{A}$  is a set of *actions* which influence the system state. Dynamics are given by  $\text{Pr} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ ; here  $\text{Pr}(s_i, a, s_j)$  denotes the probability that action  $a$ , when executed at state  $s_i$ , induces a transition to  $s_j$ .<sup>1</sup>  $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  (or simply  $R : \mathcal{S} \rightarrow \mathbb{R}$ ) is a real-valued, bounded *reward function*:  $R(s)$  is the instantaneous reward an agent receives for entering state  $s$ ;  $R(s, a)$  is the reward for entering  $s$  after doing  $a$ .<sup>2</sup> The process is *fully observable* if the agent cannot predict with certainty the state that will be reached when an action is taken, but it can observe that state precisely once it is reached. To implement full observability assumption the agent executes sensing actions that return information about the current state. We assume that sensing actions have no cost and can always be executed.

To choose actions the agent follows some *policy*. A policy specifies the decision rule (a rule for choosing actions) to be used at each stage. A deterministic Markovian (memoryless) stationary policy is a function  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  mapping the state space  $\mathcal{S}$  into the action space  $\mathcal{A}$ , with the meaning that for each state  $s$ ,  $\pi(s)$  denotes the action prescribed by this policy in state  $s$ . For an MDP with a finite horizon  $H$ , a *nonstationary* policy  $\pi : \mathcal{S} \times \{1, \dots, H\} \rightarrow \mathcal{A}$  associates with each state  $s$  and stage-to-go  $n \leq H$  an action  $\pi(s, n)$  to be executed at  $s$  with  $n$  stages remaining.

The *decision problem* faced by the agent in an MDP is to adopt a course of action  $\pi$  that maximizes expected reward accumulated by implementing that course of action over some horizon of interest. The *expected value* of policy  $\pi$  depends on the specific objectives. A *finite-horizon* decision problem with horizon  $H$  measures the value of  $\pi$  as  $V^\pi(s) = E_\pi(\sum_{n=0}^H R(s_n) \mid s_0 = s)$ , where  $E_\pi$  represents the conditional expectation, given that policy  $\pi$  is

<sup>1</sup>In general case, different actions can be available at different states:  $\mathcal{A}_s$  denotes the set of feasible actions for state  $s$ .

<sup>2</sup>Rewards can be both positive and negative; in the latter case, they can be understood as costs.

employed and the notation  $V^\pi(s)$  indicates that the *value* of a policy  $\pi$  depends on the state  $s$  in which the process begins. In a discounted infinite horizon MDP,  $V^\pi(s) = E_\pi(\sum_{n=0}^{\infty} \gamma^n R(s_n) \mid s_0 = s)$ , where  $0 \leq \gamma < 1$  is a discount factor. The function  $V^\pi$  is called the *state-value function* for policy  $\pi$ .

A policy  $\pi^*$  is *optimal* if, for all  $s \in \mathcal{S}$  and all policies  $\pi$ , we have  $V^{\pi^*}(s) \geq V^\pi(s)$ . There is always at least one policy that is better than or equal to all other policies. Optimal policies share the same state-value function, called the *optimal state-value function*, denoted  $V^*$ , and defined as  $V^*(s) = \max_{\pi} V^\pi(s)$ , for all  $s \in \mathcal{S}$  (in other words,  $V^*$  exists and is unique). It can be shown that optimal courses of action are Markovian policies: nonstationary Markovian policies for finite-horizon problems, and stationary deterministic Markovian policies for infinite-horizon problems.

The problem of finding an optimal policy for a given MDP is called the decision-theoretic planning problem. First, consider fixed, finite number of decision-making stages  $H$ . A simple algorithm for constructing optimal policies is *value iteration* (Puterman 1994). Define the  $n$ -stage-to-go value function  $V_n$  by setting  $V_0(s_i) = R(s_i)$  and, for all  $1 \leq n \leq H$ :

$$V_n(s_i) = R(s_i) + \max_{a \in \mathcal{A}} \left\{ \sum_{s_j \in \mathcal{S}} \Pr(s_i, a, s_j) V_{n-1}(s_j) \right\} \quad (1)$$

The computation of  $V_n(s_i)$  given  $V_{n-1}(s_j)$  is known as a Bellman backup and the equation (1) is called the Bellman optimality equation for a finite number of decision stages  $H$ . Using this equation, one can compute in sequence the optimal state value functions up to the horizon  $H$  of interest. Once this sequence of the value functions is computed, one can easily find an *optimal* policy. By setting  $\pi(s_i, n)$  to the action  $a$  maximizing the right-hand term, the resulting policy  $\pi$  will be optimal. In general case, an optimal policy is nonstationary because of finite horizon. Second, consider the problem of finding an optimal policy in a discounted infinite horizon MDP (when the number of decision stages is not given or not limited). In this case, an optimal policy can be determined from solving a similar system of Bellman optimality equations but with rewards discounted using  $\gamma$ . There are two most well known dynamic programming algorithms that compute an optimal policy for an MDP with an infinite horizon: policy iteration and value iteration (Bertsekas & Tsitsiklis 1996; Puterman 1994).

MDPs are well-known and widely used models for decision-theoretic planning, but traditional framework requires explicit state and action enumeration, which grow exponentially with the number of domain features. For this reason, traditional algorithms for computing optimal policies become unusable as soon as the number of features used to represent a state is large and each feature can take several different values. On the other hand, many application problems involving decision-theoretic planning can be viewed in terms of logical propositions, random variables, relations between objects, etc. Logical representations have significant advantage over traditional “flat” state-based representations because they can allow compact specifications of the system.

For this reason, many researchers studied how to represent and solve large state space MDPs using alternative representation techniques. Most of these representations and related developments of efficient methods for solving MDPs are reviewed in (Boutilier, Dean, & Hanks 1999). In this paper, we consider an approach to representation of MDPs in the predicate logic developed in (Boutilier *et al.* 2000; Reiter 2001; Soutchanski 2003).

However, before we review this logical approach to representation of an MDP, we would like to mention *decision tree search-based methods* for solving MDPs. Subsequently, we propose to use an elaborated version of a decision tree search-based method for our purposes. When the system is known to start in a given state  $s_0$ , the reachability structure of the MDP can be exploited, restricting value and policy computations to states reachable by some sequence of actions from  $s_0$ . This form of *directed value iteration* can be effected by building a search tree rooted at state  $s_0$ : its successors at level 1 of the tree are possible actions; the successors at level 2 of any action node are those states that can be reached with nonzero probability when that action is taken at state  $s_0$ ; and deeper levels of the tree are defined recursively in the same way. For an MDP with finite horizon  $k$ , the tree is built to level  $2k$ : the value of any state is given by the maximum among all values of its successor actions, and the value of an action is given by the expected value of its successor states. (States at the leaves are assigned their reward value.) It is easy to see that the value  $V(s_0)$  of the state  $s_0$  at the root of a tree with  $2n$  levels is precisely  $V_n(s_0)$  defined in the equation (1) for value iteration. This observation provides the basis for the well known relationships between heuristic search techniques and other dynamic programming algorithms (Boutilier, Dean, & Hanks 1999). To reduce the computational burden of computing an optimal policy in an unrestricted decision tree, we rely on the approach (see below) that proposes to find the optimal policy consistent with the constraints imposed by a program written in a general programming language Golog (Boutilier *et al.* 2000; Soutchanski 2003). *Interleaving* decision-theoretic planning with execution of actions is another key idea important to circumvent computational difficulties associated with decision tree search algorithms (Barto, Bradtke, & Singh 1995; Dearden & Boutilier 1997). This key idea is incorporated into the Golog approach of providing constraints on the set of policies by means of a special programming construct which limits the scopes where planning should be undertaken. In the next section, we review a predicate logic approach to representation of an MDP. In the subsequent section, we review Golog and then we review a decision-theoretic extension of Golog.

## Situation Calculus

The situation calculus is a predicate logic language for axiomatizing dynamic worlds. In recent years, it has been considerably extended beyond the original language to include stochastic actions, concurrency, continuous time, etc, but in all cases, its basic ingredients consist of *actions*, *situations* and *fluents* (Levesque, Pirri, & Reiter 1998; Reiter 2001). Fluents (playing the role of state features) are understood as

properties whose value can vary from situation to situation depending on the executed actions.

