Model-driven Design and Generation of Domain Simulators for Reinforcement Learning

No Author Given

No Institute Given

Abstract. Reinforcement learning (RL) is an important class of machine learning techniques, in which intelligent agents optimize their behavior by observing and evaluating the outcomes of their interactions with their environment. Key for successful engineering of such agents is the availability of a large number of low-cost interactions with the target environment. This is often achieved through developing simulators of such interactions, therewith different training strategies and parameters can be explored. However, specifying and implementing such simulators can be a complex endeavor requiring a systematic process for capturing and analyzing the associated goals, actions, and domain assumptions. We propose a model-driven framework for goal-oriented development of RL simulation environments. The framework utilizes a set of extensions to a standard goal modeling notation that allows concise modeling of a large number of ways by which an intelligent agent can interact with its environment. Though subsequent formalization, the model can be used by a specially constructed simulation engine to simulate agent behavior, such that off-the-shelf RL algorithms can use it as a training environment. We present the extension of the goal modeling language, sketch its semantics, and show how models built with it can become executable.

Keywords: Goal Modeling · Reinforcement Learning · DT-Golog

1 Introduction

Over the past years, the demand for efficient Artificial Intelligence (AI) systems has been on the rise. Such systems perform tasks requiring autonomy and complex decision making, such as driving vehicles, controlling devices, or making trading decisions. Some of these AI systems are based on Reinforcement Learning (RL), whereby intelligent software agents learn to optimize their behavior by continuously interacting with their environment [58]. RL has been studied in a variety of application domains including energy [4], traffic control [60], finance [52] and healthcare [27]. Intelligent *RL agents* can be seen as engaging in goal-oriented activity, whereby they perform actions to fulfill functional goals (e.g., administer a therapy, trade securities, control a heating device, etc.), while maximizing the satisfaction of higher level quality objectives (resp., health outcome, profit, occupant comfort) based on experience. Goal models [3,16,18,25,63] have been found to be suitable for model-driven analysis of the requirements for such agents [37, 38].

Key to successfully engineering an RL agent is the ability to subject it to a large number of opportunities for interaction with the target environment at a low cost. Using a *simulator* based on a model of the target environment allows RL agents to engage in a large number of training iterations that are unsafe or expensive to be performed in the real environment. When such model is accurate, the resulting policies are readily applicable to the target environment. When the model is provisional, approximate, or otherwise imprecise, simulator-based training is useful for exploring the performance of different learning algorithms under alternative problem formulations and parameter settings.

We propose a model-driven, goal-oriented framework for developing simulation environments for RL. The framework is based on developing goal models describing the required intentional structure of RL agents through representing how high-level agent goals are decomposed into low level actions, how the latter, upon their performance, give raise to stochastic outcomes, and how such outcomes, in turn, affect a variety of quality variables of interest, an aggregate of which is used to represent action/outcome rewards. The requirements model is then automatically translated into an action- and decision-theoretic formal specification which, through a set of proposed querying and simulation components, can be directly usable by RL agents as a training workbench. In this way, the training simulator is the result of a principled requirements-based approach that fully embraces modeling both for facilitating analysis and communication of the problem domain and for generating the actual simulator.

The paper is organized as follows. We describe the modeling language in Section 2 and the generation of simulators in Section 4. In Section 6 we discuss related work and we conclude in Section 7.

2 Modeling RL Domains

2.1 Motivating Example

Consider a large scale woodwork manufacturer who builds custom furniture and other wooden building structures in a make-to-order fashion. For every order they receive, they need to source the material and manufacture the requested product – among many other activities omitted here for simplicity. They have options as to how they perform these two steps. The material can be sourced from domestic or foreign sources. In the first case, the cost is higher, but in the latter case there are delay risks. Once the material is acquired, they can engage their in-house crafts-persons to build the product or subcontract to a more specialized group, who use precise manufacturing techniques but at a premium – plus they only work with domestic material. Importantly, a decision outcome of one step of the process, may affect what decision is best in a subsequent step. For example, a delay in sourcing may necessitate expedience in manufacturing to meet deadlines. In addition, choices have non-deterministic outcomes. For example, sourcing material from abroad may or may not delay, and the in-house crafts-persons may or may not produce a lower quality product.

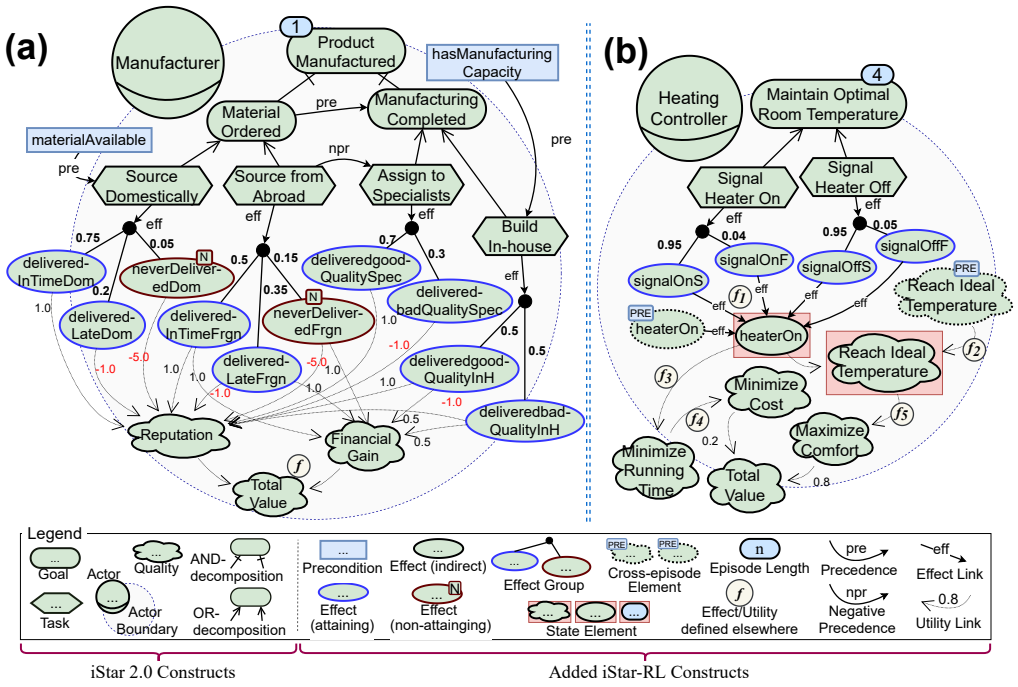

Fig. 1: An extended goal model.

The question for the manufacturer as they engage in this sequential decision making process for every incoming order is what decision they should be making at each step so that they are, on average, maximally satisfied in the end. Reinforcement learning (RL) refers to a set of techniques that have been proposed for addressing this problem via learning through experience [58]. Specifically, an RL agent actively engages with the environment by repeatedly performing *actions* which have the effect of (a) changing the *state* of the environment, typically in a non-deterministic way, and (b) offering to the agent a (positive or negative) *reward* reflecting the desirability of the action outcome or state change. They repeat until a goal state is reached and restart with a new effort to achieve the same goal. In the example case, an RL agent would, for each order, make a sourcing option, observe the outcome (delayed or not), proceed with a manufacturing option, observe that result (good quality or not) and move on to the next order. Various RL algorithms have been introduced for turning these repeated decision making sequences into a *policy*, which prescribes what action should be taken at each state, so that total reward is maximum on average.

As RL agents need to first process a potentially large number of cases before they can start making safe and good quality decisions, it is preferable that their training is not taking place in the actual environment but an executable model thereof, i.e., a simulator. Such a model should capture the decision points that are available to the RL agent, the corresponding alternative actions, the possible

effects that these actions bring about, as well as the reward structure whereby the preferability of the effects is specified. The latter is particularly challenging in RL applications within socio-technical systems, such as the one of our example, where reward structures are abstract, subjective, and multi-dimensional. We next describe an extension to a standard goal modeling language that aims at developing such models.

2.2 Extending iStar for Reinforcement Learning: iStar-RL

The proposed language, which we will refer to as *iStar-RL*, extends iStar 2.0 [16], a language for goal-oriented modeling of socio-technical systems. Two examples of iStar 2.0 models can be viewed in Figure 1. Diagram (a) on the left shows the goals of the hypothetical woodwork manufacturer we discussed above.

As per the iStar 2.0 ontology, *actors*, such as *Manufacturer*, have *goals*, such as *(Have) Material Ordered*, which represent states of affairs that actors want to bring about and/or maintain. Through *AND-* and *OR-decompositions*, high-level goals are refined into lower-level ones, whereby satisfaction of all or, respectively, one subgoal(s) is necessary (resp., sufficient) for fulfilling the parent goal. At the leaf level, *tasks* signify concrete actions to be taken for fulfilling goals – e.g. *Source Domestically*. Quality goals (*qualities*) describe attributes for which actors desire some level of achievement, without the requirement that such level is precisely defined. *Contribution links* are used to signify that achievement of a goal/task affects a quality in a way that is described using an annotation on the link.

