Towards Goal-based Generation of Reinforcement Learning Domain Simulations

Sotirios Liaskos[1], Shakil M. Khan[2], Reza Golipour[1] and John Mylopoulos[3]

[1]*School of Information Technology, York University*
[2]*Department of Computer Science, University of Regina*
[3]*Department of Computer Science, University of Toronto*

Abstract

Reinforcement learning (RL) is a much studied branch of machine learning and one in which substantial progress has taken place over the past few years. In RL, intelligent agents repeatedly interact with their environment and learn from the consequences of their actions. A key to effective RL is often the presence of a simulated environment that mimics the one that the agent is supposed to be optimized against. Such simulators allow for great numbers of training iterations in a cost-effective manner and without affecting the real environment. A systematic modeling and design process that allows efficient development of maintainable and comprehensible simulators can, hence, be beneficial for effective RL. We propose an approach for model-driven generation of RL environment simulations with discrete action spaces using goal models. The proposal utilizes earlier work for model-driven development of decision-theoretic action theories, whereby the standard iStar 2.0 notation is extended to include preconditions, stochastic effects, and reward modeling. Models in the extended notation can be translated into a formal specification for model-based reasoning based on Markov Decision Processes (MDPs). To also allow for model-free RL we introduce a module that queries the action-theoretic, stochastic action and reward structure aspects of the generated formal specification in order to guide episodic simulations of the modeled domain. The module is wrapped by a popular framework for building RL training and testing environments, making it accessible by popular RL agent frameworks.

Keywords

iStar (*i**) modeling, goal modeling, reinforcement learning, DT-Golog, Open AI

1. Introduction

Reinforcement learning (RL) is an important artificial intelligence approach whereby intelligent agents can learn to optimize their behavior through engagement with their environments [10]. Key to realizing such agents is the presence of a simulated environment, repeated interaction therewith allows the agent to improve the average value it gains from such interactions. Once the agent is trained in the simulated environment it can be transferred to the real one in which it can quickly deliver the optimal solutions.
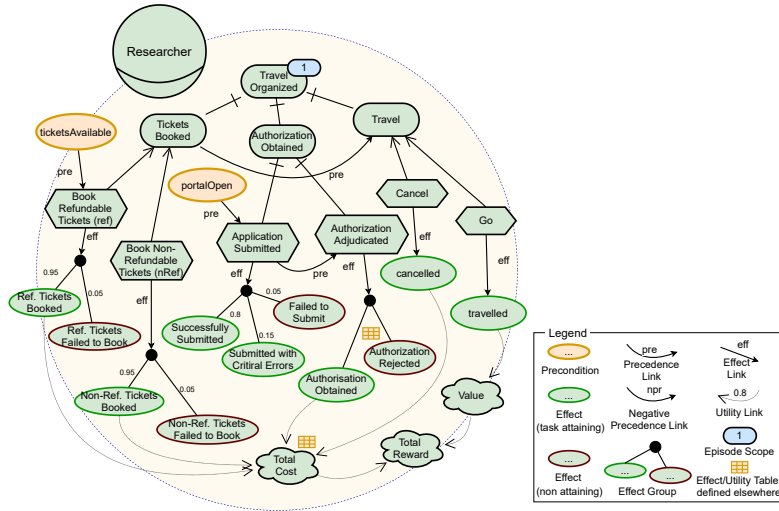

Figure 1: An extended goal model.

Simulations for RL describe how the simulated system changes states and delivers rewards or penalties in response to actions chosen and performed by an intelligent RL agent. Developing such simulations may be benefited by a model-driven approach whereby high-level models of the action and state space can be automatically translated to actual simulation components, allowing for development efficiency and outcome maintainability, comprehensibility and reproducibility.