*Actions* are first-order terms consisting of an action function symbol and its arguments. In the approach to representing time in the situation calculus of (Reiter 1998; 2001), one of the arguments to such an action function symbol—typically, its last argument—is the time of the action’s occurrence. For example,  $request(loc(3, 8), 105.7)$  might denote the action of forwarding an emergency request received from location  $loc(3, 8)$  at time 105.7 (e.g., measured in seconds) to a resource allocator (we assume that the term  $loc(x, y)$  represents locations on a x,y-grid). Following Reiter, all actions are instantaneous (i.e., with zero duration).<sup>3</sup> In most cases, agent argument of actions remains suppressed, but in the representation of an application domain there is an implicit distinction between agent’s actions and exogenous actions (they are not under direct control of the agent). For example, if we are interested only in the actions of the resource allocator (RA), then the action  $request(loc(3, 8), 5.7)$  is an exogenous (with respect to the RA) action that can be executed by an incident reviewer. Exogenous actions can include actions executed by nature. For example, when the RA decides to mobilize the ambulance  $car$  to the incident at location  $loc(3, 8)$  at time  $t$ , this can result in two possible outcomes:  $mobilizeS(car, loc(3, 8), t)$ , successful mobilization, or  $mobilizeF(car, loc(3, 8), t)$ , failed mobilization (e.g., due to failed communication link between the RA and the ambulance  $car$ ). Both these two actions are exogenous because nature decides which one of them will actually be executed when the RA tries to mobilize a car (this is represented by  $mobilize(car, l, t)$ ). However,  $mobilize(car, l, t)$  is not an action term, but is defined using an abbreviation that relates this agent action to a finite set of nature’s actions. The domain axiomatization includes definitions relating such abbreviations (e.g.,  $mobilize(car, l, t)$ ) with the finite set of possible outcomes (e.g.,  $mobilizeS(car, l, t)$  and  $mobilizeF(car, l, t)$ ). This is accomplished using a defined symbol  $choice(A) \stackrel{def}{=} \{N_1, \dots, N_m\}$  meaning that nature’s actions  $N_1, \dots, N_m$  are the only outcomes of the stochastic agent action  $A$  in every situation. This key representational trick allows to apply all reasoning mechanisms developed for the original situation calculus (with the deterministic actions only) to the case when outcomes of some agent’s actions are actually determined by nature (see details in (Boutilier *et al.* 2000; Reiter 2001; Soutchanski 2003)). In addition to physical actions, that change some properties of the world, there are also sensing actions that change what the agent knows about the world. For example, the action  $askPosition(22, l, 567)$  is a sensing action executed by the RA at time 567 that determines the location  $l$  of the ambulance with the ID 22.

A *situation* is a first-order term denoting a sequence of actions. These sequences are represented using a binary function symbol  $do$ :  $do(\alpha, s)$  denotes the sequence resulting from adding the action term  $\alpha$  to the sequence  $s$ . The special constant  $S_0$  denotes the *initial situation*, namely the

<sup>3</sup>Durations can be captured using processes. A full exposition of time is not possible here.

empty action sequence. Therefore, the situation term

$$do(mobilizeS(c, loc(3, 8), 29), do(request(loc(3, 8), 5.7), do(wait(2), S_0)))$$

denotes the following sequence of actions:  $wait(2)$ ,  $request(loc(3, 8), 5.7)$ ,  $mobilizeS(c, loc(3, 8), 29)$ . Notice that the action sequence is obtained from a situation term by reading the term from right to left. Foundational axioms for situations with time are given in (Reiter 1998; 2001).

Relations or functions whose values vary from situation to situation are called *fluents*, and are denoted by predicate or function symbols whose last argument is a situation term. For example,  $requestPending(l, s)$  might be a relational fluent, meaning that after executing the action sequence  $s$ , there is a pending emergency request from the location  $l$  (the truth value of this fluent might change depending on what action was executed last). To simplify presentation we do not consider functional fluents in this paper.

A domain theory is axiomatized in the situation calculus with four classes of axioms (see (Reiter 2001) for details). We give examples of these axioms in the case study section .

**Action precondition axioms:** There is one for each action term  $A(\vec{x})$ , with syntactic form  $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$ . Here,  $\Pi_A(\vec{x}, s)$  is a situation calculus formula with free variables among  $\vec{x}, s$ . These are the preconditions of action  $A$ :  $A$  is possible if and only if the condition  $\Pi_A(\vec{x}, s)$  is true.

**Successor state axioms:** There is one for each relational fluent  $F(\vec{x}, s)$ , with syntactic form  $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$ , where  $\Phi_F(\vec{x}, a, s)$  is a formula with free variables among  $a, s, \vec{x}$  having the syntactic form

$$a = PositiveAction \wedge \gamma^+(\vec{x}, s) \vee F(\vec{x}, s) \wedge \neg(a = NegativeAction \wedge \gamma^-(\vec{x}, s)),$$

where *PositiveAction* is an action that has positive effect on the fluent  $F$ ,  $\gamma^+(\vec{x}, s)$  is the formula expressing a context in which this positive effect can occur, and, similarly, *NegativeAction* is an action that can make the fluent  $F$  false if the formula  $\gamma^-(\vec{x}, s)$  holds in the situation  $s$ . These characterize the truth values of the fluent  $F$  in the next situation  $do(a, s)$  in terms of the current situation  $s$ , and they embody a solution to the frame problem for deterministic actions (Reiter 1991) (recall, that the solution of the frame problem is a computationally efficient way of describing all things that do not change after doing an action). There are similar axioms for functional fluents, but we shall not be using these in this paper, so we omit them.

**Unique names axioms for actions:** These state that the actions of the domain are pairwise unequal.

**Initial database:** This is a set of sentences whose only situation term is  $S_0$ ; it specifies the initial problem state (e.g., the beginning of a work shift for ambulance crews and the control staff). For example, it can include sentences saying which ambulances are located where, whether they are ready or not, etc.

## Golog

Golog (Levesque, Pirri, & Reiter 1998; Levesque *et al.* 1997; Reiter 2001) is a situation calculus-based programming language for defining complex actions in terms of a

set of primitive actions axiomatized in the situation calculus as described above. It has the standard—and some not so standard—control structures found in most Algol-like languages.

1. *Sequence*:  $\alpha_1 ; \alpha_2$ . Do action  $\alpha_1$ , followed by  $\alpha_2$ .
2. *Test actions*:  $\phi?$  Test the truth value of expression  $\phi$  in the current situation.
3. *Nondeterministic action choice*:  $\alpha_1 | \alpha_2$ . Do  $\alpha_1$  or  $\alpha_2$ .
4. *Nondeterministic choice of arguments*:  $(\pi x)\alpha$ . Nondeterministically pick a value for  $x$ , and for that value of  $x$ , do action  $\alpha$ .
5. Conditionals (*if-then-else*) and *while* loops.
6. *Procedures, including recursion*.

The semantics of Golog programs is defined by macro-expansion, using a ternary relation *Do*.  $Do(\delta, s, s')$  is an *abbreviation* for a situation calculus formula whose intuitive meaning is that  $s'$  is one of the situations reached by evaluating the program  $\delta$  beginning in situation  $s$ . Given a program  $\delta$ , one *proves*, using the situation calculus axiomatization of the background domain, the formula  $(\exists s)Do(\delta, S_0, s)$  to compute a plan. Any binding for  $s$  obtained by a constructive proof of this sentence is a legal execution trace, in terms of the primitive actions, of  $\delta$ . A Golog interpreter for the situation calculus with time, written in Prolog, is described in (Reiter 1998; 2001).

A detailed example of using a version of Golog for modeling purposes is considered in (Lespérance *et al.* 1999). Another example is discussed in the case study section below.

## Decision-Theoretic Golog

We now turn our attention to DTGolog, a decision-theoretic extension of the Golog framework that allows one to specify MDPs in a first-order language, and provide “advice” in the form of high-level programs that constrain the search for good or optimal policies. Such a program can be viewed as a partially-specified policy: its semantics can be viewed, informally, as the execution of the program that has highest expected value. DTGolog offers a synthesis of both decision-theoretic planning and programming, and is in fact general enough to accommodate both extremes. One can write purely nondeterministic programs that allow an agent to solve an MDP optimally, or purely deterministic programs that leave no decisions in the agent’s hands whatsoever.

To extend Golog to decision-theoretic contexts, where the effects of actions are uncertain and objectives are formulated using reward functions, generalizations of both the domain representation and Golog are required. First, the representational methodology for actions must be extended to allow the concise representation of rewards, probabilities and sensing-related conditions that need to be evaluated after doing sensing actions (we need sensing to implement a fully observable MDP). Second, the interpreter must be generalized so that the search is not for any execution of a program, but rather for the execution of the program that has highest expected utility—though this notion is somewhat complicated in the context of executing a program as we will see below. We discuss each of these generalizations in turn.

The specification of an MDP requires the provision of a background *action theory*—as described above—augmented with additional classes of definitions and a background *optimization theory*—consisting of the specification of a reward function and some optimality criterion (here we require only a horizon  $H$ ). The following predicates are used in additional classes of definitions that must be provided to represent an MDP (see (Soutchanski 2003) for details): (1) a defined symbol  $prob(n, p, s)$  represents the probability  $p$  of nature’s action  $n$  in situation  $s$ , and (2) the predicate  $senseCond(n, \phi)$  specifying the test condition  $\phi$  that needs to be evaluated to identify which outcome  $n$  of the last stochastic action actually occurred, (3) a defined symbol  $reward(r, do(a, s))$  to represent rewards and costs  $r$  as functions of the current situation  $do(a, s)$ , the action  $a$  or both. Examples of all these axioms are provided below in Section .

In what follows, we assume that we have been provided with an extended basic action theory and optimization theory. We interpret DTGolog programs relative to this theory. DTGolog programs are written using the same program operators as Golog programs. DTGolog programs can also use one additional operator that is called *nondeterministic finite choice of action arguments*:  $(\pi(x : \tau)\delta)$  means nondeterministically choose an element  $x$  from the finite set  $\tau = \{c_1, \dots, c_n\}$  and for that  $x$  do the program  $\delta$  (see details below).