Models such as that of Figure 1(a) can encode a great number of subsets of leaf-level tasks and temporal orderings thereof that can fulfill top-level goals. Various ways for formalizing goal models for purposes similar to identifying such task sequences have been introduced – e.g. [25, 30, 36, 46]. We adopt here a decision-theoretic extension to iStar for reasoning in the presence of probabilistic task effects [37, 38] and further extend it with constructs that facilitate RL.

These extensions are showcased in Figure 1. A set of *domain propositions* are, firstly, introduced for describing the state of the environment at different points in time – for example, *hasManufacturingCapacity* and *materialAvailable*. Domain propositions are included in the model as the contents of *precondition* and *effect* elements. A precondition contains a boolean formula of domain predicates, whereas an effect contains one such predicate. Preconditions are connected to tasks through *precedence links* $\xrightarrow{pre}$ and, respectively, *negative precedence links* $\xrightarrow{npr}$ which denote that performance of the task is not possible unless (resp., if) the formula in the precondition is satisfied. Precedence (resp., negative precedence) links can also originate from goals meaning that the task cannot be performed unless (resp., if) the origin goal has been satisfied (resp., has been attempted, i.e. at least one of its descendant tasks has been successfully performed).

Further, tasks are connected with effects through the use of *effect links* $\xrightarrow{eff}$, denoting that performance of the task from where the link originates can cause the effect which the link points at to occur, i.e., make the domain predicate contained in it true. As tasks may have several possible effects, they may be

connected to *effect groups* which represent a collection of effects each carrying a distinct probability to occur once the task is performed. The probability is added as a label on the corresponding link on the diagram. Effects are marked as *task satisfying* if their occurrence implies successful performance of the task, and *non-satisfying* otherwise. For example, an attempt to submit an order may be deemed successful (task satisfying) if it is finally delivered, despite delays, errors, etc., and non-satisfying if it is never delivered. Moreover, *indirect effects*, such as *heaterOn* in Figure 1(b), are effects whose truth status depends on other effects in a way that is defined outside the diagram – noted through an ⓕ annotation. Effect links are used to connect regular effects with indirect effects.

Further, effects, including indirect, are connected to qualities through a specialization of iStar 2.0 contribution links which we call *utility links*. These are annotated either with a number representing the amount of satisfaction that the effect, if it occurs, adds to the quality or with the ⓕ icon indicating that a more complex formula describes the relationship. Qualities, which can also be connected with each other using contribution links, form their own hierarchy with a top level goal o_{top} representing overall quality. In Figure 1, this goal is *Total Value*. Qualities in iStar-RL are considered to be continuous variables whose value represents level of satisfaction of the quality at a given state.

2.3 Task Histories, Goal Runs, and Episodes

Let us now focus tasks. Let *task (performance) history* be a sequence $tH^{|I} = [t_1, t_2, \ldots]^{|I}$ of leaf-level tasks that (a) start from an initial state in which propositions and qualities have a value configuration I, and (b) is feasible with respect to the precondition and effect constraints, i.e., t_1 is feasible under I and each subsequent task is possible given the state that the previous task brought about. Attempt of each task results in the occurrence of an effect. Hence, a task history is mapped to a set of possible *effect (occurrence) histories*, $eH^{|I} = [e_1, e_2, \ldots]^{|I}$ – we will henceforth omit the initial state superscript I unless needed. Further, let $\mathcal{H}$ be the set of all goals and $\mathcal{O}$ the set of all qualities in a model. The mappings $satG : \mathcal{H} \times \mathbf{eH} \mapsto \{true, false\}$ and $satQ : \mathcal{O} \times \mathbf{eH} \mapsto \mathbb{R}$, describe the satisfaction or not of a goal, and, respectively, the level of satisfaction of a quality, given an effect history eH from the set $\mathbf{eH}$ of all such. The former mapping reflects the AND/OR decomposition structure and the latter the structure of utility links; precisely how is discussed in a subsequent section.

Further, let $\mathcal{G}$ be a goal model with root goal $r_{\mathcal{G}}$. A *goal run* for goal $r_{\mathcal{G}}$ is an effect history $eH = [e_1, e_2, \ldots]$ such that (a) $satG(r_{\mathcal{G}}, eH)$, i.e. the root goal is satisfied, (*successful run*) *or* (b) no other task can be performed at eH, due to precondition constraints, a situation we will call a *deadlock*.

Back in Figure 1(a), consider effect histories *[deliveredInTimeDom, deliveredBadQualityInH]* (materials sourced domestically and in-house production was of bad quality) and *[neverDeliveredFrgn]* (materials ordered from foreign sources and were never delivered). Both histories are goal runs: the former satisfies the root goal, whereas the latter cannot be continued due to the $\xrightarrow{pre}$ link between

Material Ordered and *Manufacturing Completed*. However neither *[deliveredLate-Dom]* nor *[deliveredInTimeFrgn]* are goal runs, as, in both cases, the root goal is not satisfied and there are actions that are still possible. The level of satisfaction *satQ* of quality *Reputation* for each of the aforementioned four (partial) histories is +1.0 - 1.0 = 0, -5.0, -1.0, 1.0, respectively, calculated by simply adding up the annotations of utility links originating from effects included in the history.

We further define an *episode* to be a concatenation of n goal runs $eP^{|I} = [eH_1^{|I}, eH_2^{|I}, ..., eH_n^{|I}]$ such that for all $eH_i^{|I}$, $i < n$ the root goal is satisfied. In other words, an episode describes a history of repeated goal runs all of which lead to the root goal being satisfied except for the last one, that may lead to a deadlock. For $n > 1$ the episode is a *multi-run episode* and for $n = 1$ a *single-run episode*. We can specify the maximum number of runs that comprise an episode (*episode length*) as an annotation next to the root goal - see Figure 1. In multi-run episodes, each run is, by default, assumed to start from the same initial configuration I. We may however wish to designate elements that carry their values across goal runs. We accomplish this through *cross-episode elements*.

To appreciate the rationale for multi-run episodes and the role of cross-episode elements in such episodes, consider the model of Figure 1(b). It describes the function of a heating device controller, contrived here to showcase additional features of the language. The controller periodically signals wirelessly to the device to turn *on* or *off*. This *on* or *off* signal can be lost with a probability. The overall quality accrued from a sequence of signaling tasks is a function of the distance of the temperature to an ideal one and the amount of time heating is on, which represents cost and environmental impact. To make meaningful optimal decisions one needs to look at the quality value accumulated over a sequence of events of attaining the top level goal, i.e., over several goal runs. According to the diagram this is set to four (4). Hence, if the controller makes a decision every, e.g., 5 minutes, which constitutes one goal run, an entire episode spans 20 minutes and overall quality is calculated for all 4 decisions made.

In Figure 1(b), cross-episode elements (dashed outline and with a "PRE" annotation) represent the (truth) value of the enclosed proposition or quality in the previous state. The previous state is the configuration of truth values before the latest action was performed within the episode, irrespective of goal run boundaries. Thus, the truth status of the proposition within indirect effect *heaterOn* (solid line effect), depends on its truth status at the end of the previous state (dashed line effect) and the current state of four regular effects in the second run, represented through the remaining four incoming effects. The symbol $\textcircled{f_1}$ denotes that the exact formula that translates the truth value of these five propositions into the new truth value for *heaterOn* is specified outside the diagram. Likewise, the current value of *Reach Ideal Temperature* depends on its previous value and the current value of *heaterOn*. Again, a symbol $\textcircled{f_2}$ denotes that the formula for combining the two values is specified outside the diagram.

A final feature of the language is the designation of the *exported state set*, i.e., the set of propositions or qualities to be used for the calculation of policies. By default, we assume that the problem is modeled as *discrete-state* one, i.e.,

exported state set is the configuration of values of all propositions – the *discrete state set*. However, in some cases it is useful to designate the exported state to be a set of qualities of interest. In the heating controller example, whether the heater should turn on or off more obviously depends on the level to which *Reach Ideal Temperature* has been achieved than the history of *on* and *off* actions. In the diagrams of Figure 1 we put a shaded rectangle at the background of an indirect effect or a quality, to mark its inclusion in the exported state. When the exported state contains at least one non-propositional element, we model the problem as a *continuous-state* one. In Figure 1(b) we designate variables *heaterOn* and *Reach Ideal Temperature* to exclusively comprise the exported state set – a *continuous exported state set*. Note that such designation does not affect how validity of task and effect histories is established, which is based solely on the discrete state set; hence the "exported" qualification to prevent confusion.