In this paper, we describe how we reuse and extend an existing approach for translating goal models into formal specifications for model-based RL reasoning, so that simulation environments for model-free/learning-based RL can also be generated. The approach is based on extensions to the original translation routine, as well as the introduction of domain-independent integration components. Through the extension, analysts can switch between model-based and model-free analysis, to, e.g., benchmark alternative model-free techniques or assess the feasibility of a model-free approach when the model at hand is too large to be analyzed with model-based techniques. Thanks to the proposed extension, the overall framework can prove useful for developing process adaptation mechanisms within socio-technical systems that use past experience to improve ways to achieve business goals.

We offer background in Section 2, the extension in Section 3 and conclude in Section 4.

2. Background

2.1. Action- and decision-theoretic extensions to iStar

Our approach is based on a set of extensions proposed to the iStar 2.0 graphical notation [1] for capturing stochastic performance of tasks while allowing the modeling of action-theoretic aspects including preconditions, precedence constraints and effects [2, 3].

An example of the proposed notation can be viewed in Figure 1. The diagram, which represents a travel organization problem inspired by the example of the iStar 2.0 guide [1], includes many

of the basic elements of the language: *goals, tasks, qualities, AND-* and *OR-decompositions* as well as a *role* and their *actor boundary*. To this baseline, a set of constructs have been added to allow for expression of action-theoretic aspects. Firstly, a set of logical *domain predicates* are introduced for modeling the state of the world. Examples include **ticketsAvailable** or **authorizationRejected**. We borrow the *belief* construct from GRL [4] to place such domain predicates in the diagram. We further specialize these beliefs into *effects* and *preconditions*. While the former contain single predicates, the latter contain logical formulae thereof. The preconditions are connected to tasks through *precedence links* (or, respectively, their negative dual, *negative precedence links*) signifying that preconditions must be satisfied before tasks are performed (resp., that if they are satisfied the tasks cannot be performed). Such links can also be drawn between tasks (resp. goals) meaning that the target cannot be performed (resp. start to be fulfilled) unless/if the origin has been performed (resp., satisfied).

These additions are already useful for describing deterministic action-theoretic aspects of goal models allowing translations to key formalisms including Golog [5], Hierarchical Task Networks (HTNs) [6, 7] or other planners [8]. The additional aspect of interest is that tasks can be stochastic, i.e., lead to alternative effects, each with a different probability. To model this, *effect groups* are introduced which are simple tree-like structures that represent alternative effects of tasks, each annotated with its probability, when available. Finally, the framework considers that a quality is attained (or denied) not by mere attempt to execute a task or fulfill a goal, but by the observed success of task performance. Qualities are hence connected with effects rather than directly with tasks and goals. The links used for those connections are specializations of iStar 2.0's *contribution links* that we call *utility links* to highlight their decision-theoretic semantics. Utility links may also be annotated by a number signifying the level at which the truth value of an effect affects the overall utility of a state. Often, however, both probability and utility structures are too complex to be represented through annotations. In such cases *effect tables* and *utility tables* are utilized, accompanying the diagram.

2.2. DT-Golog and Model-based reasoning

The extensions introduced above allow for automatable translation of the goal models into DT-Golog, a formalism that allows representation and reasoning of action-theories with decision-theoretic components (probabilities, rewards) [9]. DT-Golog models action theories through the concept of a *situation* which, roughly, represents a history of *actions* from a distinguished initial situation S_0. State is represented through predicates called *fluents*. To map situations to truth values of fluents *successor state axioms* are used. Given an initial specification of the fluents, successor state axioms describe how their values change or remain unchanged when actions happen. Finally, actions may or may not be feasible in a given situation based on what is prescribed in *action precondition axioms*. DT-Golog distinguishes between *agent actions* and *stochastic actions*. Each action of the former type is associated with a set of alternative actions of the latter type, each with a distinct probability. Thus, upon attempt of an agent action, one of the associated stochastic actions will actually be performed based on the corresponding probability. DT-Golog also allows definition of total reward as a function of situations.