The semantics is specified in a similar fashion, with the predicate *IncrBestDo* playing the role of *Do*. Specifically, the semantics of a DTGolog program is defined by a predicate  $IncrBestDo(\delta_1, s, h, \delta_2, \pi, v, pr)$ , where  $\delta_1$  is a given Golog program,  $s$  is a starting situation,  $\pi$  is the optimal policy determined by program  $\delta_1$ ,  $v$  is the total accumulated expected value of that policy,  $pr$  is the probability that  $\pi$  will execute successfully,<sup>4</sup>  $h$  is a specified horizon and  $\delta_2$  is the program that remains to be executed after doing the first action from the policy  $\pi$ .<sup>5</sup> (The expected values are computed in the usual way by multiplying rewards with probabilities of outcomes). Generally, an interpreter implementing this definition will be called with a given program  $\delta_1$ , situation  $S_0$ , and a horizon  $h$ , and the arguments  $delta_2$ ,  $\pi$ ,  $v$  and  $pr$  will be instantiated by the interpreter. We simply call  $\pi$  the optimal policy whenever it is clear from the context that  $\pi$  is a policy optimal with respect to a given program and a finite horizon.

Note that different policies are characterized not only by their values, but also by their success probabilities. To compare them, it is assumed that a predicate  $\leq$  is available that compares pairs of the form  $\langle p, v \rangle$ , where  $p$  is a success probability and  $v$  is an expected value. How one defines this predicate will depend crucially on how one interprets the concept of advice embodied in a program. One natural implementation is to use a lexicographic preference where

<sup>4</sup>It is determined by all branches of the given program whose execution do not fail, i.e., by branches where all test expressions evaluate to true and all actions are possible.

<sup>5</sup>We use  $\pi$  to denote both a policy and nondeterministic choice operator. The meaning should always be clear from the context.

$\langle p_1, v_1 \rangle < \langle p_2, v_2 \rangle$  whenever  $p_1 = 0$  and  $p_2 > 0$  (so an agent cannot choose a policy that guarantees abnormal termination). If both  $p_1$  and  $p_2$  are zero, or both are greater than zero, then the  $v$ -terms are used for comparison.

We conclude this section with a semantics of a few operators. We include only axioms for nondeterministic choice because they are essential for understanding how the interpreter finds an optimal policy.

**The nondeterministic choice of two programs.**

$$\begin{aligned} \text{IncrBestDo}((\delta_1 \mid \delta_2); \gamma, s, h, \delta_r, \pi, v, p) &\stackrel{def}{=} h > 0 \wedge \\ \exists(\pi_1, v_1, p_1, \delta_r^1). \text{IncrBestDo}(\delta_1; \gamma, s, h, \delta_r^1, \pi_1, v_1, p_1) &\wedge \\ \exists(\pi_2, v_2, p_2, \delta_r^2). \text{IncrBestDo}(\delta_2; \gamma, s, h, \delta_r^2, \pi_2, v_2, p_2) &\wedge \\ ((p_1, v_1) \leq (p_2, v_2) \wedge \pi = \pi_2 \wedge \delta_r = \delta_r^2 \wedge v = v_2 \wedge p = p_2) &\vee \\ ((p_1, v_1) > (p_2, v_2) \wedge \pi = \pi_1 \wedge \delta_r = \delta_r^1 \wedge v = v_1 \wedge p = p_1). & \end{aligned}$$

Given the choice between two subprograms  $\delta_1$  and  $\delta_2$ , the optimal policy is determined by that subprogram with optimal execution. This is achieved by looking ahead up to the end of each of the two branches, and finding which branch is optimal;  $(p_1, v_1) < (p_2, v_2)$  is defined above.

**Nondeterministic finite choice of action arguments.**

If the program begins with  $(\pi(x : \tau)\delta); \gamma$ , the finite nondeterministic choice followed sequentially by a sub-program  $\gamma$ , the finite set  $\tau = \{c_1, \dots, c_n\}$ , and the choice binds all free occurrences of  $x$  in  $\delta$  to one of these elements, then:

$$\begin{aligned} \text{IncrBestDo}((\pi(x : \tau)\delta); \gamma, s, \delta_{rem}, h, \pi, v, pr) &\stackrel{def}{=} h > 0 \wedge \\ \text{IncrBestDo}((\delta|_{c_1}^x \mid \dots \mid \delta|_{c_n}^x); \gamma, s, \delta_{rem}, h, \pi, v, pr) & \end{aligned}$$

where  $\delta|_c^x$  means substitution of  $c$  for all free occurrences of  $x$  in  $\delta$ . Thus, the optimal policy  $\pi$  corresponds to the element  $c$  in  $\tau$  that delivers the best execution. The remaining program  $\delta_{rem}$  is the same on the both sides of the definition.

**Nondeterministic choice of arguments.**

$$\begin{aligned} \text{IncrBestDo}((\pi x)\delta(x); \gamma, s, \delta_{rem}, h, \pi, v, pr) &\stackrel{def}{=} h > 0 \wedge \\ (\exists x). \text{IncrBestDo}(\delta(x); \gamma, s, \delta_{rem}, h, \pi, v, pr) & \end{aligned}$$

This is a *non-decision-theoretic version* of nondeterministic choice: pick an argument and compute an optimal policy given this argument. We need this operator because it will be convenient to choose values of variables that satisfy certain conditions, to choose moments of time and values returned from sensors. Note that in Golog, this operator is an operator for choosing one of the alternatives, but in DTGolog it is used only for programming purposes, and not for decision making.

A detailed inductive definition of *IncrBestDo* on the structure of a given program  $\delta$ , a discussion of its semantics, off-line and on-line DTGolog interpreters are provided in (Soutchanski 2003). On-line interpreter uses the off-line *IncrBestDo* interpreter, but it interleaves off-line planning with on-line execution of the first action from the computed policy. In the finite horizon case, this execution obviously terminates after  $H$  steps, but in the infinite horizon case, this execution is an ongoing processes interleaved with occurrences of exogenous actions.

## A Case Study: London Ambulance Service Computer Aided Dispatch (LAS-CAD) System

In this section, we would like to describe a very large real-world domain where there are many stakeholders and the system goal that must be satisfied with a high probability. Subsequently, we provide a detailed logical formalization of this domain, consider three alternative designs of a decision making agent and discuss how they can be quantitatively evaluated using DTGolog. Our formalization of LAS follows (The Communications Directorate 1993), because this is the only source of information available to us and, in most cases, we make only those assumptions which are (implicitly or explicitly) stated in this report.

As described in (The Communications Directorate 1993), the job cycle of LAS comprises the following phases. (1) Call taking, reviewing and prioritization: a Call Taker gets a 999 emergency phone call requesting an ambulance service and writes down all necessary details including the location of a patient; subsequently, an Incident Reviewer (IR) removes any duplicated requests and assigns priorities. (2) Decision making: depending on the location of the request, the IR forwards each request to one of the three Resource Allocators (RA), each is in charge of one of the three London's city regions (north-west, north-east and south-west), who decides which available ambulance should be sent to serve the incident. (3) Dispatch: once the decision has been made, it will be passed on to a Dispatcher (DSP), who will communicate the mobilization instruction to the appropriate ambulance crew. (4) Mobilization: an ambulance vehicle can be mobilized either from its home base, or from a hospital, or on the road (e.g., using radio link) when a vehicle that completed a previous request is going back to its home base. (5) Travel to scene: an ambulance vehicle (from now on, we call it a car for brevity) travels as quickly as possible to the incident. (6) At scene: upon arrival, an ambulance crew should notify the DSP (e.g., by pressing buttons on the mobile terminal inside the ambulance); then, they perform on-site diagnosis. After doing a diagnosis, the crew decides whether take a patient to a hospital or not. If not, then the car returns back to its home base. (7) Travel to a hospital: if a patient needs hospitalization, then the ambulance vehicle travels to a hospital (we assume that the ambulance always travels to the hospital of that region where the patient is currently located). (8) Hand over at a hospital: after spending some time on handing a patient over to a hospital staff, the car reports that it is ready for new assignments and starts going back to its home base (this may require crossing a border between regions, if the ambulance happens to be at a hospital of another region).

One of the most important objectives of the LAS is that requests to be served within 14 minutes from the time the call is received. More specifically, according to the government targets for response times, the activation process (i.e., call taking and mobilization decision) should be less than 3 minutes, and the travel time to the incident should be, for 95% of the time, less than 11 minutes and, for 50% of the time, less than 8 minutes (The Communications Directorate 1993). Clearly, realization of this objective depends cru-

cially on the RA’s decision making strategy that can depend on the following factors: whether ambulances are allowed to cross borders between regions or not, whether the same ambulance crew can be consecutively mobilized to serve incidents without having a rest at a base, whether there is information about current locations of all available ambulances, what criteria are used to choose an ambulance, etc. We would like to show that DTGolog is a framework that is expressive enough to provide quantitative evaluation of quality of alternative designs. However, we would like to emphasize that LAS is a very complicated case study involving multiple agents. For this reason, we decide to supplement DTGolog with an elaborated (but conceptually straightforward) simulator that is responsible for generating all exogenous actions and for modeling behavior of ambulances (using assumptions formulated below). The simulator (Pham, Soutchanski, & Mylopoulos 2006) also plays an important role in collecting statistics. Recall that in a single agent case, an offline DTGolog interpreter computes not only a policy (optimal in the set of policies satisfying a given Golog program) and its value. It computes also the probability that an optimal policy successfully follows constraints imposed by a Golog program: an optimal policy, its probability of success and its value are determined by the same reasoning process. However, in a multi-agent system like LAS, this cannot be done directly. For this reason, our simulator provides a probabilistic model that we use for quantitative comparisons. All alternative designs are evaluated with respect to the same probabilistic model to guarantee a fair comparison. Finally, there is another subtle difficulty that has to be mentioned. In the real system, only partial information is available to the RA due to communication failures and other factors. Because we use fully observable MDPs, we introduce additional states that represent lack of information to deal with this complication; e.g., we say, that a location of a car is “unknown”, or a sensing action returned the value “unclear”.