2.4 iStar-RL and Reinforcement Learning

Let us now sketch how iStar-RL models can be used by an RL agent to allow for optimal decision making. Recall that RL-agents observe the state of the environment, perform an action from a set of available ones, and sense the state that results from the action along with the reward that the action yields. In our case, the state of the environment is, as we saw, the exported state, i.e., a combination of values of designated domain propositions and/or qualities. Upon sensing that state, the RL agent performs a task, which brings about the corresponding effect, which, in turn, augments the current episode eP with one more effect, and may also imply updated $satG$ and $satQ$ values. The RL agent will sense the new state and perceive the total value $satQ(o_{top}, eH_i)$ as the reward of the latest action. It will then repeatedly proceed with the next action until the episode is over, i.e., it reaches the maximum number of successful runs or a deadlock. During training, the RL agent will attempt a great number of such episodes, aimed at identifying a policy, i.e., a mapping from state to tasks that, when repeatedly followed, maximizes the average total value.

For the RL agent to accomplish such training using the iStar-RL model, the latter needs to be executable. In the next section, we describe how iStar-RL models can be formalized, aimed at both clarifying their semantics and paving the way for generating executable simulations of such models.

3 Semantics

3.1 DT-Golog

The semantics of iStar-RL are defined by means of its translation to DT-Golog, a high-level agent programming language [11, 57] based on the Situation Calculus [54]. The basic constructs of DT-Golog are fluents, stochastic (or agent) actions, nature actions, and situations. *Fluents* play the role of state features and have different truth values in different situations. They are represented through

predicates such as *materialDelivered(s)*, with the situation s as one of the parameters. *Stochastic actions* a are first-order terms signifying specific activity initiated by agents and may have several alternative outcomes, each occurring with a different probability. These outcomes are modeled through a set of *nature actions* $\hat{A} = \{\hat{a}_1, \hat{a}_2, ...\}$ associated with a through predicate $choice(a, \hat{A})$. The corresponding probabilities are represented using $prob(\hat{a}, v, s)$, where v is the probability of the occurrence of $\hat{a}$. Further, a *situation* s denotes a sequence of actions. Function $do(\hat{a}, s)$ denotes the situation which results from the performance of nature action $\hat{a}$ in situation s. A special constant S_0 denotes the initial situation, where no action has been performed.

A DT-Golog specification contains axioms that prescribe what actions are possible in different situations and how the truth value of fluents is affected by the performance of actions. The former, *precondition axioms* are of the form:

$$\forall s.Poss(a, s) \leftrightarrow \Pi_a(s)$$

where $\Pi_a(s)$ is an arbitrary formula and special predicate $Poss(a, s)$ states that action a is executable in situation s. *Successor state axioms* are of the form:

$$\forall \hat{a}, \mathbf{x}, s.f(\mathbf{x}, do(\hat{a}, s)) \leftrightarrow \Phi_f(\mathbf{x}, \hat{a}, s)$$

where $\mathbf{x}$ signifies a list of arguments, f a fluent symbol and $\Phi_f(\mathbf{x}, \hat{a}, s)$ a formula whose truth value depends on the parameters, the current situation, and the nature action in question. Finally, by defining:

$$reward(v, s) \leftrightarrow \Psi_r(v, s)$$

where $\Psi_r(v, s)$ is a formula grounded on fluents, it is possible to assign a reward value v to any situation or action.

3.2 From iStar-RL models to DT-Golog specifications

The DT-Golog-based semantics of iStar-RL is based on a treatment offered by Liaskos et al. [37], with the necessary additions for supporting RL. In what follows, let a goal model $\mathcal{G}$ containing a set $\mathcal{H}$ of goals, a set $\mathcal{T}$ of tasks, a set $\mathcal{E}$ of effects, a set $\mathcal{Q}$ of domain predicates, and a set $\mathcal{O}$ of qualities.

Primitives. Each element in the set is translated to a DT-Golog primitive as follows. For each domain proposition $q \in \mathcal{Q}$ introduce a fluent $\phi_q(s)$. For each task $t \in \mathcal{T}$ associated with an effect group $\mathcal{E}_t \subseteq \mathcal{E}$ introduce the following: a stochastic agent action a_t and a set of nature actions N_t each $\hat{a}_t \in N_t$ associated with an effect in $\mathcal{E}_t$. For each task $t \in \mathcal{T}$ and goal $h \in \mathcal{H}$, introduce fluents $\phi_t(s)$ and $\phi_h(s)$. For each quality $o \in \mathcal{O}$ introduce two fluents $\phi_o^{(r)}(v, s)$ and $\phi_o^{(m)}(v, s)$, respectively called *current satisfaction fluent* and *cumulative satisfaction fluent*; for both fluents, parameter $v \in \mathbb{R}$ represents the satisfaction value of o.

Given the 1-1 correspondence between iStar-RL effects and situation calculus nature actions, effect histories $eH = [e_{t_1}, e_{t_2}, \ldots]$ also map 1-1 to situations $s = do(\ldots do(\hat{a}_{t_2}, do(\hat{a}_{t_1}, s_0))\ldots)$, where $\hat{a}_{t_i}$ is the nature action corresponding to effect e_{t_i}. For a goal h and quality o: $satG(h, eH)$ iff $\phi_h(s)$ and $satQ(o, eH) = v$ iff $\phi_o^{(r)}(v, s)$, where s is the situation corresponding to effect history eH.

Successor State Axioms. For each domain predicate q, collect all (if any) effects $e_1, e_2, \ldots$ that mention it and consider the corresponding $\hat{a}_1, \hat{a}_2, \ldots$ nature actions. Introduce the following successor state axiom:

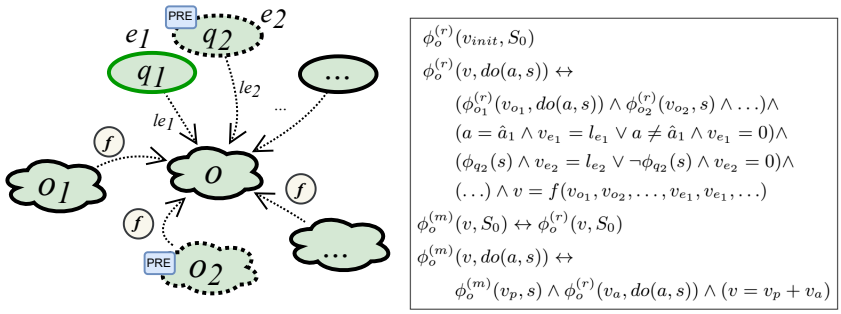

Fig. 2: Constructing quality formulae

$$\phi_q(do(a,s)) \leftrightarrow (a = \hat{a}_1 \vee a = \hat{a}_2 \vee \ldots) \vee \phi_q(s).$$

Attempt and Attainment Formulae. Consider each task t connected with an effect group $\mathcal{E}_t$. Consider *all* the effects of $\mathcal{E}_t$, and the domain predicates $q_1^{(t)}, q_2^{(t)}, \ldots$ contained in them. Generate the following *task attempt formula* for t:

$$\phi_t^{(att)}(s) \leftrightarrow \phi_{q_1^{(t)}}(s) \vee \phi_{q_2^{(t)}}(s) \vee \ldots$$

A *task attainment formula* $\phi_t(s)$ is defined similarly with the only difference being that it excludes non-satisfying effects. Introduce also the a *goal attainment formula* for each goal h as follows:

$$\phi_h(s) \leftrightarrow f_{\text{AND/OR}}(\phi_{t_1}(s), \phi_{t_2}(s), \ldots)$$

where $t_1, t_2, \ldots$ are the tasks that are descendants of h in the goal hierarchy, $\phi_{t_1}(s), \phi_{t_2}(s), \ldots$ their corresponding task attainment formulae, and $f_{\text{AND/OR}}$ an AND/OR formula reflecting the corresponding goal decomposition structure. A *goal attempt formula* for h, $\phi_h^{(att)}(s)$, is similarly defined but grounded on task attempt, rather than attainment, formulae.

Quality Formulae. Consider each quality or domain variable $o \in \mathcal{O}$, all the contribution links toward it coming from effects $e_1, e_2, \ldots$ labeled with values $l_{e_1}, l_{e_2}, \ldots$, and all contribution links coming from other qualities $o_1, o_2, \ldots$ – see Figure 2 left side. Recalling that o is associated with fluents $\phi_o^{(r)}$ and $\phi_o^{(m)}$, the value of each is defined by the formulae at the right side of Figure 2. In the formulae, $\hat{a}_i$ are the nature actions associated with effects e_i. Intuitively, the value v of quality o is a function f of the corresponding current or previous (depending on cross-episode status) values of the other qualities and/or the labels of utility links coming from effects that are(/were) currently(/previously) true.

Action Precondition Axioms. For each task t which receives precedence links from a precondition element, a set of tasks $\{t_1, t_2, \ldots\}$, a set of goals $\{h_1, h_2, \ldots\}$ and a set of effects containing predicates $\{q_1, q_2, \ldots\}$ introduce:

$$poss(a_t, s) \leftrightarrow \phi_{q_1}(s) \wedge \phi_{q_2}(s) \wedge \ldots \wedge \phi_{t_1}(s) \wedge \phi_{t_2}(s) \wedge \ldots$$
$$\wedge\phi_{h_1}(s) \wedge \phi_{h_2}(s) \wedge \ldots \wedge g_{prec}(s) \wedge \neg\phi_t^{(att)}(s)$$

where $g_{prec}(s)$ is the formula inside the precondition element, grounded on fluents of type $\phi_q(s)$, and $\phi_t^{(att)}(s)$ is the attempt formula for the task in question itself. As will become apparent below, the latter addition prevents the RL learning agent from selecting the same task more than once in a given goal run. The

above formula can be extended to include $\xrightarrow{npr}$ incoming links, omitted here for simplicity; however it must be noted that for these links specifically, we utilize task and goal *attempt* formulae rather than *attainment* ones.