Goal models drawn using the extended notation can be translated into such a DT-Golog specification through the application of a set of translation rules [2]. Roughly, these rules

map tasks into actions, domain predicates into fluents, the AND/OR decomposition into a corresponding logical formula of fluents, precedence and effect links into precondition and successor state axioms, and structures of utility links into reward structures in DT-Golog.

Through the use of the DT-Golog interpreter, the result allows identification of *policies* – roughly: situation-based action recommendations in the form of DT-Golog programs – whose adoption is understood to maximize expected reward.

2.3. Model-free reinforcement learning

DT-Golog based reasoning constitutes *model-based* analysis whereby the action probabilities are known and optimization techniques, such as DT-Golog's engine or dynamic programming, suffice for the identification of optimal courses of action – i.e., there is no real "learning" involved. Model-based techniques, however, require complete knowledge of the probability model and are computationally expensive when models are larger and more complex.

To address such shortcomings, *model-free* RL can be considered whereby intelligent agents develop optimal strategies through just repeating action attempts and observing their outcomes. In its most basic form, an RL agent is given a set of actions it is capable of performing, and a set of states that the system (i.e., the agent itself and/or its environment) can be at, based on the actions previously performed. Thus, when the RL agent chooses and performs an action, this may result in a state transition and the acquisition of a positive or negative reward, all of which (the new state, the reward and the probability of each) are initially unknown. In such a context, the RL agent develops provisional optimal policies which it repeatedly improves by trying different actions and observing the reward outcome.

In RL practice, rather than deploying an RL agent in the real target environment and allow it to perform sub-optimal actions until it learns, utilization of a *simulation* of the target environment is often more sensible, therewith agents can be trained and tested prior to such deployment. A simulation of the problem of Figure 1, for example, would be a program that receives as inputs agent tasks and returns (i) the new state in which the system is now as a consequence of the performance of the task and (ii) the reward (or penalty) accrued by performing the task and going to that state. For example, when the task *"Authorization Adjudicated"* is given as input, a simulation would stochastically change the state to one in which the authorization is obtained or rejected, and return the reward in either case based on the reward information in the model. By repeatedly interacting with this program the RL agent learns what actions are optimal at which state with respect to expected reward, i.e., it learns an optimal policy – noting that the suitability of the policy against the target system still depends on the accuracy of the probabilities embedded in the simulation program.

Moreover, model-free RL distinguishes between *continuing* problems, where the obtained reward is continuously utilized for learning, and *episodic* problems, in which there are designated terminal states which, when reached, the overall trajectory up to that point (the *episode*) becomes the unit of evaluation before a new episode starts thereafter. In our context, while the goal decomposition models are suggestive of an episodic structure whereby an episode ends when the root goal is fulfilled, analysts may define how many such root goal fulfillment occurrences signify an episode. In Figure 1 an annotation next to the root goal contains that information.

In the following, we sketch how we can re-use our translation of goal models to DT-Golog
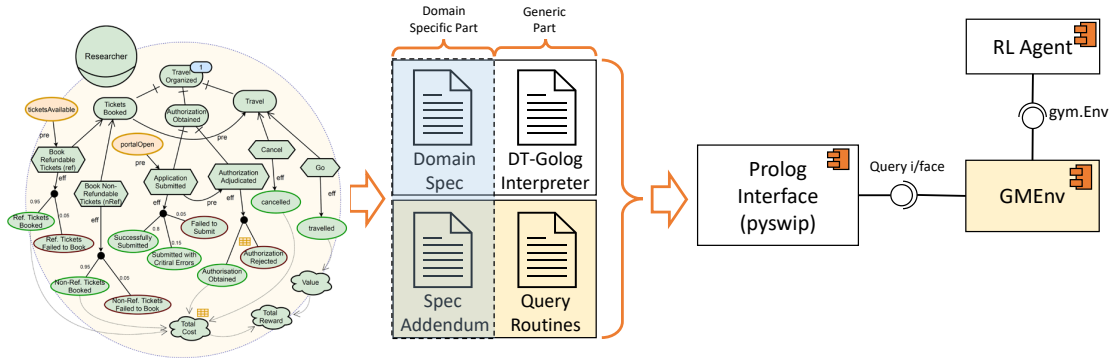
Figure 2: Solution Architecture. Components developed for model-free RL are shaded.