We will model the three city regions (we use constants NW, NE and S to name them), using three rectangular grid worlds  $10 \times 10$ . In each grid world, each cell represents a city block (an unique location), and is denoted by a term  $loc(x, y)$ , where  $x$  and  $y$  are the coordinates. All locations in the city will be referred to by the corresponding cells in which they reside, and the distance between any two locations,  $loc(x_1, y_1)$  and  $loc(x_2, y_2)$ , is defined as the Manhattan distance between the two:  $d = |x_2 - x_1| + |y_2 - y_1|$ . We assume that each region has one ambulance station (or just base for short), one hospital, and 10 ambulances (or just cars for short). The figure 1 below provides an illustration. We skip other details of our grid-world representation, but they are intuitively clear and the interested reader can find them online at (Pham, Soutchanski, & Mylopoulos 2006). It is important to understand that the size of the state space is well beyond  $30^{300} \cdot 2^{300}$  states, there are many actions in every state (which car should go to an emergency call) and, consequently, the exact solution of the problem of optimal ambulance allocation to incidents is computationally intractable. Moreover, even the task of evaluating policies on this huge state space is computationally intractable. However, we will

see below that using DTGolog we can evaluate reasonably interesting domain specific programs by taking advantage of natural constraints on the decision making process in this domain.

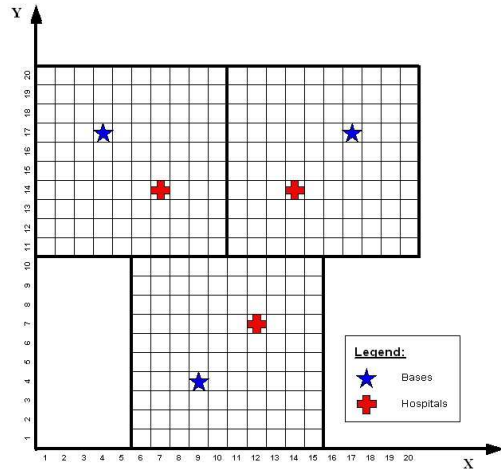


Figure 1: A grid-world model of the city.

To model, and simulate, the emergency servicing trips, we will make the following assumptions.

- The traveling speed of the ambulance in an emergency mode (i.e., when it is going to an incident, or when it is taking the patient to a hospital) is higher than that of in a normal mode (i.e., when it is going back to its home base).
- When the ambulance is outside of its home region, its speeds (both emergency and normal) are slower, due to crew members unfamiliarity with the region.
- The average amounts of time it takes to perform on-site diagnosis (*diagnosis time* from now on), and to carry the patient from an ambulance to a hospital (*unloading time* from now on) are known and the same in all regions.
- Weary crews work more slowly. Both diagnosis time and unloading time, as well as traveling time are longer in the case of consecutive trip(s), if the ambulance crew did not have any rest between trips, in comparison to amounts of time spent on average by a fresh crew.
- The rate at which emergency requests are received (*request rate* from now on) is known. (We simulate the request arrivals using a Poisson distribution.)
- The percentage at which patients need to be taken to hospital (*hospitalize rate* from now on) is a constant.
- The communication between the central ambulance control station (where the RA works) and ambulances is unreliable and it is modeled using an uniform distribution. However, each car can be reached reliably at the base.

The logical statements that capture these parameters are:  $avgTimePerBlockEmergHome(100)$  – and  $avgTimePerBlockNormHome(200)$  – home region, emergency (normal, respectively) speed per city block in seconds;  $avgTimePerBlockEmergForeign(150)$  – and  $avgTime-$

*PerBlockNormForeign*(250) – foreign region, emergency (normal, respectively) travel time per city block in seconds; *diagTime*(240) – 4 minutes, and *unloadTime*(120) – 2 minutes; *tirednessLagTime*(100) – extra time required for a fatigued crew; *requestRate*(150) – 1 request every 150 seconds, *hospitalizeRate*(0.8) – in the 80% of calls, a patient needs hospitalization. The rate at which communication fails, given the car is not at the base is represented by the predicate *commFailRate*(0.15).

We provide our logical representation of the domain in the situation calculus (Reiter 2001) as follows: first, we describe all agents, second, we describe their actions, third, we formulate all fluents, fourth, we give precondition axioms for all actions, fifth, all successor state axioms, sixth, theory to represent an MDP (specifically, probabilities of transitions and reward functions), and finally, we mention what logical statements are included in the initial database.

## Domain Representation

### Agents and actions

There are many roles in the real LAS system. Focusing on just the resource allocating and scheduling aspect of the system, however, only three roles are of significance: the IR, the RA, and the DSP (using the abbreviations from above). At the central position of the system is the RA. For simplicity, we assume that there is only one RA for the whole system. His job is to make resource allocation decisions in such a way that ambulances will arrive within the specified time limit (11 minutes) with a high probability. We will assume that in doing his job, the RA is continuously performing one of the following four actions.

*mobilize*( $c, loc, t$ ) – Send the ambulance  $c$  to  $loc$  at time  $t$ .<sup>6</sup> This is a stochastic action axiomatized as  $choice(mobilize(car, loc, t)) = \{mobilizeS(car, loc, t), mobilizeF(car, loc, t)\}$ . The first outcome *mobilizeS* corresponds to successful mobilization, the second outcome *mobilizeF* corresponds to failed mobilization (e.g., due to communication problems).

*askPosition*( $c, l, t$ ) – A sensing agent action that, if performed at time  $t$ , will tell the RA the location  $l$  of car  $c$ . To represent that this action can fail at the current communication failure rate, the simulator can return the special location *Unclear* (it is used later in the axioms).

*askStatus*( $car, status, t$ ) – Another agent sensing action that determines whether *car* is *Busy*, *Ready*, or if the current status is *Unknown* (if communication fails).

*wait*( $t$ ) – A no-cost deterministic agent action that can be performed whenever the RA has nothing to do.

### Exogenous actions

The front-end (i.e., call taking, etc) part of the system can be completely summarized and represented by the IR, because, from the point of view of the RA, this is where all emergency requests come from. We will assume that in doing his job, the IR will perform just one action:

<sup>6</sup>In the real LAS, performing this action involves not only the RA who notifies the DSP, but also the DSP who informs the appropriate ambulance crew (we gloss over details to simplify our model).

*request*( $l, t$ ) – Forward a reviewed incident request to the RA.

The back-end of the system, on the other hand, is completely summarized and represented by the DSP, since he handles all mobilization-related communications and ambulance activity reports. We will assume that, in doing his job, the DSP will perform these three actions.

*reportArrival*( $car, l, t$ ) – Report about arrival of an ambulance  $car$  to the RA. This action will tell the RA that  $car$  has arrived at location  $l$  at time  $t$ .

*reportReady*( $c, l, t$ ) – Report about readiness of an ambulance  $c$  to the RA. This action will tell the RA that  $c$  has become ready at location  $l$  at time  $t$ .

*reportLocation*( $car, loc, t$ ) – Inform about the current location  $loc$  of  $car$  at time  $t$ .

Since we consider here a DTGolog approach that accounts for a single decision maker only, we represent the behavior of the RA using a Golog program composed from his agent actions only, while treating the IR and DSP as external agents whose behaviors are simulated (using programs written in C). Note that the external agents (i.e., the IR and DSP) perform their actions in an exogenous fashion: their actions can happen any time and are outside of the direct control of our Golog program representing the RA.

To capture logical properties of the domain (state features) we introduce the following predicates.<sup>7</sup>

### Fluents

*ready*( $c, s$ ) - a car  $c$  is ready in situation  $s$

*carLocKnown*( $c, t, s$ ) - the location of a car  $c$  is known in situation  $s$  at time  $t$ .

*carLocation*( $c, l, t, s$ ) -  $c$  is located at  $l$  at time  $t$  in  $s$

*commLost*( $c, s$ ) - communication with a car  $c$  is not available in situation  $s$

*requestPending*( $l, s$ ) - there is an emergency request from a location  $l$  in situation  $s$

*atBase*( $c, s$ ) - a car  $c$  is at its home base in situation  $s$

*consecTripCount*( $c, n, s$ ) -  $c$  made  $n$  consecutive trips in  $s$

### Situation Independent Predicates:

*region*( $name, id, loc(x_1, y_1), loc(x_2, y_2)$ ) - the region  $name$  has a bottom left corner at  $loc(x_1, y_1)$  and a top right corner at  $loc(x_2, y_2)$ , e.g., the north-west region is described by *region*(*NW*, 1,  $loc(1, 11)$ ,  $loc(10, 20)$ ).

*base*( $name, id, region, l$ ) - the base  $name$  with an  $id$  is located in  $region$  at the location  $l$ , e.g., *base*(*B1*, 1, *NW*,  $loc(7, 14)$ ).

*carRange*( $region, rName$ ) -  $rName$  is the name of the set of all cars from  $region$ , e.g., *carRange*(*NW*, *NWcars*).

*range*( $rName, f$ ) -  $f$  is the finite set named  $rName$ , *range*(*NWcars*, { $c1, c2, c3, c4, c5, c6, c7, c8, c9, c10$ }).

There is also a finite set named *All* that includes all 30 cars.