Reward. The total reward calculated for each situation is simply the *current* quality value of what has been designated as o_{top} as per the quality formula:

$$reward(v, s) \leftrightarrow \phi^{(r)}_{o_{top}}(v, s)$$

Probabilities. For each link that connects an effect e with its effect group, introduce $prob(\hat{a}, p, \cdot)$, where $\hat{a}$ is the nature action associated with e, and p is the probability label of the link – independent of situation, hence $\cdot$ instead of s.

OR-decomposition exclusivity. We finally demand that all children of OR-decompositions are connected in pairs with $\xrightarrow{npr}$ links. More formally, let $g_1, g_2, \ldots$ be children (goals or tasks) of an OR-decomposition. Then, assume $g_i \xrightarrow{npr} g_j$ for all $i \neq j$. The additions appear in action precondition axioms for the involved goals, following from the semantics of $\xrightarrow{npr}$.

4 Making Models Executable

4.1 The RL Simulation Components and their Function

So far we have discussed the iStar-RL modeling language and how it can be formalized into a DT-Golog specification. We now describe how the latter specification can serve as a domain simulator usable by software agents implementing arbitrary RL algorithms. We adopt a widely used interface specification for RL simulators, Open AI's Gym framework [1], called *gym.Env*, which offers a small collection of functions that satisfy the RL observe-decide-act interaction pattern. Specifically, the most important of *gym.Env*'s functions is:

observation, reward, terminated, info $\leftarrow$ *step(action)*

The function requests the simulator to perform *action*, identified as an integer within a range, and return: (a) the state which results from the performance of *action*, which is encoded in variable *observation* as a state-identifying integer for discrete-space problems or as an array of real values for continuous-space problems, (b) the *reward* obtained for performing the action, and (c) whether the current episode is *terminated*, and (d) other miscellaneous *info*. Our goal is, hence, to use the formalized iStar-RL model to implement *step* (along with other auxiliary functions), so that the latter is compliant with the goal model, allowing thereby accurate simulation of the high-level iStar-RL model.

To achieve this we introduce two software components. The *Query Engine* (*QE*) offers a number of functions that answer queries about the domain relative to a given a history of actions, such as what fluents are true and what actions are possible. These services are used by *GMEnv*, a second component which implements the *gym.Env* standard. *GMEnv* maintains information about the current state of execution of a simulated episode, and executes actions or relays information as per the requests of the client environment. The latter is an arbitrary RL agent, implementing some RL learning algorithm (e.g, A2C [41], Deep Q Network (DQN) [42], etc.) and requiring the *gym.Env* interface for its training. We examine the development of *QE* and *GMEnv* in sequence.

4.2 The Query Engine (QE)

QE offers a set of functions whereby the iStar-RL model can be queried with respect to an effect history eH. The supported functions are listed in Table 1 and are implemented as logic programs in Prolog and in accordance to DT-Golog semantics also seen in the table. Below we present these semantics in more detail.

Extracting State Information. Consider the set $\mathcal{Q} = q_1, q_2, \ldots$ of domain predicates in the goal model and an effect history $eH = [e_1, e_2, \ldots]$. Define also list $L_{\mathcal{Q}} = [ql_1, ql_2, \ldots]$ where $ql_i = 1$ if the predicate q_i is satisfied after effect history eH has been observed, and $ql_i = 0$ otherwise. For discrete-state problems, $L_{\mathcal{Q}}$ offers a representation of discrete exported state and it is easily translatable to an integer identifier. For continuous-state problems, list $L_{\mathcal{O}} = [ol_1, ol_2, \ldots]$, where $ol_i = satQ(o_i, eH)$, represents continuous exported state for the goal model after eH for continuous exported state set $\{o_1, o_2, \ldots\} \subseteq \mathcal{O}$.

The semantics of $L_{\mathcal{Q}}$ and $L_{\mathcal{O}}$ in DT-Golog terms are understood as follows. Recall that there is a 1-1 correspondence between domain predicates $\mathcal{Q} = \{q_1, q_2, \ldots\}$ and DT-Golog fluents $\Phi_{\mathcal{Q}} = \{\phi_{q_1}, \phi_{q_2}, \ldots\}$, as well as between an effect history eH and a DT-Golog situation s. The following rule then defines $L_{\mathcal{Q}}$ in terms of situation s:

$$getState(s, L_{\mathcal{Q}}) \leftrightarrow (\phi_{q_1}(s) \wedge (ql_1 = 1) \vee \neg\phi_{q_1}(s) \wedge (ql_1 = 0)) \wedge (\phi_{q_2}(s) \wedge (ql_2 = 1) \vee \neg\phi_{q_2}(s) \wedge (ql_2 = 0)) \wedge ...$$

Thus, $getState(s, L_{\mathcal{Q}})$ holds when binary list $L_{\mathcal{Q}}$ captures the truth value of every fluent in situation s. The predicate is the semantics of *QE* function `getState(eH): bit[]` seen in entry 5 of Table 1. For continuous exported states, recall that qualities $o \in \mathcal{O}$ in the goal model are associated with fluents of the form $\phi_o^{(r)}(v, s)$ in which v is a real value representing o's satisfaction $satQ(o, eH)$. Hence, for continuous exported state set $\{o_1, o_2, \ldots\} \subseteq \mathcal{O}$:

$$getContState(s, L_{\mathcal{O}}) \leftrightarrow \phi_{o_1}^{(r)}(v_1, s) \wedge (ol_1 = v_1) \wedge \phi_{o_2}^{(r)}(v_2, s) \wedge (ol_2 = v_2) \wedge ...$$

The predicate effectively maps a situation s with the value of fluents representing the qualities included in the exported state set, and constitutes the semantics of *QE* function `getContState(eH):float[]` – entry 6 of Table 1.

Episodes and Goal Runs. We now express the semantics of goal runs and episodes in terms of DT-Golog constructs. Recall that a task history is a goal run iff the root goal is satisfied or it leads to a deadlock. Recall also that for goal h, $satG(h, eH)$ iff $\phi_h(s)$, where s is the situation corresponding to history eH. Thus, root goal $r_{\mathcal{G}}$ is satisfied iff $\phi_{r_{\mathcal{G}}}(s)$. Secondly, to decide if a situation s is a deadlock we examine if any of the action precondition axioms allow the performance of any task at s. Let $\mathcal{N}_{\mathcal{G}}$ to be the set of all nature actions $\hat{a}$ (each corresponding to an effect $e \in \mathcal{E}_{\mathcal{G}}$), a situation s is a deadlock according to the following definition:

$$deadlock(s) \equiv \forall \hat{a} \in \mathcal{N}_{\mathcal{G}}. \neg poss(\hat{a}, s)$$

where $poss(\hat{a}, s)$ are, as we saw, the left-hand sides of action precondition axioms. Given the above, we can now define predicate $done(s)$ (entry 7 of Table 1) that holds when a situation s is a complete run with respect to root goal $r_{\mathcal{G}}$:

$$done(s) \equiv \phi_{r_{\mathcal{G}}}(s) \vee deadlock(s)$$

Cross-episode Elements. Recall from Figure 2 that in the initial situation S_0, qualities are assigned an initial value v_{init}, while all other predi-

	QE Function	Semantics		QE Function	Semantics
1	`possibleAt(t,eH): bool`	$poss(a_t, s)$	7	`done(eH):bool`	$done(s) \equiv \phi^{att}_{r_\mathcal{G}} \vee deadlock(s)$
2	`getOutcomes(t): integer[]`	$choice(a_t, \{\hat{a}_t^{(1)}, \hat{a}_t^{(2)}, ...\})$	8	`deadlock(eH): bool`	$\forall \hat{a} \in \mathcal{N}_\mathcal{G}. \neg poss(\hat{a}, s)$
3	`getProbs(t,eH): float[]`	$getProbs(a_t, \{p_t^{(1)}, p_t^{(2)}, ...\}, s)$	9	`getCrState(eH): float/bool[]`	$crossState(L, s)$
4	`reward(eH): float`	$reward(s, r)$	10	`setCrState(X)` (`X`: *initializations*)	assert(X)
5	`getState(eH):bit[]`	$getState(s, L_\mathcal{Q})$	11	`rootAchieved(eH): bool`	$\phi_{r_\mathcal{G}}(s)$
6	`getContState(eH): float[]`	$getConState(s, L_\mathcal{O})$			

Table 1: Query Engine (QE) functions and their semantics. The functions assume a mapping of nature and stochastic actions to integers, hence `t` and `eH` are respectively an integer and an array thereof.