for model-based reasoning to also derive simulation environments for model-free learning. The proposed capability can help analysts in various ways. On one hand, independent of the presence of an accurate probability model, analysts can compare various RL algorithms [10] among each other and against their model-based counterparts (e.g., DT-Golog, dynamic programming) with respect to, e.g., their accuracy, training latency or sensitivity under changing parameters. On the other hand, when models are large, certain model-free RL approaches may prove to be preferable to computationally expensive model-based solutions.

3. Generating training environments

3.1. Overview

The overall architecture of the solution can be seen in Figure 2. Extended goal models are translated into a domain specification which, along with a DT-Golog interpreter, can be used for performing model-based decision theoretic reasoning [2]. From an implementation standpoint, both the specification and the DT-Golog interpreter are Prolog listings. To allow for simulations, two components need to be added to these listings: (a) additional domain specification details and (b) domain-independent query clauses. The resulting augmented specification can then be accessed from external applications for guiding simulations. In our case, a Python-to-Prolog interface allows for a Python simulation module (*GMEnv*) to query the specification for information including preconditions, effects, stochastic actions and their respective probabilities, as well as reward structures. With this capability, *GMEnv* is able to simulate step-wise performance of simulated actions, as guided by an external driver – i.e., an RL agent.

3.2. Specification Additions

Let us explore in more detail the additions that need to accompany a DT-Golog specification to allow for executing simulations. These consist of a domain-specific and a generic part. The domain-specific part includes a reproduction of the action precondition axioms for agent actions to complement those on stochastic actions that DT-Golog requires and are part of the original translation rules. Conveniently, in the goal model, preconditions are indeed specified at the

level of agent actions and translation rules apply the preconditions uniformly to all stochastic actions associated with that agent action. Thus, it is easy to expand the translation rules to also produce precondition axioms for agent actions: for every group of stochastic actions, produce a precondition axiom which includes the agent action associated with the stochastic ones.

The generic part of the specification addendum consists of a number of helper routines (Prolog rules) that allow querying of the domain specification. Letting L be a list of agent actions that have been performed from the first to the last, and a an agent action, the most important of the added predicates are: (a) *queryState(L)*, for translating an action history L, i.e., a situation, into a state, i.e., an array of fluent truth values, (b) *isFeasible(a,L)*, to check if action a is feasible after action history L, (c) *queryReward(L)*, to retrieve total reward of L, and (d) *queryDone(L)* for checking if the root goal has been fulfilled after L, through checking the corresponding logical formula constructed using fluents which represent success-signifying effects of leaf-level tasks.

Key in implementing these rules is the realization that, again, the RL concept of state is different from that of Golog's situation: the former represents a configuration of values of the fluents, i.e. the variables that represent state, and the latter is a history of agent actions. As we saw, given a situation, a state is retrievable through collecting fluents that according to the successor state axioms hold in that situation. State is encoded through a binary array, each element of which represents a fluent, and the binary value signifies if the element holds or not. Such array is easily translatable to an integer representation, as required by the client environment. Similar indexing is introduced for agent actions.

3.3. The GMEnv Component

The querying routines added in the specification can be utilized by external simulation-implementing tools. In our implementation, the client *GMEnv* is a Python class which implements the OpenAI Gym's [11] interface, *Env*, which is widely used for developing simulation environments and is required by popular RL agent development frameworks. A *GMEnv* object maintains information about episode development through a list of agent and stochastic actions that have been performed. Interface *Env* requires implementation of three important methods including: (a) an initialization routine specifying, among other things, the type of input and output spaces, (b) a *reset* routine, whereby the state is typically brought to its initial value, (c) a *step* routine, which accepts an action a as a parameter and returns the state that results from executing the action, the reward accrued from performing it, as well as a boolean value representing whether the system has reached a terminal state.