There are also several other predicates with an obvious implementation and with the meaning that is easy

<sup>7</sup>As usual in predicate logic, names starting with low-case letters are variables that are implicitly universally quantified in all axioms, and names starting with upper-case letters are constants.



to understand from their names:  $isARegion(reg)$ ,  $isAHospital(h)$ ,  $isABase(b)$ ,  $isACar(c)$ ,  $validXCoord(region, x)$ ,  $validYCoord(region, y)$ ,  $hospital(name, id, region, l)$ ,  $homeRegion(car, reg)$ ,  $homeBase(car, base)$ ,  $in(base, reg)$ ,  $in(hosp, reg)$ ,  $in(loc, reg)$ ,  $inSameRegion(a, b)$ ,  $locOf(base, loc)$ ,  $locOf(hosp, loc)$ ; see details in (Pham, Soutchanski, & Mylopoulos 2006).

### Action precondition axioms

$$\begin{aligned} & Poss(wait(t), s) \\ & Poss(mobilizeS(car, loc, t), s) \equiv \\ & \quad ready(car, s) \wedge carLocKnown(car, t, s) \\ & Poss(mobilizeF(car, loc, t), s) \equiv \\ & \quad ready(car, s) \wedge carLocKnown(car, t, s) \\ & Poss(askPosition(car, l, t), s) \\ & Poss(askStatus(car, status, t), s) \end{aligned}$$

### Successor state axioms

A car is ready if it is reported that it is ready, or if the RA asked about the current status of the car and the result of this sensing action was that it is *Ready*, or the car is ready in the previous situation  $s$  and the last action is not a sensing action informing the RA that the car is busy or its status is unknown or not a mobilization action.

$$\begin{aligned} ready(car, do(a, s)) & \equiv (\exists l, t) a = reportReady(car, l, t) \vee \\ & (\exists t) a = askStatus(car, Ready, t) \vee \\ & ready(car, s) \wedge \neg(\exists l, t) (a = mobilizeS(car, l, t) \vee \\ & \quad a = askStatus(car, Busy, t) \vee \\ & \quad a = askStatus(car, Unknown, t)). \end{aligned}$$

Communication between ambulance crews and the DSP (and hence the RA) can fail. We model this by allowing the sensing action  $askPosition(car, l, t)$  to return the constant *Unclear* instead of a genuine location. More specifically, communication with a given car is said to be lost if: the RA tried to ask for its location or for its status and the reply was unclear, and the car has not get back to the RA since then (i.e., the DSP did not execute  $reportReady$  or  $reportArrival$  to pass information from the ambulance crew to the RA). In addition, if the mobilization fails, this indicates that communication is lost.

$$\begin{aligned} commLost(car, do(a, s)) & \equiv \\ & (\exists t) a = askPosition(car, Unclear, t) \vee \\ & (\exists t) a = askStatus(car, Unknown, t) \vee \\ & (\exists t, loc) a = mobilizeF(car, loc, t) \vee \\ & commLost(car, s) \wedge \neg(\exists l, t) (a = reportReady(car, l, t) \vee \\ & \quad a = reportArrival(car, l, t)). \end{aligned}$$

When a car is stationary (e.g., parking at the home base), its location is known. When the car is on the move, its location changes, and therefore becomes unknown. However, we assume that within a certain validity period  $p$ , which we represent as  $validPeriod(p)$ , the car's location can be considered unchanged (since it did not move very far from its last known location) and therefore its location is known. In addition, if the car location is known in  $s$  at  $time$ , it remains known, if this car was not mobilized successfully (to an incident) more than  $p$  seconds ago, i.e., at a moment  $t$  such that  $t < time - p$ .

$$\begin{aligned} carLocKnown(c, time, do(a, s)) & \equiv \\ & (\exists l, t) (a = reportReady(c, l, t) \wedge t \leq time \wedge \\ & \quad (\exists n, id, r) base(n, id, r, l)) \vee \\ & (\exists l, t, p) (a = reportReady(c, l, t) \wedge t \leq time \leq t + p \wedge \\ & \quad \neg(\exists n, id, r) base(n, id, r, l) \wedge validPeriod(p)) \vee \\ & (\exists l, t, p) (a = askPosition(c, l, t) \wedge t \leq time \wedge \\ & \quad (\exists n, id, r) base(n, id, r, l)) \vee \\ & (\exists l, t, p) (a = askPosition(c, l, t) \wedge t \leq time \leq t + p \wedge \\ & \quad \neg(\exists n, id, r) base(n, id, r, l) \wedge \neq Unclear \wedge validPeriod(p)) \vee \\ & carLocKnown(c, time, s) \wedge \neg(\exists t, l, p) (a = mobilizeS(c, l, t) \\ & \quad \wedge validPeriod(p) \wedge time - t > p) \end{aligned}$$

Similar to the previous axiom,  $l$  is the car  $c$  location at  $time$  in situation  $do(a, s)$ , if the last executed action  $a$  is that the crew of the ambulance  $c$  reported (via DSP) about readiness at the home base, or the crew is reported that it is ready elsewhere (the simulator makes sure that the car can report its readiness only at a hospital, at a location in the city, if a patient does not need hospitalization, or at the home base) and started to move towards the base and less than  $p$  seconds passed since the moment when information about  $l$  was received. Also, the location is  $l$  if the RA asked the crew of  $c$  recently about its position and got a clear reply, or the location was  $l$  in the previous situation  $s$  and the car was not successfully mobilized more than  $p$  seconds ago (i.e., either it is not moving, or if it is moving, then it started less than  $p$  seconds ago).

$$\begin{aligned} carLocation(c, l, time, do(a, s)) & \equiv \\ & (\exists t) (a = reportReady(c, l, t) \wedge t \leq time \wedge \\ & \quad (\exists n, id, r) base(n, id, r, l)) \vee \\ & (\exists l, t, p) (a = reportReady(c, l, t) \wedge t \leq time \leq t + p \wedge \\ & \quad \neg(\exists n, id, r) base(n, id, r, l) \wedge validPeriod(p)) \vee \\ & (\exists l, t, p) (a = askPosition(c, l, t) \wedge t \leq time \wedge \\ & \quad (\exists n, id, r) base(n, id, r, l)) \vee \\ & (\exists l, t, p) (a = askPosition(c, l, t) \wedge t \leq time \leq t + p \wedge \\ & \quad \neg(\exists n, id, r) base(n, id, r, l) \wedge \neq Unclear \wedge validPeriod(p)) \vee \\ & (\exists l', t, p) (a = mobilizeS(c, l', t) \wedge \\ & \quad validPeriod(p) \wedge time - t > p) \wedge l = Unknown \vee \\ & carLocation(c, l, time, s) \wedge \neg(\exists l', t, p) (a = mobilizeS(c, l', t) \wedge \\ & \quad validPeriod(p) \wedge time - t > p)). \end{aligned}$$

An emergency request from the location  $l$  for an ambulance service is pending in  $do(a, s)$  if an emergency call is made from  $l$ , or if in the previous situation  $s$ , the request was pending and no ambulance is mobilized to this location  $l$ .

$$\begin{aligned} requestPending(l, do(a, s)) & \equiv (\exists t) a = request(l, t) \vee \\ & \quad \neg(\exists c, t) a = mobilizeS(c, l, t) \wedge requestPending(l, s)). \end{aligned}$$

The ambulance car  $c$  is at its home base, if the crew of  $c$  (via DSP) reported to the RA from the location  $l$  that it is ready, and  $b$  is the home base of  $c$  and  $b$  is located at  $l$ , or the RA asked the crew of  $c$  about their location and got a reply that  $c$  is at the home base, or in the previous situation  $s$ ,  $c$  is at the home base and it is not mobilized successfully to serve an incident.

$$\begin{aligned} atBase(c, do(a, s)) & \equiv (\exists l, t, b) (a = reportReady(c, l, t) \wedge \\ & \quad homeBase(c, b) \wedge locOf(b, l)) \vee \\ & (\exists l, t, b) (a = askPosition(c, l, t) \wedge \\ & \quad homeBase(c, b) \wedge locOf(b, l)) \vee \\ & atBase(c, s) \wedge \neg(\exists l, t) a = mobilizeS(c, l, t). \end{aligned}$$

An ambulance  $c$  made  $n$  consecutive trips in  $do(a, s)$ , if

it made  $n - 1$  consecutive trips in  $s$  and is successfully mobilized again, or if it made  $n$  consecutive trips in  $s$  and it is neither mobilized, nor reported that it is ready at its home base. Once the crew of  $c$  reports that  $c$  is ready at its home base, the number of consecutive trips is 0.

$$\begin{aligned} \text{consecTripCount}(c, n, \text{do}(a, s)) &\equiv \\ (\exists l, t, b)(a = \text{reportReady}(c, l, t) \wedge \\ &\quad n = 0 \wedge \text{homeBase}(c, b) \wedge \text{locOf}(b, l)) \vee \\ (\exists l, t) a = \text{mobilizeS}(c, l, t) \wedge \text{consecTripCount}(c, n-1, s) \vee \\ \text{consecTripCount}(c, n, s) \wedge \neg(\exists l, t, b)(a = \text{mobilizeS}(c, l, t) \vee \\ &\quad a = \text{reportReady}(c, l, t) \wedge \text{homeBase}(c, b) \wedge \text{locOf}(b, l)). \end{aligned}$$

### Initial Situation

In the initial situations, all ambulances are ready, they are at home bases and their locations are known:

$$\begin{aligned} \text{ready}(c, S_0) &\equiv \text{isACar}(c) \\ \text{carLocKnown}(c, t_2, S_0) &\equiv \text{isACar}(c) \wedge \text{start}(S_0, t_1) \wedge t_2 \geq t_1 \\ \text{carLoc}(c, l, t_2, s_0) &\equiv \text{isACar}(c) \wedge \text{homeBase}(c, b) \wedge \\ &\quad \text{locOf}(b, l) \wedge \text{start}(S_0, t_1) \wedge t_2 \geq t_1 \\ \text{atBase}(c, S_0) &\equiv \text{isACar}(c) \end{aligned}$$

The basic action theory for the LAS domain includes also unique names axioms for actions: they state that all physical and sensing actions (both agent and exogenous) are pairwise unequal. We need several additional groups of axioms to specify an MDP: probabilities of transitions in an MDP for all stochastic agent actions, reward functions and sense conditions that need to be evaluated to distinguish between different outcomes of every stochastic action.