cates/fluents are assumed to be false. *QE* allows the client environment to both assert these initial values and to read the values at a given situation s. This is useful for implementing cross-episode elements within multi-run episodes. Let $L = \{ol_1, ol_2, \ldots, ql_1, ql_2, \ldots\}$ represent the values of all cross-episode qualities $o_1, o_2, \ldots$ and propositions $q_1, q_2, \ldots$, where $ol_i = v$ iff $\phi_{o_i}(v, s)$ and $ql_i = 1$ iff $\phi_q(s)$, 0 otherwise. Predicate $crossState(s, L)$ (the semantics of *QE*'s `getCrState` – see entry 9 of Table 1) allows extraction of cross-state information given situation s corresponding to effect history eH:

$crossState(s, L) \leftrightarrow \phi^{(m)}_{o_1}(v_1, s) \wedge (ol_1 = v_1) \wedge \phi^{(m)}_{o_2}(v_2, s) \wedge (ol_2 = v_2) \wedge \ldots$
$[\phi_{q_1}(s) \wedge (ql_1 = 1) \vee \neg\phi_{q_1}(s) \wedge (ql_1 = 0)] \wedge [\phi_{q_2}(s) \wedge (ql_2 = 1) \vee \neg\phi_{q_2}(s) \wedge (ql_2 = 0)] \wedge \ldots$

Dynamically defining initial states is a matter of asserting in the specification, using `setCrState` the list of initializations $\{\phi_{o_1}(v'_1, s_0),\ \phi_{o_2}(v'_2, s_0), \ldots, \Phi^T_q(s_0)\}$, where v'_i are the desired initial values, $\Phi^T_q(s_0)$ the subset of fluents representing propositions that are true in s_0 – see entry 10 of Table 1. Thanks to the above two functions, *GMEnv* can implement multi-run episodes by reading the values of cross-episode elements $o_1, o_2, \ldots, q_1, q_2, \ldots$ at the end of a run using `getCrState` and setting these as the initial values of the next episode using `setCrState`.

Other predicates. The query engine offers additional functions, which can be viewed on Table 1, along with their DT-Golog semantics. One of them:

$getProbs(a_t, \{p_1, p_2, \ldots\}, s) \leftrightarrow choice(a_t, \{\hat{a}_t^{(1)}, \hat{a}_t^{(2)}, \ldots\}) \wedge$
$prob(\hat{a}_t^{(1)}, p_1, s) \wedge prob(\hat{a}_t^{(2)}, p_2, s) \wedge \ldots$

uses $choice(a_t, \hat{A})$ to collect probabilities $\{p_1, p_2, \ldots\}$ for all nature actions $\hat{A} = \{\hat{a}_t^{(1)}, \hat{a}_t^{(2)}, \ldots\}$ associated with task t's stochastic action a_t.

4.3 The GMEnv component

GMEnv implements *gym.Env* through utilizing *QE*'s services. While *QE* is stateless, *GMEnv* maintains episode information including the history of tasks that have been attempted from the beginning of an episode, the history of effects that has occurred upon such performance, the run count since the episode's beginning, as well as the state after the performance of the latest action.

Algorithm 1 Implementation of the Step function of GMEnv.

```
1: GLOBAL: eH,tH = []                                  ▷ run-wide effect and task history
2:         eH_Ep,tH_Ep =[]                             ▷ episode-wide effect and task history
3:         curRun = 0                                  ▷ the goal run under current consideration
4:         penaltyReward                               ▷ default penalty for infeasible actions
5:         qe                                          ▷ reference to a QE implementing object
6: function STEP(t)
7:    if qe.possibleAt(t,eH) then
8:       E_t = qe.getOutcomes(t)                      ▷ E_t : list of effects
9:       P_t = qe.getProbs(t,eH)                      ▷ P_t: list of effect probabilities
10:      e_t = pickRndAction(E_t, P_t)                ▷ randomly pick effect
11:      eH = append(eH,e_t)                          ▷ append effect to history
12:      tH = append(tH,t)                            ▷ append task to history
13:      reward = qe.reward(eH)                       ▷ retrieve reward
14:      state = qe.getState(eH)                      ▷ retrieve new discrete state
15:      c_state = qe.getConState(eH)                 ▷ optional: retrieve
                                                        new continuous state
16:      n_state = bitToInt(State)                    ▷ bit array to int
17:   else
18:      reward = penatlyReward                       ▷ penalize infeasible action
19:   end if
20:   if qe.rootAchieved(eH) then
21:      X = constructInitClauses(qe.getCrState(eH))
22:      qe.setCrState(X)
23:      eH_Ep.append(eH),tH_Ep.append(tH)
24:      eH,tH = []
25:      curRun = curRun + 1
26:   end if
27:   return n_state, reward, DONE, [c_state]
28: end function
29:
30: function DONE
31:   return ((curRun == N) or qe.deadlock(eH))
32: end function
```

Of the *gym.Env* functions that *GMEnv* implements, the most important is, as we saw, `step(task): state, reward, terminate, info`. Algorithm 1 sketches the implementation of the function. The function is iteratively called by the RL agent, with parameter a task `t` of its choosing. Upon its call, `step` checks first if the task is feasible given the current history of effect occurrences, stored in `eH`, through the use of *QE*'s `possibleAt(t,eH)`. If yes, through `getOutcomes(t)` it retrieves a list of possible effects $\mathtt{E_t}$ that `t` may have and through `getProbs(t,eH)` their probabilities $\mathtt{P_t}$. A random choice on the basis of the probabilities is made and both task `t` and the chosen effect $\mathtt{e_t}$ are appended to the corresponding lists, `tH` and `eH` respectively. The state resulting from the performance of `t` is calculated through `getState(eH)`, which translates a history of effect occurrences to an array of bits representing the truth values of domain

	Model Characteristics				Learning Tests									
					Learning Reward			Training Steps			Training Time (s)			
	Model	Size	Run #	State Space	A2C	DQN	PPO	A2C	DQN	PPO	A2C	DQN	PPO	Rnd.*
Discrete	Material Ordered	1,12	1	2^6	0.924	0.924	0.924	10K	10K	10K	6.813	5.000	24.00	0.771
			2	2^{12}	1.711	1.712	1.715	10K	10K	10K	106.2	4.469	113.9	1.465
	Product Manuf.	2,20	1	2^{10}	0.477	0.480	0.465	10K	150K	20K	81.97	848	147.8	-78.42
			2	2^{20}	0.946	0.705	0.710	10K	80K	20K	8,123	23.49K	20.32K	-90.95
	Organize Travel	3,29	1	2^{14}	0.789	0.694	0.788	20K	150K	10K	400.3	1,957	436.7	-94.46
	HVAC Control	1,13	4	2^{16}	-1.36	-1.37	-1.36	10K	70K	10K	585	702.4	724.2	-5.000
Conts.	Product Manuf.	2,20	1	$\mathbb{R}^2$	0.456	0.472	0.048	50K	80K	50K	778.4	972.9	181.3	-78.42
			2	$\mathbb{R}^2$	0.286	0.667	0.325	70K	4.5M	70K	62.59	30,379	343.5	-90.95
	HVAC Control	1,13	4	$\mathbb{R}^2$	-1.36	-1.36	-1.37	10K	110K	10K	709.5	390.1	769.8	-5.000

Table 2: Training with off-the-shelf RL agents. Times in CPU seconds. **Rnd*** excludes deadlock penalty. Model size n, m: n is total # of OR-nodes, m total # of elements.

propositions. An integer representation of the bit array, `n_state`, is, in turn, returned to the calling environment as per the *gym.Env* requirements. Likewise, `getConState(eH)` is called to retrieve any continuous exported state set.

If task `t` is not feasible at `eH`, `step` does not proceed with any changes to history lists and state, but may, based on user configuration, result in a negative reward to bias the learning procedure against performance of the task in the specific state. On the other hand, if the goal is achieved, the current run has concluded at `eH`. Consequently (a) the cross-run elements at `eH` are retrieved and reasserted as initial state for the next run, (b) the action and effect lists are added to the episode-wide record and (c) reset, and (d) the run counter increases by one. As we saw above, the episode is `done` if the root of the Nth goal run has been achieved or a deadlock has been detected.

Finally, the second important *gym.Env* function to be implemented by *GMEnv* `reset()`, simply empties the history lists and resets state to its initial values.

5 Evaluation

5.1 Objectives and Methodology

We perform an empirical evaluation of the proposed language and simulator generation technique aimed at answering two research questions:

Q1: Do the generated simulators correctly simulate the goal model with respect to the resulting DT-Golog semantics?

Q2: Is the framework usable by existing RL algorithms?