Our implementation of *step* utilizes the routines mentioned in the previous section in order to calculate the feasibility, probability profiles, and rewards of proposed actions as well as whether they lead to root goal fulfillment, which marks the end of an episode. Implementation of *reset* trivially returns the state of *GMEnv* to the initial one in which no action has been performed.

4. Concluding Remarks and Future Work

We described an extension to a framework for performing decision-theoretic reasoning with goal models with additional components that allow the automatable generation of discrete-action simulation environments for RL. Through the extension, goal models can be used as

the basis for model-driven engineering both of reasoners for model-based identification of optimal policies and of simulations that allow model-free learning of optimal policies. The specific extension to derive simulations for model-free learning can be utilized in different ways, including benchmarking model-free RL algorithms against a known model, or testing model-free solutions when models that are too large for model-based reasoning. While the overall framework has been motivated towards design-time identification of optimal task sequences under uncertainty within socio-technical analysis contexts, the proposed extension also paves the way for model-driven design and prototyping of run-time process/workflow management components that learn from experience.

Of great priority for future investigation is whether there are RL techniques that are indeed more effective and efficient than DT-Golog for any size or type of goal model, and, furthermore, whether there are synergies between the two, e.g., training against the simulation to assist parallel search in the state space. Furthermore, we are exploring ways to model continuous state spaces using iStar extensions. Should that be possible, the modeling framework could prove useful for iStar-driven development of RL agents for physical or cyber-physical systems.

References

[1] F. Dalpiaz, X. Franch, J. Horkoff, iStar 2.0 Language Guide, The Computing Research Repository (CoRR) abs/1605.0 (2016). URL: http://arxiv.org/abs/1605.07767. arXiv:1605.07767.

[2] S. Liaskos, S. M. Khan, J. Mylopoulos, Modeling and reasoning about uncertainty in goal models: a decision-theoretic approach, Software and Systems Modeling (2022). doi:https://doi.org/10.1007/s10270-021-00968-w.

[3] S. Liaskos, S. M. Khan, M. Soutchanski, J. Mylopoulos, Modeling and Reasoning with Decision-Theoretic Goals, in: Proceedings of the 32th International Conference on Conceptual Modeling, (ER'13), Hong-Kong, China, 2013, pp. 19–32.

[4] E. S. Yu, GRL - Goal-oriented Requirement Language, 2001. URL: https://www.cs.toronto.edu/km/GRL/.

[5] X. Wang, Y. Lespérance, Agent-oriented requirements engineering using ConGolog and i*, in: In Bi-Conference Workshop at Agents 2001 and CAiSE'01 (AOIS-2001)., 2001.

[6] S. Liaskos, S. McIlraith, S. Sohrabi, J. Mylopoulos, Representing and reasoning about preferences in requirements engineering, Requirements Engineering Journal (REJ) 16 (2011) 227–249.

[7] S. Liaskos, S. A. McIlraith, S. Sohrabi, J. Mylopoulos, Integrating Preferences into Goal Models for Requirements Engineering, in: Proceedings of the 10th IEEE International Requirements Engineering Conference (RE'10), Sydney, Australia, 2010.

[8] S. Liaskos, S. M. Khan, M. Litoiu, M. D. Jungblut, V. Rogozhkin, J. Mylopoulos, Behavioral adaptation of information systems through goal models, Informations Systems (IS) 37 (2012) 767–783.

[9] M. Soutchanski, High-Level Robot Programming in Dynamic and Incompletely Known Environments, Ph.D. thesis, Department of Computer Science, University of Toronto, 2003.

[10] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 2018.

[11] Open AI Gym, 2022. URL: https://github.com/openai/gym.