Probabilities of outcomes are defined as follows:

$$\begin{aligned} \text{prob}(\text{mobilizeS}(c, l, t), p, s) &\stackrel{\text{def}}{=} \text{carLocation}(c, l, t, s) \wedge \\ ((\exists n, id, r) \text{base}(n, id, r, l) \wedge p = 1 \vee \\ &\quad \neg(\exists n, id, r) \text{base}(n, id, r, l) \wedge \text{commFailRate}(f) \wedge p = 1-f). \\ \text{prob}(\text{mobilizeF}(c, l, t), p, s) &\stackrel{\text{def}}{=} \text{carLocation}(c, l, t, s) \wedge \\ ((\exists n, id, r) \text{base}(n, id, r, l) \wedge p = 0 \vee \\ &\quad \neg(\exists n, id, r) \text{base}(n, id, r, l) \wedge \text{commFailRate}(f) \wedge p = f). \end{aligned}$$

The most essential part of our theory is the set of definitions for rewards.<sup>8</sup> We will use two different reward functions. The first one takes into account the traveling distance only: it is intended to encourage mobilization of the car nearest to an incident without regard to all other factors. The idea is to provide the DTGolog's decision making operator  $\pi(\text{var} : \tau)\delta$  (nondeterministic finite choice of action argument), where  $\tau = \{c_1, \dots, c_n\}$  is the finite set of alternative cars that can be chosen for mobilization, with the ability to pick the car with highest chance of getting to the incident on time. Essentially, given an available car  $c$  and a location  $l$ , the reward  $r$  that the program can expect to receive for mobilizing  $c$  to  $l$  is directly proportional to the probability that the travel time is no more than 11 minutes (or 660 seconds):  $r = c \cdot \text{Prob}\{0 \leq t \leq 660\}$ , where  $c$  is a constant 100 (defined by  $r\text{Ontime}(100)$  in the model), and  $t$  is a random variable that represents the travel time. Rewards for doing other actions (except of mobilization) are defined as 0. For simplicity, we assume that travel time  $t$  has Gaussian distribution. More specifically,

<sup>8</sup>We talk about reward functions, but we define them using predicate symbols instead of function symbols to make their implementation in Prolog straightforward.

$t = (d + 1) \cdot (\mathcal{N}(0, 1) + v)$ , where  $\mathcal{N}(0, 1)$  is the Gaussian random variable with the mean 0 and variance 1,  $d$  is the Manhattan distance between a car and an incident (i.e.,  $d$  is the number of city blocks that a car has to travel) and  $v$  is the average travel speed (in seconds per city block). (Note we assume that time is needed to travel from a city block to itself, this is why we write  $d + 1$  instead of  $d$ .) Consequently,  $\text{Prob}\{0 \leq t \leq 660\} = \text{Prob}\{t \leq 660\} - \text{Prob}\{t \leq 0\} = \text{Prob}\{\mathcal{N}(0, 1) \leq \frac{660 - (d+1) \cdot v}{d+1}\} - \text{Prob}\{\mathcal{N}(0, 1) \leq -v\}$ . The reward function provided in the model, shown below, captures this equation and serves as a measure of how likely a given car, if mobilized, will make it to the incident on time.

$$\begin{aligned} \text{reward}(r, S_0) &\stackrel{\text{def}}{=} r = 0 \\ \text{reward}(r, \text{do}(a, s)) &\stackrel{\text{def}}{=} r = 0 \wedge \neg(\exists c, l, t) a = \text{mobilizeS}(c, l, t) \\ \text{reward}(r, \text{do}(\text{mobilizeS}(c, \text{loc}(x, y), t), s)) &\stackrel{\text{def}}{=} \\ &\quad (\exists x_1, y_1, d, v, r_{on}) \text{carLocation}(c, \text{loc}(x_1, y_1), t, s) \wedge \\ &\quad \text{distance}(\text{loc}(x_1, y_1), \text{loc}(x, y), d) \wedge r\text{Ontime}(r_{on}) \wedge \\ &\quad \text{avgTimePerBlockEmergHome}(v) \wedge \\ &\quad r = (\text{Prob}\{\mathcal{N} \leq \frac{660 - (d+1) \cdot v}{d+1}\} - \text{Prob}\{\mathcal{N} \leq -v\}) \cdot r_{on}. \end{aligned}$$

This first reward function reflects only the system goal that requests have to be served quickly, but neglects take into account human factors (e.g., desire of crews to have rest between assignments). Note also that if the RA uses this reward function, then he does not account for unfamiliarity with foreign regions by assuming that the travel speed is always the emergency speed in the home region (in the simulator, it varies from the home region to a "foreign" region). In other words, by using this reward function, the RA overlooks important features of the domain and assumes that ambulance crews are equally comfortable to travel in any region. We define it intentionally in this way to demonstrate that more carefully designed reward functions are possible too.

The second reward function is more sophisticated. It takes into account not only the distance, but also crew fatigue which introduces a lag in the response time (recall the predicate  $\text{tirednessLagTime}(\text{lag})$  defined above), region familiarity (using the predicate  $\text{avgTimePerBlockEmergForeign}(v)$  defined above), and the fluent  $\text{consecTripCount}(c, m, s)$  that helps to determine the number  $m$  of consecutive trips done by the car  $c$ . The definition of this reward function parallels the previous definition.

$$\begin{aligned} \text{reward}(r, \text{do}(\text{mobilizeS}(c, \text{loc}(x, y), t), s)) &\stackrel{\text{def}}{=} \\ (\exists x_1, y_1, d, m, v, \text{lag}, r_{on}) \text{carLocation}(c, \text{loc}(x_1, y_1), t, s) \wedge \\ &\quad \text{distance}(\text{loc}(x_1, y_1), \text{loc}(x, y), d) \wedge \\ &\quad \text{consecTripCount}(c, m, s) \wedge \\ &\quad \text{tirednessLagTime}(\text{lag}) \wedge r\text{Ontime}(r_{on}) \wedge \\ &\quad (\forall g)(\text{homeRegion}(g) \wedge \text{in}(\text{loc}(x, y), g) \supset \\ &\quad \quad \text{avgTimePerBlockEmergHome}(v)) \wedge \\ &\quad (\forall g)(\text{homeRegion}(g) \wedge \neg \text{in}(\text{loc}(x, y), g) \supset \\ &\quad \quad \text{avgTimePerBlockEmergForeign}(v)) \wedge \\ r = &(\text{Prob}\{\mathcal{N} \leq \frac{660 - m \cdot \text{lag} - (d+1) \cdot v}{d+1}\} - \text{Prob}\{\mathcal{N} \leq -v\}) \cdot r_{on}. \end{aligned}$$

According to this reward function, if  $\text{lag}$  is sufficiently large number (say,  $\text{lag}=100$ ) and an ambulance  $c$  did already several trips (say,  $m = 6$  trips), then the reward for choosing

this ambulance is low, because chances for this ambulance to arrive in time are less than  $Prob\{\mathcal{N}(0, 1) \leq \frac{60-(d+1) \cdot v}{d+1}\}$ , which is a very small number even if an incident happened in the same city block, i.e.,  $d=0$  (recall that  $v=100$ ). Thus, this reward function takes into account not only the system goals, but also interests of crews.

Finally, our theory of the domain includes axioms specifying: (1) what sensing actions has to be done to distinguish one outcome of the stochastic agent action  $mobilize(c, l, t)$  from another outcome (we require that the sensing action  $askStatus(c, st, t)$  should be performed); (2) axioms specifying situation suppressed logical conditions that need to be evaluated after doing a sensing action (see (Reiter 2001)):

$$\begin{aligned} senseCond(n, \phi) &\stackrel{def}{=} (\exists c, l, t) (n = mobilizeS(c, l, t) \wedge \\ &\phi = (isACar(c) \wedge \neg ready(c) \wedge \neg commLost(c)) \vee \\ &n = mobilizeF(c, l, t) \wedge \\ &\phi = (isACar(c) \wedge (ready(c) \vee commLost(c))) ) \end{aligned}$$

This completes our logical representation of the domain and the MDP associated with the problem of allocating ambulances.