To answer these questions we first develop a number of complete goal models. Then, to address Q1 we perform *simulation tests*. Specifically, for discrete-state models, we calculate the optimal policy using exact techniques, specifically the DT-Golog solver, and verify if repeated executions of the same policy using the simulator yields an average reward that is close to the calculated one. Presence of deviations would imply errors in theoretical or implementation aspects of

	Model Characteristics				Learning Tests									
					Learning Reward			Training Steps			Training Time (s)			
	Model	Size	Run #	State Space	A2C	DQN	PPO	A2C	DQN	PPO	A2C	DQN	PPO	Rnd.*
Discrete	Material Ordered	1,12	1	2^6	0.924	0.924	0.924	10K	10K	10K	6.813	5.000	24.00	0.771
			2	2^{12}	1.711	1.712	1.715	10K	10K	10K	106.2	4.469	113.9	1.465
	Product Manuf.	2,20	1	2^{10}	0.477	0.480	0.465	10K	150K	20K	81.97	848	147.8	-78.42
			2	2^{20}	0.946	0.705	0.710	10K	80K	20K	8,123	23.49K	20.32K	-90.95
	Organize Travel	3,29	1	2^{14}	0.789	0.694	0.788	20K	150K	10K	400.3	1,957	436.7	-94.46
	HVAC Control	1,13	4	2^{16}	-1.36	-1.37	-1.36	10K	70K	10K	585	702.4	724.2	-5.000
Conts.	Product Manuf.	2,20	1	$\mathbb{R}^2$	0.456	0.472	0.048	50K	80K	50K	778.4	972.9	181.3	-78.42
			2	$\mathbb{R}^2$	0.286	0.667	0.325	70K	4.5M	70K	62.59	30,379	343.5	-90.95
	HVAC Control	1,13	4	$\mathbb{R}^2$	-1.36	-1.36	-1.37	10K	110K	10K	709.5	390.1	769.8	-5.000

Table 3: Training with off-the-shelf RL agents. Times in CPU seconds. **Rnd*** excludes deadlock penalty. Model size n, m: n is total # of OR-nodes, m total # of elements.

the proposal, i.e. the developed components would not properly simulate the modeled domains.

To answer Q2, we perform *learning tests*. Specifically, we utilize third-party RL agent implementations on the models and observe: (a) whether training and testing is at all possible, and (b) if any of those techniques identify the optimal policy that the DT-Golog solver outputs or, failing that, at least constitute an improvement compared to a random policy. Failure to have at least some of the algorithms identify the optimal in at least some of the models would indicate the possibility of erroneous conceptualization or implementation.

The models against which we perform the tests are chosen to allow tractable exact reasoning using the DT-Golog interpreter and manual verification of outputs. We particularly consider: (i) a simple single decision problem with one goal decomposition *Material Ordered* corresponding to the left subgoal of Figure 1(a) for one run and two runs, (ii) the complete model of Figure 1(a) in single-run and two-run configurations, (iii) a travel arrangement problem reproduced as-is from the literature [37], (iv) a four-run version of the HVAC controller problem discussed earlier (Figure 1(b)).

A complete reproducibiliy package including implementation of *GMEnv* and the *Query Engine* can be found at [53].

5.2 Results

The outcomes of the tests can be found in Tables 4 and 3. Starting from the simulation test results of Table 4, which could only be meaningfully run for the discrete-space cases, in all cases, the reward generated by 10,000 iterations of the DT-Golog optimal policy ("Simulation Reward" in the table) was very close to the optimal reward ("DT-Golog Reward"), and higher than the one generated through execution of a random policy ("Random Reward").

For the learning tests, Table 3 reports the expected rewards (as per 5,000 episode simulations) of policies learned through the application of three different algorithms, along with the number of training steps required and the CPU time

Model	Model Size	Run #	State Size	DT-Golog Reward	Simulation Reward	Random Reward
Order Material	1,12	1	2^6	0.924	0.925	0.773
		2	2^{12}	1.708	1.715	1.466
Build Roof	2,20	1	2^{10}	0.476	0.482	0.058
		2	2^{20}	0.928	0.938	0.065
Organize Travel	3,29	1	2^{14}	0.791	0.793	0.059
HVAC Control	1,13	4	2^{16}	-1.361	-1.359	-5.041

Table 4: Simulating DT-Golog calculated optimal and random policy (discrete state). Model size numbers n, m denote that the goal model has n OR-decompositions and m elements overall, DT-Golog reward is the expected reward of the optimal policy calculated by the model-based reasoner, Simulation Reward is the average of the rewards of 10,000 stochastic executions of that optimal policy and Random Reward is the corresponding average for random choices of actions.

in seconds on an Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 4 Core(s), 16Gb RAM computer. Of the four model cases initially tested as discrete state space problems, two are also tested with continuous state space, consisting of two real-valued variables, and the results are presented at the bottom three rows. The algorithms attempted are Advantage Actor Critic (A2C) [41], Deep Q Network (DQN) [42] and Proximal Policy Optimization (PPO) [56], as implemented for Stable-Baselines3 [51] and using default parameters. The algorithms were chosen based on their ability to deal with both discrete and continuous state problems.

In all cases, a series of training sessions was attempted starting from 10,000 steps (task trials) and gradually increased until the learned policy approached the DT-Golog one (where available) or substantially improved the corresponding random. For discrete problems, the trained RL agent yields an average reward that is consistently very close to the DT-Golog optimal, and substantially departs from random. For continuous problems, where there is no optimal standard to compare with, we look for improvements compared to the average reward of the random policy, which we indeed observe. To allow for fair comparison, in Table 3 rewards under random policy ("Rnd.*") account for a user-defined -100 penalty provided by the simulator in cases an action leads to a deadlock; such penalties are excluded in the "Rnd." column of table 4.

Overall, the outcome of this preliminary evaluation offers support for the applicability of the proposed approach. In response to RQ1, the alignment between simulation and DT-Golog output constitutes evidence that the simulator accurately simulates the DT-Golog domain. Further, a third-party RL training implementation is able to improve the random reward and, in most cases, reach the optima with unremarkable training effort. Thus not only the generated simulators appear to be usable by RL training agents (RQ2), but also small problems can readily be solved through off-the-shelf agents with default parameters without intense training and engineering effort (e.g., hyperparameter tuning, specialized algorithms, advanced hardware configurations etc.). Further

the three algorithms perform differently both in terms of optimization outcome and in terms of numbers of steps and time required. For example, DQN turns out to train slower than the other two algorithms given our hardware and the hyper-parameter configuration, which prompts exploration as to why this is the case and how it would affect real application. This seems to support the usefulness of the toolset for assessing the feasibility of various training techniques.

6 Related Work

Substantial interest has developed over the past few years in studying the intersection of conceptual modeling (CM) with artificial intelligence (AI), a space referred to as CMAI [9, 10]. The latter describes both the application of AI to support CM tasks (AI4CM), and reversely, the application of CM to systematize, streamline, and support various qualities AI-intensive systems (CM4AI). In the AI4CM context, machine learning has been used in tasks such as supporting extraction of models or schemas from data (e.g., [55, 64–66]), supporting model transformation [13, 23], classification of models or repositories thereof [40, 47], model auto-completion [21], or assessment of realism of generated models [39]. In the CM4AI space, where our work finds a natural fit, the idea of using models to organize and support the ML development pipeline from input collection and preparation to training and inference, has been underlined [17]. Here, models have been proposed for quality assuring the AI/ML process [33], detecting bias [62], supporting meta-learning [29], chatbot generation [50] or generation of neural architectures [35].

The problem of systematically devising AI solutions has also been studied from the RE point of view. Major theme in that context is the quality of the end-result [28], with an emphasis on explainability [12,14,15], how such systems can be specified [8,59], and how the RE process can be organized [49]. In search for a solution to the *RE4AI* [2] problem, Nalchigar et al. [44,45] propose a goal-oriented conceptual framework for expressing machine learning requirements and designs organized around three main modeling views (business, analytics design, and data preparation). Ahmad et al. [2] review, among other things, modeling approaches for conducting RE for AI, to find that Goal Oriented RE languages, along with UML/SySML, are the most popular for modeling in that space.

Goal models have indeed been known to be effective vehicles for allowing formal reasoning about requirements and designs in various ways [19, 20, 25, 30, 36].This capability of goal models has been utilized also in the area of adaptive systems [5, 7, 24] and multi-agent systems [26]. The modeling approach proposed here, iStar-RL, is heavily based on an iStar dialect proposed for modeling decision-theoretic domains and, through formalization, perform search-based reasoning thereof [37, 38]. The strength of the modeling approach we adopt, compared to common approaches for modeling probabilistic transition systems (e.g., [32]) lies in the combined ability of goal models both to concisely encode high-variability processes and behaviors, stemming from overarching stakeholder goals, and to compare variants vis-à-vis intricate, multi-dimensional qual-

ity requirements structures [43]. Hence, efforts such as on probabilistic logic shields [61] using ProbLog [34], or on using DT-Golog for Q-learning [6], indicate promising destinations of iStar-RL transformations. Probabilistic reasoning using conceptual models other than goal models, such as BPMN [48], has also been proposed [22, 31], with not clear connection to RL however.