## Golog procedures

We model three allocation strategies using two Golog procedures and two reward functions. The first procedure resembles the strategy used by the human RA of the manual system as described in (The Communications Directorate 1993). For this reason, we call it a *manual* system, and implement it by a deterministic Golog program that does not do any decision theory. The main idea of this procedure is that the RA picks the ambulance car nearest to the incident by comparing distances of all cars from the incident. However, this choice is subject to important restriction that ambulances cannot cross borders between regions, and they serve incidents (and hospitals) from their home regions only. The second Golog program resembles the strategy used by an *automated* system as described in the LAS report, and this program is coupled with the first reward function. This second Golog program makes nondeterministic choice from a finite set of cars and, consequently, allows non-trivial decision making: the car nearest to an incident is picked using the first reward function. Any ambulance can be sent to any region, as long as it is deemed as the ambulance that will reach the incident in time with the highest probability. The third strategy (a combination of the same second Golog program with the second reward function) is a hypothetical *optimized* system, in which the ambulance allocation task is also casted as a decision theoretic problem, but interests of ambulance crews are also reflected in the reward function.

We could also implement in DTGolog more sophisticated strategies (e.g., those which require planning for several steps ahead), but for simplicity of presentation we limit ourselves with strategies mentioned above. We do quantitative comparison of these 3 strategies using our simulator. To present out results, we explain the structure of both Golog programs and then we provide tables with numerical data.

The first Golog program does the following. For a duration of  $d$  seconds, it checks continuously if there is a region with at least one request. If yes, and if there is at least one car

with the home base in this region that is currently ready and whose location is known, then procedure chooses a region and an incident in this region, finds a car that is the nearest to the incident, the distance between this car and the incident, and finds also the distance between the incident and the base. If there is a car at the base and the distance from the base to the incident is no more than 2 city blocks greater than the distance from the nearest car to the incident, the procedure mobilizes a car from the base; otherwise, it mobilizes the nearest car. In the case when there is no pending request, or there are pending requests, but all cars are busy, the procedure just performs the no-cost action  $wait()$  and then calls itself recursively. The procedure does all its work inside the scope of the DTGolog operator  $limit()$  that makes sure that program inside must be executed to the completion on-line before any decisions can be made about subsequent choices. If more than  $d$  seconds passed (the end of working shift), then the procedure executes no cost  $noOp()$  action that does nothing and quits.

```

proc allocResManual(d)
   $\pi(t)$  [ (now(t))?;
  if  $t < d$  then limit( queryAllCars(t) ;
    if % At least one region has a pending request,
      % and one of its local cars is mobilizable
       $(\exists r, x, y, c_1) (isARegion(r) \wedge validXCoord(r, x) \wedge$ 
         $validYCoord(r, y) \wedge requestPending(loc(x, y)) \wedge$ 
         $isACar(c_1) \wedge homeRegion(c_1, r) \wedge$ 
         $ready(c_1) \wedge carLockKnown(c_1, t))$ 
      then  $\pi(r, x, y, nCar, nD, bD)$  [ incidentLoc(r, x, y);
        nearestMobilizableCar(r, x, y, nCar, nD);
        distanceToBase(r, x, y, bD) ;
      if % Some car at the base is mobilizable and the
        % base is not very far away from the incident
        % in comparison to the nearest car.
         $(\exists c_2) (isACar(c_2) \wedge homeRegion(c_2, r) \wedge$ 
           $atBase(c_2) \wedge ready(c_2) \wedge carLockKnown(c_2, t) \wedge$ 
           $bD - nD = < 2)$ 
        then % Mobilize the car from the base
           $(\pi c) [ (isACar(c) \wedge homeRegion(c, r) \wedge$ 
             $atBase(c) \wedge ready(c) \wedge carLockKnown(c, t))?$ ;
            mobilize(c, loc(x, y), t) ]
          else % Mobilize the nearest car
            mobilize(nCar, loc(x, y), t) ]
        else wait(t) % end of "limit"
      ) ; allocResManual(d) % Recursive call
    else noOp(t) ]
  endProc

```

This Golog program calls several other procedures; all of them are straightforward and do not require any decision theory (the interested reader can find in (Pham, Soutchanski, & Mylopoulos 2006) the details of their implementation). The procedure  $queryAllCars(t)$  asks about positions of all those cars in the city for which locations are not known (but communication with them is available). The procedure  $incidentLoc(region, x, y)$  consists of the

test expression that grounds variables  $region, x, y$  to the name of a region where there is at least one request and the location of this request is at  $loc(x, y)$ . The procedure  $nearestMobilizableCar(region, x, y, nCar, nDist)$  also consists of the single test expression that for given values of  $region, x, y$  grounds the variable  $nCar$  (the variable  $nDist$ , respectively) to the nearest car (in terms of the Manhattan distance) from the incident  $loc(x, y)$  (to the distance between the nearest car and the incident, respectively). The procedure  $distanceToBase(region, x, y, bDist)$  takes values of  $region, x, y$  and grounds the value of the variable  $bDist$  to the Manhattan distance between the base in  $region$  and the incident at  $loc(x, y)$ .

The automated system does not take into account human factors (such as crew fatigue and unfamiliarity with “foreign” regions) and uses the first reward function. (Note: The simulator always does calculations that take all these factors into account, regardless of what allocation strategy is used.) The second Golog program that implements the automated system works in much the same way as the Golog program for the manual system, except a few important details. The main difference is that it picks (via DTGolog’s nondeterministic finite choice  $\pi(car, rangeName)$ ) the car that it believes to have the highest chance of getting to the incident on time and mobilizes it. Because this non-deterministic operator occurs inside the scope of the  $limit()$  operator, the choice of the car to mobilize is made by solving a simple decision task with the horizon 1. Note that DTGolog can also accommodate more far-sighted decision making (DTGolog was designed for finite horizon decision-theoretic planning with constraints), and consequently, more sophisticated decision making agents can be considered as well. The second Golog program gives no preference to cars at the base, and to serve a request, it considers all cars from all regions. Thus, this second program provides a good illustration how DTGolog can take advantage of the structure of this domain by providing natural constraints on the set of policies that need to be considered.

```

proc allocResAuto(d)
   $\pi(t) \left[ (now(t))?;$ 
  if  $t < d$  then  $limit( queryAllCars(t) ;$ 
  if % At least one request is pending
    % and at least one car is mobilizable
     $(\exists r, x, y, c_1)( isARegion(r) \wedge validXCoord(r, x) \wedge$ 
     $validYCoord(r, y) \wedge requestPending(loc(x, y)) \wedge$ 
     $isACar(c_1) \wedge ready(c_1) \wedge carLocKnown(c_1, t) )$ 
    then  $\pi(r, x, y, rName) [ incidentLoc(r, x, y) ;$ 
    % Pick the best car and mobilize it
     $(range(All, rName))?;$ 
     $\pi(car, rName) mobilize(car, loc(x, y), t) ]$ 
    else  $wait(t)$  % end of “limit”
   $); allocResAuto(d)$  % Recursive call
  else  $noOp(t) ]$ 
endProc

```

As we mentioned above, an “optimized” system uses also the second Golog program, but it makes better decisions than

the automated system by relying on the second reward function. The interested reader can find all details about these Golog programs, their implementation in Prolog (as well as numerical data we collected) in (Pham, Soutchanski, & Mylopoulos 2006).

## Simulation Results

The simulator calculates travel times for ambulances using a simple assumption that travel time of any ambulance along a city block has the Gaussian distribution (because travel time cannot be negative the simulator samples until it gets a positive value). To simulate the travel time between  $loc(x_1, y_1)$  and  $loc(x_2, y_2)$ , let  $d$  be the Manhattan distance between them,  $v$  be the average number of seconds it takes for the car to travel one city block, and apply this formula:  $t(loc(x_1, y_1), loc(x_2, y_2)) = \sum_{i=1}^d \mathcal{N}(v, 1)$ , where  $t(loc_1, loc_2)$  is the time we want to calculate, and  $\mathcal{N}(v, 1)$  is a positive random number drawn from the Gaussian distribution with mean  $v$  and variance 1. The simulator takes a short trajectory between two locations and if it intersects a border between the home region and another region, the simulator uses the appropriate value of speed  $v$  for each city block (as defined above). In addition, the simulator uses a data structure that allows appropriate exogenous actions to be generated at the right time and be inserted into the situation term.