7 Concluding Remarks and Future Work

We presented a framework for goal-oriented modeling and generation of RL simulators. Through an extension to a standard goal modeling language, designers can systematically identify the action, effect, and state spaces of the environments to be simulated through analysis of agent goals, while utilizing quality contribution structures for designing reward functions. Models constructed using the proposed language are subsequently translated to a formal specification, which is used by a set of interpreting components to allow step-wise execution of the model. Such step-wise execution lies at the heart of the RL process, hence the result is directly usable by a variety of RL algorithms. The framework is aimed at supporting investigations of the feasibility and performance of different training techniques against different action, reward, and probability models, and, when these models are deemed accurate, identifying a directly usable optimal or near-optimal policy when dealing with continuous state spaces or, potentially, with problems that are too large for model-based techniques to solve. The approach is, hence, promising for systematically developing adaptive high-variability systems, such as adaptive business processes or behaviorally customizable software, while following well-studied goal oriented requirements engineering processes.

Future research effort can be dedicated towards exploring how the goal model representations can assist the learning process. Of particular interest is the exploration of novel RL algorithms that are allowed partial access to the domain specification (e.g., the precondition and contribution structures, or some presumed probability intervals) to be used for informing the training process. Further, potential may exist in applying some version of our proposal to the problems of AI safety and explainability. Firstly, the framework ensures that AI agent training is based on interaction data stemming from well-studied and well-behaved models, arguably confining the learned behavior to what is permitted by the model that trained it, or, conversely using the model as the source of safety standards (e.g., non violation of temporal constraints) for the trained agent. Secondly, by being faithful to i^*'s fundamental principle of centering the analysis around action and decision rationale, the iStar-RL model can serve as a suitable explanatory device for the respective actions and decisions performed by the acting RL agent that was trained against the model. Applications and case studies in various problem domains can be helpful in assessing the utility of the approach in these areas.

References

1. Open AI Gym (2022), `https://github.com/openai/gym`

2. Ahmad, K., Bano, M., Abdelrazek, M., Arora, C., Grundy, J.: What's up with Requirements Engineering for Artificial Intelligence Systems? In: Proceedigns of the 29th IEEE International Requirements Engineering Conference (RE). pp. 1–12 (2021). https://doi.org/10.1109/RE51729.2021.00008
3. Amyot, D., Mussbacher, G.: User Requirements Notation: The First Ten Years, The Next Ten Years (Invited Paper). Journal of Software (JSW) **6**(5), 747–768 (2011)
4. Anderson, R.N., Boulanger, A., Powell, W.B., Scott, W.: Adaptive Stochastic Control for the Smart Grid. Proceedings of the IEEE **99**(6), 1098–1115 (2011). https://doi.org/10.1109/JPROC.2011.2109671
5. Angelopoulos, K., Papadopoulos, A.V., Silva Souza, V.E., Mylopoulos, J.: Model Predictive Control for Software Systems with CobRA. In: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'16). pp. 35–46. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2897053.2897054, https://doi.org/10.1145/2897053.2897054
6. Beck, D., Lakemeyer, G.: Reinforcement learning for Golog programs with first-order state-abstraction. Logic Journal of the IGPL **20**(5), 909–942 (2012)
7. Bencomo, N., Belaggoun, A.: Supporting Decision-Making for Self-Adaptive Systems: From Goal Models to Dynamic Decision Networks. In: Doerr, J., Opdahl, A.L. (eds.) Requirements Engineering: Foundation for Software Quality. pp. 221–236. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
8. Berry, D.M.: Requirements Engineering for Artificial Intelligence: What Is a Requirements Specification for an Artificial Intelligence? In: Gervasi, V., Vogelsang, A. (eds.) Proceedings of the 28th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2022). pp. 19–25. Springer International Publishing, Cham (2022)
9. Bork, D., Ali, S.J., Roelens, B.: Conceptual Modeling and Artificial Intelligence: A Systematic Mapping Study. Conceptual Modeling and Artificial Intelligence: A Systematic Mapping Study **1**(1), 27 (2023), https://arxiv.org/abs/2303.06758v1
10. Bork, D., Fettke, P., Maass, W., Reimer, U., Schuetz, C.G., Tropmann-Frick, M., Yu, E.S.: 1st Workshop on Conceptual Modeling Meets Artificial Intelligence and Data-Driven Decision Making (CMAI 2020). In: Grossmann, G., Ram, S. (eds.) Advances in Conceptual Modeling. ER 2020 Workshops CMAI, CMLS, CMOMM4FAIR, CoMoNoS, EmpER. Vienna, Austria (2020)
11. Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In: Proceedings of the 17th Conference on Artificial Intelligence (AAAI-00). pp. 355–362. AAAI Press, Austin, TX (2000), http://www.cs.toronto.edu/cogrobo/Papers/dtgologaaai00.ps.Z
12. Brunotte, W., Chazette, L., Klös, V., Speith, T.: Quo Vadis, Explainability? – A Research Roadmap for Explainability Engineering. In: Gervasi, V., Vogelsang, A. (eds.) Requirements Engineering: Foundation for Software Quality. pp. 26–32. Springer International Publishing, Cham (2022)
13. Burgueño, L., Cabot, J., Gérard, S.: An LSTM-Based Neural Network Architecture for Model Transformations. Proceedings - 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems, MODELS 2019 pp. 294–299 (2019). https://doi.org/10.1109/MODELS.2019.00013
14. Chazette, L., Brunotte, W., Speith, T.: Exploring Explainability: A Definition, a Model, and a Knowledge Catalogue. In: Proceedings fo the 29th IEEE International

Requirements Engineering Conference (RE'21). pp. 197–208 (2021). https://doi.org/10.1109/RE51729.2021.00025
15. Chazette, L., Schneider, K.: Explainability as a non-functional requirement: challenges and recommendations. Requirements Engineering **25**(4), 493–514 (2020). https://doi.org/10.1007/s00766-020-00333-1, https://doi.org/10.1007/s00766-020-00333-1
16. Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 Language Guide. The Computing Research Repository (CoRR) **abs/1605.0** (2016), http://arxiv.org/abs/1605.07767
17. Damiani, E., Frati, F.: Towards Conceptual Models for Machine Learning Computations. In: Trujillo, J.C., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (eds.) Conceptual Modeling. pp. 3–9. Springer International Publishing, Cham (2018)
18. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computer Programming **20**(1-2), 3–50 (1993)
19. Dell'Anna, D., Dalpiaz, F., Dastani, M.: Validating Goal Models via Bayesian Networks. In: Proceedings of the 5th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE 2018). pp. 39–46 (2018). https://doi.org/10.1109/AIRE.2018.00012
20. Dell'Anna, D., Dalpiaz, F., Dastani, M.: Requirements-driven evolution of sociotechnical systems via probabilistic reasoning and hill climbing. Automated Software Engineering **26**(3), 513–557 (2019). https://doi.org/10.1007/s10515-019-00255-5, https://doi.org/10.1007/s10515-019-00255-5
21. Di Rocco, J., Di Sipio, C., Di Ruscio, D., Nguyen, P.T.: A GNN-based Recommender System to Assist the Specification of Metamodels and Models. Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS 2021 pp. 70–81 (2021). https://doi.org/10.1109/MODELS50736.2021.00016
22. Durán, F., Rocha, C., Salaün, G.: Stochastic analysis of BPMN with time in rewriting logic. Science of Computer Programming **168**, 1–17 (2018). https://doi.org/https://doi.org/10.1016/j.scico.2018.08.007, https://www.sciencedirect.com/science/article/pii/S0167642318303307
23. Eisenberg, M., Pichler, H.P., Garmendia, A., Wimmer, M.: Towards Reinforcement Learning for In-Place Model Transformations. Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS 2021 pp. 82–88 (2021). https://doi.org/10.1109/MODELS50736.2021.00017
24. Félix Solano, G., Diniz Caldas, R., Nunes Rodrigues, G., Vogel, T., Pelliccione, P.: Taming Uncertainty in the Assurance Process of Self-Adaptive Systems: a Goal-Oriented Approach. In: Proceedings of the 14th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'19). pp. 89–99 (may 2019). https://doi.org/10.1109/SEAMS.2019.00020
25. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Formal Reasoning Techniques for Goal Models. In: Spaccapietra, S., March, S., Aberer, K. (eds.) Journal on Data Semantics I, pp. 1–20. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39733-5_1, https://doi.org/10.1007/978-3-540-39733-5_1
26. Gonçalves, E., Araujo, J., Castro, J.: iStar4RationalAgents: Modeling Requirements of Multi-agent Systems with Rational Agents. In: Laender, A.H.F., Pernici, B., Lim, E.P., de Oliveira, J.P.M. (eds.) Conceptual Modeling. pp. 558–566. Springer International Publishing, Cham (2019)