To collect the statistics, we performed simulation of the three allocation strategies using several request rates. For each request rate, each strategy was called 10 times, each time for approximately 300 requests. We run simulation on an AMD 1800 Mhz machine with 1GB of memory running Linux kernel 2.6.8. For each request rate, the process of collecting data for 300 requests takes approximately 5 hours. This indicates that decision making for one request takes less than 1 minute on average (because the simulator takes some time to do required computations). This time is mostly due to unoptimized implementation of the DTGolog interpreter in Prolog. The on-line DTGolog interpreter that we use is implemented using ECLiPSe Prolog 5.7 that is bundled with a linear constraint solver XPRESS 1427icp (it provides reasoning about time). The interpreter calls external functions and gets results from our simulator using C-Prolog interface.<sup>9</sup> The resulting values (presented in the tables) are averages over 10 runs. The entries in the table,  $A(B + C + D)$ , mean that in the given design at the given request rate,  $A$  percents of the time, it took more than 8 or 11 minutes for the ambulance to reach its incident’s location. Out of this  $A$  percents,  $B$  percents was caused by long travel time (ie, the car simply spent more than 11 minutes in traffic),  $C$  percents was caused by mobilization delay (i.e., all cars were busy at

<sup>9</sup>One can be tempted to implement the three allocation strategies considered above in a programming language like C, but this approach would be inflexible, would not allow to consider easily other designs that require decision making with longer horizons, and would be difficult to adapt to other applications with constraints on decision making process. At the same time, a variety of other Golog programs can be easily evaluated using DTGolog interpreter.

the time the incident occurred), and  $D$  percents was the result of both mobilization delay and long travel time. The first table represents percentage (rounded to an integer) of those trips that take more than 8 minutes, the second table - percentage of those trips that take more than 11 minutes. As

Rates	Manual	Automated	Optimized
60	70(14+35+20)	68(16+11+42)	62(16+9+36)
70	55(21+21+13)	66(23+9+35)	57(26+5+26)
80	45(25+13+7)	56(34+4+18)	32(32+0+0)
90	39(29+6+4)	37(37+0+0)	31(31+0+0)
120	30(30+0+0)	35(35+0+0)	32(32+0+0)
150	31(31+0+0)	33(33+0+0)	28(28+0+0)

Rates	Manual	Automated	Optimized
60	56(4+45+7)	58(7+15+36)	53(9+13+31)
70	36(5+27+4)	53(10+12+31)	43(12+7+24)
80	24(6+17+2)	35(14+6+16)	11(11+0+0)
90	16(7+8+1)	11(11+0+0)	9(9+0+0)
120	8(8+0+0)	8(8+0+0)	8(8+0+0)
150	8(8+0+0)	8(8+0+0)	7(7+0+0)

one might expected, the performance of different strategies are in the right order: the “optimized” allocation strategy is better than “automated”, and “manual” allocation strategy is somewhat better than “automated”. Additional experimentation (with different sets of parameters) might be necessary to provide statistically significant comparison between “optimized” and “manual” strategies. We did not do this in our paper, because our intention was not to advocate the superiority of one particular strategy, but that each design of a decision making agent can be expressed and compared with other alternatives using DTGolog.

## Discussion and Conclusion

We consider applicability of DTGolog to the task of evaluation of alternative designs of a decision making agent. In particular, we discuss several designs of a resource allocating agent in LAS and use DTGolog for making decisions which ambulance to allocate when there is a request. Each design is represented as a combination of a Golog program and a reward function. We use different reward functions to show that preferences of stakeholders can be captured differently in the model. We also intentionally use different Golog programs to show that different decision making styles can be represented using this approach. We show that domain dependent constraints on the set of policies can range from a purely deterministic program (no decision making at all) to a Golog program with a limited number of decision points. In the latter case, a finite horizon planning can be accomplished by the off-line DTGolog interpreter that resolves nondeterministic choices in an optimal way using one of the given reward functions. In this paper, we consider only very simple Golog programs, but DTGolog can handle also more sophisticated strategies and longer horizons. We demonstrate that DTGolog is a very expressive framework that seamlessly combines programming with decision-theoretic planning in the fully observable MDPs. To represent a process-oriented problem that continues indefinitely (as long as new exogenous requests arrive), we supplement DTGolog with

a simulator that generates randomly requests and several other exogenous actions (e.g., arrivals of ambulances) and also computes traveling time. Because we choose a domain where the state space has well beyond  $30^{300} \cdot 2^{300}$  states (3 grid-worlds  $10 \times 10$  with at least 1 request anywhere and 30 cars located anywhere) and there are many actions available in every state, it is doubtful that even state-of-the-art MDP solvers (such as SPURD) can solve the decision-theoretic planning problem in this domain. For this reason, quality of alternative designs cannot be evaluated by comparison with policies computed by the decision-theoretic planners. However, our simulator provides statistical data about traveling time, and using these data, we can determine the number of cases when ambulances arrive in time or too late, and, as a consequence, it is possible to evaluate quantitatively each design. It might be surprising that some of our designs are non-deterministic Golog program where non-determinism is resolved at run time to make the best decision (with respect to program constraints). However, we believe this is important, because resolving this non-determinism at design time can be problematic due to the huge size of the state space.

The main contributions of our paper are the following. First, we developed an extensive logical formalization of a non-trivial domain. Second, we demonstrated that DTGolog is well suited to the task of evaluation of alternative designs of a decision making agent. Third, we did experimental analysis of three different designs of a decision making agent using the same simulator for a fair comparison.

The large number of choices and the large number of actions with uncertain outcomes present computational challenges that have to be addressed in future work. The most important direction for future research is overcoming computational challenges of the DTGolog framework: using sampling to deal with large branching factor (in the version of directed value iteration that provides semantics for a DTGolog interpreter (Boutilier *et al.* 2000)) and using progression to deal with growing situation terms (Reiter 2001). Our research goal is a more advanced framework to handle models that are large enough to be of use in software design applications such as this one. In 2004, the real LAS-CAD system included about 30 regions, about 400 vehicles and was the largest public ambulance system in the world.<sup>10</sup>

## References

- Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1–2):81–138.
- Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*. Belmont, Mass.: Athena Scientific.
- Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the 17th Conference on Artificial Intelligence (AAAI-00)*, 355–362.
- Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *J. of Artificial Intelligence Research* 11:1–94.

<sup>10</sup> <http://www.lond-amb.sthames.nhs.uk/news/performance/performance.html>

- Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision theoretic planning. *Artificial Intelligence* 89:219–283.
- Ferrein, A.; Fritz, C.; and Lakemeyer, G. 2005. Using Golog for deliberation and team coordination in robotic soccer. *KI Künstliche Intelligenz* 05(1):24–43.
- Finzi, A., and Lukasiewicz, T. 2004. Game-theoretic agent programming in Golog. In *Proceedings of the 16th biennial European Conference on Artificial Intelligence (ECAI 2004)*. Valencia, Spain: IOS.
- Fritz, C., and McIlraith, S. 2006. Compiling qualitative preferences into decision-theoretic golog programs. In Mylopoulos, J., ed., *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*.
- Haddawy, P.; Doan, A.; and Goodwin, R. 1995. Efficient decision-theoretic planning: Techniques and empirical analysis. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI-95)*, 229–236.
- Hanks, S., and McDermott, D. 1994. Modeling a dynamic and uncertain world I: Symbolic and probabilistic reasoning about change. *Artificial Intelligence* 65(2).
- Kramer, J., and Wolf, A. 1996. Succeedings of the 8th international workshop on software specification and design. *ACM SIGSOFT Software Engineering Notes* 21(5):21–35.
- Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1-2):239–286.
- Lespérance, Y.; Kelly, T. G.; Mylopoulos, J.; and Yu, E. S. 1999. Modeling dynamic domains with ConGolog. In *Proceedings of CAISE-99*.
- Letier, E., and van Lamsweerde, A. 2000. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering* 26(10).
- Letier, E., and van Lamsweerde, A. 2004. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of the 12th International Symposium on the Foundation of Software Engineering FSE-04*, 53–62. Newport Beach, CA: ACM Press.
- Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.
- Levesque, H.; Pirri, F.; and Reiter, R. 1998. Foundations for a calculus of situations. *Electronic Transactions of AI (ETAI)* 2(3–4):159–178.
- Madani, O.; Hanks, S.; and Condon, A. 2003. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence, Special Issue on Planning with Uncertainty and Incomplete Information* 147(1–2):5–34.
- Majercik, S., and Littman, M. 2003. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence* 147(1–2):119–162.
- Manna, Z., and Waldinger, R. 1987. How to clear a block: A theory of plans. *J. of Automated Reasoning* 3(4):343–377.
- Pham, H.; Soutchanski, M.; and Mylopoulos, J. 2006. A simulator and a Golog implementation of the London Ambulance Service (LAS) Computer-Aided Dispatch (CAD) system. Technical report, Department of Computer Science, Ryerson University, <http://www.cs.toronto.edu/~mes/papers/LAS/index.html>.
- Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley.
- Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academ Press. 359–380.
- Reiter, R. 1998. Sequential, temporal GOLOG. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 6th International Conference*, 547–556.
- Reiter, R. 2001. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.
- Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. HTN planning for web service composition using SHOP2. *Journal of Web Semantics* 1(4):377–396.
- Soutchanski, M. 2001. An on-line decision-theoretic Golog interpreter. In *Proc. of the 17th International Joint Conference on Artificial Intelligence IJCAI-2001*, 19–24.
- Soutchanski, M. 2003. *High-Level Robot Programming in Dynamic and Incompletely Known Environments*. <http://www.cs.toronto.edu/~mes/papers/index.html>: Ph.D. Thesis, Computer Science Dep., University of Toronto.
- The Communications Directorate. 1993. *Report of the Inquiry Into The London Ambulance Service (South West Thames Regional Health Authority)*. The 8th International Workshop on Software Specification and Design Case Study. Electronic Version prepared by Anthony Finkelstein. Available at <http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las.html>.
- Waldinger, R. 1977. Achieving several goals simultaneously. In Elcock, E., and Michie, D., eds., *Machine Intelligence*, volume 8. Ellis Horwood, Edinburgh, Scotland. 94–136.
- Wang, X., and Lespérance, Y. 2001. Agent-oriented requirements engineering using ConGolog and *i\**. In Wagner, G.; Karlapalem, K.; Lespérance, Y.; and Yu, E., eds., *Proc. of the 3rd International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS)*, 59–78.
- You, J. Z. 2003. Applying the GRL framework to the london ambulance service (LAS) computer-aided dispatch (CAD) system case study. Technical report, Department of Computer Science, University of Toronto, <http://www.cs.toronto.edu/~janeyou/>.
- Younes, H.; Littman, M.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research* 24:851–887.