27. Gottesman, O., Johansson, F., Komorowski, M., Faisal, A., Sontag, D., Doshi-Velez, F., Celi, L.A.: Guidelines for reinforcement learning in healthcare. Nature Medicine **25**(1), 16–18 (2019). https://doi.org/10.1038/s41591-018-0310-5, https://doi.org/10.1038/s41591-018-0310-5
28. Habibullah, K.M., Horkoff, J.: Non-functional Requirements for Machine Learning: Understanding Current Use and Challenges in Industry. In: Proceedings fo the 29th IEEE International Requirements Engineering Conference (RE'21). pp. 13–23 (2021). https://doi.org/10.1109/RE51729.2021.00009
29. Hartmann, T., Moawad, A., Schockaert, C., Fouquet, F., Le Traon, Y.: Meta-Modelling Meta-Learning. Proceedings - 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems, MODELS 2019 pp. 300–305 (2019). https://doi.org/10.1109/MODELS.2019.00014
30. Heaven, W., Letier, E.: Simulating and optimising design decisions in quantitative goal models. In: Proceedings of the 19th IEEE International Requirements Engineering Conference (RE'11). pp. 79–88. Trento, Italy (2011)
31. Herbert, L.T., Hansen, Z.N.L., Jacobsen, P.: SBOAT: A Stochastic BPMN Analysis and Optimisation Tool. In: Karlaftis, M.G., Lagaros, N.D., Papadrakakis, M. (eds.) Proceedings of the 1st International Conference on Engineering and Applied Sciences Optimization (OPT-i). pp. 1136–1152. National Technical University of Athens (2014), http://www.opti2014.org/
32. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A Tool for Automatic Verification of Probabilistic Systems. In: Hermanns, H., Palsberg, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science (LNCS), vol. 3920, pp. 441–444. Springer Berlin/Heidelberg (2006)
33. Ishikawa, F.: Concepts in Quality Assessment for Machine Learning - From Test Data to Arguments. In: Trujillo, J.C., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (eds.) Conceptual Modeling. pp. 536–544. Springer International Publishing, Cham (2018)
34. Kimmig, A., Demoen, B., De Raedt, L., Costa, V.S., Rocha, R.: On the implementation of the probabilistic logic programming language ProbLog. Theory and Practice of Logic Programming **11**(2-3), 235–262 (2011). https://doi.org/DOI:10.1017/S1471068410000566, https://www.cambridge.org/core/product/21037609B99F5DC8033DDF56D07BF839
35. Kusmenko, E., Nickels, S., Pavlitskaya, S., Rumpe, B., Timmermanns, T.: Modeling and Training of Neural Processing Systems. In: Proceedings of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS 2019). pp. 283–293 (2019). https://doi.org/10.1109/MODELS.2019.00012
36. Letier, E., van Lamsweerde, A.: 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). pp. 53–62. ACM Press, Newport Beach, CA (nov 2004), http://www2.info.ucl.ac.be/people/eletier/
37. Liaskos, S., Khan, S.M., Mylopoulos, J.: Modeling and reasoning about uncertainty in goal models: a decision-theoretic approach. Software & Systems Modeling **21**, 1–24 (2022). https://doi.org/https://doi.org/10.1007/s10270-021-00968-w
38. Liaskos, S., Khan, S.M., Soutchanski, M., Mylopoulos, J.: Modeling and Reasoning with Decision-Theoretic Goals. In: Proceedings of the 32th International Conference on Conceptual Modeling, (ER'13). pp. 19–32. Hong-Kong, China (2013)

39. Lopez, J.A.H., Cuadrado, J.S.: Towards the Characterization of Realistic Model Generators using Graph Neural Networks. Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS 2021 pp. 58–69 (2021). https://doi.org/10.1109/MODELS50736.2021.00015
40. López, J.A.H., Rubei, R., Cuadrado, J.S., Di Ruscio, D.: Machine Learning Methods for Model Classification: A Comparative Study. Proceedings - 25th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2022 pp. 165–175 (2022). https://doi.org/10.1145/3550355.3552461
41. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: Proceedings of the 33rd International Conference on Machine Learning (ICML'16). pp. 1928–1937. JMLR.org (2016)
42. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236, https://doi.org/10.1038/nature14236
43. Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Exploring Alternatives During Requirements Analysis. IEEE Software **18**(1), 92–96 (2001). https://doi.org/http://dx.doi.org/10.1109/52.903174
44. Nalchigar, S., Yu, E.: Business-driven data analytics: A conceptual modeling framework. Data & Knowledge Engineering **117**, 359–372 (2018). https://doi.org/https://doi.org/10.1016/j.datak.2018.04.006, https://www.sciencedirect.com/science/article/pii/S0169023X18301691
45. Nalchigar, S., Yu, E., Keshavjee, K.: Modeling machine learning requirements from three perspectives: a case report from the healthcare domain. Requirements Engineering **26**(2), 237–254 (2021). https://doi.org/10.1007/s00766-020-00343-z, https://doi.org/10.1007/s00766-020-00343-z
46. Nguyen, C.M., Sebastiani, R., Giorgini, P., Mylopoulos, J.: Multi-objective reasoning with constrained goal models. Requirements Engineering **23**(2), 189–225 (2018). https://doi.org/10.1007/s00766-016-0263-5, https://doi.org/10.1007/s00766-016-0263-5
47. Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Pierantonio, A., Iovino, L.: Automated Classification of Metamodel Repositories: A Machine Learning Approach. Proceedings - 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems, MODELS 2019 pp. 272–282 (2019). https://doi.org/10.1109/MODELS.2019.00011
48. Object Management Group: Business Process Model And Notation (v2.0). Tech. rep. (2011)
49. Pei, Z., Liu, L., Wang, C., Wang, J.: Requirements Engineering for Machine Learning: A Review and Reflection. In: 2022 IEEE 30th International Requirements Engineering Conference Workshops (REW). pp. 166–175 (2022). https://doi.org/10.1109/REW56159.2022.00039
50. Pérez-Soler, S., Guerra, E., de Lara, J.: Model-Driven Chatbot Development. In: Dobbie, G., Frank, U., Kappel, G., Liddle, S.W., Mayr, H.C. (eds.) Proceedigns of the International Conference on Conceptual Modeling (ER 2020). pp. 207–222. Springer International Publishing (2020)
51. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine

Learning Research **22**(268), 1–8 (2021), http://jmlr.org/papers/v22/20-1364.html
52. Rao, A., Jelvis, T.: Foundations of Reinforcement Learning with Applications in Finance. Chapman and Hall/CRC (2022)
53. [Redacted]: Reproducibility package for: Model-driven design and generation of domain simulators for reinforcement learning (anonymized) (2024), https://github.com/for-review-purposes/RLGen
54. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
55. Saini, R., Mussbacher, G., Guo, J.L., Kienzle, J.: Machine learning-based incremental learning in interactive domain modelling. Proceedings - 25th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2022 (C), 176–186 (2022). https://doi.org/10.1145/3550355.3552421
56. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms (2017). https://doi.org/10.48550/ARXIV.1707.06347, https://arxiv.org/abs/1707.06347
57. Soutchanski, M.: High-Level Robot Programming in Dynamic and Incompletely Known Environments. Ph.D. thesis, Department of Computer Science, University of Toronto (2003)
58. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press (2018)
59. Vogelsang, A., Borg, M.: Requirements Engineering for Machine Learning: Perspectives from Data Scientists. In: Proceedigns of the 6th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE 2019). pp. 245–251 (2019). https://doi.org/10.1109/REW.2019.00050
60. Wei, H., Zheng, G., Yao, H., Li, Z.: IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18). pp. 2496–2505. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3219819.3220096, https://doi.org/10.1145/3219819.3220096
61. Yang, W.C., Marra, G., Rens, G., De Raedt, L.: Safe Reinforcement Learning via Probabilistic Logic Shields. IJCAI International Joint Conference on Artificial Intelligence **2023-Augus**, 5739–5749 (2023). https://doi.org/10.24963/ijcai.2023/637
62. Yohannis, A., Kolovos, D.: Towards model-based bias mitigation in machine learning. Proceedings - 25th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2022 pp. 143–153 (2022). https://doi.org/10.1145/3550355.3552401
63. Yu, E.S.K.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97). pp. 226–235. Annapolis, MD (1997)
64. Yuan, G., Lu, J., Yan, Z.: Effective Generation of Relational Schema from Multi-Model Data with Reinforcement Learning. In: Ralyté, J., Chakravarthy, S., Mohania, M., Jeusfeld, M.A., Karlapalem, K. (eds.) Conceptual Modeling. pp. 224–235. Springer International Publishing, Cham (2022)
65. Zhou, Q., Li, T., Wang, Y.: Assisting in Requirements Goal Modeling: A Hybrid Approach based on Machine Learning and Logical Reasoning. Proceedings - 25th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2022 pp. 199–209 (2022). https://doi.org/10.1145/3550355.3552415

66. Zuo, Y., Zhan, M., Zhou, Y., Zhan, P.: Bidirectional Relation Attention for Entity Alignment Based on Graph Convolutional Network. In: Ralyté, J., Chakravarthy, S., Mohania, M., Jeusfeld, M.A., Karlapalem, K. (eds.) Conceptual Modeling. pp. 295–309. Springer International Publishing, Cham (2022)